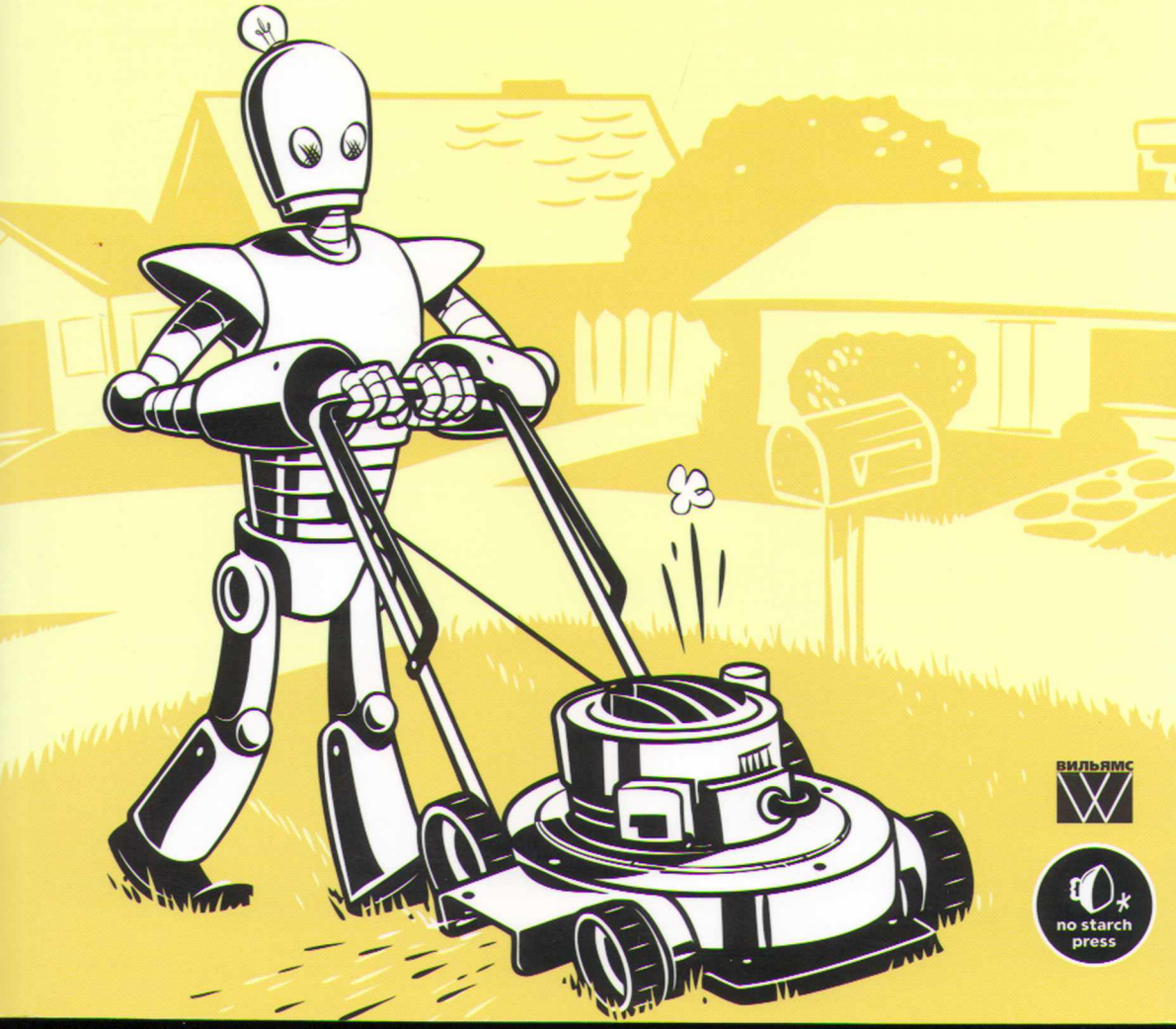
АВТОМАТИЗАЦИЯ РУТИННЫХ ЗАДАЧ С ПОМОЩЬЮ PYTHON

ПРАКТИЧЕСКОЕ РУКОВОДСТВО ДЛЯ НАЧИНАЮЩИХ

ЭЛ СВЕЙГАРТ



ВИЛЬЯМС

no starch press

AUTOMATE THE BORING STUFF WITH PYTHON

Practical Programming for Total Beginners

by Al Sweigart

no starch
press

San Francisco

АВТОМАТИЗАЦИЯ РУТИННЫХ ЗАДАЧ С ПОМОЩЬЮ PYTHON

Практическое руководство для начинающих

Эл Свейгарт

Москва • Санкт-Петербург • Киев
2017

ББК 32.973.26-018.2.75
С24
УДК 681.3.07

Издательский дом "Вильямс"
Главный редактор *С.Н. Тригуб*
Зав. редакцией *В.Р. Гинзбург*
Перевод с английского и редакция канд. хим. наук *А.Г. Гузикевича*

По общим вопросам обращайтесь в Издательский дом "Вильямс" по адресу:
info@williamspublishing.com, http://www.williamspublishing.com

Свейгарт, Эл.
С24 Автоматизация рутинных задач с помощью Python: практическое руководство для начинающих.: Пер. с англ. – М.: ООО "И.Д. Вильямс", 2017. – 592 с.: ил. – Парал. тит. англ.
ISBN 978-5-8459-2090-4 (рус.)

ББК 32.973.26-018.2.75

Научно-популярное издание

Эл Свейгарт

Автоматизация рутинных задач с помощью Python: практическое руководство для начинающих

Литературный редактор *Л.Н. Красножон*
Верстка *Л.В. Чернокозинская*
Художественный редактор *В.Г. Павлютин*
Корректоры *Л.А. Гордиенко*

Подписано в печать 05.08.2016. Формат 70x100/16
Усл. печ. л. 47,73. Уч.-изд. л. 27,4.
Тираж 400 экз. Заказ № 8141

Отпечатано в АО "Первая Образцовая типография"
Филиал "Чеховский Печатный Двор"
142300, Московская область, г. Чехов, ул. Полиграфистов, д.1

ООО "И. Д. Вильямс", 127055, г. Москва, ул. Лесная, д. 43, стр. 1

ISBN 978-5-8459-2090-4 (рус.)
ISBN 978-1-593-27599-0 (англ.)

ОГЛАВЛЕНИЕ

СОДЕРЖАНИЕ

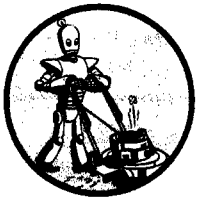
Об авторе

Эл Свейгарт – разработчик ПО, автор книг по программированию, живет в Сан-Франциско. Его любимый язык программирования – Python, для которого он разработал несколько модулей с открытым исходным кодом. Другие его книги доступны на условиях бесплатной лицензии Creative Commons (`http://www.inventwithpython.com`).

О техническом рецензенте

Ари Лаценски разрабатывает приложения для Android и программное обеспечение на языке Python. Живет в Сан-Франциско, где публикует статьи по программированию для Android на сайте `http://gradlewhy.ghost.io` и занимается преподавательской деятельностью в некоммерческой организации "Women Who Code". Ее хобби – игра на гитаре.

ВВЕДЕНИЕ



"За каких-то пару часов ты сделал то, на что у нас троих ушло бы целых три дня". В начале 2000-х годов мой сосед по комнате в колледже работал в магазине электроники. Время от времени они получали электронные таблицы с ценниками своих конкурентов, включающие тысячи наименований. Распечатка одной таблицы представляла собой толстую стопку бумажных листов. Обработкой данных занимались три сотрудника магазина. Они сравнивали цены, указанные в таблице, с ценами в своем магазине и отмечали тот товар, который конкуренты продавали по более низкой цене. На эту работу у них уходило примерно два дня.

"А знаете что? Если вы дадите мне исходный файл таблицы, то я напишу программу, которая выполнит всю работу вместо вас", — сказал мой товарищ, увидев, как они копошатся среди груды разбросанных на полу и сложенных в стопки бумажных листов.

Через пару часов у него была готова небольшая программа, которая читала данные о ценах конкурентов из файла, находила для каждого продукта аналог в базе данных магазина и отмечала весь товар конкурентов, цена которого была ниже. Здесь уместно сказать, что мой товарищ был всего лишь начинающим программистом, и бо́льшую часть этого времени он потратил на просмотр нужных разделов в книге по программированию. Собственно выполнение программы заняло всего лишь несколько секунд, что дало возможность моему товарищу и его коллегам по работе насладиться в тот день удлиненным обеденным перерывом.

Этот пример наглядно демонстрирует всю мощь программирования. Компьютер подобен армейскому ножу, который можно использовать в самых разных ситуациях. Многие люди часами работают за клавиатурой, вводя данные для выполнения повторяющихся задач, и даже не догадываются, что их компьютер, если снабдить его соответствующими инструкциями, способен выполнить ту же работу за считанные секунды.

Для кого предназначена эта книга

В наши дни трудно найти сферу человеческой деятельности, в которой вообще не используется программное обеспечение (ПО). Почти каждый из нас общается в социальных сетях, телефоны многих из нас — это по сути компьютеры, подключенные к Интернету, а бо́льшая часть офисного персонала для выполнения своих функциональных обязанностей нуждается в компьютерной технике. Как следствие, это привело к необычайно высокому спросу на специалистов, способных писать программный код.

Бесчисленные книги по программированию, интерактивные онлайновые руководства, практические семинары для разработчиков – все это направлено на превращение амбициозных новичков в специалистов программной индустрии, заработная плата которых выражается шестизначными числами.

Эта книга предназначена не для них. Она предназначена для всех остальных.

Прочтение одной только этой книги не сможет сделать из вас разработчика-профессионала, точно так же как пяти уроков игры на гитаре вряд ли будет достаточно для того, чтобы стать рок-звездой. Но если вы офисный работник, администратор, преподаватель или вообще один из тех, кто использует компьютер для работы или развлечения, то изучения основ программирования в том объеме, который предлагается в данной книге, вам хватит для автоматизации следующих простых задач:

- перемещение и переименование тысяч файлов и их сортировка по папкам;
- заполнение онлайновых форм без ввода данных вручную;
- загрузка файлов или копирование текста с веб-сайта при его обновлении;
- вывод компьютером заранее подготовленных уведомлений;
- обновление и форматирование электронных таблиц Excel;
- проверка электронной почты и отправка заранее подготовленных ответных писем.

Все эти задачи просты, но отнимают у человека массу времени. Кроме того, зачастую они настолько тривиальны или узкоспециальны, что подыскать какую-то готовую программу для их выполнения не удается. Вооружившись даже минимальными знаниями в области программирования, вы сможете заставить свой компьютер выполнять эти задачи вместо вас.

Исходные предположения

Эта книга – не справочник, а руководство для начинающих. Используемый в ней стиль программирования иногда идет вразрез с принципами наилучшей практики (например, в некоторых программах используются глобальные переменные), но это компромиссное решение, позволяющее сделать код более легким для изучения. Книга предназначена для тех, кому будет достаточно научиться писать простой одноразовый код, поэтому стилю оформления программ и приданию им элегантного вида не уделяется особого внимания. Такие понятия "продвинутого" программирования, как "объектно-ориентированный подход", "списковые включения" и "генераторы", не рассматриваются, дабы не усложнять излагаемый материал.

Возможно, опытные программисты легко укажут в книге те места, где код следовало бы изменить, чтобы сделать его более эффективным, но в этой книге нас в основном заботит создание работоспособных программ с минимальными усилиями.

Что такое программирование

В телевизионных шоу и фильмах часто показывают программистов, пристально вглядывающихся в потоки загадочных нулей и единиц, бегущих по экрану, но современное программирование далеко не столь таинственно. *Программирование* — всего лишь снабжение компьютера инструкциями, приказывающими ему что-либо сделать. Это может быть оперирование числами, изменение текста, поиск информации в файлах или обмен данными с другими компьютерами через Интернет.

Во всех программах в качестве строительных блоков используются элементарные инструкции. Вот как выглядят некоторые из наиболее распространенных инструкций такого рода, если перевести их на обычный человеческий язык.

"Сделай это; затем сделай то".
"Если данное условие удовлетворяется, выполни такое-то действие; в противном случае выполни такое-то действие".
"Выполни это действие такое-то количество раз".
"Продолжай выполнять эти действия до тех пор, пока удовлетворяется данное условие".

Эти строительные блоки можно комбинировать для реализации более сложных решений. В качестве примера ниже приведены инструкции (так называемый *исходный код*) простой программы на языке Python. Программное обеспечение последовательно выполняет каждую строку кода, начиная с первой (при этом некоторые строки выполняются лишь при соблюдении определенных условий, иначе выполняется другая строка), пока не будет достигнут конец программы.

```
❶ passwordFile = open('SecretPasswordFile.txt')
❷ secretPassword = passwordFile.read()
❸ print('Введите пароль.')
  typedPassword = input()
❹ if typedPassword == secretPassword:
❺    print('Доступ разрешен.')
❻    if typedPassword == '12345':
❼        print('Рекомендуем установить более сложный пароль!')
  else:
❽    print('В доступе отказано.')
```

Даже если вы ничего не смыслите в программировании, вы все равно сможете сделать разумные предположения относительно того, что делает этот код, просто читая его. Сначала открывается файл *SecretPasswordFile.txt* ❶, из которого считывается секретный пароль ❷. Затем пользователю предлагается ввести свой пароль (с помощью клавиатуры) ❸. Далее оба пароля сравниваются между собой ❹, и в случае их совпадения программа выводит на экран текст `Доступ разрешен` ❺. Далее программа проверяет, является ли введенный пароль числом `12345` ❻, и, если это так, подсказывает, что такой вариант выбора пароля не является оптимальным ❼. В случае несовпадения паролей программа выводит на экран сообщение `В доступе отказано` ❽.

Что означает название Python

Название *Python* относится как к языку программирования Python (с его собственным синтаксисом, определяющим правила написания корректного кода), так и к интерпретатору Python – программе, предназначенной для чтения исходного кода (написанного на языке Python) и выполнения его инструкций. Различные версии интерпретатора Python, ориентированные на платформы Linux, OS X и Windows, доступны для бесплатной загрузки по адресу `http://python.org`.

Своим названием язык Python обязан вовсе не одноименному виду пресмыкающихся (питон), а комедийной группе из Великобритании "Monty Python"[1], работавшей в жанре сюрреалистического юмора. Программистов на Python шутливо называют *питонистами* (Pythonistas), а многочисленные руководства и документация по языку Python пестрят ссылками как на группу "Monty Python", так и на рептилию.

Программисту вовсе не обязательно в совершенстве знать математику

По моим наблюдениям, наибольшую озабоченность у тех, кто собирается учиться программированию, вызывает то, что, по их мнению, для этого надо хорошо знать математику. На самом деле большинству программистов не нужно знать ничего кроме элементарной арифметики. В этом смысле хорошему программисту понадобится не намного больший объем математических знаний по сравнению с тем, который требуется для решения головоломок судоку.

[1] Читается как "Монти Пайтон". Грамматически правильное звучание названия языка Python – также [пай-тон], однако среди программистов принято произносить его как [пи-тон]. – *Примеч. ред.*

Суть головоломки судоку заключается в заполнении цифрами от 1 до 9 каждого из внутренних квадратов размером 3×3, расположенных на игровом поле размером 9×9, таким образом, чтобы ни одна строка и ни один столбец большого квадрата не содержали повторяющихся цифр. Для нахождения решения необходимо использовать дедуктивный логический метод, исходя из заданной начальной конфигурации цифр. Например, поскольку в головоломке, показанной на рис. 1, цифра 5 находится в левом верхнем углу игрового поля, она не может появиться ни в верхней строке, ни в крайнем слева столбце, ни в левом верхнем квадрате размером 3×3. Последовательное применение подобной логики к строкам, столбцам и внутренним квадратам будет предоставлять вам подсказки, позволяющие заполнять пустые клетки головоломки.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Рис. 1. Головоломка судоку (слева) и ее решение (справа). Несмотря на то что в головоломке используются числа, никаких особых математических знаний для нахождения решения не требуется (изображения предоставлены компанией Wikimedia Commons)

Из того факта, что в головоломке судоку используются числа, вовсе не следует, что для нахождения решения необходимо быть хорошим математиком. То же самое справедливо и в отношении программирования. Как и в судоку, написание программ предусматривает разбиение задачи на ряд отдельных этапов. Аналогичным образом при *отладке программ* (процесс обнаружения и предотвращения возможности возникновения ошибок) вы кропотливо анализируете результаты интересующих вас действий, выполненных программой, пытаясь выявить причину ошибки. И, как это характерно для любого другого вида деятельности, чем больше вы программируете, тем лучше у вас это будет получаться.

Программирование — творческий вид деятельности

Программирование — это вид творчества, несколько напоминающий возведение замков из элементов LEGO. Сначала вы формулируете для себя основные идеи относительно того, что собой должна представлять будущая программа и какие строительные элементы имеются в вашем распоряжении. После этого вы приступаете к построению программы. Завершив построение программы, вы наводите окончательный порядок в своем коде, аналогично тому как по окончании строительства замка принялись бы за уборку его территории.

Различие между программированием и другими творческими видами деятельности заключается в том, что все необходимые исходные материалы находятся в вашем компьютере, и вам не нужно дополнительно закупать какие-либо холсты, краску, пленку, нитки, блоки LEGO или электронные компоненты. Написав программу, вы можете легко поделиться ею через Интернет с целым миром. И даже если в процессе программирования вы будете допускать неизбежные ошибки, это занятие доставит вам массу удовольствия.

Структура книги

В части I книги рассмотрены основы программирования на языке Python, тогда как часть II посвящена различным задачам, решение которых можно автоматизировать. Каждая глава части II включает программные проекты, с которыми вам предстоит работать. Ниже приведено краткое описание глав.

Часть I. Основы программирования на языке Python

- **Глава 1. Основные понятия языка Python.** В этой главе рассматриваются выражения — базовый тип инструкций Python, а также описывается использование интерактивной программной оболочки Python для экспериментирования с кодом.
- **Глава 2. Поток управления.** В этой главе речь идет о том, как заставить программу принимать решения, касающиеся последовательности выполнения инструкций, чтобы код мог самостоятельно реагировать на возникновение различных условий.
- **Глава 3. Функции.** В этой главе показано, как использовать собственные функции для разбиения кода на отдельные части, с которыми проще работать.
- **Глава 4. Списки.** Вводится понятие *списка*, одного из встроенных типов данных Python, и рассказывается о том, как использовать списки для организации данных.

- **Глава 5. Словари и структурирование данных.** Вводится понятие другого встроенного типа данных Python, *словаря*, и демонстрируются более совершенные способы организации данных.
- **Глава 6. Манипулирование строками.** Описываются методы работы с текстовыми данными (которые в языке Python принято называть *строками*).

Часть II. Автоматизация задач

- **Глава 7. Поиск по шаблону с помощью регулярных выражений.** Обсуждаются приемы обработки строк и способы поиска образцов текста, соответствующих заданному шаблону, с помощью регулярных выражений.
- **Глава 8. Чтение и запись файлов.** В этой главе речь идет о том, как организовать в программе чтение данных из текстовых файлов и сохранить информацию на жестком диске.
- **Глава 9. Управление файлами.** Рассматриваются автоматизированные способы копирования, перемещения, переименования и удаления файлов, позволяющие выполнять данные операции быстрее, чем это можно сделать вручную.
- **Глава 10. Отладка.** Рассматриваются средства Python, предназначенные для обнаружения и устранения логических ошибок.
- **Глава 11. Автоматический сбор данных в Интернете.** Показано, как писать программы, выполняющие автоматическую загрузку веб-страниц и их синтаксический анализ с целью извлечения полезной информации. Для этого процесса часто используют термин *веб-скрапинг*. Соответствующие программы называют *интернет-ботами*.
- **Глава 12. Работа с электронными таблицами Excel.** Рассматриваются методы манипулирования электронными таблицами Excel, ориентированные на обработку данных без их чтения человеком. Такая возможность чрезвычайно полезна в тех случаях, когда количество обрабатываемых документов исчисляется сотнями и даже тысячами.
- **Глава 13. Работа с документами в форматах PDF и Word.** Рассматриваются программные методы чтения документов, подготовленных в форматах PDF и Word.
- **Глава 14. Работа с CSV-файлами и данными в формате JSON.** Рассматриваются методы манипулирования CSV-файлами и JSON-данными.
- **Глава 15. Обработка значений даты и времени, планировщик заданий и запуск программ.** Рассказывается о способах обработки информации, связанной с датой и временем, и выполнении задач по расписанию. Также показано, как запускать программы, написанные на языках, отличных от Python, из Python-программ.

- **Глава 16. Отправка сообщений электронной почты и текстовых сообщений.** Обсуждается написание программ, осуществляющих автоматическую рассылку сообщений электронной почты и текстовых сообщений.
- **Глава 17. Работа с изображениями.** Рассматриваются способы программного манипулирования изображениями, сохраненными в различных форматах, таких как JPEG или PNG.
- **Глава 18. Управление клавиатурой и мышью с помощью средств автоматизации графического интерфейса пользователя.** Речь идет о возможностях управления клавиатурой и мышью путем программной эмуляции щелчков мышью и нажатий клавиш.

Исходный код примеров доступен в виде архивного файла *Automate_the_Boring_Stuff_onlinematerials.zip* на сайте `https://www.nostarch.com/automatestuff/`. Там же содержатся другие полезные ресурсы с примерами, предлагаемые автором в дополнение к данной книге, с указанием глав, к которым они относятся, а также публикуется информация об ошибках и опечатках, обнаруженных в книге.

Загрузка и установка Python

Версии Python для Windows, OS X и Ubuntu доступны для бесплатной загрузки по адресу `http://python.org/downloads/`. Если вы загрузите текущую версию для своей системы, то все примеры программ, приведенные в книге, должны будут работать.

Предупреждение

Обязательно загрузите версию Python 3 (например, 3.4.5). Все примеры программ в книге написаны с использованием Python 3, и если вы попытаетесь запускать их в версии Python 2, то они могут выполняться неправильно или вообще не выполняться.

На указанной странице загрузки для каждой операционной системы предлагаются отдельные установщики, рассчитанные на 64- и 32-разрядные версии, поэтому предварительно определитесь, какой именно установщик вам нужен. Если вы приобрели компьютер в 2007 году или позже, то, скорее всего, на нем установлена 64-разрядная операционная система. В противном случае можно полагать, что вы пользуетесь 32-разрядной версией, но лучше убедиться в этом непосредственно, выполнив следующие действия.

- Если вы используете Windows, выберите пункты меню Пуск⇨Панель управления⇨Система и проверьте, какая система указана в качестве значения параметра Тип системы — 64- или 32-разрядная.
- Если вы используете OS X, перейдите в меню Apple, выберите пункты меню About This Mac⇨More Info⇨System Report⇨Hardware, а затем проверьте значение поля Processor Name. Если в этом поле отображается текст "Core Solo" или "Intel Core Duo", то у вас 32-разрядный компьютер. Если же в поле отображается какой-либо другой текст (включая "Intel Core 2 Duo"), то у вас 64-разрядный компьютер.
- Если вы используете Ubuntu Linux, откройте терминал и выполните команду `uname -m`. Ответ `i686` означает 32-разрядный компьютер, ответ `x86_64` — 64-разрядный.

Для Windows загрузите установщик Python (файл с расширением `.msi`) и дважды щелкните на нем. Чтобы установить Python, следуйте инструкциям, которые установщик отображает на экране.

1. Выберите вариант Install for All Users, а затем щелкните на кнопке Next.
2. Выполните установку в папку *C:\Python34*, щелкнув на кнопке Next.
3. Вновь щелкните на кнопке Next, чтобы пропустить раздел Customize Python.

Для MAC OS X загрузите файл с расширением *.dmg*, соответствующий вашей версии OS X, и дважды щелкните на нем. Чтобы установить Python, следуйте инструкциям, которые установщик отображает на экране.

1. Когда в новом окне откроется пакет DMG, дважды щелкните на файле *Python.mpkg*. Возможно, вам придется ввести пароль администратора.
2. Щелкните на кнопке Continue для прохождения раздела Welcome, а затем на кнопке Agree для принятия условий лицензии.
3. Выделите имя жесткого диска, на который выполняется установка, и щелкните на кнопке Install.

В случае использования Ubuntu можете установить Python из окна терминала, выполнив следующие действия.

1. Откройте окно Terminal.
2. Введите команду `sudo apt-get install python3`.
3. Введите команду `sudo apt-get install idle3`.
4. Введите команду `sudo apt-get install python3-pip`.

Запуск IDLE

Если *интерпретатор Python* — это программное обеспечение, предназначенное для выполнения программ на Python, то *интерактивная среда разработки (IDLE)* — это программное обеспечение, с помощью которого можно вводить текст программ примерно так же, как это делается с помощью текстового процессора. Приступим к запуску IDLE.

- Если ваш компьютер работает под управлением операционной системы Windows 7 или более новой версии Windows, щелкните на кнопке Пуск в левом нижнем углу экрана, введите **IDLE** в строке поиска и выберите в раскрывшемся меню пункт IDLE (Python GUI).
- Если ваш компьютер работает под управлением операционной системы Windows XP, щелкните на кнопке Пуск и выберите последовательно пункты меню Programs⇨Python 3.4⇨IDLE (Python GUI).
- Если ваш компьютер работает под управлением операционной системы MAC OS X, откройте окно Finder, выберите последовательно Applications и Python 3.4, а затем щелкните на значке IDLE.
- Если ваш компьютер работает под управлением операционной системы Ubuntu, выберите Applications⇨Accessories⇨Terminal, а затем введите команду **idle3**. (Вы также можете щелкнуть на кнопке Applications в верхней части экрана, выбрать раздел Programming и щелкнуть на пункте IDLE 3.)

Интерактивная оболочка

Независимо от того, в какой операционной системе вы работаете, окно IDLE при его первом открытии будет в основном пустым, не считая текста, который будет выглядеть примерно так.

```
Python 3.4.0 (v3.4.0:04f714765c13, Mar 16 2014, 19:25:23)
[MSC v.1600 64 bit (AMD64)] on win32Type "copyright", "credits"
or "license()" for more information.
>>>
```

Это окно называется *интерактивной оболочкой*. Оболочка — это программа, которая позволяет вводить инструкции в компьютер во многом аналогично тому, как это делается в окне терминала или командной строки на компьютерах OS X и Windows соответственно. Команды, которые вы вводите в интерактивной оболочке, выполняются интерпретатором Python. Компьютер читает введенные инструкции (команды) и немедленно выполняет их.

Чтобы увидеть, как это работает на практике, введите в интерактивной оболочке сразу же за приглашением к вводу (>>>) следующую инструкцию:

```
>>> print('Hello world!')
```

После того как вы введете эту инструкцию и нажмете клавишу <Enter>, интерактивная оболочка должна отреагировать на это выводом следующей строки:

```
>>> print('Hello world!')
Hello world!
```

Как получить справку

Самостоятельно находить решения проблем, возникающих в процессе программирования, гораздо легче, чем вы думаете. Чтобы убедить вас в этом, давайте намеренно вызовем ошибку при попытке выполнить инструкцию. Введите в интерактивной оболочке инструкцию `'42' + 3`. Вам необязательно знать сейчас, что она означает, но результат должен выглядеть так.

```
   >>> '42' + 3
❶  Traceback (most recent call last):
     File "<pyshell#0>", line 1, in <module>
       '42' + 3
❷  TypeError: Can't convert 'int' object to str implicitly
   >>>
```

Появление здесь сообщения об ошибке ❷ обусловлено тем, что смысл введенной вами инструкции остался непонятным для Python. В той части сообщения, которая касается текущего стека вызовов (`Traceback`) ❶, отображаются конкретная инструкция и номер строки, в которой Python столкнулся с проблемой. Если сообщение об ошибке ни о чем вам не говорит, выполните поиск в Интернете по точному тексту сообщения. Введите текст **`"TypeError: Can't convert 'int' object to str implicitly"`** (включая кавычки) в своем поисковике, и вы увидите тысячи ссылок, по которым можно узнать о том, что означает данное сообщение и что породило ошибку (рис. 2).

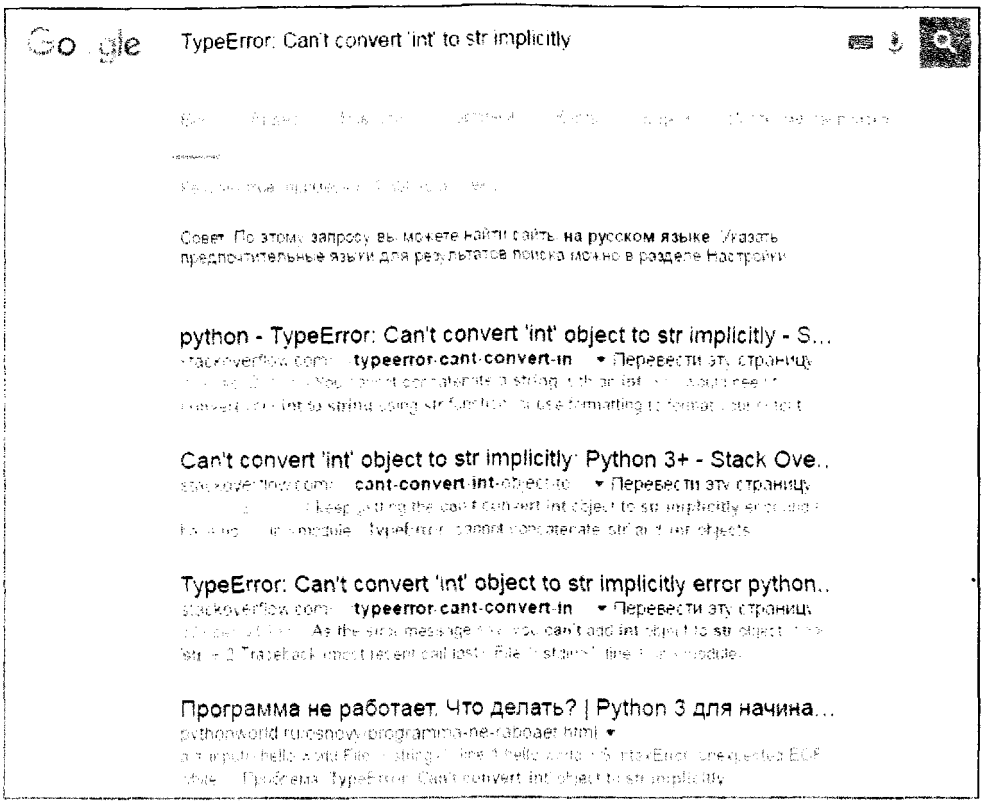


Рис. 2. Получение дополнительной информации о природе ошибки путем поиска в Google с использованием текста сообщения об ошибке в качестве ключа

Часто вы будете сталкиваться с тем, что у кого-то уже возникал тот же вопрос, что и у вас, и на этот вопрос уже был дан ответ. В программировании никому не дано знать абсолютно все, поэтому свыкнитесь с мыслью о том, что поиск ответов на различные вопросы технического характера — неотъемлемая часть ежедневной деятельности программиста-разработчика.

Правильно формулируйте вопросы, ответы на которые ищете

Если онлайновый поиск не позволил получить ответ на интересующий вас вопрос, то поспрашивайте людей на таких форумах, как Stack Overflow (`http://stackoverflow.com`), или посетите учебный раздел сайта Reddit (`http://reddit.com/r/learnprogramming`). Однако имейте в виду, что при обращении за помощью очень важно правильно формулировать свои вопросы. Обязательно посетите разделы Frequently Asked Questions (часто

задаваемые вопросы) этих сайтов, где вы сможете ознакомиться с формулировками вопросов, которые послужат вам образцом для подражания.

Задавая вопросы, касающиеся программирования, старайтесь придерживаться следующих рекомендаций.

- Объясните, что именно вы *пытаетесь* сделать, а не только то, что вы *делали*. Это позволит тому, кто хочет вам помочь, определить, находитесь вы на верном или на неверном пути.
- Укажите, когда именно возникает ошибка: сразу после запуска программы или после того, как вы выполняете определенные действия.
- Скопируйте и вставьте *полный* текст сообщения об ошибке и ваш код в буферное хранилище по адресу `http://pastebin.com/` или `http://gist.github.com/`.

  Указанные веб-сайты упрощают обмен большими объемами кода через Интернет без риска потерять форматирование текста. Затем можете переслать URL-адрес размещенного в буферном хранилище кода нужному человеку по электронной почте или опубликовать его на форуме. Чтобы увидеть, как это работает, просмотрите код, который я разместил в хранилищах по следующим адресам: `http://pastebin.com/SzP2DbFx/` и `https://gist.github.com/asweigart/6912168/`.

- Объясните, какие меры вы предпринимали для разрешения возникшей проблемы. Тем самым вы покажете, что уже приложили усилия со своей стороны, стараясь самостоятельно выяснить причину неполадок.
- Укажите версию Python, которую используете. (Между интерпретаторами, входящими в состав версий Python 2 и 3, имеются важные различия.) Также укажите используемую вами операционную систему и ее версию.
- Если ошибка появилась после того, как вы внесли изменения в код, детально опишите, какие именно изменения были вами внесены.
- Расскажите, воспроизводится ли ошибка всякий раз, когда вы выполняете программу, или она возникает лишь после того, как вы совершаете определенные действия. В последнем случае опишите, в чем именно заключаются эти действия.

Кроме того, строго соблюдайте правила сетевого этикета. Например, размещая на форуме свои вопросы, не набирайте весь текст прописными буквами, пытаясь сделать его более заметным, и не выдвигайте необоснованных требований к людям, которые пытаются вам помочь.

Резюме

Для большинства людей компьютер — это всего лишь полезное устройство, а не инструмент. Вместе с тем, научившись программировать, вы получите доступ к одному из наиболее мощных инструментов в современном мире, работа с которым доставит вам, кроме всего прочего, огромное удовольствие. Программирование вовсе не сродни нейрохирургии — оно предоставляет новичкам великолепную возможность экспериментировать и при этом не бояться, что допущенные ошибки могут быть чреваты катастрофическими последствиями.

Мне нравится помогать людям открывать для себя Python. Я пишу руководства по программированию на своем блоге по адресу `http://inventwithpython.com/blog/`, и вы можете связаться со мной и задать вопросы, отправив сообщение электронной почты по адресу `al@inventwithpython.com`.

Эта книга лишь помогает преодолеть начальный барьер в изучении программировании, поэтому вы не всегда найдете в ней ответы на все свои вопросы. Не забывайте о том, что умение правильно формулировать вопросы и знание того, как находить ответы на них, окажут вам неоценимую помощь в вашем путешествии в мир программирования.

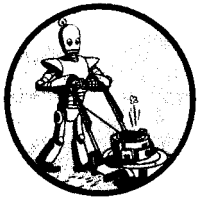
Итак, приступим!

ЧАСТЬ I

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON

1

ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА PYTHON



Язык программирования Python предлагает богатейший набор синтаксических конструкций, функций стандартной библиотеки и средств интерактивной разработки. К счастью, без большинства этих средств можно спокойно обойтись, ведь все, что вам нужно, — это научиться писать короткие полезные программы.

Прежде чем вы сможете что-либо сделать, вам следует усвоить некоторые базовые понятия программирования. Поскольку на данном этапе вы еще только учитесь, рассматриваемый в этой главе материал поначалу может показаться вам скучным и чересчур сложным. Но даже самых скромных знаний и небольшой практики, которые вы приобретете, вам хватит для того, чтобы управлять компьютером подобно магу, который вооружен волшебной палочкой и способен совершать самые невероятные подвиги.

В этой главе мы разберем несколько примеров, которые пробудят в вас интерес к работе с интерактивной оболочкой. С ее помощью вы сможете выполнять по одной команде Python за один раз и сразу же видеть результаты. Интерактивная оболочка будет вашим надежным помощником в изучении основных инструкций Python, поэтому отказываться от такой возможности не стоит. Когда делаешь что-то своими руками, а не просто читаешь книгу, все запоминается гораздо лучше.

Ввод выражений в интерактивной оболочке

Для вызова интерактивной оболочки необходимо запустить интерактивную среду IDLE, процедура установки которой описана во введении. В Windows откройте меню Пуск и выберите пункты меню Все программы⇨Python 3.3, а затем IDLE (Python GUI). В OS X выберите пункты меню Applications⇨MacPython 3.3⇨IDLE. В Ubuntu откройте новое окно терминала и введите команду `idle3`.

В результате должно открыться окно с приглашением ко вводу (>>>); это и есть интерактивная оболочка. Введите в командной строке инструкцию `2 + 2` в ответ на приглашение, чтобы Python выполнил для вас простое математическое действие.

```
>>> 2 + 2
4
```

Теперь в окне IDLE должен отображаться примерно такой текст.

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53) [MSC
v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 2 + 2
4
>>>
```

В Python запись `2 + 2` называется *выражением*. Выражение — это наиболее фундаментальная разновидность программных инструкций языка. Выражения состоят из *значений* (таких, как `2`) и *операторов* (таких, как `+`) и всегда могут *вычисляться* (сводиться) до получения единственного значения. Отсюда следует, что в коде Python выражения могут использоваться везде, где допускается использование значения.

В предыдущем примере выражение `2 + 2` вычисляется, сводясь к единственному значению — `4`. Одиночное значение без операторов также считается выражением, результатом вычисления которого является само значение.

Программные ошибки не представляют угрозы для оборудования компьютера!

Если компьютеру встречается непонятный для него программный код, то программа завершается аварийно, и Python выводит сообщение об ошибке. Эти сообщения не могут причинить вред вашему компьютеру, так что не бойтесь совершать ошибки. *Аварийное завершение*, или *крах*, программы — это просто неожиданное прекращение ее выполнения.

Если вы хотите получить более подробную информацию о каком-то сообщении об ошибке, используйте точный текст сообщения для проведения соответствующего поиска в Интернете. Кроме того, вы сможете воспользоваться ресурсами, находящимися по адресу `http://nostarch.com/automatestuff/`, для просмотра списка наиболее часто встречающихся сообщений об ошибках в Python вместе с их описаниями.

```
>>> 2
2
```

Кроме оператора +, существует множество других операторов, допустимых в выражениях Python. Полный список математических операторов Python приведен в табл. 1.1.

Таблица 1.1. Математические операторы Python в порядке уменьшения их приоритета

Оператор	Операция	Пример	Результат
**	Возведение в степень	2 ** 3	8
%	Деление по модулю/остаток	22 % 8	6
//	Целочисленное деление с отбрасыванием дробной части	22 // 8	2
/	Деление	22 / 8	2.75
*	Умножение	3 * 5	15
-	Вычитание	5 - 2	3
+	Сложение	2 + 2	4

Очередность (строгое название — *приоритет*) выполнения математических операторов Python совпадает с очередностью, принятой в математике. Сначала выполняется оператор **; затем, в порядке следования слева направо, выполняются операторы *, /, // и %; и наконец, последними выполняются операторы + и - (также в порядке следования слева направо). Для изменения очередности выполнения операций используются круглые скобки. Введите в интерактивной оболочке следующие выражения.

```
>>> 2 + 3 * 6
20
>>> (2 + 3) * 6
30
>>> 48565878 * 578453
28093077826734
>>> 2 ** 8
256
>>> 23 / 7
3.2857142857142856
>>> 23 // 7
3
>>> 23 % 7
2
>>> 2 + 2
4
>>> (5 - 1) * ((7 + 1) / (3 - 1))
16.0
```

Во всех этих случаях вы как программист лишь вводите выражения, тогда как всю тяжелую работу по сведению выражения к единственному значению берет на себя интерпретатор. Python последовательно вычисляет отдельные части выражения до тех пор, пока не преобразует его в единственное значение (рис. 1.1).

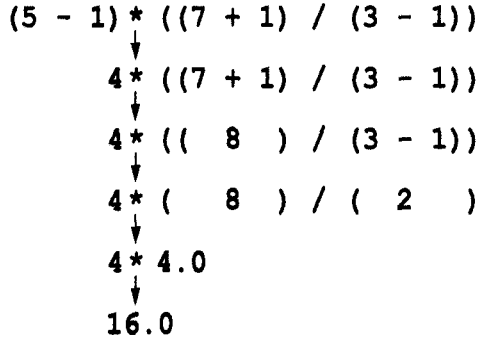


Рис. 1.1. Результат вычисления выражения сводится к единственному значению

Вышеперечисленные правила совместного использования операторов и значений для формирования выражений являются фундаментальной частью языка программирования Python точно так же, как правила обычной грамматики обеспечивают нам возможность общения на родном языке. Рассмотрим соответствующий пример.

> Это предложение на русском языке грамматически корректно.
> Это корректно предложение не на языке грамматически русском.

Понять смысл второй строки практически невозможно, поскольку она не следует правилам построения предложений, принятым в русском языке. Точно так же и Python, встретив неправильно составленную инструкцию, не сможет ее однозначно интерпретировать и выведет сообщение о синтаксической ошибке (`SyntaxError`).

```
>>> 5 +
  File "<stdin>", line 1
    5 +
      ^
SyntaxError: invalid syntax
>>> 42 + 5 + * 2
  File "<stdin>", line 1
    42 + 5 + * 2
             ^
SyntaxError: invalid syntax
```

Вы всегда можете проверить, работает ли та или иная инструкция, введя ее в интерактивной оболочке. Вам не о чем волноваться — компьютер вы не сломаете. В самом худшем случае Python отреагирует на неправильную инструкцию выводом сообщения об ошибке. Даже профессиональные разработчики программного обеспечения постоянно получают подобные сообщения в процессе написания кода.

Типы данных: целые числа, вещественные числа, строки

Вспомните о том, что выражения — это просто сочетания значений и операторов и что результат их вычисления всегда сводится к единственному значению. *Тип данных* — это определенная категория значений, причем каждое значение относится к одному и только одному типу данных. Наиболее простые типы данных Python приведены в табл. 1.2. О значениях `-2` и `30` говорят, что они являются *целочисленными*. Целочисленному (`int`) типу данных соответствуют значения в виде целых чисел. Числа с десятичной точкой, такие, например, как `3.14`, называются *числами с плавающей точкой* (тип данных `float`) или *вещественными числами*. Заметьте, что значение `42` — целое число, тогда как значение `42.0` — вещественное.

Таблица 1.2. Простые типы данных Python

Тип данных	Примеры
Целые числа	`-2, -1, 0, 1, 2, 3, 4, 5`
Числа с плавающей точкой (вещественные)	`-1.25, -1.0, -0.5, 0.0, 0.5, 1.0, 1.25`
Строки	`'a', 'aa', 'aaa', 'Hello!', '11 cats'`

Программируя на языке Python, вы сможете использовать и текстовые значения, так называемые *строки* (тип данных `str`). Всегда заключайте строку в апострофы (`'`), чтобы Python мог распознать, где она начинается и заканчивается (например, `'Hello'` или `'Goodbye cruel world!'`). Строка может вообще не содержать ни одного символа (`''`); такие строки называются *пустыми*. Более подробно строки рассматриваются в главе 4.

Если вам когда-либо приходилось сталкиваться с сообщением об ошибке `SyntaxError: EOL while scanning string literal`, то, вероятно, это было вызвано тем, что вы забыли ввести символ апострофа, завершающий строку, как в приведенном ниже примере.

```
>>> 'Hello world!
SyntaxError: EOL while scanning string literal
```

Конкатенация и репликация строк

Смысл оператора может меняться в зависимости от типа соседних с ним данных (контекста). Например, если оператор + применяется к двум значениям, являющимся числами (целыми или вещественными), то он трактуется как оператор сложения. Но если его применить к двум строковым значениям, то он объединит их в одну строку, играя роль оператора *конкатенации строк*. Введите в интерактивной оболочке следующее выражение.

```
>>> 'Alice' + 'Bob'
'AliceBob'
```

Данное выражение сводится к новому строковому значению, объединяющему текст обеих исходных строк. Если же попытаться применить оператор + к строке и целому числу, то Python не сможет определить, чего именно от него хотят, и выведет сообщение об ошибке.

```
>>> 'Alice' + 42
Traceback (most recent call last):
  File "<pyshell#26>", line 1, in <module>
    'Alice' + 42
TypeError: Can't convert 'int' object to str implicitly
```

Сообщение `Can't convert 'int' object to str implicitly` означает, что Python предположил, будто вы пытаетесь конкатенировать строку `'Alice'` с целочисленным значением. В подобных ситуациях ваш код должен явно преобразовывать целые числа в строки, так как Python не может автоматически выполнить такое преобразование. (Преобразования типов данных рассматриваются в разделе "Анализ программы" при обсуждении функций `str()`, `int()` и `float()`.)

Если оператор * применяется к двум целочисленным или вещественным значениям, то он трактуется как оператор умножения. Но если одно из значений — строка, а второе — целое число, то он становится оператором *репликации строк*. Введите в интерактивной оболочке строку, умноженную на число, чтобы увидеть, как это работает.

```
>>> 'Alice' * 5
'AliceAliceAliceAliceAlice'
```

Результатом вычисления этого выражения является строковое значение, представляющее собой многократно повторенную исходную строку, число повторов которой равно данному целочисленному значению. Ре-

пликация строки — очень полезный прием, хотя и используется реже, чем конкатенация строк.

Оператор * может использоваться только в отношении двух числовых значений (умножение) или одной строки и одного целочисленного значения (репликация строк). В противном случае Python отобразит сообщение об ошибке.

```
>>> 'Alice' * 'Bob'
Traceback (most recent call last):
  File "<pyshell#32>", line 1, in <module>
    'Alice' * 'Bob'
TypeError: can't multiply sequence by non-int of type 'str'
>>> 'Alice' * 5.0
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    'Alice' * 5.0
TypeError: can't multiply sequence by non-int of type 'float'
```

Совершенно очевидно, почему Python не смог определить смысл этих выражений: невозможно умножить одно слово на другое, равно как и повторить строку дробное количество раз.

Сохранение значений в переменных

· *Переменная* — это область памяти компьютера, в которой может храниться одиночное значение. Если вы предполагаете, что результат вычисления выражения впоследствии будет использоваться в программе, то можете сохранить его в переменной.

Инструкции присваивания

Для сохранения значений в переменных используется *инструкция присваивания*. В ней указываются имя переменной, знак равенства (называемый *оператором присваивания*) и сохраняемое значение. Например, инструкция присваивания `spam = 42` обеспечивает сохранение значения `42` в переменной `spam`.

Образно говоря, переменную можно уподобить ящику с меткой, в который переменная помещается для постоянного или временного хранения (рис. 1.2).

Введите в интерактивной оболочке следующие инструкции.

```
❶ >>> spam = 40
  >>> spam
  40
  >>> eggs = 2
```

```
❷ >>> spam + eggs
   42
   >>> spam + eggs + spam
   82
❸ >>> spam = spam + 2
   >>> spam
   42
```

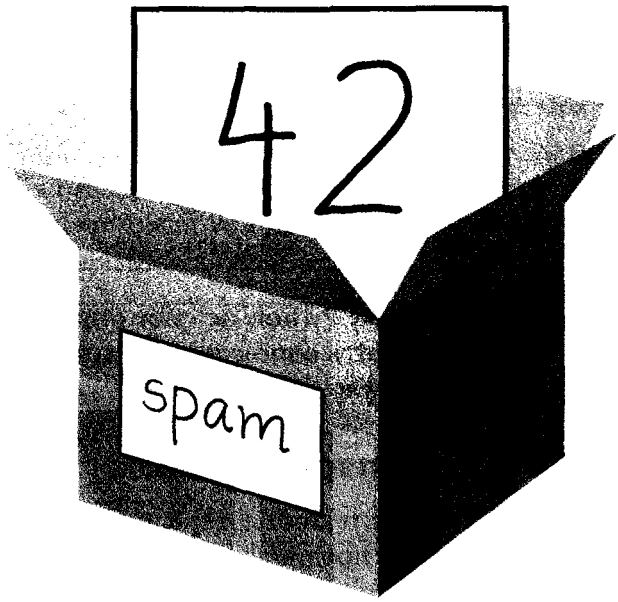


Рис. 1.2. Инструкция spam = 42 сообщает программе: "Теперь в переменной spam хранится целочисленное значение 42"

Термин *инициализация* (или создание) *переменной* относится к первоначальному присвоению переменной какого-либо значения ❶. После этого переменную можно использовать в выражениях совместно с другими переменными и значениями ❷. В процессе присваивания переменной нового значения ее прежнее значение теряется, и именно поэтому значением переменной `spam` в конце примера является `42`, а не `40`. Изменение значения называется *перезаписью* переменной. Попробуйте перезаписать строку, введя в интерактивной оболочке следующий код.

```
>>> spam = 'Hello'
>>> spam
'Hello'
>>> spam = 'Goodbye'
>>> spam
'Goodbye'
```

В этом примере значение `'Hello'` хранится в переменной `spam` лишь до тех пор, пока вы не замените его значением `'Goodbye'` (рис. 1.3).

Имена переменных

В табл. 1.3 приведены примеры допустимых имен переменных. Переменным можно присваивать любые имена, при условии, что они удовлетворяют следующим трем требованиям.

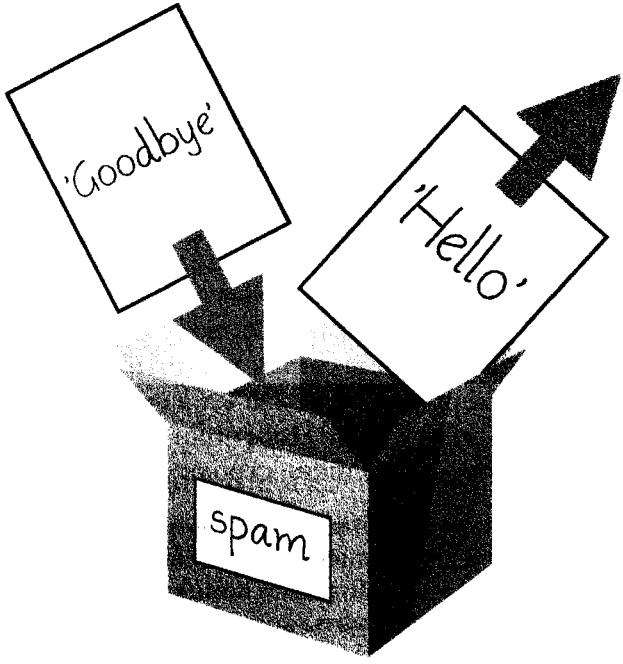


Рис. 1.3. Когда переменной присваивается новое значение, старое значение теряется

1. Имя переменной должно представлять собой одно слово.
2. В имени переменной могут использоваться только буквы, цифры и символ подчеркивания (_).
3. Имя переменной не может начинаться с цифры.

Таблица 1.3. Допустимые и недопустимые имена переменных

Допустимые имена переменных	Недопустимые имена переменных
`balance`	`current-balance` (недопустимый дефис)
`currentBalance`	`current balance` (недопустимый пробел)
`current_balance`	`4account` (имя не может начинаться с цифры)

Окончание табл. 1.3

Допустимые имена переменных	Недопустимые имена переменных
`_spam`	`42` (имя не может начинаться с цифры)
`SPAM`	`total_$um` (специальные символы, такие как `$`, в именах недопустимы)
`account4`	`'hello'` (специальные символы, такие как `'`, в именах недопустимы)

Имена переменных чувствительны к регистру. Это означает, что `spam`, `SPAM`, `Spam` и `sPaM` — четыре разных имени. В соответствии с принятым в Python соглашением имена переменных должны начинаться с маленькой буквы.

В этой книге вместо стиля именования, предполагающего использование символа подчеркивания, используется так называемый "верблюжий стиль", в котором имена наподобие `looking_like_this` преобразуются в имена наподобие `lookingLikeThis`. Некоторые опытные программисты могли бы упрекнуть меня в том, что я не следую официальному руководству Python по стилям, PEP8, поощряющему использование символов подчеркивания в именах переменных. Совершенно верно, я самым непочтительным образом предпочитаю использовать "верблюжий стиль", а в свое оправдание сошлюсь на раздел "A Foolish Consistency Is the Hobgoblin of Little Minds"[1] самого же руководства PEP8[2], в котором есть такие строки:

> "Согласованность с этим руководством очень важна. Согласованность внутри одного проекта еще важнее. А согласованность внутри модуля или функции — самое важное. Но важно помнить, что иногда это руководство неприменимо, и понимать, когда можно отойти от рекомендаций. Когда вы сомневаетесь, просто посмотрите на другие примеры и решите, какой выглядит лучше".

Хорошими считаются описательные имена, которые раскрывают смысл данных, содержащихся в переменных. Представьте, что, переезжая в новый дом, вы разложили все вещи по ящикам и пометили каждый из них одной и той же надписью — "Вещи". Легко ли вам будет после этого что-либо найти? В данной книге, как, впрочем, и в большей части документации по Python, в качестве типичных имен в примерах используются такие имена, как `spam`, `eggs` или `bacon` (здесь явно сказалось влияние скетча "Spam"

[1] "Глупая последовательность — пугало маленьких умов" — цитата из эссе "Доверие к себе" известного американского писателя Ральфа Уолдо Эмерсона (см. перевод на русский язык по адресу `https://raw.githubusercontent.com/boretskiy/self-reliance-ru/master/self-reliance-ru.txt`). — *Примеч. ред.*

[2] `https://www.python.org/dev/peps/pep-0008/` (см. перевод на русский язык по адресу `http://pep8.ru/doc/pep8/`). — *Примеч. ред.*

группы "Monty Python"), но в своих программах вам лучше использовать описательные имена, что повысит удобочитаемость вашего кода.

Ваша первая программа

Интерактивная оболочка отлично подходит для выполнения инструкций Python по одной за один раз, но при написании полноценных программ на Python вы будете подготавливать их текст с помощью какого-либо файлового редактора. *Файловые редакторы* аналогичны текстовым, таким как Notepad (Блокнот) или TextMate, но предлагают дополнительные возможности при вводе исходного кода. Чтобы открыть файловый редактор в IDLE, выберите пункты меню File⇨New file (Файл⇨Создать файл).

В открывшемся окне вы увидите курсор, ожидающий ввода, но это окно отличается от окна интерактивной оболочки, которое выполняет введенную вами инструкцию Python сразу же после нажатия клавиши <Enter>. Файловый редактор позволяет ввести множество инструкций, сохранить файл и выполнить программу. Ниже указано, чем различаются эти два окна:

- признаком окна интерактивной оболочки служит приглашение к вводу вида >>>;
- в окне файлового редактора приглашение к вводу вида >>> отсутствует.

А теперь пришло время создать вашу первую программу! Открыв окно файлового редактора, введите в нем следующий текст.

```
❶ # Эта программа приветствует пользователя и запрашивает
  # ввод информации.

❷ print('Hello world!')
  print('What is your name?') # запрос имени
❸ myName = input()
❹ print('It is good to meet you, ' + myName)
❺ print('The length of your name is:')
  print(len(myName))
❻ print('What is your age?') # запрос возраста
  myAge = input()
  print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

Введя исходный код, сохраните его, чтобы набирать его заново при следующем запуске IDLE. Выберите в меню, расположенном в верхней части окна файлового редактора, пункты File⇨Save As (Файл⇨Сохранить как), введите `hello.py` в поле File Name (Имя файла) и щелкните на кнопке Save (Сохранить).

В процессе ввода текста программы периодически сохраняйте файл. Это позволит избежать потери уже введенного кода, если работа компьютера

завершится аварийно или вы случайно выйдете из IDLE. Вместо сохранения файла с помощью меню можно нажать комбинацию клавиш <Ctrl+S> (Windows и Linux) или <⌘+S> (OS X).

После того как файл будет сохранен, запустите программу. Выберите пункты меню Run⇨Run Module (Выполнить⇨Выполнить модуль) или просто нажмите клавишу <F5>. Ваша программа должна запуститься в окне интерактивной оболочки, которое открывалось, когда вы впервые запускали IDLE. Не забывайте о том, что клавишу <F5> следует нажимать не в окне интерактивной оболочки, а в окне файлового редактора. Введите свое имя в ответ на приглашение программы. Вывод программы в окне интерактивной оболочки должен выглядеть примерно так.

```
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:06:53)
[MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========================== RESTART ================
==========
>>>
Hello world!
What is your name?
Al
It is good to meet you, Al
The length of your name is:
2
What is your age?
4
You will be 5 in a year.
>>>
```

Исчерпав все строки кода, подлежащие выполнению, программа *завершается*, т.е. ее выполнение прекращается. (В этом случае также говорят о *выходе* из программы.)

Чтобы закрыть окно файлового редактора, щелкните на кнопке × в верхней части окна. Чтобы перезагрузить сохраненную программу, выберите в меню пункты File⇨Open (Файл⇨Открыть). Проделайте это сейчас, выберите в открывшемся окне имя файла *hello.py* и щелкните на кнопке Open (Открыть). В окне файлового редактора должна открыться программа, которую вы перед этим сохранили в файле *hello.py*.

Анализ программы

Сейчас мы кратко рассмотрим все инструкции, которые используются в новой программе, открытой в окне файлового редактора, и проанализируем, что именно делает каждая строка кода.

Комментарии

Приведенные ниже строки кода называются *комментарием.*

```
❶ # Эта программа приветствует пользователя и запрашивает
  # ввод информации.
```

Python игнорирует комментарии, и вы сможете использовать их для записи примечаний или напоминаний самому себе относительно того, что делает данный код. Любой текст от символа "решетка" (#) до конца строки считается частью комментария.

Иногда программисты помещают символ # перед строкой кода для того, чтобы фактически исключить ее из кода, или, как говорят, *закомментировать*, на время тестирования программы. Этот прием оказывается очень полезным при выяснении причин нарушения нормальной работы программы. Впоследствии вы сможете восстановить отключенный код, удалив символ #, или, как говорят, *раскомментировав* строку.

Пустые строки после комментария игнорируются. Вы можете добавить в свою программу столько пустых строк, сколько пожелаете. Такое форматирование кода, напоминающее разбивку книжного текста на абзацы, облегчает его чтение.

Функция `print()`

Функция `print()` отображает указанное в скобках строковое значение на экране.

```
❷ print('Hello world!')
  print('What is your name?') # запрос имени
```

Строка `print('Hello world!')` означает "Вывести текст, хранящийся в строке `'Hello world!'`". При словесном описании этой строки говорят, что Python *вызывает* функцию `print()`, *передавая* ей строковое значение. Значение, передаваемое функции, называется *аргументом.* Обратите внимание на то, что кавычки не выводятся на экран. Они лишь обозначают начало и конец строки и не являются частью строкового значения.

Примечание

Эту же функцию можно использовать для вывода на экран пустой строки; для этого достаточно выполнить вызов `print()` *без указания аргументов.*

Наличие пары скобок после имени указывает на то, что данное имя обозначает функцию. Именно поэтому в книге везде используется запись `print()`, а не `print`. Более подробно функции рассматриваются в главе 2.

Функция input()

Функция input() ожидает ввода пользователем текста с последующим нажатием клавиши <Enter>.

```
❸ myName = input()
```

В этой строке кода текст, введенный пользователем, преобразуется в строковое значение, которое присваивается переменной myName.

Вызов функции input() можно рассматривать как выражение, значением которого является строка, введенная пользователем. Если пользователь ввел 'Al', то предыдущая строка кода эквивалентна строке myName = 'Al'.

Вывод имени пользователя

В следующем вызове функции print() в скобках указано выражение 'It is good to meet you, ' + myName.

```
❹ print('It is good to meet you, ' + myName)
```

Вспомните о том, что результатом вычисления выражения всегда является одиночное значение. Если 'Al' — это значение, сохраненное в переменной myName в предыдущей строке, то значением данного выражения является 'It is good to meet you, Al'. Затем это одиночное строковое значение передается функции print(), которая выводит его на экран.

Функция len()

Вы можете передать функции len() строковое значение (или переменную, содержащую строку), и она возвратит целое число, равное количеству символов в данной строке.

```
❺ print('The length of your name is:')
  print(len(myName))
```

Чтобы увидеть, как это работает на практике, введите в интерактивной оболочке следующий код.

```
>>> len('hello')
5
>>> len('My very energetic monster just scarfed nachos.')
46
>>> len('')
0
```

Точно так же, как в этих примерах, вычисление выражения `len(myName)` дает целое число. Далее это число передается функции `print()`, которая выводит его на экран. Заметьте, что функции `print()` можно передавать любые целочисленные или строковые значения. Но если вы введете в интерактивной оболочке приведенный ниже код, то получите сообщение об ошибке.

```
>>> print('I am ' + 29 + ' years old.')
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    print('I am ' + 29 + ' years old.')
TypeError: Can't convert 'int' object to str implicitly
```

Ошибка связана не с самой функцией `print()`, а с выражением, которое вы пытаетесь ей передать. То же самое произойдет и в том случае, если вы введете в интерактивной оболочке одно только это выражение.

```
>>> 'I am ' + 29 + ' years old.'
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    'I am ' + 29 + ' years old.'
TypeError: Can't convert 'int' object to str implicitly
```

- Python выводит сообщение об ошибке по той причине, что оператор + можно использовать лишь для сложения двух чисел или конкатенации двух строк. Сложить число со строкой невозможно, потому что это запрещено грамматикой Python. Данную проблему можно устранить, используя вместо целого числа его строковую версию, о чем пойдет речь в следующем разделе.

Функции `str()`, `int()` и `float()`

Если вы хотите конкатенировать целое число, такое как `29`, со строкой для передачи функции `print()`, вы должны получить значение `'29'`, являющееся строковой формой числа `29`. Функции `str()` можно передать целое число, и она преобразует его в соответствующую строку.

```
>>> str(29)
'29'
>>> print('I am ' + str(29) + ' years old.')
I am 29 years old.
```

Поскольку вычисление `str(29)` дает строку `'29'`, мы получаем в результате вычисления выражения `'I am ' + str(29) + ' years old.'` значение `'I am ' +`

'29' + ' years old.', которое, в свою очередь, возвращает результат в виде строки 'I am 29 years old.'. Это значение и передается функции print().

Функции str(), int() и float() соответственно возвращают строковую, целочисленную и вещественную формы передаваемого им значения. Попробуйте выполнить в интерактивной оболочке преобразование некоторых значений с помощью этих функций и посмотрите, что при этом будет происходить.

```
>>> str(0)
'0'
>>> str(-3.14)
'-3.14'
>>> int('42')
42
>>> int('-99')
-99
>>> int(1.25)
1
>>> int(1.99)
1
>>> float('3.14')
3.14
>>> float(10)
10.0
```

В этих примерах функции str(), int() и float() вызываются для получения строковой, целочисленной и вещественной форм значений, представляющих другие типы данных.

Функцию str() удобно использовать в тех случаях, когда у вас есть целое или вещественное число, которое вы хотите конкатенировать со строкой. Функция int() будет полезной, если имеется число в строковой форме, которое вы хотите использовать в математических вычислениях. Например, функция input() всегда возвращает строку, даже если пользователь вводит число. Введите в интерактивной оболочке инструкцию spam = input(), а затем в режиме ожидания ввода — число 101.

```
>>> spam = input()
101
>>> spam
'101'
```

Сохраненное в переменной spam значение является не числом 101, а строкой '101'. Если вы хотите выполнить вычисления с использованием значения, хранящегося в spam, воспользуйтесь функцией int() для получения целочисленной формы значения spam и сохраните новое значение в этой же переменной.

```
>>> spam = int(spam)
>>> spam
101
```

Теперь вы сможете обрабатывать значение spam не как строку, а как число.

```
>>> spam * 10 / 5
202.0
```

Имейте в виду, что в случае передачи функции `int()` значения, которое она не может привести к целочисленному виду, Python отобразит сообщение об ошибке.

```
>>> int('99.99')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    int('99.99')
ValueError: invalid literal for int() with base 10: '99.99'
>>> int('twelve')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    int('twelve')
ValueError: invalid literal for int() with base 10: 'twelve'
```

· Функцию `int()` удобно использовать для округления вниз вещественных чисел.

```
>>> int(7.7)
7
>>> int(7.7) + 1
8
```

В вашей программе функции `int()` и `str()` используются в последних трех строках кода для приведения значений к соответствующим типам данных.

```
❻ print('What is your age?') # запрос возраста
  myAge = input()
  print('You will be ' + str(int(myAge) + 1) + ' in a year.')
```

В переменной `myAge` содержится значение, возвращенное функцией `input()`. Поскольку функция `input()` всегда возвращает строку (даже если пользователь ввел число), для преобразования строкового значения, хранящегося в переменной `myAge`, в целочисленное следует использовать код `int(myAge)`. Затем это значение увеличивается на `1` в выражении `int(myAge) + 1`.

Сравнение строковых и числовых значений

В то время как строковое значение числа считается полностью отличающимся от его целочисленной или вещественной версии, целое значение может быть равно вещественному значению.

```
>>> 42 == '42'
False
>>> 42 == 42.0
True
>>> 42.0 == 0042.000
True
```

Python учитывает это различие, поскольку строки — это текст, тогда как и целый, и вещественный типы — числа.

Результат сложения передается функции `str()`: `str(int(myAge) + 1)`. После этого возвращенное строковое значение конкатенируется со строками `'You will be '` и `' in a year.'` для получения единого строкового значения. Наконец, это строковое значение передается функции `print()` для вывода на экран.

Предположим, что пользователь вводит в качестве значения переменной `myAge` строку `'4'`. Далее эта строка преобразуется в целое число, к которому можно прибавить единицу. В результате мы получаем значение `5`. Функция `str()` преобразует этот результат обратно в строку, которую можно конкатенировать со второй строкой, `' in a year.'`, для создания окончательного сообщения. Последовательность выполнения этих действий представлена на рис. 1.4.

```
print('You will be ' + str(int(myAge) + 1) + ' in a year.')
print('You will be ' + str(int( '4' ) + 1) + ' in a year.')
print('You will be ' + str(    4 + 1    ) + ' in a year.')
print('You will be ' + str(      5      ) + ' in a year.')
print('You will be ' +           '5'      + ' in a year.')
print('You will be 5'                     + ' in a year.')
print('You will be 5 in a year.')
```

Рис. 1.4. Последовательность вычислений для случая, когда значение myAge равно 4

Резюме

Для вычисления выражений можно использовать калькулятор, а для объединения строк в связный текст — текстовый редактор. Вы также можете реплицировать строки путем их копирования и вставки. Однако выражения и их компоненты — операторы, переменные и вызовы функций — являются теми строительными блоками, на основе которых создаются программы. Научившись обрабатывать эти элементы, вы сможете с помощью Python оперировать большими объемами данных.

Вам следует запомнить различные операторы (`+`, `-`, `*`, `/`, `//`, `%` и `**` для математических операций и `+` и `*` для операций над строками), а также три типа данных (целые числа, вещественные числа и строки), описанные в этой главе.

Вы также познакомились с несколькими функциями. Функции `print()` и `input()` предназначены для обработки простого текстового вывода (на экран) и ввода (с клавиатуры). Функция `len()` принимает строку и вычисляет количество содержащихся в ней символов. Функции `str()`, `int()` и `float()` вычисляют соответственно строковую, целочисленную и вещественную формы переданного им значения.

Из следующей главы вы узнаете, как организовать в программе на Python принятие решений относительно того, какой код выполнить, какой пропустить, а какой повторить, исходя из проверки выполнения заданных условий. Это обеспечивается управляющими конструкциями языка Python, с помощью которых вы сможете писать программы, способные принимать гибкие решения.

Контрольные вопросы

1. Какие из приведенных ниже синтаксических элементов являются операторами, а какие — значениями?

```
*
'hello'
-88.8
-
/
+
5
```

2. Что из приведенного ниже является переменной, а что — строкой?

```
spam
'spam'
```

3. Назовите три типа данных.
4. Из чего состоит выражение? К чему сводится любое выражение?
5. В этой главе введены выражения присваивания наподобие `spam = 10`. В чем суть различия между выражением и инструкцией?
6. Каким будет содержимое переменной `bacon` после выполнения следующей инструкции?

```
bacon = 20
bacon + 1
```

7. Каким будет результат вычисления следующих двух выражений?

```
'spam' + 'spamspam'
'spam' * 3
```

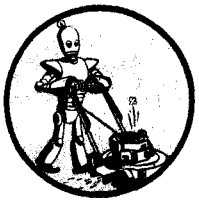
8. Почему `eggs` — допустимое имя переменной, а `100` таковым не является?
9. Назовите три функции, которые могут быть использованы для получения целочисленной, вещественной и строковой версий значения?
10. Что приводит к возникновению ошибки при попытке вычисления приведенного ниже выражения? Как избавиться от этой ошибки?

```
'Я съел ' + 99 + ' лепешек.'
```

Дополнительное задание. Изучите подробнее функцию `len()`, проведя поиск в онлайновой документации Python. Нужная вам информация находится на веб-странице, озаглавленной "Built-in Functions". Ознакомьтесь со списком других функций Python, обратив особое внимание на функцию `round()`, и самостоятельно поэкспериментируйте с этой функцией, используя интерактивную оболочку.

2

ПОТОК УПРАВЛЕНИЯ



Итак, вам уже известны основы синтаксиса отдельных инструкций, и вы знаете, что последовательность таких инструкций образует программу. Однако реальная мощь программирования заключается не в простом поочередном выполнении инструкций программы, напоминающем заранее спланированное расписание встреч с друзьями в выходные дни. Исходя из результатов вычисления выражений, программа может самостоятельно принимать решения относительно того, какие инструкции следует пропустить, а какие повторить, а также выбирать одну из нескольких инструкций для выполнения. В действительности практически не существует программ, которые выполняли бы строго последовательно каждую строку кода, начиная с первой и заканчивая последней. Python предоставляет синтаксис *управляющих инструкций*, или *инструкций ветвления* (другие названия — *инструкции передачи управления*, *инструкции перехода*, *условные инструкции*), которые предназначены для изменения естественного последовательного порядка выполнения инструкций программы, позволяя коду решать, какая инструкция должна выполняться следующей, исходя из соблюдения определенных условий. Последовательность передач управления в процессе выполнения программы образует ее *поток управления* (также *поток выполнения*).

Управляющие инструкции непосредственно соответствуют определенным условным символам, используемым в блок-схемах процессов, поэтому при обсуждении программ в этой главе я буду обращаться к их графическому представлению в виде блок-схем. На рис. 2.1 показана блок-схема процесса принятия решений, определяющего порядок действий, которые должны предприниматься в зависимости от того, идет ли дождь.

Обычно на таких блок-схемах отображается несколько возможных маршрутов, ведущих от начальной точки к конечной. То же самое справедливо

и для строк кода в компьютерной программе. Блоки условий, в которых происходит ветвление программы, обозначаются на блок-схеме ромбами, блоки действий – прямоугольниками. Конечный и начальный блоки программы представляются скругленными прямоугольниками.

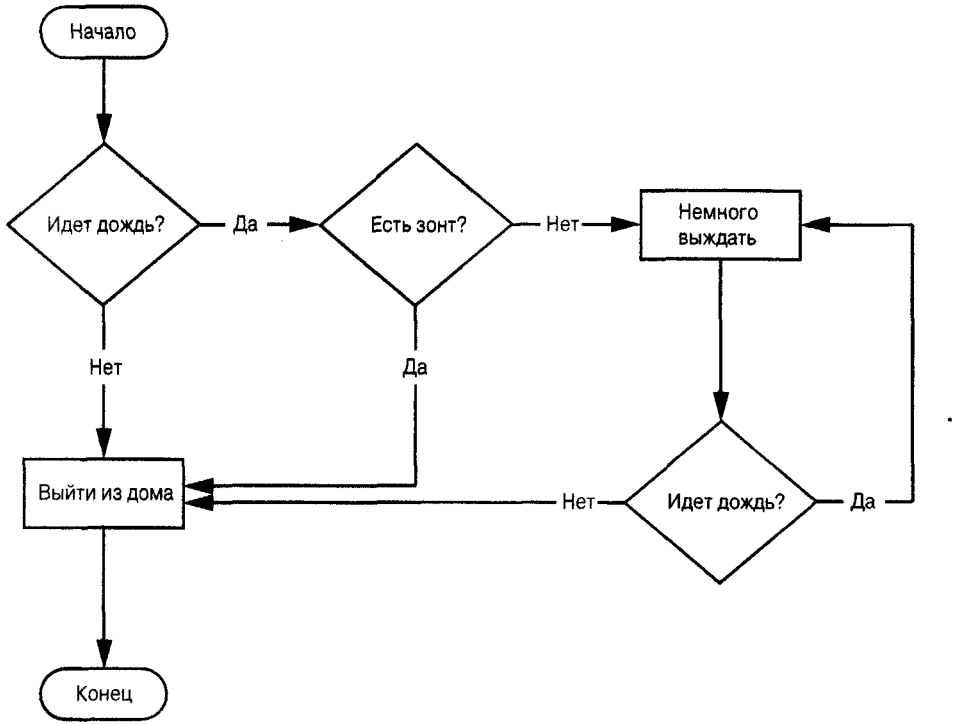


Рис. 2.1. Блок-схема, определяющая порядок действий на случай дождя

Однако, прежде чем приступить к изучению управляющих инструкций, вам предстоит ознакомиться со способами представления отображаемых на блок-схемах вариантов *да* и *нет*, а также записи точек ветвления в виде кода на языке Python. В качестве основы для этого рассмотрения мы будем использовать булевы значения, операторы сравнения и булевы операторы.

Булевы значения

В то время как целый, вещественный и строковый типы данных могут иметь неограниченное количество возможных значений, *логический*, или *булев*, тип данных может принимать только два значения: `True` (истина) и `False` (ложь). (Булев тип данных получил свое название в честь английского математика, основателя математической логики Джорджа Буля.) При использовании в коде Python булевы значения `True` и `False` не заключаются в кавычки и всегда начинаются с большой буквы `T` или `F`, тогда как остальная

часть слова записывается маленькими буквами. Введите в интерактивной оболочке следующие инструкции (некоторые из них намеренно сделаны некорректными и будут сопровождаться выводом сообщений об ошибке при попытке их выполнения).

```
❶ >>> spam = True
  >>> spam
  True
❷ >>> true
  Traceback (most recent call last):
    File "<pyshell#2>", line 1, in <module>
      true
  NameError: name 'true' is not defined
❸ >>> True = 2 + 2
  SyntaxError: assignment to keyword
```

Как и любые другие значения, булевы значения могут входить в выражения и сохраняться в переменных ❶. В случае применения неправильного регистра букв ❷ или попытки использования идентификатора `True` или `False` в качестве имени переменной ❸ Python выведет сообщение об ошибке.

Операторы сравнения

Операторы сравнения сравнивают два значения между собой и возвращают результат в виде булева значения. Операторы сравнения приведены в табл. 2.1.

Таблица 2.1. Операторы сравнения

Оператор	Название
==	Равно
!=	Не равно
<	Меньше чем
>	Больше чем
<=	Меньше или равно
>=	Больше или равно

Результатом действия этих операторов, в зависимости от предоставленных им значений, является значение `True` или `False`. Посмотрим, как они работают, начав с операторов == и !=.

```
>>> 42 == 42
True
>>> 42 == 99
```

```
False
>>> 2 != 3
True
>>> 2 != 2
False
```

Как и следовало ожидать, результат применения оператора == (равно) равен `True` в случае равенства значений, записанных слева и справа от него, тогда как результат применения оператора `!=` (не равно) равен `True`, если эти значения различаются. Операторы == и `!=` могут работать, по сути, с любыми типами значений.

```
>>> 'hello' == 'hello'
True
>>> 'hello' == 'Hello'
False
>>> 'dog' != 'cat'
True
>>> True == True
True
>>> True != False
True
>>> 42 == 42.0
True
❶ >>> 42 == '42'
False
```

Обратите внимание на то, что целые и вещественные значения никогда не могут быть равны строковым. Результат выражения `42 == '42'` ❶ равен `False`, поскольку для Python целое число `42` и строка `'42'` — разные значения.

С другой стороны, операторы <, >, <= и >= работают надлежащим образом лишь с целыми и вещественными значениями.

```
>>> 42 < 100
True
>>> 42 > 100
False
>>> 42 < 42
False
>>> eggCount = 42
❶ >>> eggCount <= 42
True
>>> myAge = 29
❷ >>> myAge >= 10
True
```

Различие между операторами == и =

Возможно, вы обратили внимание на то, что оператор проверки равенства (==) обозначается двумя знаками равенства, тогда как оператор присваивания (=) — одним. Эти операторы можно легко перепутать. Чтобы этого не произошло, запомните следующее.

- Оператор == (равно) проверяет равенство двух значений.
- Оператор = (присвоение) помещает значение, указанное справа, в переменную, указанную слева.

Вам будет легче запомнить, что есть что, прибегнув к следующей мнемонике: оба родственных оператора сравнения (== и !=) состоят из двух символов.

Операторы сравнения часто используют для того, чтобы сравнить значение переменной с некоторым другим значением, как в строках кода `eggCount <= 42` ❶ и `myAge >= 10` ❷. (В конце концов, если бы речь шла лишь о сравнении двух значений, то, например, вместо инструкции `'dog' != 'cat'` в своем коде вы могли бы просто использовать значение `True`.) Со многими аналогичными примерами вы еще встретитесь далее, когда будете изучать управляющие инструкции.

Булевы операторы

Для сравнения булевых значений используются три булева оператора: `and`, `or` и `not`. Подобно операторам сравнения, они вычисляют булевы выражения, сводя их к единственному булеву значению. Приступим к более подробному рассмотрению этих операторов, начав с оператора `and`.

Бинарные булевы операторы

Операторы `and` (логическое И) и `or` (логическое ИЛИ) всегда работают с двумя булевыми значениями (или выражениями), и поэтому их называют *бинарными*. Оператор `and` возвращает значение `True` только в том случае, если одновременно *оба* булева значения равны `True`; в противном случае результат равен `False`. Чтобы увидеть, как это работает, выполните в интерактивной оболочке следующие примеры.

```
>>> True and True
True
>>> True and False
False
```

Для представления всех возможных результатов применения булевых операторов используют *таблицы истинности*. Таблица истинности для оператора and приведена в табл. 2.2.

Таблица 2.2. Таблица истинности для оператора and

Выражение	Результат
True and True	True
True and False	False
False and True	False
False and False	False

Возможные результаты применения оператора or приведены в его таблице истинности (табл. 2.3).

Таблица 2.3. Таблица истинности для оператора or

Выражение	Результат
True or True	True
True or False	True
False or True	True
False or False	False

Оператор not

Оператор not, в отличие от операторов and и or, воздействует только на одно булево значение (выражение) и поэтому является *унарным*. Этот оператор обращает булево значение в его противоположность.

```
>>> not True
False
❶ >>> not not not not True
True
```

В полной аналогии с использованием двойных отрицаний в устной речи и на письме допускается использование вложенных операторов not ❶, хотя в реальных программах в этом вряд ли возникнет необходимость. Возможные результаты применения оператора not приведены в его таблице истинности (табл. 2.4).

Таблица 2.4. Таблица истинности для оператора not

Выражение	Результат
not True	True
not False	False

Сочетание операторов сравнения с булевыми операторами

Поскольку результатом вычисления выражений, включающих операторы сравнения, являются булевы значения, их можно использовать в выражениях совместно с булевыми операторами.

Вспомните, что операторы `and`, `or` и `not` называются булевыми, поскольку всегда воздействуют на булевы значения `True` и `False`. В то время как выражения наподобие `4 < 5` не являются булевыми значениями, будучи вычисленными, они дают как раз такие значения. Чтобы увидеть, как это работает, выполните в интерактивной оболочке следующие примеры.

```
>>> (4 < 5) and (5 < 6)
True
>>> (4 < 5) and (9 < 6)
False
>>> (1 == 2) or (2 == 2)
True
```

Сначала компьютер вычисляет левое выражение, а затем — правое. Когда оба этих результата становятся известными, вычисляется конечный результат всего выражения в целом, который будет представлять собой единственное булево значение. Используемый компьютером процесс вычисления выражения `(4 < 5) and (5 < 6)` проиллюстрирован на рис. 2.2.

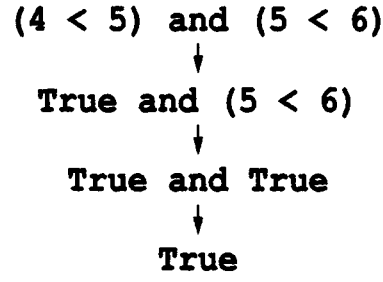


Рис. 2.2. Процесс вычисления выражения `(4 < 5) and (5 < 6)`, приводящий к значению `True`

Допускается совместное использование в одном выражении как булевых операторов, так и операторов сравнения.

```
>>> 2 + 2 == 4 and not 2 + 2 == 5 and 2 * 2 == 2 + 2
True
```

Как и для математических операторов, для булевых операторов определен порядок выполнения. После того как будут вычислены все

математические операторы и операторы сравнения, первыми выполняются операторы `not`, затем — операторы `and` и только после этого — операторы `or`.

Элементы потока управления

Начальная часть инструкций, управляющих порядком выполнения других инструкций программы, часто является *условием*, за которым следует блок кода. Прежде чем приступить к изучению конкретных управляющих инструкций Python, рассмотрим, что собой представляют условие и ассоциированный с ним блок кода.

Условия

Все булевы выражения, с которыми вы к этому времени успели познакомиться, могут рассматриваться как условия. *Условие* — это всего лишь более специализированное название выражения в контексте управляющих инструкций. Вычисление условия всегда дает булево значение, `True` или `False`. Управляющая инструкция принимает решение относительно дальнейших действий в зависимости от того, какое из этих двух значений принимает условие, и условия используются почти во всех управляющих инструкциях.

Блоки кода

Строки кода Python могут группироваться в *блоки*. О том, где находятся начало и конец блока, можно судить по отступам строк кода в тексте программы. В отношении блоков действуют следующие три правила.

Признаком начала блока служит увеличение отступа.

Блоки могут содержать другие блоки.

Признаком конца блока служит уменьшение отступа до нулевой величины или до величины отступа содержащего (внешнего) блока.

Вам будет легче понять, что такое блоки, определив, где они находятся в приведенной ниже части небольшой игровой программы.

```
  if name == 'Mary':
❶     print('Hello Mary')
      if password == 'swordfish':
❷         print('Access granted.')
      else:
❸         print('Wrong password')
  else:
❹     print('User not registered')
```

Первый блок кода ❶ начинается строкой `print('Hello Mary')` и содержит все последующие строки, предшествующие второй инструкции `else`. В этом блоке содержится второй блок ❷, содержащий только одну строку: `print('Access granted.')`. Третий блок ❸ также состоит только из одной строки: `print('Wrong password.')`. Четвертый блок, состоящий из строки `print('User not registered')`, относится ко второй инструкции `else`, ассоциированной с первой инструкцией `if`.

Выполнение программы

В программе *hello.py* из предыдущей главы Python последовательно выполнял инструкции одну за другой от начала и до конца программы. *Выполнение программы* — это термин, относящийся к текущим выполняемым инструкциям. Если вы напечатаете на бумажных листах исходный код программы и будете перемещать палец по строкам сверху вниз, то это и будет схематической иллюстрацией выполнения программы.

Однако не все программы выполняются столь прямолинейно. Если вы приметесь отслеживать пальцем порядок выполнения программы, в которой есть управляющие инструкции, то вашему пальцу придется перемещаться в пределах всего исходного кода в соответствии с заданными условиями и, возможно, пропускать целые блоки.

Управляющие инструкции

Приступим к самым важным элементам потока управления: собственно управляющим инструкциям. На блок-схеме, которая была показана на рис. 2.1, эти инструкции представлены ромбами, и именно они реализуют принятие решений вашей программой.

Инструкция `if`

Наиболее общим типом управляющих инструкций является инструкция `if`. Следующий за инструкцией `if` блок кода будет выполняться только в том случае, если результат вычисления условия равен `True` (условие истинно). Если же результат вычисления условия равен `False` (условие ложно), то блок не будет выполняться.

На обычном языке инструкция `if` интерпретируется следующим образом: "Если условие истинно, то выполнить данный блок кода". В Python инструкция `if` включает такие элементы:

- ключевое слово `if`;
- условие (т.е. выражение, вычисление которого дает значение `True` или `False`);

- двоеточие;
- блок кода с отступом, начинающийся в следующей строке.

Предположим, у вас имеется код, проверяющий совпадение с именем "Alice". (Здесь предполагается, что переменной `name` ранее было присвоено некоторое значение.)

```
if name == 'Alice':
    print('Hi, Alice.')
```

Все управляющие инструкции заканчиваются двоеточием, за которым следует блок кода (тело инструкции). В данном случае блок состоит из одной инструкции: `print('Hi, Alice.')`. Блок-схема рассматриваемого кода приведена на рис. 2.3.

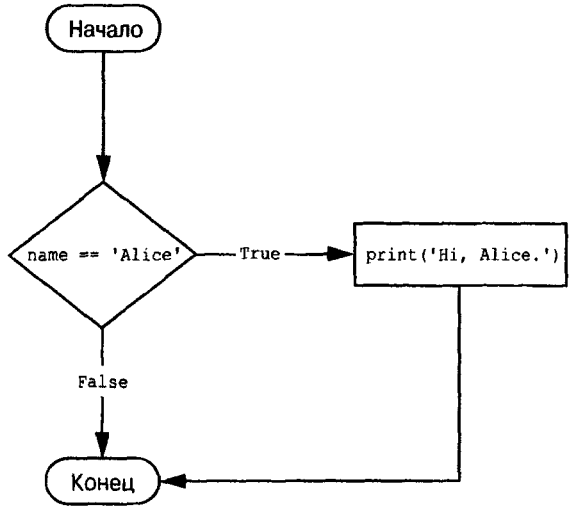


Рис. 2.3. Блок-схема инструкции `if`

Инструкция `else`

За блоком кода инструкции `if` может следовать необязательная инструкция `else` со своим блоком кода, который выполняется лишь в том случае, если условие `if` ложно. На обычном языке конструкция `if/else` может быть прочитана так: "Если условие истинно, выполнить данный блок кода. В противном случае выполнить следующий блок кода". В Python инструкция `else` не имеет условия и всегда состоит из следующих элементов:

- ключевое слово `else`;
- двоеточие;
- блок кода с отступом, начинающийся на следующей строке.

Возвращаясь к примеру с именем "Alice", рассмотрим код, содержащий инструкцию `else`, которая выводит другое приветствие, если имя пользователя — не "Alice".

```
if name == 'Alice':
    print('Hi, Alice.')
else:
    print('Hello, stranger.')
```

Блок-схема этого кода представлена на рис. 2.4.

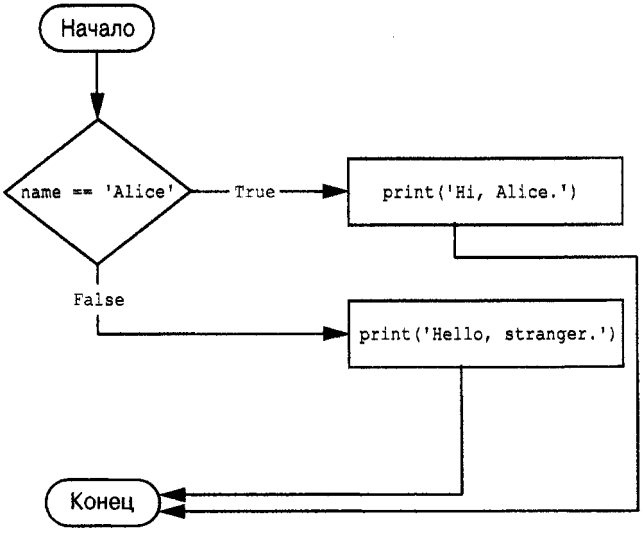


Рис. 2.4. Блок-схема инструкции `else`

Инструкция `elif`

В то время как из двух блоков кода, ассоциированных с инструкциями `if` и `else`, выполняется только один, иногда желательно выбирать для выполнения один из *нескольких* возможных блоков кода. Инструкция `elif` (сокращение от *else if*) может помещаться только после инструкции `if` или другой инструкции `elif`. Она предоставляет еще одно условие, которое проверяется лишь в том случае, если все предыдущие условия оказались ложными. В Python инструкция `elif` всегда состоит из следующих элементов:

- ключевое слово `elif`;
- условие (т.е. выражение, вычисление которого дает значение `True` или `False`);
- двоеточие;
- блок кода с отступом, начинающийся в следующей строке.

Добавим инструкцию `elif` в код, проверяющий имя, чтобы увидеть, как это работает на практике.

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
```

На этот раз дополнительно проверяется возраст пользователя, и если он меньше 12, то текст выводимого сообщения будет другим. Блок-схема этого кода представлена на рис. 2.5.

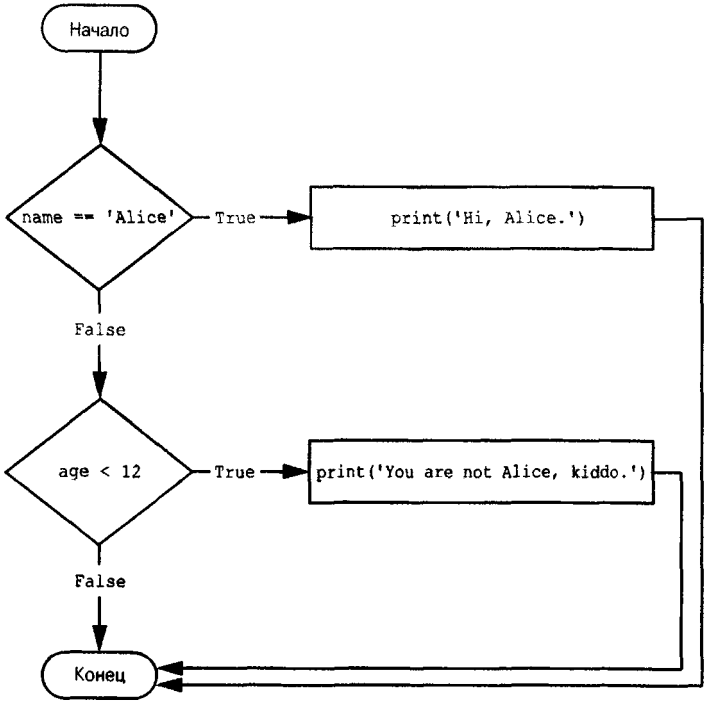


Рис. 2.5. Блок-схема инструкции `elif`

Блок кода `elif` выполняется, если выражение `age < 12` истинно, а выражение `name == 'Alice'` ложно. Но если оба условия ложны, то пропускаются оба блока кода. Никаких гарантий того, что выполнится хотя бы один из этих двух блоков кода, нет. Если имеется цепочка инструкций `elif`, то будет выполнен либо только один блок кода, либо ни один из них. Как только обнаруживается, что одно из условий истинно, все остальные блоки кода `elif` автоматически пропускаются. В качестве примера откройте в файловом редакторе новое окно, введите в нем приведенный ниже код и сохраните его в файле *vampire.py*.

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
elif age > 2000:
    print('Unlike you, Alice is not an undead, immortal vampire.')
elif age > 100:
    print('You are not Alice, grannie.')
```

В этом коде добавлены две дополнительные инструкции `elif`, обеспечивающие вывод разных сообщений в зависимости от возраста (`age`), указанного пользователем. Блок-схема этого кода показана на рис. 12.6.

Имейте, однако, в виду, что порядок расположения инструкций `elif` имеет значение. Сейчас мы намеренно внесем в код ошибку, переставив инструкции местами. Как вам уже известно, если одно из условий оказывается истинным, все остальные блоки кода `elif` автоматически пропускаются, и поэтому перестановка условий может создавать проблемы в коде. Измените код, как показано ниже, и сохраните его в файле *vampire2.py*.

```
    if name == 'Alice':
        print('Hi, Alice.')
    elif age < 12:
        print('You are not Alice, kiddo.')
❶   elif age > 100:
.       print('You are not Alice, grannie.')
    elif age > 2000:
        print('Unlike you, Alice is not an undead, immortal vampire.')
```

Предположим, что до выполнения кода в переменной `age` содержится значение `3000`. Вы ожидаете, что код выведет на экран строку `'Unlike you, Alice is not an undead, immortal vampire.'`. Однако, поскольку вычисление условия `age > 100` дает значение `True` (ведь `3000` больше `100`) ❶, на экран будет выведена строка `'You are not Alice, grannie.'`, тогда как остальные инструкции `elif` будут автоматически пропущены. Вспомните о том, что выполняется не более чем один из блоков инструкций `elif`, а порядок следования этих инструкций имеет значение!

Блок-схема предыдущего кода представлена на рис. 2.7. Обратите внимание на то, что ромбы для условий `age > 100` и `age > 2000` переставлены местами.

При необходимости вслед за последней инструкцией `elif` можно поместить инструкцию `else`. В этом случае гарантируется, что будет выполнен хотя бы один (и только один) блок кода. Если условия во всех инструкциях `if` и `elif` окажутся ложными, то выполнится блок кода инструкции `else`. Например, переделаем программу для распознавания имени "Alice" таким образом, чтобы в ней использовались инструкции `if`, `elif` и `else`.

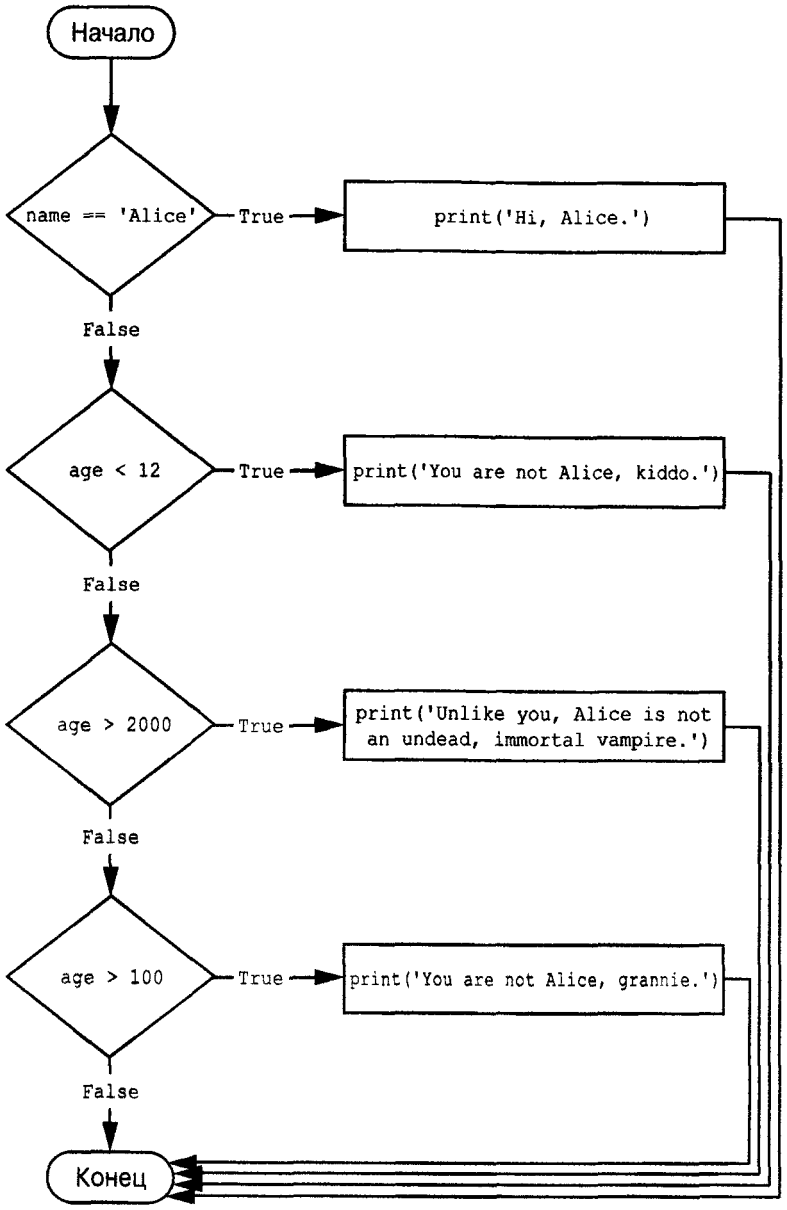


Рис. 2.6. Блок-схема кода с несколькими инструкциями `elif` в программе vampire.py

```
if name == 'Alice':
    print('Hi, Alice.')
elif age < 12:
    print('You are not Alice, kiddo.')
else:
    print('You are neither Alice nor a little kid.')
```

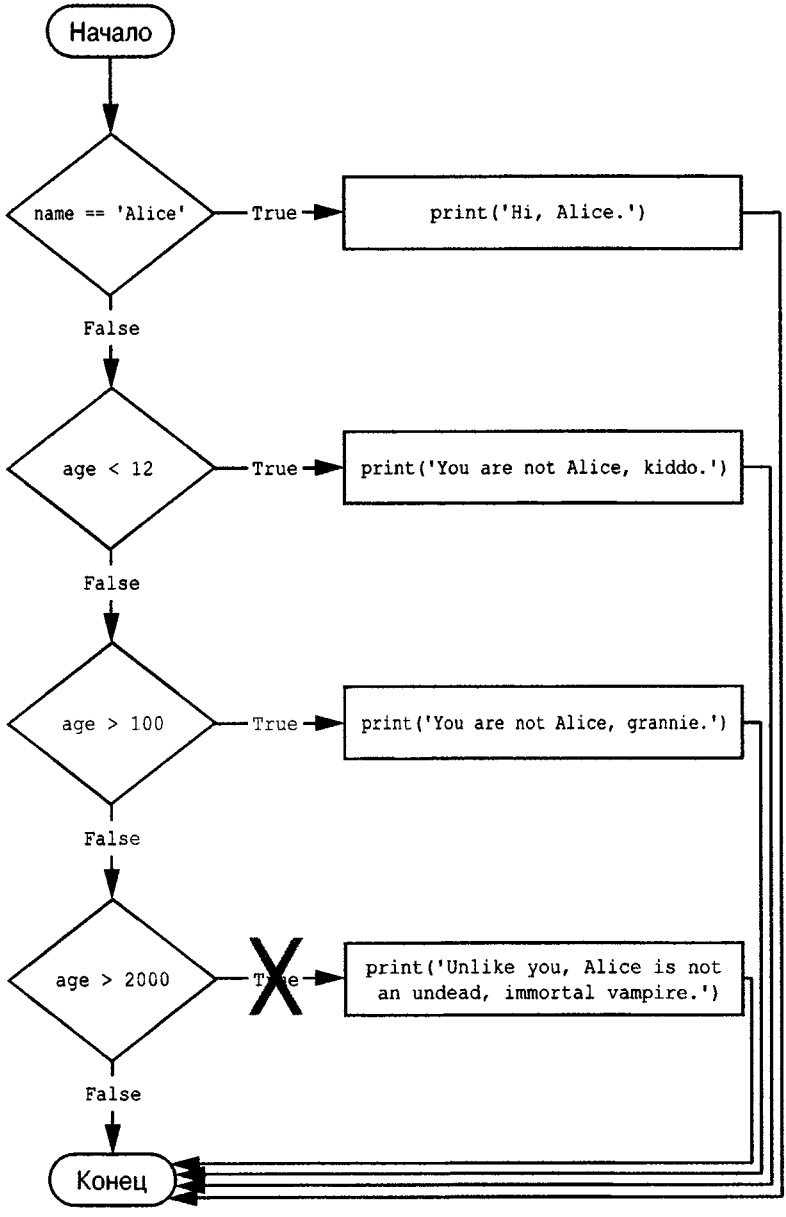


Рис. 2.7. Блок-схема программы vampire2.py. Выполнение программы вдоль перечеркнутого пути логически невозможно, поскольку если значение `age` *больше* `2000`*, то оно заведомо больше* `100`

На рис. 2.8 показана блок-схема нового кода, который мы сохраним в файле *littleKid.py*.

Пользуясь обычным языком, смысл управляющей конструкции этого типа можно передать так: "Если первое условие истинно, выполнить этот

блок кода. Иначе, если второе условие истинно, выполнить этот блок кода. В противном случае выполнить этот блок кода". В случае совместного использования всех трех инструкций не забывайте о правилах, устанавливающих последовательность их расположения, чтобы избежать ошибок, подобных той, которая проиллюстрирована на рис. 2.7. Во-первых, должна быть только одна инструкция `if`. Любые инструкции `elif`, которые вам могут понадобиться, должны следовать за инструкцией `if`. Во-вторых, если вы хотите быть уверены в том, что выполнится хотя бы один блок кода, замкните всю управляющую структуру инструкцией `else`.

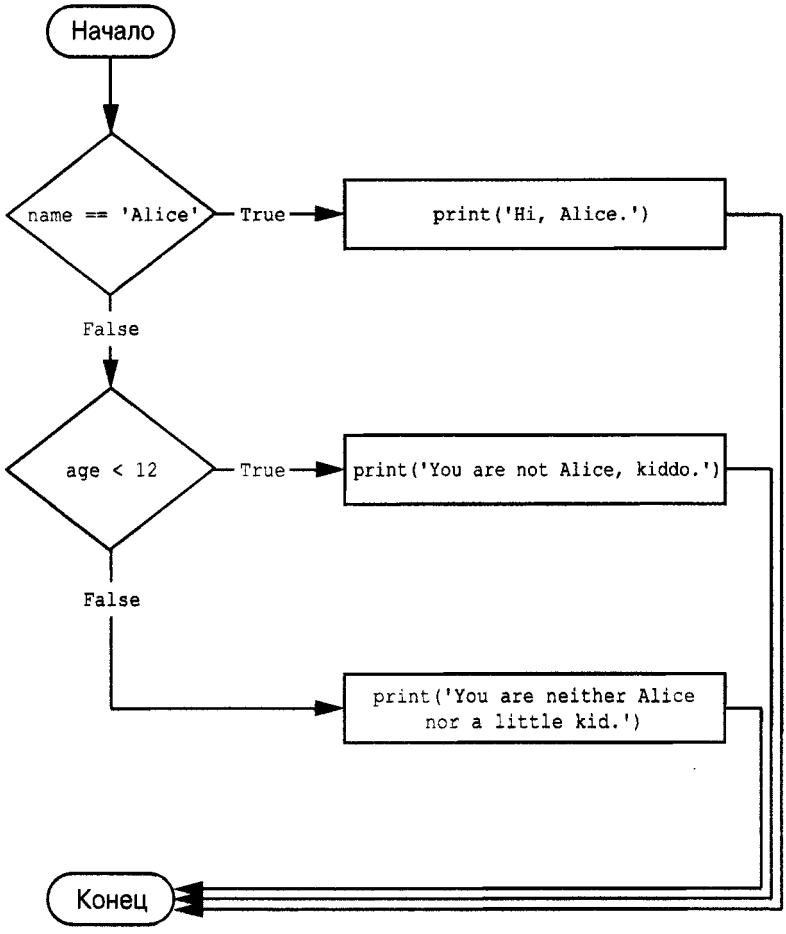


Рис. 2.8. Блок-схема программы littleKid.py

Цикл while

Инструкция `while` позволяет организовать многократное повторное выполнение блока кода. Блок кода инструкции `while` выполняется до тех пор,

пока выполняется указанное в ней условие. В Python инструкция `while` всегда состоит из следующих элементов:

- ключевое слово `while`;
- условие (т.е. выражение, вычисление которого дает значение `True` или `False`);
- двоеточие;
- блок кода с отступом, начинающийся в следующей строке.

Нетрудно заметить, что инструкции `while` имеет сходную с инструкцией `if` структуру, однако ведет себя иначе. По достижении конца блока кода `if` управление выполнением передается следующей инструкции. Однако по достижении конца блока кода инструкции `while` управление передается в ее начало, и программа продолжает выполнение того же блока кода. Инструкцию `while` часто называют *циклом* `while` или просто *циклом*.

Сравним, как работают инструкция `if` и цикл `while`, использующие одно и то же условие, на основании проверки которого в обоих случаях предпринимаются одни и те же действия. Вот так выглядит код с инструкцией `if`.

```
spam = 0
if spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

А вот так выглядит код с инструкцией `while`.

```
spam = 0
while spam < 5:
    print('Hello, world.')
    spam = spam + 1
```

Эти инструкции весьма похожи: как `if`, так и `while` проверяют значение переменной `spam`, и если оно меньше пяти, то выводится сообщение. Тем не менее эти фрагменты кода выполняются совершенно по-разному. Если в инструкции `if` вывод представляет собой одиночное сообщение `"Hello, world."`, то в инструкции `while` это сообщение выводится целых пять раз! Чтобы разобраться в том, почему так происходит, обратимся к соответствующим блок-схемам, представленным на рис. 2.9 и 2.10.

Код с инструкцией `if` тестирует условие и выводит текст `Hello, world.`, если условие истинно. С другой стороны, код с инструкцией `while` выводит этот текст пять раз. После этого он прекращает свою работу, поскольку целочисленное значение, хранящееся в переменной `spam`, увеличивается на единицу на каждой итерации цикла, а это означает, что цикл выполнится именно пять раз, прежде чем условие `spam < 5` примет значение `False`.

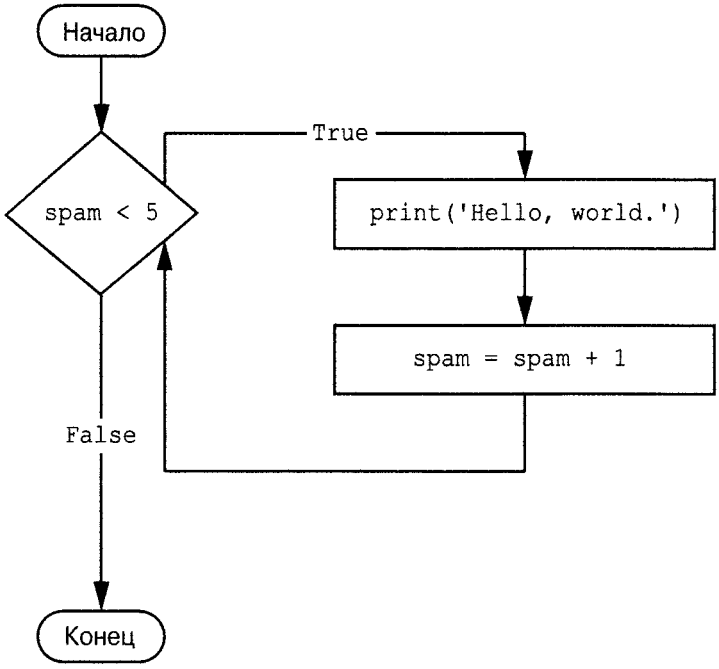


Рис. 2.9. Блок-схема кода с инструкцией `if`

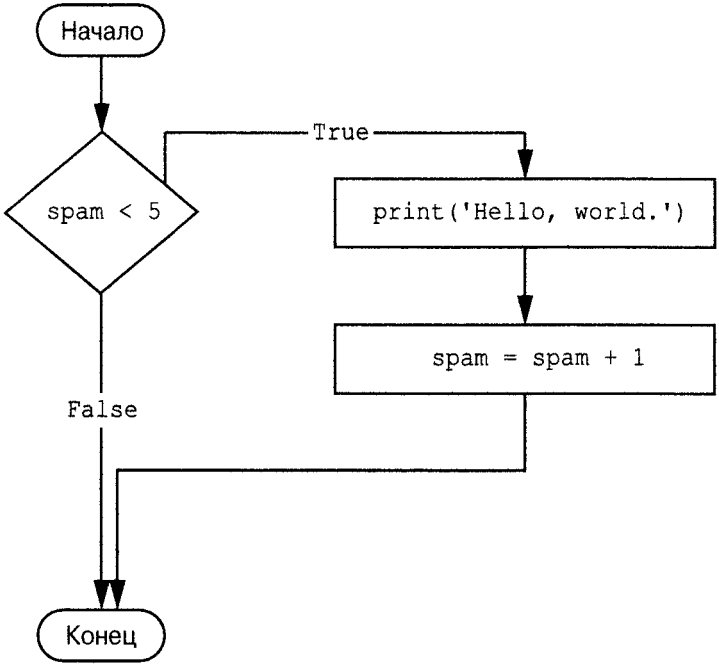


Рис. 2.10. Блок-схема кода с инструкцией `while`

В цикле while условие всегда проверяется в начале каждой *итерации* (т.е. каждый раз, когда выполняется цикл). Если условие интерпретируется как True, то блок кода выполняется, а затем вновь проверяется условие. Сразу же после того как обнаруживается, что условие равно False, блок кода while пропускается.

Назойливый цикл while

Ниже приведен пример простой программы, которая без устали просит вас ввести свое имя, но на самом деле ожидает ввода не вашего имени, а буквального текста 'your name' ('ваше имя'). Выберите пункты меню File⇨New Window (Файл⇨Новое окно) для открытия нового окна файлового редактора, введите приведенный ниже код и сохраните его в файле *yourName.py*.

```
❶ name = ''
❷ while name != 'your name':
      print('Please type your name.')
❸     name = input()
❹ print('Thank you!')
```

Сначала программа присваивает переменной name ❶ значение в виде пустой строки. Это сделано для того, чтобы вычисление условия name != 'your name' дало значение True и программа приступила к выполнению блока кода цикла while ❷.

Затем код в теле цикла запрашивает ввод имени пользователя и присваивает его переменной name ❸. Поскольку это последняя строка блока, выполнение возобновляется с самого начала цикла while, где вновь тестируется условие. Если значение name *не равно* строке 'your name', то вычисление условия дает значение True, в результате чего вновь начинает выполняться тело цикла.

Но как только пользователь введет текст your name, условие примет следующий вид: 'your name' != 'your name', что эквивалентно значению False. Поскольку теперь условие ложно, выполнение цикла прекращается и управление передается коду, следующему за циклом ❹. Блок-схема программы *yourName.py* приведена на рис. 2.11.

Давайте проверим, как работает программа *yourName.py*. Нажмите клавишу <F5> и несколько раз введите текст, отличный от your name, прежде чем предоставить программе то, что она требует.

```
Please type your name.
Al
Please type your name.
Albert
Please type your name.
```

```
Please type your name.
your name
Thank you!
```

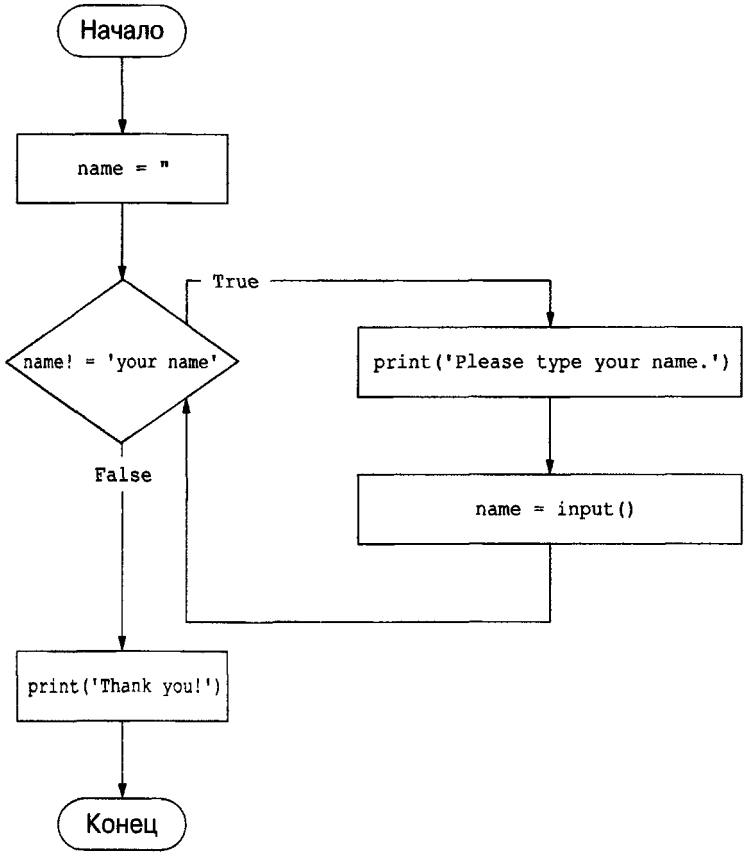


Рис. 2.11. Блок-схема программы yourName.py

Если вы так и не введете текст `your name`, то условие цикла `while` никогда не сможет принять значение `False`, и программа будет бесконечно повторять свой запрос. В данном случае вызов функции `input()` предоставляет пользователю возможность ввести строку, которая позволит программе покинуть цикл. В других программах возможны случаи, когда значение условия никогда не сможет измениться, и это станет проблемой. Рассмотрим, как можно организовать принудительный выход из цикла `while`.

Инструкция `break`

Существует быстрый способ заставить программу преждевременно выйти из цикла `while`. Если программа в процессе выполнения достигает

инструкции `break`, то выполнение цикла немедленно прерывается (`break` — ключевое слово Python).

Довольно просто, не правда ли? Ниже приведен пример программы, которая делает то же самое, что и предыдущая, но использует инструкцию `break` для выхода из цикла. Введите следующий код и сохраните его в файле *yourName2.py*.

```
❶ while True:
      print('Please type your name.')
❷     name = input()
❸     if name == 'your name':
❹        break
❺ print('Thank you!')
```

Первая строка ❶ создает *бесконечный цикл*, т.е. цикл `while`, условие которого всегда истинно (`True`). Программа всегда будет входить в цикл и выйдет из него только в том случае, если выполнится инструкция `break`. (Бесконечный цикл, выйти из которого невозможно, — распространенная программная ошибка.)

Так же, как и ранее, программа запрашивает ввод пользователем текста `your name` ❷. Однако теперь в цикле `while` присутствует инструкция `if` ❸, которая проверяет, совпадает ли значение переменной `name` со строкой `your name`. В случае истинности этого условия выполняется инструкция `break` ❹, и управление передается инструкции `print('Thank you')` ❺. В противном случае предложение `if`, содержащее инструкцию `break`, пропускается, программа переходит в конец цикла `while` и сразу же возвращается в его начало для проверки условия. Поскольку условие — это просто булево значение `True`, программа вновь входит в цикл и запрашивает ввод текста `your name`. Блок-схема этой программы приведена на рис. 2.12.

Запустите программу *yourNumber2.py* и введите тот же текст, который вы вводили для программы *yourNumber.py*. Новая версия программы должна реагировать на ваш ввод так же, как и исходная.

Инструкция `continue`

Как и инструкция `break`, инструкция `continue` используется в циклах. Когда программа в процессе выполнения достигает инструкции `continue`, управление немедленно передается в начало цикла, где условие вычисляется заново. (То же самое происходит и при обычном достижении программой конца цикла.)

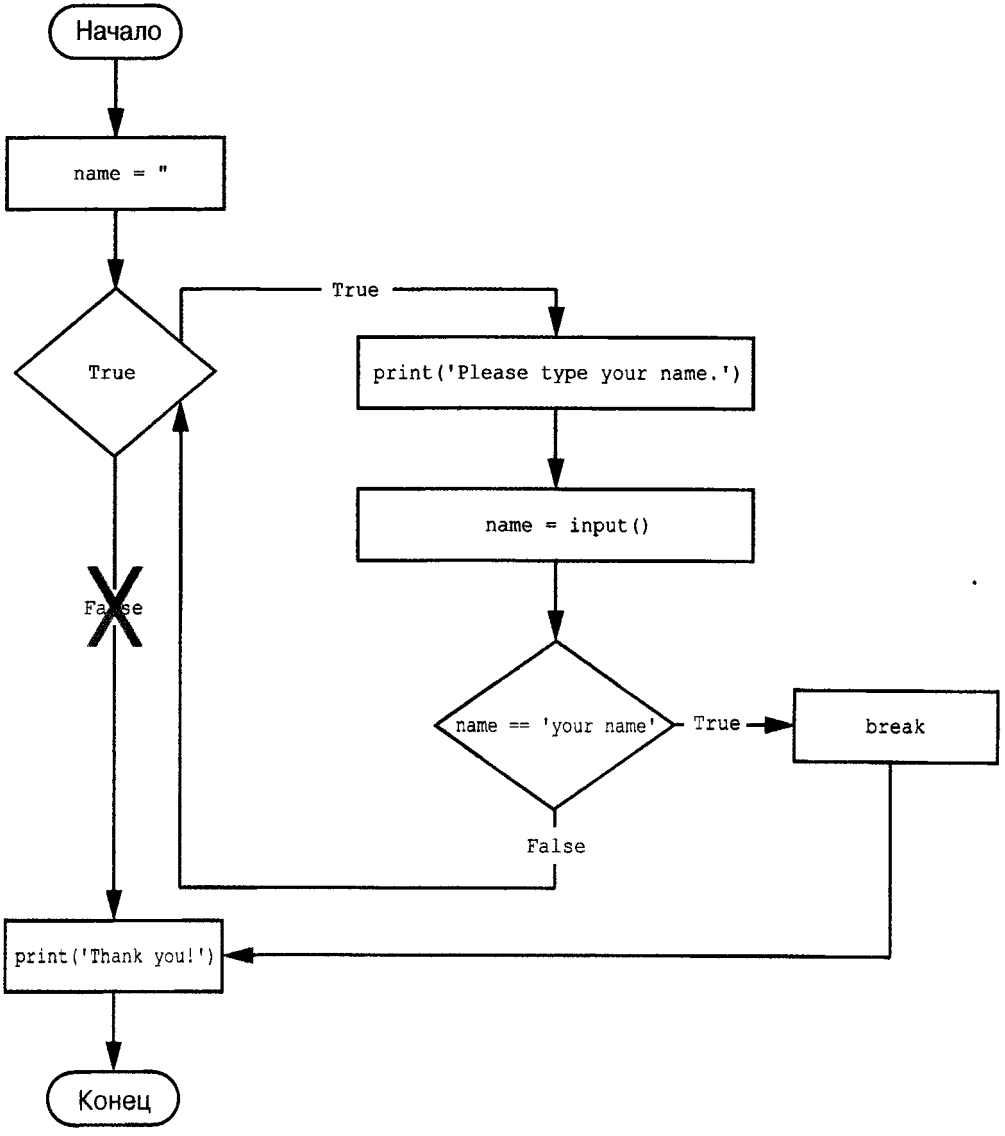


Рис. 2.12. Блок-схема программы yourName2.py с бесконечным циклом. Обратите внимание на то, что перечеркнутый путь выполнения программы логически никогда не может быть реализован, поскольку условие цикла всегда истинно

Продолжим написание программы, запрашивающей ввод имени и пароля. Введите следующий код в новом окне файлового редактора и сохраните его в файле *swordfish.py*.

Попали в ловушку бесконечного цикла?

Если вы обнаружили, что ваша программа увязла в бесконечном цикле, нажмите комбинацию клавиш <Ctrl+C>. В результате будет сгенерирована ошибка `KeyboardInterrupt`, что приведет к немедленному прекращению работы программы. Проверьте, как это работает на практике, создав простой бесконечный цикл в файловом редакторе и сохранив его в файле *infiniteloop.py*.

```
while True:
    print('Hello world!')
```

Когда вы запустите эту программу, она начнет безостановочно выводить на экран приветствие `Hello world!`, поскольку условие инструкции `while` всегда остается истинным. В интерактивной оболочке IDLE существуют два способа принудительного завершения работы программы: нажатие клавиш <Ctrl+C> и выбор пунктов меню **Shell⇨Restart Shell** (Оболочка⇨Перезапустить оболочку). Первый способ удобно использовать в любой ситуации, когда требуется немедленно прекратить работу программы, даже если это никак не связано с необходимостью прервать выполнение бесконечного цикла.

```
while True:
    print('Who are you?')
    name = input()
❶   if name != 'Joe':
❷       continue
    print('Hello, Joe. What is the password? (It is a fish.)')
❸   password = input()
    if password == 'swordfish':
❹       break
❺ print('Access granted.')
```

Если пользователь вводит любое другое имя, кроме "Joe" ❶, инструкция `continue` ❷ заставляет программу перейти в начало цикла. После повторной проверки условия программа всегда входит в тело цикла, поскольку условие всегда истинно (`True`). В случае же прохождения инструкции `if` запрашивается ввод пароля ❸. Если пользователь вводит пароль `swordfish`, то выполняется инструкция `break` ❹, программа покидает цикл `while` и выводит на экран текст `Access granted` ❺. В противном случае управление выполнением программы передается в конец цикла `while` и сразу же возвращается в его начало. Блок-схема этой программы представлена на рис. 2.13.

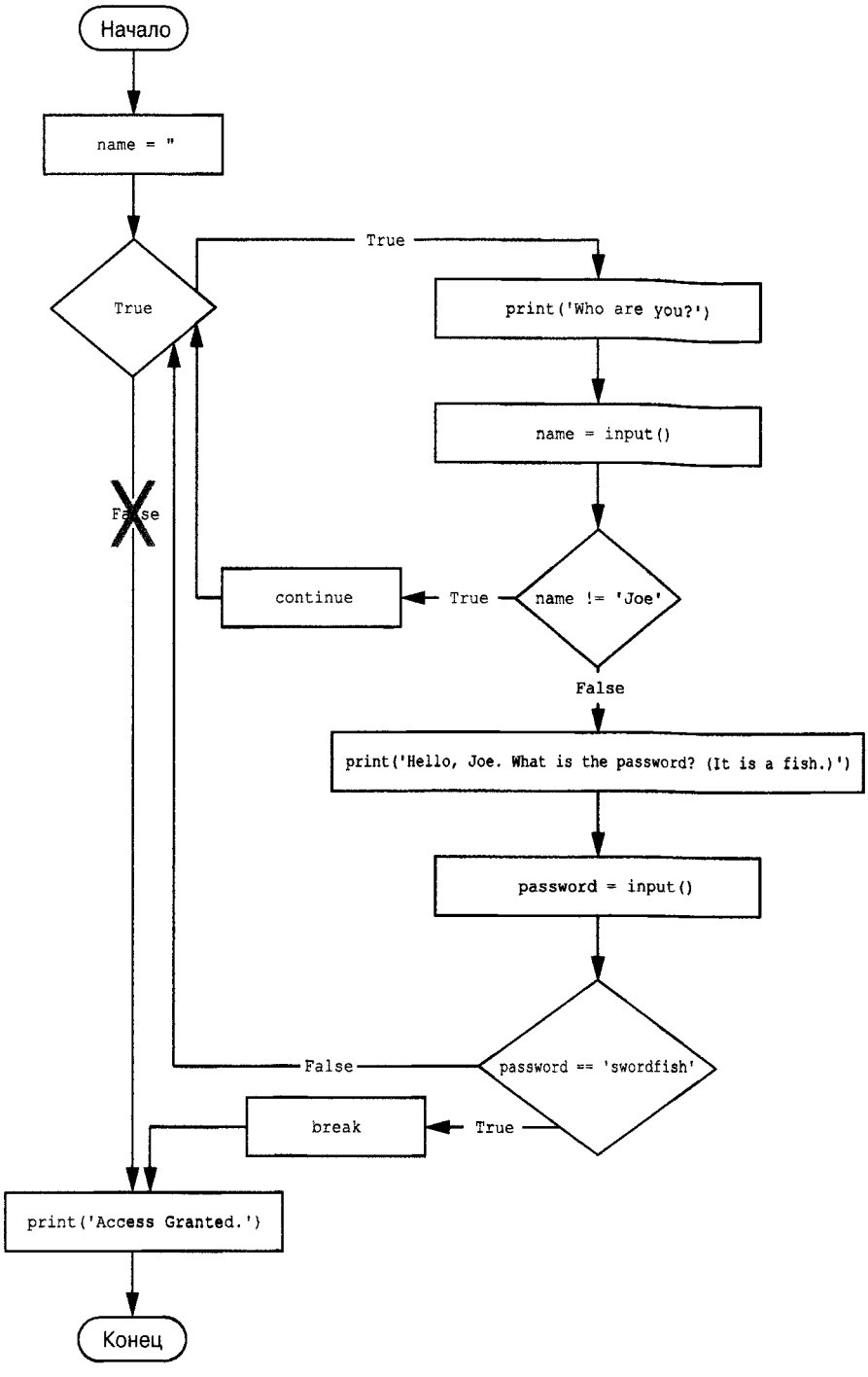


Рис. 2.13. Блок-схема программы swordfish.py. Перечеркнутый путь выполнения программы логически никогда не может быть реализован, поскольку условие цикла всегда истинно

"Истинные" и "ложные" значения

Существуют значения других типов данных, которые при проверке условий считаются эквивалентными значениям `True` и `False`. При использовании в выражениях условий значения `0`, `0.0` и `''` (пустая строка) считаются ложными (`False`), тогда как все другие значения считаются истинными (`True`). Взгляните, например, на следующую программу.

```
name = ''
while not name:❶
    print('Enter your name:')
    name = input()
print('How many guests will you have?')
numOfGuests = int(input())
if numOfGuests:❷
    print('Be sure to have enough room for all your guests.')❸
print('Done')
```

Если пользователь вводит пустую строку в качестве имени, то условие цикла `while` принимает значение `True` ❶, и программа продолжает запрашивать имя. Если значение переменной `numOfGuests` не равно `0` ❷, то значение условия считается равным `True`, и программа выводит напоминание ❸.

Вместо `not name` вы могли бы ввести `not name != ''`, а вместо `numOfGuests` — `numOfGuests != 0`, но использование истинных и ложных значений облегчает чтение кода.

Запустите программу и введите какой-либо текст. Программа не запросит ввод пароля до тех пор, пока вы не подтвердите, что ваше имя — "Joe"; после ввода правильного пароля она должна завершить выполнение.

```
Who are you?
I'm fine, thanks. Who are you?
Who are you?
Joe
Hello, Joe. What is the password? (It is a fish.)
Mary
Who are you?
Joe
Hello, Joe. What is the password? (It is a fish.)
swordfish
Access granted.
```

Цикл for и функция range()

Цикл while продолжает выполняться до тех пор, пока условие остается истинным. Но что если вы хотите выполнить блок кода лишь определенное количество раз? Это можно сделать с помощью цикла for или функции range().

Инструкция for выглядит в коде примерно как for i in range(5): и всегда включает следующие элементы:

- ключевое слово for;
- имя переменной;
- ключевое слово in;
- вызов функции range(), которой можно передать до трех целых чисел;
- двоеточие;
- блок кода с отступом, начинающийся в следующей строке.

Чтобы проверить на практике, как работает цикл for, создадим новую программу в файле fiveTimes.py.

```
print('My name is')
for i in range(5):
    print('Jimmy Five Times (' + str(i) + ')')
```

Блок кода цикла for выполняется пять раз. На первой итерации значение переменной i устанавливается равным 0. Вызов функции print() в теле цикла выводит текст Jimmy Five Times (0). Когда Python завершает итерацию, выполнив весь блок кода, управление передается в начало цикла, где инструкция for увеличивает значение переменной i на единицу. Именно поэтому вызов функции range(5) обеспечивает пятикратное выполнение блока кода цикла, устанавливая для i последовательные значения 0, 1, 2, 3 и 4. Целое значение, указанное при вызове функции range(), в этот ряд не входит. Блок-схема программы *fiveTimes.py* показана на рис. 2.14.

Когда вы запустите эту программу, она должна вывести пять строк текста Jimmy Five Times, каждая из которых заканчивается текущим значением i, и покинуть цикл for.

```
My name is
Jimmy Five Times (0)
Jimmy Five Times (1)
Jimmy Five Times (2)
Jimmy Five Times (3)
Jimmy Five Times (4)
```

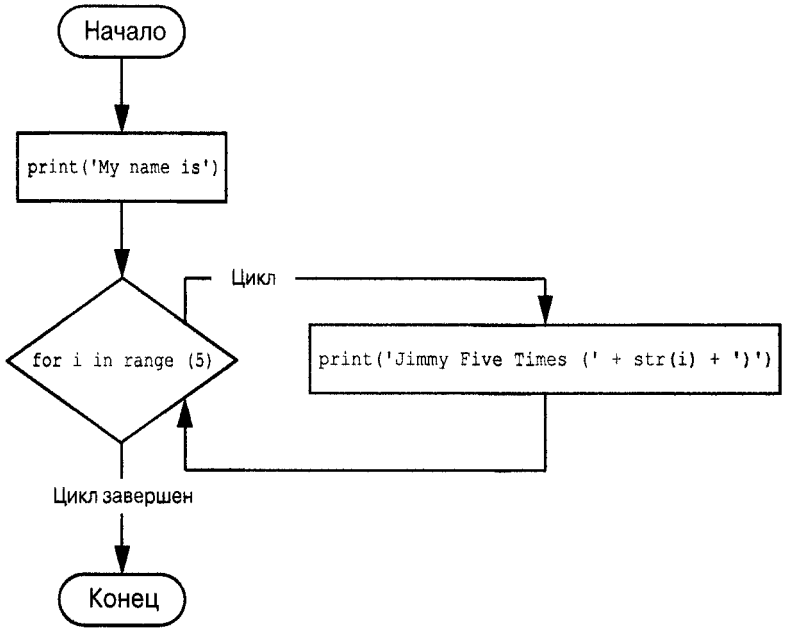


Рис. 2.14. Блок-схема программы fiveTimes.py

Примечание

В циклах `for` *также допускается использовать инструкции* `break` *и* `continue`. *Инструкция* `continue` *возвращает управление в начало цикла для выполнения следующей итерации, как если бы программа достигла конца цикла и вернулась к его началу обычным способом. В действительности инструкции* `continue` *и* `break` *могут использоваться только в циклах* `while` *и* `for`. *Если вы попытаетесь использовать их где-либо еще, Python выведет сообщение об ошибке.*

В основу еще одного примера цикла `for` положен исторический факт, связанный с известным математиком Карлом Фридрихом Гауссом. Как-то, когда Гаусс еще учился в школе, учитель дал классу следующее задание: найти сумму всех чисел от 0 до 100. Используя остроумный прием, юный Гаусс нашел нужную сумму в течение всего лишь нескольких секунд, но вы можете проделать соответствующие вычисления самостоятельно, написав программу на языке Python, в которой используется цикл `for`.

```
❶ total = 0
❷ for num in range(101):
❸     total = total + num
❹ print(total)
```

Правильный ответ — 5050. Сразу после запуска программы значение переменной `total` устанавливается равным `0` ❶. Затем в цикле `for` ❷ код `total = total + num` ❸ выполняется 100 раз. По завершении 100 итераций цикла в переменной `total` будет сохранена сумма всех целых чисел от 0 до 100. После этого значение `total` выводится на экран ❹. Даже на самых медленных компьютерах выполнение этой программы займет менее секунды.

(Юный Гаусс догадался, что всего имеется 50 пар чисел, сумма которых дает 100: 1 + 99, 2 + 98, 3 + 97 + ...; 49 + 51. Поскольку 50 × 100 = 5000, то после прибавления оставшегося без пары числа 50 мы получаем окончательный результат — 5050. Сообразительный ребенок!)

Эквивалентный цикл while

Все то, что делает цикл `for`, можно сделать с помощью цикла `while`; просто циклы `for` более компактны в записи. Перепишем код программы *fiveTimes.py*, использовав в новой версии цикл `while`, эквивалентный циклу `for`.

```
print('My name is')
i = 0
while i < 5:
    print('Jimmy Five Times (' + str(i) + ')')
    i = i + 1
```

Если вы запустите эту программу, то увидите, что она позволяет получить те же результаты, что и ее версия, в которой используется цикл `for`.

Аргументы начала, конца и шага диапазона функции range()

Некоторые функции могут вызываться с передачей им нескольких аргументов, разделенных запятыми, и функция `range()` — одна из них. Это позволяет изменять диапазон целых чисел, задаваемый функцией `range()`, включая его начальный и конечный элементы.

```
for i in range(12, 16):
    print(i)
```

Первый аргумент определяет, с какого значения начинает изменяться переменная цикла `for`, а второй — верхнюю границу диапазона изменения этой переменной, но сам в этот диапазон не включается.

```
12
13
14
15
```

Также допускается вызов функции `range()` с тремя аргументами. Первые два аргумента определяют начало и конец диапазона изменения переменной цикла `for` в соответствии с приведенным выше описанием, а третий задает шаг. *Шаг* – это приращение переменной в конце каждой итерации цикла.

```
for i in range(0, 10, 2):
    print(i)
```

Следовательно, вызов `range(0, 10, 2)` обеспечивает изменение переменной цикла от нуля до восьми с шагом два.

```
0
2
4
6
8
```

Функция `range()` очень гибкая в отношении формирования последовательностей целых чисел для циклов `for`. Например, задание отрицательного числа в качестве шага обеспечивает изменение переменной цикла от бо́льших значений к меньшим.

```
for i in range(5, -1, -1):
.   print(i)
```

Выполнение цикла `for` для вывода значений переменной `i` с использованием вызова `range(5, -1, -1)` даст следующий результат.

```
5
4
3
2
1
0
```

Импортирование модулей

Любой программе на языке Python доступен базовый набор функций, которые называются *встроенными*, в число которых входят такие функции, как `print()`, `input()` и `len()`, с которыми вы уже успели познакомиться. Кроме того, в поставку Python входит набор модулей, который называется *стандартной библиотекой*. Каждый модуль – это программа на Python, содержащая группу родственных функций, которые вы можете внедрять в свои

программы. Например, модуль `math` включает математические функции, модуль `random` — функции для работы со случайными числами, и т.д.

Прежде чем вы сможете использовать функции, входящие в модуль, его необходимо импортировать с помощью инструкции `import`. Инструкция `import` в коде состоит из следующих элементов:

- ключевое слово `import`;
- имя модуля;
- необязательные дополнительные имена модулей, разделенные запятыми.

Как только модуль импортирован, вы можете использовать любую из входящих в него функций. Давайте проверим, как работает модуль `random`, предоставляющий доступ к функции `random.ranint()`.

Введите в файловом редакторе приведенный ниже код и сохраните его в файле `printRandom.py`.

```
import random
for i in range(5):
    print(random.randint(1, 10))
```

Выполнив программу, вы должны получить следующий вывод.

```
4
1
8
4
1
```

Функция `random.randint()` возвращает случайное число, лежащее в диапазоне между двумя целочисленными значениями, которые передаются функции в качестве аргументов. Поскольку функция `randint()` находится в модуле `random`, его имя должно указываться в виде префикса (через точку) перед именем функции, чтобы Python мог определить, что данную функцию следует искать в модуле `random`.

Вот пример инструкции `import`, которая импортирует четыре различных модуля:

```
import random, sys, os, math
```

Теперь мы можем использовать любую из функций, находящихся в этих четырех модулях.

Инструкция from import

Альтернативная форма инструкции import состоит из ключевого слова from, за которым следуют имя модуля, ключевое слово import и символ "звездочка", например from random import *.

При использовании этой формы импорта добавлять префикс random. к имени функции, вызываемой из модуля random, не требуется. Однако использование полного имени функции повышает удобочитаемость кода, поэтому лучше использовать обычную форму инструкции import.

Преждевременное прекращение выполнения программы с помощью вызова sys.exit()

И в завершение хочу познакомить вас с тем, как прекратить выполнение программы. Это всегда происходит автоматически после выполнения последней инструкции программы. Однако существует возможность принудительно прекратить работу программы, или, как говорят, выйти из нее, с помощью вызова функции sys.exit(). Поскольку эта функция находится в модуле sys, вы должны импортировать его до того, как он будет использоваться.

Откройте новое окно файлового редактора, введите в него приведенный ниже код и сохраните его в файле *exitExample.py*.

```
import sys

while True:
    print('Type exit to exit.')
    response = input()
    if response == 'exit':
        sys.exit()
    print('You typed ' + response + '.')
```

Запустите эту программу в IDLE. В программе имеется бесконечный цикл, в котором отсутствует инструкция break. Единственная возможность завершить работу этой программы — это ввести exit в ответ на запрос, что приведет к вызову функции sys.exit(). В случае совпадения значения переменной response со строкой 'exit' выполнение программы прекращается. Поскольку значение переменной response устанавливается функцией input(), для завершения программы пользователь должен ввести слово exit.

Резюме

Используя выражения, результатом вычисления которых является значение `True` или `False` (такие выражения называют условиями), можно писать программы, способные принимать решения относительно того, какие участки кода должны выполняться, а какие — пропускаться. Кроме того, можно организовать многократное выполнение кода в цикле до тех пор, пока вычисление условия дает значение `True`. В тех случаях, когда требуется осуществить выход из цикла или возврат к его началу, используются инструкции `break` и `continue`.

Управляющие инструкции позволяют писать намного более интеллектуальные программы. Существуют и другие возможности по управлению выполнением программ, обеспечиваемые написанием собственных функций, о чем пойдет речь в следующей главе.

Контрольные вопросы

1. Каковы два возможных значения данных булева типа? Как они записываются?
2. Назовите три булевых оператора.
3. Напишите таблицы истинности (т.е. запишите результаты для всех возможных комбинаций оператора и двух булевых значений) для каждого из булевых операторов.
4. Каковы результаты вычисления приведенных ниже выражений?

```
(5 > 4) and (3 == 5)
not (5 > 4)
(5 > 4) or (3 == 5)
not ((5 > 4) or (3 == 5))
(True and True) and (True == False)
(not False) or (not True)
```

5. Назовите шесть операторов сравнения.
6. В чем суть различия между оператором равенства и оператором присваивания?
7. Объясните, что такое условие и где используются условия.
8. Идентифицируйте три блока в приведенном ниже коде.

```
spam = 0
if spam == 10:
    print('eggs')
    if spam > 5:
        print('bacon')
    else:
```

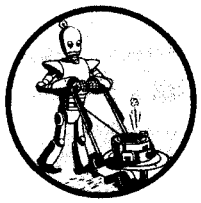
```
        print('ham')
    print('spam')
print('spam')
```

9. Напишите код, который выводит разные сообщения в зависимости от значения, хранящегося в переменной `spam`: `Hello` — при значении `1`, `Howdy` — при значении `2` и `Greetings!` — при любом другом значении.
10. Какую комбинацию клавиш следует нажать, чтобы вывести программу из бесконечного цикла?
11. Чем различаются инструкции `break` и `continue`?
12. Чем различаются вызовы функций `range(10)`, `range(0, 10)` и `range(0, 10, 1)` в цикле `for`?
13. Напишите короткую программу, выводящую числа от 1 до 10 с помощью цикла `for`. Затем напишите аналогичную программу, в которой используется цикл `while`.
14. Как бы вы вызвали функцию `bacon()`, хранящуюся в модуле `spam`, после того, как импортировали этот модуль?

Дополнительное задание. Поищите в Интернете информацию о функциях `round()` и `abs()` и выясните, что они делают. Самостоятельно поэкспериментируйте с ними в интерактивной оболочке.

3

ФУНКЦИИ



В предыдущих главах вы уже познакомились с функциями `print()`, `input()` и `len()`. Python предоставляет встроенные функции наподобие этих, но вы можете писать и собственные функции. *Функция* — это нечто вроде мини-программы в рамках большой программы.

Чтобы лучше понять, как работают функции, обратимся к конкретному примеру. Введите в файловом редакторе исходный код приведенной ниже программы и сохраните его в файле *helloFunc.py*.

```
❶ def hello():
❷     print('Howdy!')
      print('Howdy!!!')
      print('Hello there.')

❸     hello()
      hello()
      hello()
```

Первая строка — это инструкция `def` ❶, которая определяет функцию с именем `hello()`. Блок кода, следующий за инструкцией `def` ❷, образует тело функции. Этот код выполняется при вызове функции, а не при ее первоначальном определении.

Последующие три строки `hello()` ❸ — это вызовы функции. В коде вызов функции обозначается указанием ее имени с последующей парой круглых скобок, которые могут содержать некоторое количество аргументов. Когда программа в процессе выполнения достигает вызова функции, управление передается первой строке кода этой функции, после чего выполняется весь ее код. По достижении конца функции управление возвращается строке, которая ее вызвала, и продолжается выполнение дальнейшего кода.

Поскольку в данной программе функция `hello()` вызывается три раза, столько же выполняется и код этой функции. Выполнив эту программу, вы должны получить следующий вывод.

```
Howdy!
Howdy!!!
Hello there.
Howdy!
Howdy!!!
Hello there.
Howdy!
Howdy!!!
Hello there.
```

Основное назначение функции — группирование многократно выполняемого кода. Если бы мы не определили функцию, то пришлось бы копировать этот код в буфер обмена и вставлять его в программу везде, где он используется, в результате чего программа выглядела бы примерно так.

```
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
print('Howdy!')
print('Howdy!!!')
print('Hello there.')
```

В целом старайтесь избегать дублирования кода, поскольку, если понадобится обновить программу (например, для исправления ошибок), вам придется вносить изменения в код везде, куда вы его копировали.

По мере накопления программистского опыта вы заметите, что все чаще занимаетесь тем, что избавляетесь от дублирования кода, сводя к минимуму использование метода копирования и вставки. Такие меры позволяют сократить размер программ, а также упростить их чтение и обновление.

Инструкции `def` с параметрами

Вызывая функции наподобие `print()` или `len()`, вы передаете им значения, которые называются *аргументами* в данном контексте, указывая их в скобках. Вы также можете определять собственные функции, принимающие аргументы. Введите в файловом редакторе приведенный ниже пример и сохраните его в файле *helloFunc2.py*.

```
❶ def hello(name):
❷     print('Hello ' + name)

❸ hello('Alice')
  hello('Bob')
```

Выполнив эту программу, вы должны получить следующий вывод.

```
Hello Alice
Hello Bob
```

В данной программе определение функции `hello()` содержит параметр `name` ❶. *Параметр* — это переменная, в которой сохраняется аргумент функции при ее вызове. Когда функция вызывается в первый раз, ей передается аргумент `'Alice'` ❸. Когда поток управления переходит в функцию, переменной `name` автоматически присваивается значение `'Alice'`, которое и выводится на экран инструкцией `print()` ❷.

Следует особо отметить то обстоятельство, что после выполнения возврата из функции сохраненное в параметре значение теряется. Например, если вы добавите вызов `print(name)` после вызова `hello('Bob')` в предыдущей программе, то будет выведено сообщение об ошибке `NameError`, поскольку вне функции переменной с именем `name` не существует. При возврате из функции после вызова `hello('Bob')` эта переменная уничтожается, поэтому вызов `print(name)` будет ссылаться на несуществующую переменную.

Это аналогично тому, как при завершении программы информация о ее переменных теряется. Более подробно о том, почему так происходит, речь пойдет далее, когда мы будем обсуждать локальную область видимости переменных внутри функции.

Инструкция `return` и возвращаемые значения

Когда вы вызываете функцию `len()` и передаете ей аргумент, например `'Hello'`, функция вычисляет целое значение `5`, представляющее длину строки, которая была передана функции. В общем случае значение, вычисляемое в результате вызова функции, называется *возвращаемым значением* данной функции.

Создавая функцию с использованием инструкции `def`, вы можете указать, какое значение должно возвращаться, с помощью инструкции `return`. Инструкция `return` состоит из следующих частей:

- ключевое слово `return`;
- значение или выражение, которое должна вернуть функция.

Если в инструкции return используется выражение, то возвращается результат вычисления данного выражения. Например, в следующей программе определяется функция, которая каждый раз возвращает другую строку, в зависимости от того, какое число передано в качестве аргумента. Введите в файловом редакторе следующий код и сохраните его в файле *magic8Ball.py*.

```
❶ import random

❷ def getAnswer(answerNumber):
❸     if answerNumber == 1:
          return 'It is certain'
      elif answerNumber == 2:
          return 'It is decidedly so'
      elif answerNumber == 3:
          return 'Yes'
      elif answerNumber == 4:
          return 'Reply hazy try again'
      elif answerNumber == 5:
          return 'Ask again later'
      elif answerNumber == 6:
          return 'Concentrate and ask again'
      elif answerNumber == 7:
          return 'My reply is no'
      elif answerNumber == 8:
          return 'Outlook not so good'
      elif answerNumber == 9:
          return 'Very doubtful'

❹ r = random.randint(1, 9)
❺ fortune = getAnswer(r)
❻ print(fortune)
```

Когда запускается эта программа, Python в первую очередь импортирует модуль random ❶. Затем определяется функция getAnswer() ❷. Поскольку функция лишь определяется (а не вызывается), ее код не выполняется. Далее вызывается функция random.randint() с двумя аргументами — 1 и 9 ❹. Эта функция возвращает случайное целое число, принадлежащее диапазону чисел от 1 до 9 (включая сами эти значения), и это число сохраняется в переменной r.

Функция getAnswer() вызывается с передачей ей переменной r в качестве аргумента ❺. Выполнение программы переходит в начало функции getAnswer() ❸, и значение r сохраняется в параметре answerNumber. Затем функция возвращает одно из множества возможных строковых значений, зависящих от значения answerNumber. Выполнение возвращается строке в нижней части программы, которая первоначально вызвала функцию getAnswer() ❺. Возвращенная строка присваивается переменной fortune, которая затем передается функции print() ❻ и выводится на экран.

Поскольку возвращаемые значения могут передаваться в качестве аргументов другим вызываемым функциям, вместо строк

```
r = random.randint(1, 9)
fortune = getAnswer(r)
print(fortune)
```

можно использовать следующую эквивалентную им строку:

```
print(getAnswer(random.randint(1, 9)))
```

Вспомните, что выражения состоят из значений и операторов. Вызов функции можно использовать в выражениях, поскольку это эквивалентно использованию возвращаемого значения функции.

Значение None

В языке Python определено значение None, представляющее отсутствие значения. None — единственный представитель типа данных NoneType. (Аналогичные значения в других языках программирования фигурируют под именами null, nil и undefined.) Точно так же, как имена булевых значений True и False, имя значения None всегда должно вводиться с использованием прописной буквы N.

Это "значение, не имеющее значения" может быть очень полезным, если вам нужно сохранить нечто такое, что невозможно перепутать с настоящим значением переменной. В частности, оно используется в качестве возвращаемого значения функции print(). Последняя отображает текст на экране, и ей необязательно возвращать значение так, как это делают функции len() и input(). Однако, поскольку вычисление любой функции должно давать возвращаемое значение, функция print() возвращает значение None. Чтобы увидеть, как это работает, введите в интерактивной оболочке следующий код.

```
>>> spam = print('Hello!')
Hello!
>>> None == spam
True
```

Незаметно для пользователя, "за кулисами", Python добавляет инструкцию return None в конец определения любой функции, в которой инструкция return отсутствует. Это аналогично тому, как цикл while или for заканчивается неявной инструкцией continue. Кроме того, если вы используете инструкцию, не указывая значения (т.е. записываете только само ключевое слово return), то функция возвращает значение None.

Именованные аргументы и функция `print()`

Большинство аргументов идентифицируется по их позиции в вызове функции. Например, вызов `random.randint(1, 10)` отличается от вызова `random.randint(10, 1)`. При вызове функции `random.randint(1, 10)` будет возвращено случайное целое число из диапазона от `1` до `10`, поскольку первый аргумент имеет смысл нижней границы диапазона, а второй — верхней (в то время как вызов `random.randint(10, 1)` приводит к возникновению ошибки).

В то же время *именованные аргументы* идентифицируются по имени, указываемому перед ними при вызове функции. Именованные аргументы часто используются в качестве необязательных параметров. Например, функция `print()` имеет необязательные параметры `end` и `sep`, с помощью которых можно задать соответственно текст, который должен выводиться в конце аргументов, и текст, который должен выводиться между аргументами (разделитель).

Если вы запустите программу

```
print('Hello')
print('World')
```

то вывод будет таким:

```
Hello
World
```

Две строки появятся в виде отдельных строк вывода, поскольку функция `print()` автоматически добавляет символы новой строки в конец каждой строки, которая ей передается. В случае необходимости вместо символа новой строки можно использовать другую строку, задав ее с помощью именованного аргумента `end`. Например, для программы

```
print('Hello', end='')
print('World')
```

вывод будет таким:

```
HelloWorld
```

Здесь текст выводится в одной строке ввиду отсутствия символа новой строки после текста `'Hello'`. Вместо него выводится пустая строка. Этот прием пригодится вам в тех случаях, когда потребуется отмена использования символов новой строки, автоматически добавляемых в конце каждого вывода, осуществляемого с помощью функции `print()`.

Точно так же, если вы передаете функции `print()` несколько строковых значений, то по умолчанию в качестве разделителя используется пробел. Введите в интерактивной оболочке следующий код.

```
>>> print('cats', 'dogs', 'mice')
cats dogs mice
```

Однако вместо заданной по умолчанию пустой строки можно использовать другую, передав функции именованный аргумент `sep`. Введите в интерактивной оболочке следующий текст.

```
>>> print('cats', 'dogs', 'mice', sep=',')
cats,dogs,mice
```

Именованные аргументы также можно добавлять в функции, написанные вами самостоятельно, однако сначала необходимо изучить такие типы данных, как списки и словари, о которых пойдет речь в двух следующих главах. А пока что достаточно знать лишь то, что у некоторых функций имеются именованные аргументы, которые можно задавать при вызове функции.

Локальная и глобальная области видимости

О параметрах и переменных, получающих значения в теле вызванной функции, говорят, что они существуют в *локальной области видимости* этой функции. О переменных, значения которым присваиваются вне функций, говорят, что они существуют в *глобальной области видимости*. Переменные, существующие в локальной области видимости, называются *локальными переменными*, тогда как переменные, существующие в глобальной области видимости, называются *глобальными переменными*. Переменная может относиться только к одному из этих типов; она не может быть локальной и глобальной одновременно.

Вы можете представлять себе область видимости как контейнер для переменных. При уничтожении области видимости все значения, которые хранятся в относящихся к ней переменных, теряются. Существует только одна глобальная область видимости, и она создается в момент начала выполнения программы. Когда программа завершает работу, глобальная область видимости уничтожается, и все ее переменные теряются. В противном случае при запуске программы в очередной раз переменные содержали бы те же значения, что и во время последнего сеанса выполнения программы.

Локальная область видимости создается всякий раз, когда вызывается функция. Любая переменная, которой присваивается значение в этой функции, существует в данной локальной области видимости. При возврате из функции локальная область видимости уничтожается, и эти переменные

теряются. Когда вы в следующий раз вызовете эту же функцию, локальные переменные не будут помнить те значения, которые хранились в них при последнем вызове функции.

Области видимости переменных играют важную роль по следующим причинам:

- локальные переменные не могут использоваться в коде, относящемся к глобальной области видимости;
- в то же время локальная область видимости имеет доступ к глобальным переменным;
- код, находящийся в локальной области видимости функции, не может использовать переменные из любой другой локальной области видимости;
- разные переменные могут иметь одно и то же имя, если они относятся к разным областям видимости. Это означает, что одновременно могут существовать как локальная переменная с именем `spam`, так и глобальная переменная с таким же именем.

Использование в Python различных областей видимости и отказ от того, чтобы считать все переменные глобальными, обеспечивает то преимущество, что функции, изменяющие переменные, могут взаимодействовать с остальным кодом программы только через свои параметры и возвращаемое значение. При таком подходе контролировать поведение программы намного легче и безопаснее. Если бы все переменные в вашей программе были только глобальными, то ошибочное значение какой-либо переменной могло бы передаваться в другие части программы, тем самым затрудняя поиск причины ошибки. Значение глобальной переменной может устанавливаться в любом месте программы, а вся программа может насчитывать тысячи строк! Но если ошибка связана с неверным значением локальной переменной, то для нахождения причины ошибки вам достаточно исследовать лишь код функции, в которой устанавливается значение данной переменной.

В использовании глобальных переменных в небольших программах нет ничего предосудительного, но придерживаться такого же подхода в случае крупных программ — плохая привычка.

Локальные переменные не могут использоваться в глобальной области видимости

Рассмотрим следующую программу, попытка выполнения которой приводит к ошибке.

```
def spam():
    eggs = 31337
```

```
spam()
print(eggs)
```

Выполнив эту программу, вы получите следующий вывод.

```
Traceback (most recent call last):
  File "C:/test3784.py", line 4, in <module>
    print(eggs)
NameError: name 'eggs' is not defined
```

Ошибка возникла потому, что переменная `eggs` существует только в локальной области видимости, созданной при вызове функции `spam()`. Как только осуществляется возврат из функции `spam()`, эта локальная область видимости уничтожается, и переменная `eggs` прекращает существование. Поэтому, когда ваша программа пытается выполнить вызов `print(eggs)`, Python выводит сообщение об ошибке, информируя вас о том, что переменная `eggs` не определена. Если вдуматься, то в этом есть смысл: когда программа выполняется в глобальной области видимости, локальные области видимости не существуют, а это означает, что нет никаких локальных переменных. Вот почему в глобальной области видимости могут использоваться только глобальные переменные.

В локальных областях видимости не могут использоваться переменные из других локальных областей видимости

Новая локальная область видимости создается всякий раз, когда вызывается функция, включая случаи, когда функция вызывается из другой функции. Рассмотрим следующую программу.

```
   def spam():
❶      eggs = 99
❷      bacon()
❸      print(eggs)

   def bacon():
       ham = 101
❹      eggs = 0
❺ spam()
```

Когда она запускается, вызывается функция `spam()` ❺ и создается локальная область видимости. Локальной переменной `eggs` ❶ присваивается значение `99`. Затем вызывается функция `bacon()` ❷ и создается вторая локальная область видимости. В одно и то же время могут существовать несколько локальных областей видимости. В этой новой локальной области

видимости значение локальной переменной ham устанавливается равным 101, а кроме того, создается и устанавливается равной 0 локальная переменная eggs ❹, которая отличается от одноименной переменной, созданной в локальной области видимости функции spam().

После возврата из функции bacon() ее локальная область видимости уничтожается. Выполнение программы продолжается в функции spam(), которая выводит значение переменной eggs ❸, а поскольку локальная область видимости для вызова функции spam() в это время все еще существует, то значение переменной eggs устанавливается равным 99. Именно это значение и выводит программа.

Таким образом, локальные переменные одной функции полностью отделены от локальных переменных другой функции.

Глобальные переменные могут читаться из локальной области видимости

Рассмотрим следующую программу.

```
def spam():
    print(eggs)
eggs = 42
spam()
print(eggs)
```

Поскольку имя eggs отсутствует в списке параметров функции spam() и в коде этой функции отсутствует присваивание значения переменной с таким именем, то Python, встретив эту переменную в теле функции spam(), полагает, что в данном случае имеется в виду ссылка на глобальную переменную eggs. Именно поэтому при выполнении данной программы на экран будет выведено значение 42.

Локальные и глобальные переменные с одинаковыми именами

Чтобы не усложнять себе жизнь, избегайте использования локальных переменных, имена которых совпадают с именами глобальных или других локальных переменных. Но с технической точки зрения это вполне допустимо в Python. Чтобы посмотреть, что при этом происходит, введите в файловом редакторе приведенный ниже код и сохраните его в файле *sameName.py*.

```
  def spam():
❶     eggs = 'spam local'
      print(eggs) # выводится строка 'spam local'
```

```
  def bacon():
❷     eggs = 'bacon local'
      print(eggs) # выводится строка 'bacon local'
      spam()
      print(eggs) # выводится строка 'bacon local'

❸     eggs = 'global'
      bacon()
      print(eggs) # выводится строка 'global'
```

Выполнив эту программу, вы должны получить следующий вывод.

```
bacon local
spam local
bacon local
global
```

В этой программе существуют фактически три разные переменные с одним и тем же именем `eggs`, что может сбивать с толку. Перечислим эти переменные.

❶ Переменная `eggs`, которая существует в локальной области видимости, когда вызывается функция `spam()`.

❷ Переменная `eggs`, которая существует в локальной области видимости, когда вызывается функция `bacon()`.

❸ Переменная `eggs`, которая существует в глобальной области видимости.

Тот факт, что все три независимые переменные имеют одно и то же имя, может сбивать с толку, если вы хотите отслеживать, какая из них используется в любой заданный момент времени. Именно поэтому старайтесь избегать использования одних и тех же имен переменных в разных областях видимости.

Инструкция `global`

Если возникает потребность изменить в коде функции глобальную переменную, используйте инструкцию `global`. Например, наличие инструкции `global eggs` в начале функции сообщает Python следующее: "В этой функции имя `eggs` ссылается на глобальную переменную, поэтому создавать локальную переменную с таким же именем не следует". Введите в файловом редакторе следующий код и сохраните его в файле *sameName2.py*.

```
  def spam():
❶     global eggs
❷     eggs = 'spam'

  eggs = 'global'
```

```
spam()
print(eggs)
```

При выполнении этой программы завершающий вызов функции `print()` должен вывести следующий текст:

```
spam
```

Переменная `eggs` объявлена как глобальная в начале функции `spam()` ❶, и поэтому, когда `eggs` присваивается значение `'spam'` ❷, эта операция выполняется по отношению к глобальной переменной `eggs`. Никакая локальная переменная `eggs` не создается.

Существуют четыре правила, позволяющие судить о том, в какой области видимости находится переменная – локальной или глобальной.

1. Если переменная используется в глобальной области видимости (т.е. вне какой-либо функции), то она всегда является глобальной переменной.
2. Если переменная была объявлена в функции с использованием инструкции `global`, то она является глобальной.
3. В противном случае, если переменная используется в операции присваивания в функции, то она является локальной.
4. Но если переменной нигде в функции не присваивается значение, то она является глобальной.

Рассмотрим пример программы, который поможет вам усвоить эти правила. Введите в файловом редакторе приведенный ниже код и сохраните его в файле *sameName3.py*.

```
def spam():
❶    global eggs
     eggs = 'spam' # это глобальная переменная

def bacon():
❷    eggs = 'bacon' # это локальная переменная

def ham():
❸    print(eggs) # это глобальная переменная

eggs = 42 # это глобальная переменная
spam()
print(eggs)
```

В функции `spam()` переменная `eggs` – глобальная, поскольку в начале функции для `eggs` используется инструкция `global` ❶. В функции `bacon()`

переменная eggs — локальная, поскольку в этой функции она вводится с помощью операции присваивания ❷. В функции ham() ❸ переменная eggs — глобальная, поскольку в этой функции для нее отсутствует инструкция global и она не вводится с помощью операции присваивания. Выполнив программу *sameName3.py*, вы должны получить следующий вывод:

```
spam
```

В функции любая переменная будет либо всегда глобальной, либо всегда локальной. Не может быть такого, чтобы в одной и той же функции переменная использовалась сначала как локальная, а впоследствии как глобальная.

Примечание

Если вы хотите иметь возможность изменять с помощью кода функции значение, хранящееся в глобальной переменной, то примените к этой переменной инструкцию `global`.

Если вы попытаетесь использовать в функции локальную переменную до того, как присвоите ей какое-либо значение, то Python выведет сообщение об ошибке. Чтобы в этом убедиться, введите в файловом редакторе следующий код и сохраните его в файле *sameName4.py*.

```
def spam():
    print(eggs) # ERROR!
❶   eggs = 'spam local'

❷ eggs = 'global'
  spam()
```

Выполнив эту программу, вы получите следующее сообщение об ошибке.

```
Traceback (most recent call last):
  File "C:/test3784.py", line 6, in <module>
    spam()
  File "C:/test3784.py", line 2, in spam
    print(eggs) # ERROR!
UnboundLocalError: local variable 'eggs' referenced before
assignment
```

Эта ошибка обусловлена тем, что Python, обнаружив присваивание переменной eggs значения в функции spam() ❶, предполагает, что это локальная переменная. Однако, поскольку функция print(eggs) выполняется до того, как переменной eggs присваивается какое-либо значение, в момент ее вызова такой переменной не существует. В этой ситуации Python не будет пытаться использовать одноименную глобальную переменную eggs ❷.

Функции как "черные ящики"

Зачастую все, что вам нужно знать о функции, — это какие входные данные (параметры) ей следует предоставить и каково ее выходное значение. Вам не всегда нужно обременять себя знанием того, как в действительности работает ее код. Если вы применяете к функциям такой высокоуровневый подход, то можно сказать, что вы рассматриваете любую функцию как "черный ящик".

Эта идея имеет фундаментальное значение для современного программирования. В последующих главах вы познакомитесь с некоторыми модулями, которые содержат функции, написанные другими людьми. Если вы любознательны, то можете заглянуть в их исходный код, однако для того, чтобы использовать эти функции, вам вовсе не обязательно знать их внутреннюю структуру. А поскольку написание функций, в которых глобальные переменные не используются, только приветствуется, вам, как правило, не приходится беспокоиться относительно того, что код этих функций будет нежелательным образом взаимодействовать с остальным кодом вашей программы.

Обработка исключений

На данном этапе возникновение ошибки, или *исключения*, в вашей программе на Python означает крах программы, т.е. ее аварийное завершение. Однако для реальных программ такое поведение недопустимо. Поэтому в них используются средства, позволяющие обнаруживать ошибки, обрабатывать их и после этого продолжать выполнение программы.

В качестве примера рассмотрим программу, в которой возникает ошибка "деление на 0". Введите в файловом редакторе следующий код и сохраните его в файле *zeroDivide.py*.

```
def spam(divideBy):
    return 42 / divideBy

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Мы определили функцию `spam()`, предоставили ей параметр, а затем попытались вывести на экран возвращаемое этой функцией значение при различных параметрах, чтобы понаблюдать, что при этом происходит. Запустив на выполнение этот код, вы получите следующий вывод.

```
21.0
3.5
```

```
Traceback (most recent call last):
  File "C:/zeroDivide.py", line 6, in <module>
    print(spam(0))
  File "C:/zeroDivide.py", line 2, in spam
    return 42 / divideBy
ZeroDivisionError: division by zero
```

Сообщение об ошибке `ZeroDivisionError` выводится всякий раз, когда предпринимается попытка разделить число на 0. По указанному в сообщении об ошибке номеру строки можно легко определить, что виновницей является инструкция `return` функции `spam()`.

Ошибки можно обрабатывать с помощью инструкций `try` и `except`. Код, относительно которого у вас есть подозрения, что он может привести к ошибке, помещается в блок `try`. В случае возникновения ошибки выполнение программы передается в начало блока `except`.

Вы можете поместить предыдущий код в блок инструкции `try` и организовать обработку соответствующей ошибки в блоке инструкции `except`.

```
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')

print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

Когда в коде, помещенном в блок `try`, возникает ошибка, выполнение программы немедленно переходит к коду в блоке `except`. После выполнения этого кода дальнейшее выполнение программы продолжается, как обычно. Вывод предыдущей программы должен быть таким.

```
21.0
3.5
Error: Invalid argument.
None
42.0
```

Обратите внимание на то, что ошибки, которые могут возникать при вызовах функций в блоке `try`, также будут перехватываться. Рассмотрим следующую программу, в которой вызовы функции `spam()` помещены в блок `try`.

```
def spam(divideBy):
    return 42 / divideBy

try:
    print(spam(2))
    print(spam(12))
    print(spam(0))
    print(spam(1))
except ZeroDivisionError:
    print('Error: Invalid argument.')
```

Вывод этой программы должен быть таким.

```
21.0
3.5
Error: Invalid argument.
```

Инструкция `print(spam(1))` не была выполнена по той причине, что после выполнения кода в блоке `except` возврата в блок `try` не происходит. Вместо этого далее, как обычно, выполняются инструкции, следующие за блоком `except`.

Короткая программа: угадай число

Те небольшие примеры, которые приводились до сих пор, были вполне пригодны для знакомства с базовыми понятиями, но теперь настал подходящий момент показать вам, как на основе того, что вы уже успели изучить, можно составить более интересную программу. В этом разделе я продемонстрирую вам простую программу, реализующую игру в угадывание чисел. Запустив ее, вы получите примерно следующий вывод.

```
Мною задумано число в интервале от 1 до 20. Попробуйте его
угадать.
Ваш вариант:
10
Предложенное число меньше задуманного.
Ваш вариант:
15
Предложенное число меньше задуманного.
Ваш вариант:
17
Предложенное число больше задуманного.
Ваш вариант:
16
Верно! Количество попыток: 4
```

Введите в файловом редакторе следующий код и сохраните его в файле *guessTheNumber.py*.

```
# Игра в угадывание чисел.
import random
secretNumber = random.randint(1, 20)
print('Мною задумано число в интервале от 1 до 20. Попробуйте его
↳ угадать.')

# Предоставить игроку 6 попыток для угадывания числа.
for guessesTaken in range(1, 7):
    print('Ваш вариант:')
    guess = int(input())

    if guess < secretNumber:
        print('Предложенное число меньше задуманного.')
    elif guess > secretNumber:
        print('Предложенное число больше задуманного.')
    else:
        break # Соответствует правильному ответу!

if guess == secretNumber:
    print('Верно! Количество попыток: ' + str(guessesTaken))
else:
    print('Нет. Было задумано число ' + str(secretNumber))
```

Проанализируем этот код с самого начала, строка за строкой.

```
# Игра в угадывание чисел.
import random
secretNumber = random.randint(1, 20)
```

Первая строка — это строка комментария, содержащая описание назначения программы. Далее программе необходимо импортировать модуль `random`, чтобы иметь возможность использовать функцию `random.randint()` для генерирования случайного числа, которое пользователь должен угадать. Возвращаемое функцией значение, которое будет представлять собой целое число в диапазоне от `1` до `20`, сохраняется в переменной `secretNumber`.

```
print('Мною задумано число в интервале от 1 до 20. Попробуйте его
↳ угадать.')
# Предоставить игроку 6 попыток для угадывания числа.
for guessesTaken in range(1, 7):
print('Ваш вариант:')
guess = int(input())
```

Программа сообщает игроку, что она задумала секретное число, и предоставляет возможность угадать его не более чем за шесть попыток. Код,

который предлагает ввести число и осуществляет проверку этого числа, помещен в цикл for, выполняющий не более шести итераций. Первое, что происходит в цикле, — это ввод игроком пробного числа. Поскольку функция input() возвращает строку, ее возвращаемое значение передается непосредственно функции int(), которая преобразует строку в целочисленное значение. Это значение сохраняется в переменной guess.

```
if guess < secretNumber:
    print('Предложенное число меньше задуманного.')
elif guess > secretNumber:
    print('Предложенное число больше задуманного.')
```

В этих нескольких строках кода пробное число сравнивается с секретным и проверяется, больше ли первое из них, чем второе, или меньше. В обоих случаях на экран выводится соответствующая подсказка.

```
else:
break # Соответствует правильному ответу!
```

Если оказывается, что пробное число одновременно не больше и не меньше секретного числа, то это означает, что оно и есть секретное число, и в таком случае инструкция break осуществляет выход из цикла for.

```
if guess == secretNumber:
    print('Верно! Количество попыток: ' + str(guessesTaken))
else:
    print('Нет. Было задумано число ' + str(secretNumber))
```

Располагающаяся вслед за циклом for инструкция if/else проверяет, является ли введенное пользователем число правильным, и выводит на экран соответствующее сообщение. В обоих случаях программа отображает переменную, содержащую целочисленное значение (guessesTaken и secretNumber). Поскольку эти значения необходимо конкатенировать со строками, они передаются функции str(), которая возвращает полученные числа в виде строк. Теперь эти строки можно конкатенировать с помощью операторов + и передать результирующую строку функции print().

Резюме

Функции — это основной способ разбиения кода на логические группы. Поскольку переменные в функциях существуют в собственных локальных областях видимости, код одной функции не может непосредственно воздействовать на значения переменных другой функции. Эти ограничения, налагаемые на возможность изменения значений переменных, могут быть полезными при отладке кода.

Функции являются великолепным средством организации кода. Можете представлять себе любую функцию как "черный ящик". Для вас имеет значение лишь то, какие входные данные необходимо предоставить функции в качестве аргументов и какое значение она возвращает, а также то, что код функции не может воздействовать на переменные в коде других функций.

В примерах, приведенных в предыдущих главах, единственная ошибка могла приводить к краху программы. В этой главе вы изучили инструкции `try` и `except`, обеспечивающие дальнейшее выполнение программы, даже если в ней возникла ошибка. Это позволяет вам обеспечить устойчивую работу своих программ при возникновении в них распространенных ошибок.

Контрольные вопросы

1. Что дает использование функций в программах?
2. Когда именно выполняется код функции: когда она определяется или когда вызывается?
3. С помощью какой инструкции создаются функции?
4. Чем отличается определение функции от ее вызова?
5. Сколько глобальных областей видимости может иметь программа на языке Python? Сколько локальных?
6. Что происходит с переменными, находящимися в локальной области видимости, при возврате из функции?
7. Что такое возвращаемое значение? Может ли возвращаемое значение быть частью выражения?
8. Каково возвращаемое значение функции, если в ней отсутствует инструкция `return`?
9. Как заставить переменную в функции ссылаться на глобальную переменную?
10. Что такое тип данных `None`?
11. Что делает инструкция `import areallyourpetsnamederic`?
12. Если бы у вас была функция `bacon()`, содержащаяся в модуле `spam`, то как бы вы ее вызвали после импортирования этого модуля?
13. Как можно предотвратить аварийное завершение программы при возникновении в ней ошибки?
14. Какой код помещается в блок `try`? Какой код помещается в блок `except`?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Последовательность Коллатца

Напишите функцию `collatz()`, принимающую один параметр: `number`. Если `number` — четное число, функция `collatz()` должна вывести на экран и возвратить значение `number // 2`. Если же `number`— нечетное число, то функция должна вывести на экран и возвратить значение `3 * number + 1`.

После этого напишите программу, которая предлагает пользователю ввести целое число, а затем последовательно вызывает функцию `collatz()` для этого числа и значений, возвращаемых очередным вызовом этой функции, пока на каком-то этапе не будет возвращено значение `1`. (Любопытно отметить, что, независимо от выбора начального числа, вы все равно рано или поздно получите `1`! Даже математики не могут объяснить, почему так происходит. Числовая последовательность, которую вы исследуете с помощью этой программы, называется *последовательностью Коллатца*[1] и иногда характеризуется как "простейшая из неразрешенных проблем математики".)

Не забывайте о том, что возвращаемое функцией `input()` значение нуждается в преобразовании в целое число с помощью функции `int()`, иначе это будет строковое значение.

Подсказка. Условие четности значения — `number % 2 == 0`, условие нечетности — `number % 2 == 1`.

Примерный вывод этой программы показан ниже.

```
Enter number:
3
10
5
16
8
4
2
1
```

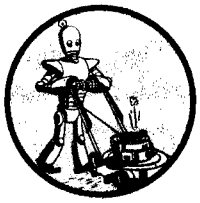
Проверка корректности ввода

Добавьте в предыдущий проект инструкции `try` и `except` с целью обнаружения ввода пользователем нецелочисленных значений. Обычно при передаче функции `int()` строки, представляющей нецелочисленное значение, как, например, при вызове `int('puppy')`, генерируется ошибка `ValueError`. Поместите в блок `except` код, который выводит для пользователя сообщение о том, что требуется ввод целого числа.

1 См. https://ru.wikipedia.org/wiki/Гипотеза_Коллатца. — *Примеч. ред.*

4

СПИСКИ



Еще одна тема, с которой вам обязательно следует познакомиться, прежде чем приступать к написанию собственных серьезных программ, — это списки и родственный им тип данных: кортежи. Списки и кортежи могут содержать более чем одно значение, что упрощает написание программ, обрабатывающих большие объемы данных. А поскольку списки сами могут содержать другие списки, вы сможете использовать их для создания иерархических структур данных.

В этой главе приведены основные сведения о списках. Вы также узнаете о методах, которые представляют собой функции, связанные с данными определенного типа. Затем мы вкратце рассмотрим такие типы данных, как кортежи и строки, и проанализируем, как они связаны со списками, аналогами которых они являются. В следующей главе речь пойдет о других типах данных — словарях.

Что такое список

Список — это значение, которое представляет собой коллекцию значений, образующих упорядоченную последовательность. Термин *списковое значение* относится к списку как единому целому (значение которого может сохраняться в переменной или передаваться функции подобно значению любого другого типа), а не к отдельным значениям, которые в нем содержатся. Например, список может иметь следующее значение:

```
['cat', 'bat', 'rat', 'elephant']
```

Подобно тому как строковые значения заключаются в кавычки, показывающие, где начинается и заканчивается строка, список заключается в

квадратные скобки, `[]`. Значения, образующие список, называются *элементами списка*. Элементы списка разделяются запятыми.

Введите в интерактивной оболочке следующие команды.

```
>>> [1, 2, 3]
[1, 2, 3]
>>> ['cat', 'bat', 'rat', 'elephant']
['cat', 'bat', 'rat', 'elephant']
>>> ['hello', 3.1415, True, None, 42]
['hello', 3.1415, True, None, 42]
❶ >>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam
['cat', 'bat', 'rat', 'elephant']
```

Здесь переменной `spam` ❶ присвоено лишь одно значение — списковое. Но это значение само содержит другие значения. Значению `[]` соответствует пустой список, в котором отсутствуют элементы, аналогично тому, как значению `''` соответствует пустая строка.

Доступ к отдельным элементам списка с помощью индексов

Предположим, у вас есть список `['cat', 'bat', 'rat', 'elephant']`, сохраненный в переменной `spam`. Python интерпретирует выражение `spam[0]` как `'cat'`, выражение `spam[1]` — как `'bat'` и т.д. Целое число, которое указывается в квадратных скобках после имени списка, называется *индексом*. Первому из значений, входящих в список, соответствует индекс `0`, второму — индекс `1`, третьему — индекс `2` и т.д. На рис. 4.1 представлен список, значение которого присвоено переменной `spam`, и показано, какие элементы соответствуют различным индексным выражениям.

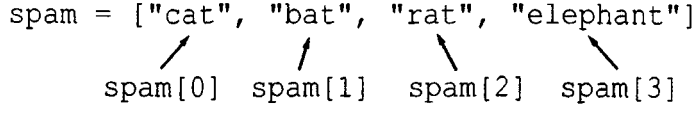


Рис. 4.1. Список, хранящийся в переменной `spam`, и индексные выражения, соответствующие его отдельным элементам

Введите в интерактивной оболочке следующие выражения. Начните с присваивания списка переменной `spam`.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0]
'cat'
>>> spam[1]
'bat'
>>> spam[2]
```

```
'rat'
>>> spam[3]
'elephant'
>>> ['cat', 'bat', 'rat', 'elephant'][3]
'elephant'
❶ >>> 'Hello ' + spam[0]
❷ 'Hello cat'
>>> 'The ' + spam[1] + ' ate the ' + spam[0] + '.'
'The bat ate the cat.'
```

Обратите внимание на то, что выражение `'Hello ' + spam[0]` ❶ вычисляется как `'Hello ' + 'cat'`, поскольку значением `spam[0]` является строка `'cat'`. Таким образом, конечным результатом вычисления данного выражения является строка `'Hello cat'` ❷.

Если вы попытаетесь использовать индекс, значение которого превышает количество элементов в списке, то Python выведет сообщение об ошибке `IndexError`.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[10000]
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    spam[10000]
IndexError: list index out of range
```

Индексы могут принимать только целочисленные значения (не вещественные). В следующем примере возникает ошибка `TypeError`.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1]
'bat'
>>> spam[1.0]
Traceback (most recent call last):
    File "<pyshell#13>", line 1, in <module>
    spam[1.0]
TypeError: list indices must be integers, not float
>>> spam[int(1.0)]
'bat'
```

Элементы списков сами могут быть списками. Доступ к значениям в таких списках списков осуществляется с помощью нескольких индексов.

```
>>> spam = [['cat', 'bat'], [10, 20, 30, 40, 50]]
>>> spam[0]
['cat', 'bat']
>>> spam[0][1]
'bat'
```

```
>>> spam[1][4]
50
```

Первый индекс указывает, какой элемент-список следует использовать, а второй — к значению какого элемента в этом списке осуществляется доступ. Например, для выражения `spam[0][1]` будет выведено значение `'bat'`, т.е. второе значение в первом списке. Если вы используете только один индекс, то программа выведет в качестве значения полный список, соответствующий данному индексу.

Отрицательные индексы

Несмотря на то что отсчет индексов начинается с нуля, в качестве индексов разрешается использовать отрицательные значения. Отрицательному значению `-1` соответствует последний элемент списка, значению `-2` — предпоследний и т.д. Введите в интерактивной оболочке следующие команды.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[-1]
'elephant'
>>> spam[-3]
'bat'
>>> 'The ' + spam[-1] + ' is afraid of the ' + spam[-3] + '.'
'The elephant is afraid of the bat.'
```

Получение части списка с помощью среза

В то время как с помощью индексов можно извлекать из списка одиночные элементы, *срезы* позволяют получать сразу несколько значений в виде нового списка. Срез списка обозначается, как и при индексном доступе, квадратными скобками, однако в скобках указываются два индекса, разделенные двоеточием. Обратите внимание на различие между индексами и срезами:

- `spam[2]` — список с индексом (одно целое число);
- `spam[1:4]` — список со срезом (два целых числа).

Первое целое число в срезе — это индекс, с которого начинается срез. Второе целое число — это индекс, который обозначает конец среза, но сам в срез не включается. Значением среза является новый список.

Введите в интерактивной оболочке следующие команды.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[0:4]
['cat', 'bat', 'rat', 'elephant']
```

```
>>> spam[1:3]
['bat', 'rat']
>>> spam[0:-1]
['cat', 'bat', 'rat']
```

Допускается сокращенная запись среза с пропуском одного или двух индексов по обе стороны двоеточия. Отсутствующий первый индекс равносилен использованию значения 0, т.е. соответствует началу списка. Отсутствующий второй индекс означает расширение среза до конца списка. Введите в интерактивной оболочке следующие команды.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[:2]
['cat', 'bat']
>>> spam[1:]
['bat', 'rat', 'elephant']
>>> spam[:]
['cat', 'bat', 'rat', 'elephant']
```

Получение длины списка с помощью функции `len()`

Функция `len()` возвращает количество значений, содержащихся в переданном ей списке, а в случае передачи ей строкового значения – количество символов в строке. Введите в интерактивной оболочке следующие команды.

```
>>> spam = ['cat', 'dog', 'moose']
>>> len(spam)
3
```

Изменение значений в списках с помощью индексов

Обычно слева от оператора присваивания располагается имя переменной, например `spam = 42`. Однако для изменения значения в списке, характеризующегося определенным индексом, можно использовать индексацию. Например, инструкция `spam[1] = 'aardvark'` означает следующее: "Назначить элементу с индексом `1` в списке `spam` строковое значение `'aardvark'`". Введите в интерактивной оболочке следующие команды.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam[1] = 'aardvark'
>>> spam
['cat', 'aardvark', 'rat', 'elephant']
>>> spam[2] = spam[1]
>>> spam
```

```
['cat', 'aardvark', 'aardvark', 'elephant']
>>> spam[-1] = 12345
>>> spam
['cat', 'aardvark', 'aardvark', 12345]
```

Конкатенация и репликация списков

С помощью оператора + можно объединить два списка в новый список аналогично тому, как с помощью этого же оператора можно объединить два строковых значения в одну строку. Кроме того, умножая список с помощью оператора * на целое число, можно повторить список заданное количество раз. Введите в интерактивной оболочке следующие команды.

```
>>> [1, 2, 3] + ['A', 'B', 'C']
[1, 2, 3, 'A', 'B', 'C']
>>> ['X', 'Y', 'Z'] * 3
['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
>>> spam = [1, 2, 3]
>>> spam = spam + ['A', 'B', 'C']
>>> spam
[1, 2, 3, 'A', 'B', 'C']
```

Удаление значений из списка с помощью инструкции `del`

Инструкция `del` удаляет из списка значение с заданным индексом. Все значения, находящиеся после удаленного, сдвигаются к началу списка на одну позицию. Например, введите в интерактивной оболочке следующие команды.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat', 'elephant']
>>> del spam[2]
>>> spam
['cat', 'bat']
```

Инструкция `del` также может удалять простые переменные. Если вы попытаетесь использовать удаленную переменную, то получите сообщение об ошибке `NameError`, поскольку такой переменной больше не существует. На практике вам почти никогда не придется удалять простые переменные. Основное назначение инструкции `del` — удаление значений из списков.

Работа со списками

У тех, кто впервые приступает к написанию программ, возникает соблазн создавать множество отдельных переменных для группы родственных значений. Например, если бы я захотел сохранить имена своих котов и кошек, то мог бы это сделать с помощью примерно такого кода.

```
catName1 = 'Zophie'
catName2 = 'Pooka'
catName3 = 'Simon'
catName4 = 'Lady Macbeth'
catName5 = 'Fat-tail'
catName6 = 'Miss Cleo'
```

(Клянусь, на самом деле у меня не так уж и много кошек в доме.) Однако этот способ далеко не самый удачный. К примеру, если количество кошек изменится, программа не сможет сохранить имен больше, чем имеется переменных. К тому же в программах этого типа часто наблюдается повторение одних и тех же или почти одинаковых фрагментов кода. Посмотрите, как часто дублируется код в следующей программе, которую вы должны ввести в файловом редакторе и сохранить в файле *allMyCats1.py*.

```
print('Enter the name of cat 1:')
catName1 = input()
print('Enter the name of cat 2:')
catName2 = input()
print('Enter the name of cat 3:')
catName3 = input()
print('Enter the name of cat 4:')
catName4 = input()
print('Enter the name of cat 5:')
catName5 = input()
print('Enter the name of cat 6:')
catName6 = input()
print('The cat names are:')
print(catName1 + ' ' + catName2 + ' ' + catName3 + ' ' +
      catName4 + ' ' + catName5 + ' ' + catName6)
```

Вместо множества однотипных переменных лучше использовать одну переменную, содержащую список. Ниже приведен пример улучшенной версии программы *allMyCats1.py*. В этой новой версии используется всего один список, в котором может храниться любое количество имен, введенных пользователем. Откройте в файловом редакторе новое окно, наберите в нем приведенный ниже текст и сохраните его в файле *allMyCats2.py*.

```
catNames = []
while True:
    print('Enter the name of cat ' + str(len(catNames) + 1) +
          ' (Or enter nothing to stop.):')
    name = input()
    if name == '':
        break
    catNames = catNames + [name] # list concatenation
print('The cat names are:')
for name in catNames:
    print(' ' + name)
```

Запустив эту программу, вы получите следующий вывод.

```
Enter the name of cat 1 (Or enter nothing to stop.):
Zophie
Enter the name of cat 2 (Or enter nothing to stop.):
Pooka
Enter the name of cat 3 (Or enter nothing to stop.):
Simon
Enter the name of cat 4 (Or enter nothing to stop.):
Lady Macbeth
Enter the name of cat 5 (Or enter nothing to stop.):
Fat-tail
Enter the name of cat 6 (Or enter nothing to stop.):
Miss Cleo
Enter the name of cat 7 (Or enter nothing to stop.):
```

Вот все кошачьи имена.

```
Zophie
Pooka
Simon
Lady Macbeth
Fat-tail
Miss Cleo
```

Список дает то преимущество, что теперь ваши данные сосредоточены в одной структуре, а сама программа стала намного более гибкой по сравнению с тем ее вариантом, в котором использовалось множество однотипных переменных.

Использование циклов `for` со списками

Из главы 2 вы узнали о том, как использовать циклы `for` для выполнения одного и того же блока кода определенное количество раз. С технической точки зрения цикл `for` выполняет блок кода по одному разу для каждого значения из списка или подобной ему структуры данных. Например, если выполнить код

```
for i in range(4):
    print(i)
```

то его вывод будет выглядеть так:

```
0
1
2
3
```

Это происходит потому, что возвращаемое вызовом функции `range(4)` значение Python трактует как список `[0, 1, 2, 3]`. Поэтому предыдущий вывод можно воспроизвести с помощью следующей программы.

```
for i in [0, 1, 2, 3]:
    print(i)
```

Данный цикл многократно выполняет блок кода, используя на каждой итерации переменную `i`, которая принимает последовательный ряд значений из списка `[0, 1, 2, 3]`.

Примечание

Используя в этой книге выражение "тип данных наподобие списка", я подразумеваю данные, для которых существует термин "последовательность". Однако знать техническое определение этого термина вам вовсе необязательно.

Один из часто применяемых в Python приемов — использование выражения `range(len(someList))` с циклом `for` для итерирования по индексам в списке. Например, введите в интерактивной оболочке следующие команды.

```
>>> supplies = ['pens', 'staplers', 'flame-throwers', 'binders']
>>> for i in range(len(supplies)):
    print('Index ' + str(i) + ' in supplies is: ' + supplies[i])

Index 0 in supplies is: pens
Index 1 in supplies is: staplers
Index 2 in supplies is: flame-throwers
Index 3 in supplies is: binders
```

Использовать выражение `range(len(supplies))` в представленном выше цикле `for` очень удобно, поскольку код в цикле может иметь доступ к индексу (в виде переменной `i`) и к значению по этому индексу (предоставляемому выражением `supplies[i]`). Что самое главное, выражение

`range(len(supplies))` обеспечивает итерирование по всем индексам списка `supplies`, независимо от количества содержащихся в нем значений.

Операторы `in` и `not in`

Определить, находится ли какое-либо значение в списке, можно с помощью операторов `in` и `not in`. Как и другие операторы, `in` и `not in` используются в выражениях, соединяя два значения: то, поиск которого выполняется в списке, и список, в котором это значение может находиться. Результатом вычисления этих выражений является булево значение. Введите в интерактивной оболочке следующие команды.

```
>>> 'howdy' in ['hello', 'hi', 'howdy', 'heyas']
True
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> 'cat' in spam
False
>>> 'howdy' not in spam
False
>>> 'cat' not in spam
True
```

В качестве примера рассмотрим программу, которая предлагает пользователю ввести имя своего домашнего питомца и проверяет, содержится ли оно в списке `pets`. Откройте в файловом редакторе новое окно, введите в него следующий код и сохраните его в файле *myPets.py*.

```
myPets = ['Zophie', 'Pooka', 'Fat-tail']
print('Enter a pet name:')
name = input()
if name not in myPets:
    print('I do not have a pet named ' + name)
else:
    print(name + ' is my pet.')
```

Вывод этой программы может выглядеть примерно так.

```
Enter a pet name:
Footfoot
I do not have a pet named Footfoot
```

Трюк с групповым присваиванием

Используя трюк с *групповым присваиванием*, можно быстро присвоить значения ряду переменных в одной строке кода. Итак, вместо выполнения последовательности инструкций

```
>>> cat = ['fat', 'black', 'loud']
>>> size = cat[0]
>>> color = cat[1]
>>> disposition = cat[2]
```

можно ограничиться следующим кодом:

```
>>> cat = ['fat', 'black', 'loud']
>>> size, color, disposition = cat
```

Число переменных должно совпадать с длиной списка, иначе Python выведет сообщение об ошибке.

```
>>> cat = ['fat', 'black', 'loud']
>>> size, color, disposition, name = cat
Traceback (most recent call last):
  File "<pyshell#84>", line 1, in <module>
    size, color, disposition, name = cat
ValueError: need more than 3 values to unpack
```

Комбинированные операторы присваивания

В ходе присваивания значения переменной справа от оператора присваивания часто используется эта же переменная. Например, если переменной `spam` необходимо присвоить значение `42`, а затем увеличить его на `1`, то это можно сделать с помощью следующего кода.

```
>>> spam = 42
>>> spam = spam + 1
>>> spam
43
```

Однако, используя комбинированный оператор присваивания `+=`, можно немного сократить этот код:

```
>>> spam = 42
>>> spam += 1
>>> spam
43
```

Комбинированные операторы присваивания существуют для операторов `+`, `-`, `*`, `/` и `%` (табл. 4.1).

Таблица 4.1. Комбинированные операции присваивания

Присваивание	Комбинированное присваивание
spam = spam + 1	spam += 1
spam = spam - 1	spam -= 1
spam = spam * 1	spam *= 1
spam = spam / 1	spam /= 1
spam = spam % 1	spam %= 1

Кроме того, оператор += может применяться для конкатенации, а оператор *= — для репликации строк и списков. Введите в интерактивной оболочке следующие команды.

```
>>> spam = 'Hello'
>>> spam += ' world!'
>>> spam
'Hello world!'
>>> bacon = ['Zophie']
>>> bacon *= 3
>>> bacon
['Zophie', 'Zophie', 'Zophie']
```

Методы

Метод — это то же самое, что и функция, но он вызывается для значения. Например, если список хранится в переменной spam, то вы можете вызвать для нее метод index() списка (о чем далее будет рассказано более подробно): spam.index('hello'). Метод указывается после значения и отделяется от него точкой.

Каждый тип данных имеет собственный набор методов. Так, для списков предусмотрен ряд полезных методов, позволяющих выполнять поиск, добавление, удаление элементов и другие манипуляции со значениями, образующими список.

Поиск значения в списке с помощью метода index()

Списки имеют метод index(), который принимает значение и возвращает его индекс, если оно содержится в списке. Если указанное значение отсутствует в списке, то Python генерирует ошибку ValueError. Введите в интерактивной оболочке следующие команды.

```
>>> spam = ['hello', 'hi', 'howdy', 'heyas']
>>> spam.index('hello')
0
```

```
>>> spam.index('heyas')
3
>>> spam.index('howdy howdy howdy')
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    spam.index('howdy howdy howdy')
ValueError: 'howdy howdy howdy' is not in list
```

В случае наличия в списке дубликатов данного значения возвращается индекс первого из элементов, в котором встречается это значение. Введите в интерактивной оболочке следующие команды (обратите внимание на то, что метод `index()` возвращает 1, а не 3).

```
>>> spam = ['Zophie', 'Pooka', 'Fat-tail', 'Pooka']
>>> spam.index('Pooka')
1
```

Добавление значений в список с помощью методов `append()` и `insert()`

Для добавления новых значений в список используются методы `append()` и `insert()`. Введите в интерактивной оболочке следующий код, чтобы вызвать метод `append()` для списка, хранящегося в переменной `spam`.

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.append('moose')
>>> spam
['cat', 'dog', 'bat', 'moose']
```

Здесь вызов метода `append()` добавляет аргумент в конец списка. Метод `insert()` позволяет добавить в список элемент с определенным индексом. Первый аргумент метода `insert()` — это индекс для нового значения, а второй — вставляемое значение. Введите в интерактивной оболочке следующие команды.

```
>>> spam = ['cat', 'dog', 'bat']
>>> spam.insert(1, 'chicken')
>>> spam
['cat', 'chicken', 'dog', 'bat']
```

Обратите внимание на то, в каком виде записан код: `spam.append('moose')` и `spam.insert(1, 'chicken')`, а не `spam = spam.append('moose')` и `spam = spam.insert(1, 'chicken')`. Ни метод `append()`, ни метод `insert()` не предоставляют новое значение `spam` в качестве возвращаемого значения.

(В действительности оба метода возвращают значение None, однако вряд ли вы захотите присваивать его в качестве нового значения переменной.) Вместо этого список изменяется, как говорят, *на месте*. Более подробно изменение списка на месте обсуждается в разделе "Изменяемые и неизменяемые типы данных".

Методы принадлежат к одному определенному типу данных. Методы append() и insert() являются методами списков и могут вызываться только для списков. Вызывать их для других типов значений, таких, например, как строки или целые числа, нельзя. Введите в интерактивной оболочке следующие команды и обратите внимание на появление сообщения об ошибке AttributeError.

```
>>> eggs = 'hello'
>>> eggs.append('world')
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    eggs.append('world')
AttributeError: 'str' object has no attribute 'append'
>>> bacon = 42
>>> bacon.insert(1, 'world')
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    bacon.insert(1, 'world')
AttributeError: 'int' object has no attribute 'insert'
```

Удаление значений из списка с помощью метода remove()

Методу remove() передается значение, подлежащее удалению из списка, для которого он вызывается. Введите в интерактивной оболочке следующие команды и обратите внимание на появление сообщения об ошибке.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('bat')
>>> spam
['cat', 'rat', 'elephant']
```

При попытке удалить значение, отсутствующее в списке, будет сгенерирована ошибка ValueError. Чтобы в этом убедиться, введите в интерактивной оболочке следующие команды.

```
>>> spam = ['cat', 'bat', 'rat', 'elephant']
>>> spam.remove('chicken')
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    spam.remove('chicken')
ValueError: list.remove(x): x not in list
```

Если в списке имеется несколько одинаковых значений, будет удалено лишь то из них, которое встретится первым. Введите в интерактивной оболочке следующие команды.

```
>>> spam = ['cat', 'bat', 'rat', 'cat', 'hat', 'cat']
>>> spam.remove('cat')
>>> spam
['bat', 'rat', 'cat', 'hat', 'cat']
```

Инструкция `del` используется в тех случаях, когда вам известен индекс значения, которое вы хотите удалить, а метод `remove()` — когда известно значение, подлежащее удалению из списка.

Сортировка значений в списке с помощью метода `sort()`

Для сортировки списков, содержащих числа или строки, используется метод `sort()`. Введите в интерактивной оболочке следующий код.

```
>>> spam = [2, 5, 3.14, 1, -7]
>>> spam.sort()
>>> spam
[-7, 1, 2, 3.14, 5]
>>> spam = ['ants', 'cats', 'dogs', 'badgers', 'elephants']
>>> spam.sort()
>>> spam
['ants', 'badgers', 'cats', 'dogs', 'elephants']
```

Чтобы задать сортировку в обратном порядке, следует указать именованный аргумент `reverse` со значением `True` при вызове метода `sort()`. Введите в интерактивной оболочке следующий код.

```
>>> spam.sort(reverse=True)
>>> spam
['elephants', 'dogs', 'cats', 'badgers', 'ants']
```

Относительно метода `sort()` следует сделать три замечания. Во-первых, метод `sort()` выполняет сортировку списка на месте; не пытайтесь использовать его возвращаемое значение с помощью кода наподобие `spam = spam.sort()`.

Во-вторых, невозможно отсортировать список, который содержит одновременно и числа, и строки, поскольку Python не знает, как сравнивать разные типы значений между собой. Введите в интерактивной оболочке следующий код и обратите внимание на появление сообщения об ошибке `TypeError`.

```
>>> spam = [1, 3, 2, 4, 'Alice', 'Bob']
>>> spam.sort()
Traceback (most recent call last):
  File "<pyshell#70>", line 1, in <module>
    spam.sort()
TypeError: unorderable types: str() < int()
```

В-третьих, метод `sort()` сортирует строки не в алфавитном порядке, а в соответствии с таблицей ASCII. Это означает, что буквы в верхнем регистре предшествуют буквам в нижнем регистре. Поэтому, например, буква `a` будет располагаться в процессе сортировки после буквы `Z`. Введите в интерактивной оболочке следующий код.

```
>>> spam = ['Alice', 'ants', 'Bob', 'badgers', 'Carol', 'cats']
>>> spam.sort()
>>> spam
['Alice', 'Bob', 'Carol', 'ants', 'badgers', 'cats']
```

Если вам необходимо отсортировать строку в обычном алфавитном порядке, то передайте методу `sort()` именованный аргумент `key` со значением `str.lower`.

```
>>> spam = ['a', 'z', 'A', 'Z']
>>> spam.sort(key=str.lower)
>>> spam
['a', 'A', 'z', 'Z']
```

Это приведет к тому, что метод `sort()` будет обрабатывать все элементы списка так, как если бы они были записаны только с использованием нижнего регистра, не изменяя при этом самих элементов.

Пример программы: Magic 8 Ball со списком

Используя списки, можно написать гораздо более элегантную версию программы Magic 8 Ball. Вместо того чтобы использовать несколько строк, содержащих почти идентичные инструкции `elif`, можно создать единственный список, с которым будет работать код. Откройте в файловом редакторе новое окно, введите в нем следующий код и сохраните его в файле *magic8Ball2.py*.

```
import random

messages = ['It is certain',
    'It is decidedly so',
    'Yes definitely',
```

```
    'Reply hazy try again',
    'Ask again later',
    'Concentrate and ask again',
    'My reply is no',
    'Outlook not so good',
    'Very doubtful']

print(messages[random.randint(0, len(messages) - 1)])
```

Запустив этот код, вы увидите, что он работает точно так же, как и предыдущая версия программы *magic8Ball.py*.

Обратите внимание на выражение, которое используется в качестве индекса: `random.randint(0, len(messages) - 1)`. Оно позволяет получать случайные числа в нужном диапазоне, независимо от длины списка `messages`.

Исключения из правил использования отступов в коде Python

В большинстве случаев величина отступа строки кода сообщает интерпретатору Python, к какому блоку она относится. Однако из этого правила есть исключения. Например, список может располагаться в нескольких строках в файле исходного кода. В подобных случаях величина отступа не имеет значения; Python знает, что до тех пор, пока не встретится закрывающая квадратная скобка, список еще не закончился. Например, у вас может быть код следующего вида.

```
spam = ['apples',
    'oranges',
                'bananas',
'cats']
print(spam)
```

Разумеется, большинство людей используют эту особенность поведения Python для того, чтобы придать аккуратный вид спискам, подобным списку сообщений в программе Magic 8 Ball, и облегчить чтение кода.

Также допускается разбиение инструкции на несколько строк с помощью символа \, который ставится в конце строки и указывает на то, что далее следует ее продолжение. Считайте, что символ \ эквивалентен такому утверждению: "Эта инструкция продолжается в следующей строке". Величина отступа строки, следующей за символом \, не играет никакой роли. Вот пример корректного кода Python.

```
print('Four score and seven ' + \
      'years ago...')
```

Эти приемы пригодятся вам для реорганизации длинных строк с целью повышения удобочитаемости кода.

В данном случае имеется в виду, что вы получаете случайное число в диапазоне от 0 до значения len(messages) - 1. Преимуществом такого подхода является то, что вы можете легко добавлять и удалять строки из списка, не изменяя другие строки кода. Если впоследствии вы захотите обновить код, то вам придется изменить меньшее количество строк кода, а это означает, что уменьшится и вероятность внесения ошибок при вводе.

Типы данных, подобные спискам: строки и кортежи

Списки — не единственный тип данных, который представляет упорядоченные последовательности значений. Например, строки фактически аналогичны спискам, если рассматривать строку как "список" одиночных текстовых символов. Многое из того, что можно делать со списками, можно делать и со строками: индексирование, получение срезов, а также использование в циклах for с функцией len() и с операторами in и not in. Чтобы убедиться в этом, введите в интерактивной оболочке следующие команды.

```
>>> name = 'Zophie'
>>> name[0]
'Z'
>>> name[-2]
'i'
>>> name[0:4]
'Zoph'
>>> 'Zo' in name
True
>>> 'z' in name
False
>>> 'p' not in name
False
>>> for i in name:
        print('* * * ' + i + ' * * *')

* * * Z * * *
* * * o * * *
* * * p * * *
* * * h * * *
* * * i * * *
* * * e * * *
```

Изменяемые и неизменяемые типы данных

Вместе с тем между списками и строками существует одно важное различие. Список — это *изменяемый* тип данных: значения, хранящиеся в списках, можно добавлять, удалять и изменять. Строки же относятся к *неизменяемому* типу данных: строку нельзя изменить. Попытка заменить одиночный

символ в строке приведет к возникновению ошибки TypeError, в чем вы сможете убедиться, введя в интерактивной оболочке следующий код.

```
>>> name = 'Zophie a cat'
>>> name[7] = 'the'
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    name[7] = 'the'
TypeError: 'str' object does not support item assignment
```

Правильный способ "изменения" строки заключается в использовании операций среза и конкатенации для создания *новой* строки путем копирования частей исходной строки. Введите в интерактивной оболочке следующие команды.

```
>>> name = 'Zophie a cat'
>>> newName = name[0:7] + 'the' + name[8:12]
>>> name
'Zophie a cat'
>>> newName
'Zophie the cat'
```

Здесь для того, чтобы сослаться на символы, которые мы не намерены заменять, использованы срезы [0:7] и [8:12]. Заметьте, что исходная строка 'Zophie a cat' не подверглась изменению ввиду неизменяемости строкового типа данных.

Несмотря на то что списки — изменяемый тип данных, в приведенном ниже коде вторая строка не изменяет список eggs.

```
>>> eggs = [1, 2, 3]
>>> eggs = [4, 5, 6]
>>> eggs
[4, 5, 6]
```

Списковое значение, которое хранилось в eggs, здесь не изменилось. В действительности было просто создано совершенно новое значение ([4, 5, 6]), которое заменило прежнее ([1, 2, 3]). Эту ситуацию поясняет рис. 4.2.

Если бы мы хотели действительно изменить первоначальный список, хранящийся в переменной eggs, чтобы он содержал значения [4, 5, 6], то должны были бы сделать примерно следующее.

```
>>> eggs = [1, 2, 3]
>>> del eggs[2]
>>> del eggs[1]
```

```
>>> del eggs[0]
>>> eggs.append(4)
>>> eggs.append(5)
>>> eggs.append(6)
>>> eggs
[4, 5, 6]
```

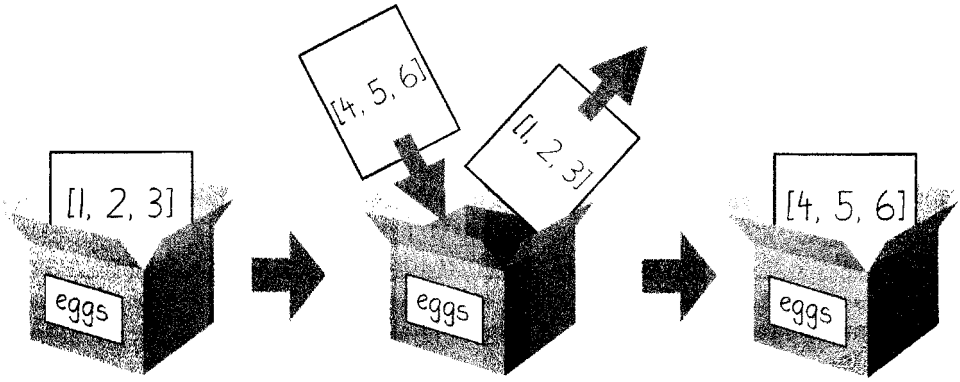


Рис. 4.2. При выполнении инструкции `eggs = [4, 5, 6]` *содержимое* `eggs` *заменяется новым списковым значением*

В первом примере списковое значение, которое в конечном счете содержится в переменной `eggs`, является тем же значением, которое эта переменная имела изначально. Суть в том, что данный список лишь изменился, а не был перезаписан. На рис. 4.3 в иллюстративной форме представлены семь изменений, которые выполняются первыми семью строками кода в последнем примере.

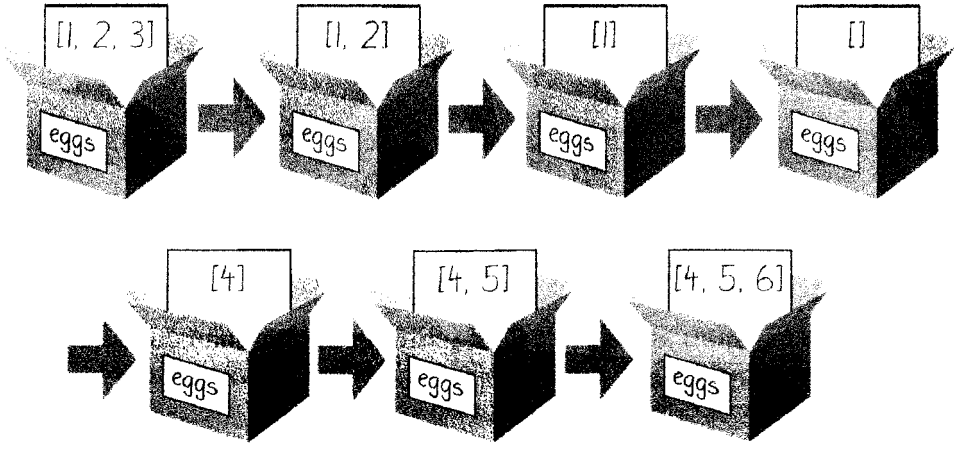


Рис. 4.3. Инструкция `del` *и метод* `append()` *изменяют элементы списка на месте*

В случае изменяемых типов данных изменение значений осуществляется на месте (как это делают инструкция `del` и метод `append()` в предыдущем примере), поскольку значение переменной не заменяется новым списковым значением.

Разграничение изменяемых и неизменяемых типов данных может показаться не имеющим особого смысла, однако, как будет показано в разделе "Передача ссылок", различия между вызовами функций с изменяемыми и неизменяемыми аргументами весьма существенны. А сейчас нам предстоит рассмотреть кортежи — тип данных, который является неизменяемой формой списков.

Кортежи

Кортежи почти идентичны спискам и отличаются от них лишь в двух отношениях. Во-первых, кортежи заключаются в круглые скобки, `(` и `)`, а не в квадратные, `[` и `]`. Введите в интерактивной оболочке следующие команды.

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[0]
'hello'
>>> eggs[1:3]
(42, 0.5)
>>> len(eggs)
3
```

Однако главным отличием кортежей от списков является то, что кортежи, подобно строкам, относятся к неизменяемому типу данных. Их значения не могут изменяться, добавляться и удаляться. Введите в интерактивной оболочке следующие команды.

```
>>> eggs = ('hello', 42, 0.5)
>>> eggs[1] = 99
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    eggs[1] = 99
TypeError: 'tuple' object does not support item assignment
```

Если кортеж состоит всего лишь из одного значения, после него надо ставить запятую внутри скобок. В противном случае Python будет полагать, что вы ввели обычное значение и просто заключили его в скобки. Запятая укажет Python, что данное значение является кортежем. (В отличие от некоторых других языков программирования, в Python использование завершающей запятой в кортежах и списках вполне допустимо.) Введите в интерактивной оболочке следующие команды, чтобы увидеть, к чему приводит отсутствие такой запятой.

```
>>> type(('hello',))
<class 'tuple'>
>>> type(('hello'))
<class 'str'>
```

Используя кортежи, вы тем самым сообщаете тому, кто читает код вашей программы, что не намереваетесь изменять значения, входящие в данную последовательность. Если вам требуется упорядоченная последовательность значений, которые не будут изменяться, то используйте кортеж. Еще одним преимуществом кортежей по сравнению со списками является то, что, благодаря неизменяемости их содержимого, Python может реализовать некоторые схемы оптимизации, ускоряющие работу кода.

Преобразование типов с помощью функций `list()` и `tuple()`

Подобно тому как вызов функции `str(42)` возвращает значение `'42'`, являющееся строковым представлением целого числа `42`, функции `list()` и `tuple()` возвращают версии переданных им значений соответственно в виде списка и кортежа. Введите в интерактивной оболочке следующие команды и обратите внимание на различие типов передаваемых и возвращаемых значений функций.

```
>>> tuple(['cat', 'dog', 5])
('cat', 'dog', 5)
>>> list(('cat', 'dog', 5))
['cat', 'dog', 5]
>>> list('hello')
['h', 'e', 'l', 'l', 'o']
```

Преобразование кортежа в список удобно использовать в тех случаях, когда необходимо получить изменяемую версию значений кортежа.

Ссылки

Как вы уже знаете, переменные хранят строковые и целочисленные значения. Введите в интерактивной оболочке следующие команды.

```
>>> spam = 42
>>> cheese = spam
>>> spam = 100
>>> spam
100
>>> cheese
42
```

Здесь переменной spam сначала присваивается значение 42, а затем это значение копируется и присваивается переменной cheese. Когда впоследствии переменной spam присваивается значение 100, это никак не отражается на значении, хранящемся в переменной cheese. Такое поведение объясняется тем, что spam и cheese — это различные переменные, хранящие различные значения.

Однако списки работают не так. Присваивая переменной список, на самом деле вы присваиваете ей *ссылку* на этот список. Ссылка — это значение, которое указывает на некоторую порцию данных, а ссылка на список — это значение, которое указывает на список. Приведенный ниже код позволит вам лучше понять суть этого различия. Введите в интерактивной оболочке следующие команды.

```
❶ >>> spam = [0, 1, 2, 3, 4, 5]
❷ >>> cheese = spam
❸ >>> cheese[1] = 'Hello!'
  >>> spam
  [0, 'Hello!', 2, 3, 4, 5]
  >>> cheese
  [0, 'Hello!', 2, 3, 4, 5]
```

Возможно, полученные результаты вас несколько озадачивают. Несмотря на то что изменялся лишь список cheese, изменение затронуло также список spam.

Создавая список ❶, вы присваиваете ссылку на него переменной spam. В следующей строке ❷ в переменную cheese копируется не сам список, а только ссылка на него, хранящаяся в переменной spam. Это означает, что теперь оба значения, хранящиеся в переменных spam и cheese, ссылаются на один и тот же список. Собственно список существует лишь в единственном экземпляре, поскольку он никуда не копировался. Поэтому, изменяя первый элемент списка с помощью переменной cheese ❸, вы изменяете тот же список, на который ссылается переменная spam.

Вспомните о том, что переменные можно уподобить ящикам, в которых хранятся значения. Представленная на предыдущих рисунках в этой главе аналогия с ящиками для списков не совсем точна, поскольку на самом деле в переменных хранятся не сами списки, а лишь ссылки на них. (Эти ссылки снабжаются числовыми идентификаторами (ID), которые используются внутренними механизмами Python, но для вас это несущественно.) На рис. 4.4, также с использованием ящика в качестве метафоры для переменной, показано, что происходит, когда переменной spam присваивается список.

Далее на рис. 4.5 представлен процесс копирования ссылки из переменной spam в переменную cheese. При этом создается и сохраняется в пере-

менной cheese новая ссылка, а не новый список. Заметьте, что обе ссылки ссылаются на один и тот же список.

Если вы измените список, на который ссылается переменная cheese, то список, на который ссылается переменная spam, также изменится, поскольку обе эти переменные ссылаются на один и тот же список. Эта ситуация проиллюстрирована на рис. 4.6.

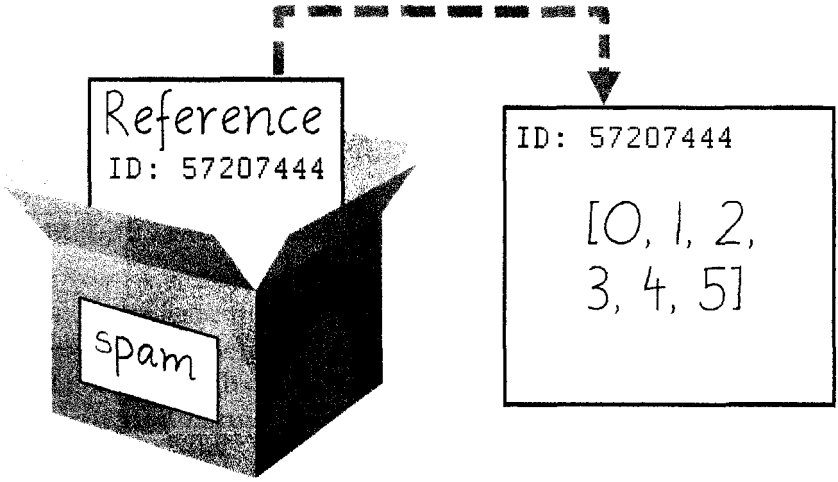


Рис. 4.4. Инструкция spam = [0, 1, 2, 3, 4, 5] означает сохранение ссылки на список, а не самого списка

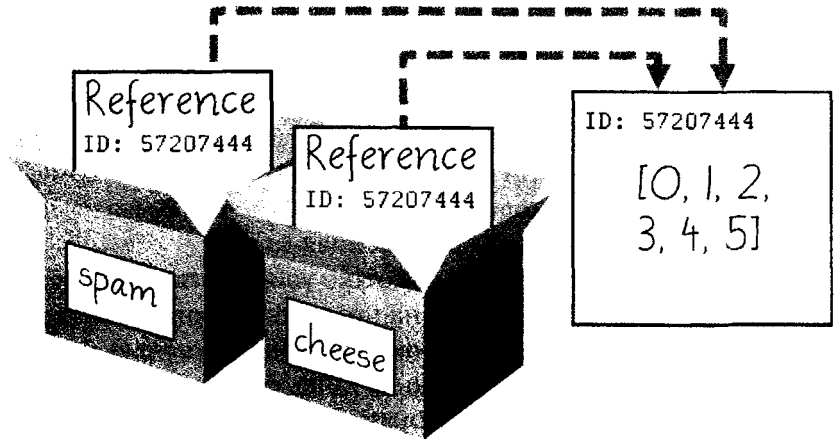


Рис. 4.5. Инструкция spam = cheese означает копирование ссылки на список, а не самого списка

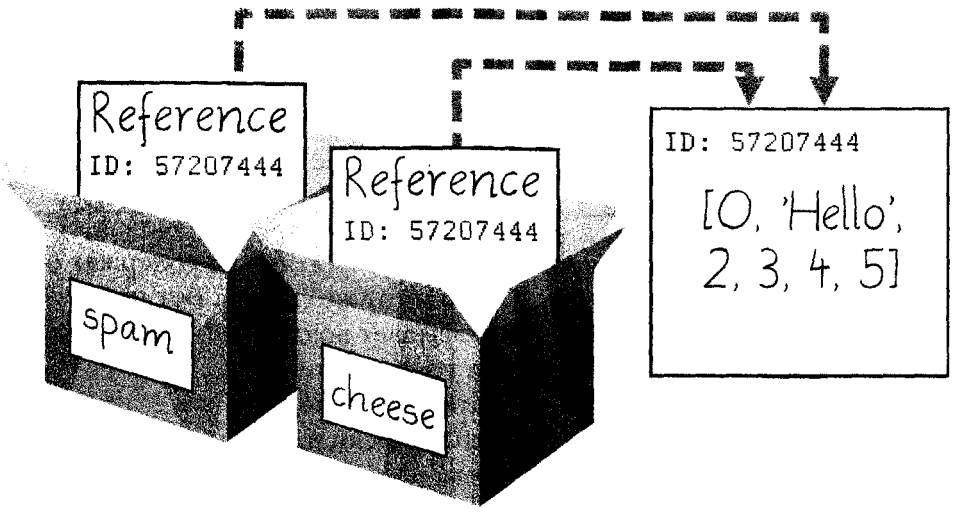


Рис. 4.6. Инструкция `cheese[1] = 'Hello!'` *изменяет список, на который ссылаются обе переменные*

Переменные содержат ссылки на списки, а не сами списки. Однако в случае строк и целых чисел переменные содержат непосредственно сами строковые и целочисленные значения. Python всегда использует ссылки в тех случаях, когда в переменных должны храниться изменяемые типы данных, такие как списки или словари. В случае неизменяемых типов данных, таких как строки, целые числа или кортежи, в переменных Python хранятся сами значения.

Несмотря на то что с технической точки зрения в переменных Python хранятся лишь ссылки на списки и словари, часто просто говорят, что переменная содержит список или словарь.

Передача ссылок

Ссылки особенно важны для понимания механизма передачи аргументов функциям. Когда вызывается функция, значения аргументов копируются в переменные параметров. Для списков (и словарей, о которых пойдет речь в следующей главе) это означает, что параметры используют ссылки. Чтобы увидеть, к каким следствиям это приводит, введите следующий код и сохраните его в файле *passingReference.py*.

```
def eggs(someParameter):
    someParameter.append('Hello')

spam = [1, 2, 3]
eggs(spam)
print(spam)
```

Обратите внимание на то, что присваивание нового значения переменной spam осуществляется не за счет использования значения, возвращаемого вызовом eggs(). Вместо этого список изменяется непосредственно на месте. Запустив эту программу, вы получите следующий вывод:

```
[1, 2, 3, 'Hello']
```

Несмотря на то что переменные spam и someParameter содержат отдельные ссылки, они обе ссылаются на один и тот же список. Вот почему вызов метода append('Hello') в функции оказывает воздействие на список даже после того, как был выполнен возврат из функции.

Имейте в виду, что забывчивость относительно того, как Python обрабатывает переменные, в которых хранятся списки и словари, чревата возникновением самых неожиданных ошибок.

Функции copy() и deepcopy() модуля copy

Несмотря на то что передача ссылок нередко оказывается самым удобным способом работы со списками и словарями, в тех случаях, когда их изменяет функция, возможны ситуации, в которых изменение исходного списка или словаря является нежелательным. По этой причине Python предоставляет модуль copy, в котором имеются функции copy() и deepcopy(). Первая из них, copy.copy(), может использоваться для создания дубликата изменяемого значения, такого как список или словарь, а не просто копии ссылки. Введите в интерактивной оболочке следующие команды.

```
>>> import copy
>>> spam = ['A', 'B', 'C', 'D']
>>> cheese = copy.copy(spam)
>>> cheese[1] = 42
>>> spam
['A', 'B', 'C', 'D']
>>> cheese
['A', 42, 'C', 'D']
```

Теперь переменные spam и cheese ссылаются на разные списки, чем и объясняется тот факт, что в результате присваивания значения 42 элементу с индексом 7 в списке cheese изменяется только этот список. Как показано на рис. 4.7, числовые идентификаторы ID ссылок, представляемых обеими переменными, больше не совпадают, поскольку теперь переменные ссылаются на два независимых списка.

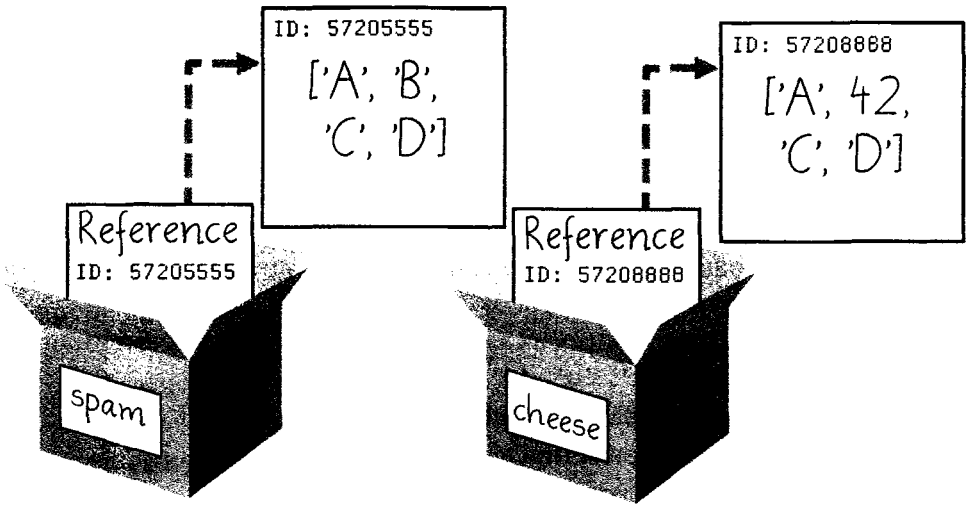


Рис. 4.7. Инструкция `cheese = copy.copy(spam)` создает второй список, который можно изменять независимо от первого

Если список, копию которого нужно создать, сам содержит списки, то в этом случае вместо функции `copy.copy()` следует использовать функцию `copy.deepcopy()`. Эта функция выполняет копирование основного списка вместе со всеми внутренними.

Резюме

Списки — весьма полезный тип данных, поскольку позволяют писать код, способный работать с варьируемым количеством значений в одной переменной. Далее вы познакомитесь с примерами программ, позволяющих сделать то, что без использования списков было бы трудно или вообще невозможно выполнить.

Списки — изменяемый тип данных в том смысле, что их содержимое может изменяться. Кортежи и строки, несмотря на их сходство со списками, являются неизменяемыми типами данных и не могут изменяться. Значение переменной, содержащей кортеж или строку, может быть переопределено путем присвоения ей нового значения в виде кортежа или строки, но это не то же самое, что изменение существующего значения на месте, как это, например, делают методы `append()` и `remove()` в отношении списков.

В переменных хранятся не непосредственно сами списки, а ссылки на них. Это чрезвычайно важное обстоятельство следует учитывать при копировании переменных или передаче аргументов функциям при их вызове. Поскольку копируемое значение представляет собой ссылку на список, остерегайтесь того, чтобы внести непреднамеренные изменения в другие переменные, имеющиеся в вашей программе. Если вы хотите иметь

возможность изменять копию списка таким образом, чтобы это не влияло на исходный список, то используйте функцию `copy()` или `deepcopy()`.

Контрольные вопросы

1. Что означают эти скобки: `[]`?
2. Как бы вы присвоили значение `'hello'` в качестве третьего элемента списка, хранящегося в переменной `spam`? (Предполагается, что в переменной `spam` содержится список `[2, 4, 6, 8, 10]`.)

   В следующих трех вопросах предполагается, что переменная `spam` содержит список `['a', 'b', 'c', 'd']`.
3. Каково значение выражения `spam[int('3' * 2) / 11]`?
4. Каково значение выражения `spam[-1]`?
5. Каково значение выражения `spam[:2]`?

   В следующих трех вопросах предполагается, что переменная `bacon` содержит список `[3.14, 'cat', 11, 'cat', True]`.
6. Каково значение выражения `bacon.index('cat')`?
7. Как будет выглядеть список, хранящийся в переменной `bacon`, после следующего вызова: `bacon.append(99)`?
8. Как будет выглядеть список, хранящийся в переменной `bacon`, после следующего вызова: `bacon.remove('cat')`?
9. Какие операторы используются для конкатенации и репликации списков?
10. В чем состоит различие между предусмотренными для списков методами `append()` и `insert()`?
11. Назовите два способа удаления значений из списков.
12. Назовите несколько общих признаков списков и строк.
13. Чем кортежи отличаются от списков?
14. Как бы вы записали кортеж, содержащий единственное значение в виде целого числа `42`?
15. Как преобразовать список в кортеж? Как преобразовать кортеж в список?
16. Переменные, которые "содержат" список, на самом деле не содержат непосредственно сам список. Что же тогда они содержат?
17. В чем состоит различие между функциями `copy.copy()` и `copy.deepcopy()`?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Запятая в качестве разделителя

Предположим, у вас имеется список наподобие следующего:

```
spam = ['apples', 'bananas', 'tofu', 'cats']
```

Напишите функцию, которая принимает список в качестве аргумента и возвращает строку, в которой все элементы списка разделены запятой и пробелом, а перед последним элементом вставлено слово and. Например, передав функции предыдущий список spam, вы должны получить строку 'apples, bananas, tofu, and cats'. Но ваша функция должна работать с любыми списками.

Рисование символами

Предположим, у вас есть список списков, в котором каждое значение внутренних списков представляет собой строку, состоящую из одного символа, как в приведенном ниже примере.

```
grid = [['.', '.', '.', '.', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['.', 'O', 'O', 'O', 'O', 'O'],
        ['O', 'O', 'O', 'O', 'O', '.'],
        ['O', 'O', 'O', 'O', '.', '.'],
        ['.', 'O', 'O', '.', '.', '.'],
        ['.', '.', '.', '.', '.', '.']]
```

Элемент grid[x][y] можно интерпретировать как символ с координатами x и y в составе "рисунка", нарисованного текстовыми символами. Начало координат (0, 0) находится в верхнем левом углу, координата x увеличивается слева направо, а координата y — сверху вниз.

Скопируйте предыдущее значение grid и напишите код, который использует его для вывода следующего изображения.

```
..OO.OO..
.OOOOOOO.
.OOOOOOO.
..OOOOO..
...OOO...
....O....
```

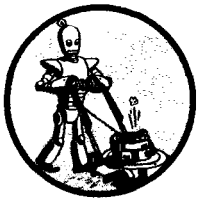
Подсказка. Используйте цикл в цикле для вывода элементов grid[0][0], grid[1][0], grid[2][0] и так далее вплоть до элемента grid[8][0]. Этим вы

заполните первую строку, после чего вам следует вывести символ новой строки. Затем программа должна вывести элементы `grid[0][1]`, `grid[1][1]`, `grid[2][1]` и т.д. Последний элемент, который должна вывести программа — `grid[8][5]`.

Кроме того, не забудьте передать функции именованный аргумент `end`, если хотите отменить автоматический вывод символа новой строки при каждом вызове функции `print()`.

5

СЛОВАРИ И СТРУКТУРИРОВАНИЕ ДАННЫХ



В этой главе речь пойдет о словарях — типе данных, который обеспечивает гибкие возможности доступа к данным и их эффективную организацию. Затем, объединив словари со списками из предыдущей главы, вы создадите структуру данных для игры "крестики-нолики".

Что такое словарь

Как и список, *словарь* — это коллекция многих значений. Однако в словарях, в отличие от списков, индексами могут служить не только целые числа, но и ряд других типов данных. Индексы в словарях называются *ключами*, а ключ вместе с ассоциированным с ним значением — *парой "ключ-значение"*.

В коде словари обозначаются фигурными скобками — `{}`. Введите в интерактивной оболочке следующий код.

```
>>> myCat = {'size': 'fat', 'color': 'gray',
↳ disposition': 'loud'}
```

Здесь переменной `myCat` присваивается словарь. Ключами в нем служат строки `'size'`, `'color'` и `'disposition'`. Значениями ключей являются соответственно строки `'fat'`, `'gray'` и `'loud'`. Доступ к значениям осуществляется посредством их ключей.

```
>>> myCat['size']
'fat'
>>> 'My cat has ' + myCat['color'] +' fur.'
'My cat has gray fur.'
```

Индексами в словарях могут служить также целые числа, как и в списках, однако их отсчет не обязательно должен вестись от 0, и они могут быть любыми числами.

```
>>> spam = {12345: 'Luggage Combination', 42: 'The Answer'}
```

Сравнение словарей и списков

В отличие от списков, в словарях элементы не упорядочены. Первым элементом в списке spam был бы spam[0]. Однако к словарям понятие "первый элемент" неприменимо. В то время как порядок элементов играет существенную роль при проверке совпадения списков, для словарей не имеет ни малейшего значения, в каком порядке в них были включены пары "имя–значение". Введите в интерактивной оболочке следующие команды.

```
>>> spam = ['cats', 'dogs', 'moose']
>>> bacon = ['dogs', 'moose', 'cats']
>>> spam == bacon
False
>>> eggs = {'name': 'Zophie', 'species': 'cat', 'age': '8'}
>>> ham = {'species': 'cat', 'age': '8', 'name': 'Zophie'}
>>> eggs == ham
True
```

Поскольку словари не упорядочены, они не допускают создание срезов, как это возможно в случае списков.

Попытки получения доступа к ключу, отсутствующему в словаре, приводят к возникновению ошибки KeyError, что аналогично возникновению ошибки IndexError error при попытке выхода за пределы допустимого диапазона индексов в случае списков. Введите в интерактивной оболочке следующий код и обратите внимание на сообщение об ошибке, которое выводится, поскольку в словаре отсутствует ключ 'color'.

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> spam['color']
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    spam['color']
KeyError: 'color'
```

Несмотря на то что словари не упорядочены, возможность извлекать произвольное значение по его ключу позволяет использовать гибкие способы организации данных. Предположим, вы хотите, чтобы ваша программа сохраняла данные о днях рождения ваших друзей. Для этой цели вполне подойдет словарь, в котором ключами являются имена друзей, а значениями – даты их

дней рождения. Откройте в файловом редакторе новое окно, введите в нем следующий код и сохраните его в файле *birthdays.py*.

```
❶ birthdays = {'Alice': 'Apr 1', 'Bob': 'Dec 12',
               'Carol': 'Mar 4'}

  while True:
      print('Enter a name: (blank to quit)')
      name = input()
      if name == '':
          break

❷     if name in birthdays:
❸         print(birthdays[name] + ' is the birthday of ' + name)
      else:
          print('I do not have birthday information for ' + name)
          print('What is their birthday?')
          bday = input()
❹         birthdays[name] = bday
          print('Birthday database updated.')
```

Здесь создается исходный словарь, который сохраняется в переменной `birthdays` ❶. Проверить, содержится ли введенное имя в качестве ключа в словаре, можно с помощью ключевого слова `in` ❷, точно так же, как и в случае списков. Если имя содержится в словаре, то доступ к ассоциированному с ним значению осуществляется посредством квадратных скобок ❸; если же данное имя в словаре отсутствует, вы сможете добавить его, используя тот же синтаксис квадратных скобок в сочетании с оператором присваивания ❹.

Выполнив эту программу, вы получите примерно следующий вывод.

```
Enter a name: (blank to quit)
Alice
Apr 1 is the birthday of Alice
Enter a name: (blank to quit)
Eve
I do not have birthday information for Eve
What is their birthday?
Dec 5
Birthday database updated.
Enter a name: (blank to quit)
Eve
Dec 5 is the birthday of Eve
Enter a name: (blank to quit)
```

Разумеется, по завершении работы программы все ранее введенные вами данные теряются. О том, как сохранить данные в файле на жестком диске, вы узнаете в главе 8.

Методы keys(), values() и items()

Существуют три метода для работы со словарями — keys(), values() и items(), возвращающие соответственно ключи, значения и пары "ключ–значение" в виде коллекций, подобных спискам. Возвращаемые этими методами значения не являются истинными списками: их нельзя изменить, и они не имеют метода append(), однако эти типы данных (dict_keys, dict_values и dict_items соответственно) можно использовать в циклах for. Чтобы увидеть, как работают эти методы, введите в интерактивной оболочке следующие команды.

```
>>> spam = {'color': 'red', 'age': 42}
>>> for v in spam.values():
        print(v)

red
42
```

Здесь цикл for используется для итерирования по всем значениям, содержащимся в словаре spam. Однако циклы for можно также использовать для итерирования по ключам или одновременно по ключам и значениям.

```
>>> for k in spam.keys():
        print(k)

color
age
>>> for i in spam.items():
        print(i)

('color', 'red')
('age', 42)
```

Используя методы keys(), values() и items(), можно организовать с помощью цикла for итерации по ключам, значениям и парам "ключ–значение" соответственно. Обратите внимание на то, что элементами значения dict_items, возвращаемого методом items(), являются кортежи, образуемые ключами и ассоциированными с ними значениями.

Если вы хотите получить результат в виде истинного списка, то передайте возвращаемое любым из этих трех методов значение функции list(). Введите в интерактивной оболочке следующие команды.

```
>>> spam = {'color': 'red', 'age': 42}
>>> spam.keys()
dict_keys(['color', 'age'])
```

```
>>> list(spam.keys())
['color', 'age']
```

В строке `list(spam.keys())` значение `dict_keys`, возвращенное функцией `keys()`, передается функции `list()`, которая возвращает список `['color', 'age']`.

Кроме того, можно воспользоваться групповым присваиванием в цикле `for` для присваивания ключей и ассоциированных с ними значений отдельным переменным. Введите в интерактивной оболочке следующие команды.

```
>>> spam = {'color': 'red', 'age': 42}
>>> for k, v in spam.items():
        print('Key: ' + k + ' Value: ' + str(v))

Key: age Value: 42
Key: color Value: red
```

Проверка существования ключа или значения в словаре

Как вам уже известно из предыдущей главы, операторы `in` и `not in` позволяют проверить, существует ли в списке данное значение. Эти же операторы можно использовать и для того, чтобы проверить, содержится ли в словаре определенный ключ или значение. Введите в интерактивной оболочке следующие команды.

```
>>> spam = {'name': 'Zophie', 'age': 7}
>>> 'name' in spam.keys()
True
>>> 'Zophie' in spam.values()
True
>>> 'color' in spam.keys()
False
>>> 'color' not in spam.keys()
True
>>> 'color' in spam
False
```

Обратите внимание на то, что использованная в этом примере инструкция `'color' in spam` по сути является сокращенной записью инструкции `'color' in spam.keys()`. Это общее правило: если вам нужно проверить, является ли (или не является) данное значение ключом в словаре, то для этого достаточно указать после ключевого слова `in` (или `not in`) одно только имя словаря.

Метод get()

Проверять каждый раз при доступе к ключу, есть ли он в словаре, довольно утомительно. К счастью, для словарей предусмотрен метод `get()`, принимающий два аргумента: ключ извлекаемого значения и значение по умолчанию, возвращаемое в случае отсутствия данного ключа в словаре. Введите в интерактивной оболочке следующие команды.

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems.get('cups', 0)) + ' cups.'
'I am bringing 2 cups.'
>>> 'I am bringing ' + str(picnicItems.get('eggs', 0)) + ' eggs.'
'I am bringing 0 eggs.'
```

Поскольку ключ `'eggs'` отсутствует в словаре `picnicItems`, метод `get()` возвращает заданное по умолчанию значение `0`. Если не использовать метод `get()`, то в коде возникнет ошибка.

```
>>> picnicItems = {'apples': 5, 'cups': 2}
>>> 'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
Traceback (most recent call last):
  File "<pyshell#34>", line 1, in <module>
    'I am bringing ' + str(picnicItems['eggs']) + ' eggs.'
KeyError: 'eggs'
```

Метод setdefault()

Часто приходится устанавливать значение для определенного ключа лишь в том случае, если значение ему еще не присваивалось. В коде это выглядит примерно так.

```
spam = {'name': 'Pooka', 'age': 5}
if 'color' not in spam:
    spam['color'] = 'black'
```

С помощью метода `setdefault()` это можно сделать в одной строке кода. Данный метод принимает два аргумента. Первый из них — это проверяемый ключ, а второй — значение, устанавливаемое для этого ключа в случае его отсутствия. Если же ключ существует, то метод `setdefault()` возвращает его значение. Введите в интерактивной оболочке следующий код.

```
>>> spam = {'name': 'Pooka', 'age': 5}
>>> spam.setdefault('color', 'black')
'black'
>>> spam
```

```
{'color': 'black', 'age': 5, 'name': 'Pooka'}
>>> spam.setdefault('color', 'white')
'black'
>>> spam
{'color': 'black', 'age': 5, 'name': 'Pooka'}
```

Первый вызов метода `setdefault()` изменяет словарь, который теперь выглядит так: `{'color': 'black', 'age': 5, 'name': 'Pooka'}`. Метод `setdefault()` возвращает значение `'black'`, которое было установлено данным вызовом для ключа `'color'`. Значение этого ключа не изменяется при последующем вызове `spam.setdefault('color', 'white')`, поскольку в словаре уже имеется ключ `'color'`.

Метод `setdefault()` очень удобно использовать в ситуациях, когда требуется гарантированное наличие ключа. Ниже приведена короткая программа, которая подсчитывает, сколько раз встречается в строке каждая из входящих в нее букв. Откройте новое окно в файловом редакторе, введите в него следующий код и сохраните его в файле *characterCount.py*.

```
message = 'It was a bright cold day in April, and the clocks
↪were striking thirteen.'
count = {}

for character in message:
    count.setdefault(character, 0)
    count[character] = count[character] + 1

print(count)
```

Программа циклически просматривает все символы строки в переменной `message` и подсчитывает, как часто встречается каждый из них. Вызов метода `setdefault()` гарантирует существование ключа в словаре (значение которого по умолчанию равно `0`), и поэтому при выполнении инструкции `count[character] = count[character] + 1` ошибка `KeyError` возникать не будет. Выполнив эту программу, вы получите примерно следующий вывод.

```
{' ': 13, ',': 1, '.': 1, 'A': 1, 'I': 1, 'a': 4, 'c': 3, 'b': 1,
'e': 5, 'd': 3, 'g': 2, 'i':6, 'h': 3, 'k': 2, 'l': 3, 'o': 2,
'n': 4, 'p': 1, 's': 3, 'r': 5, 't': 6, 'w': 2, 'y': 1}
```

Отсюда, например, видно, что, как и следовало ожидать, буква `c` в нижнем регистре встречается 3 раза, пробел — 13 раз, а буква `A` в верхнем регистре — 1 раз. Эта программа будет работать со строкой любой длины, даже если в переменной `message` хранится строка, содержащая миллионы символов!

Красивая печать

Импортировав в свою программу модуль `pprint`, вы получите доступ к функциям `pprint()` и `pformat()`, осуществляющим "красивую печать" значений словаря. Такая возможность может быть полезной, если требуется более аккуратный вывод элементов словаря, чем это обеспечивает функция `print()`. Измените предыдущую программу *characterCount.py*, как показано ниже, и сохраните ее в файле *prettyCharacterCount.py*.

```
import pprint
message = 'It was a bright cold day in April, and the clocks
↳were striking thirteen.'
count = {}

for character in message:
    count.setdefault(character, 0)
    count[character] = count[character] + 1

pprint.pprint(count)
```

На этот раз вывод выглядит намного аккуратнее, и к тому же он рассортирован по ключам.

```
{' ': 13,
',': 1,
'.': 1,
'A': 1,
'I': 1,
'a': 4,
'b': 1,
'c': 3,
'd': 3,
'e': 5,
'g': 2,
'h': 3,
'i': 6,
'k': 2,
'l': 3,
'n': 4,
'o': 2,
'p': 1,
'r': 5,
's': 3,
't': 6,
'w': 2,
'y': 1}
```

Функция `pprint.pprint()` особенно полезна в тех случаях, когда сам словарь содержит вложенные списки или словари.

Если вы хотите получить аккуратно оформленный текст в виде строкового значения, а не выводить его на экран, то воспользуйтесь функцией `pprint.pformat()`. Следующие две строки эквивалентны.

```
pprint.pprint(someDictionaryValue)
print(pprint.pformat(someDictionaryValue))
```

Использование структур данных для моделирования реальных объектов

Возможность играть в шахматы с партнером, находящимся на другой стороне земного шара, существовала еще до того, как появился Интернет. Каждый из игроков, сидя у себя дома за шахматной доской, сообщал партнеру о ходах, которые он делал своими фигурами, по почте. Это требовало применения способа записи шахматных партий, позволяющего однозначно описывать положения фигур на доске и их перемещения.

В алгебраической шахматной нотации клетки шахматной доски обозначаются с использованием буквенно-цифровой записи (рис. 5.1).

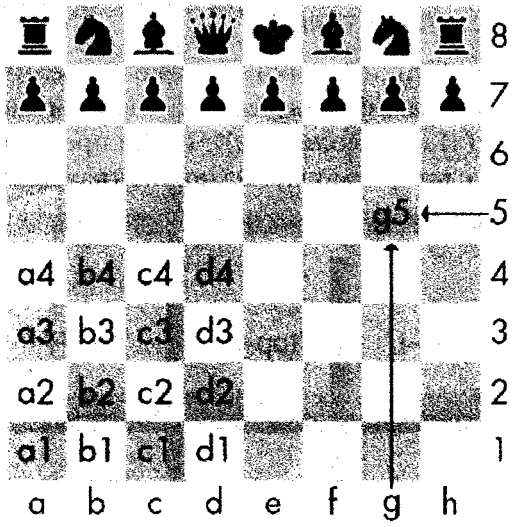


Рис. 5.1. Система координат на шахматной доске, в которой используется буквенно-цифровая нотация

Шахматные фигуры обозначаются буквами: K (king) — король, Q (queen) — ферзь (королева), R (rook) — ладья, B (bishop) — слон и N (knight) — конь. Описание хода включает букву, соответствующую фигуре, которая делает ход, и координаты поля, в которое перемещается данная фигура в результате выполнения хода. Запись пары таких ходов информирует

о том, что происходит при выполнении игроками хода и ответного хода (первый ход за белыми). Например, запись `2. Nf3 Nc6` означает, что на втором ходу белые переместили коня на поле f3, а черные — своего коня на поле c6.

Это далеко не полное описание системы записи шахматных партий, однако для нас важнее всего тот факт, что существует возможность однозначно описывать шахматную партию, даже не находясь за шахматной доской. Вы можете просто читать ходы противника, которые он пересылает вам по почте, и мысленно обновлять положения фигур на шахматной доске.

Компьютеры обладают хорошей памятью. Современные программы позволяют хранить в компьютере миллиарды строк наподобие `'2. Nf3 Nc6'`. Это позволяет компьютеру играть в шахматы без использования шахматной доски. Он моделирует данные для представления шахматной доски, а вы можете писать код, который работает с этой моделью.

Вот тут-то и приходят на помощь списки и словари. Их можно использовать для моделирования объектов реального мира, таких как шахматы. В качестве первого примера мы используем более простую, чем шахматы, игру "крестики-нолики".

Поле для игры в "крестики-нолики"

Поле для игры в "крестики-нолики" похоже на увеличенный символ "решетки" (`#`) с девятью клетками, каждая из которых может быть пустой или содержать "крестик" (`X`) или "нолик" (`O`). Чтобы представлять клетки игрового поля с помощью словаря, можно назначить каждой из них ключ в виде строки (рис. 5.2).

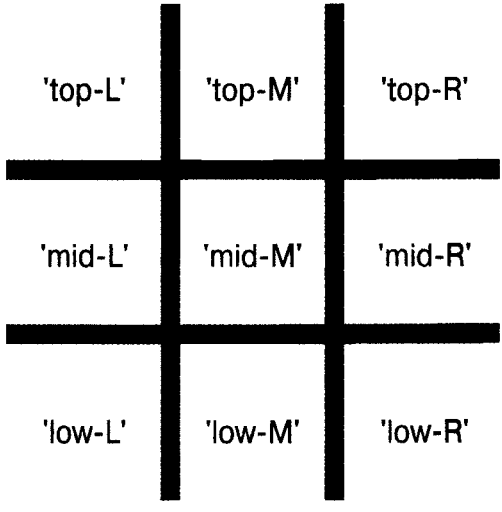


Рис. 5.2. Клетки игры в "крестики-нолики" с указанием соответствующих ключей

Для представления содержимого клеток можно использовать строки: 'X', 'O' и ' ' (символ пробела). Таким образом, всего нам понадобится девять таких строковых значений. Чтобы связать эти значения с клетками игрового поля, используем словарь. Состояние верхнего правого угла можно представить с помощью строкового значения, ассоциированного с ключом 'top-R', состояние нижнего левого угла — с помощью строкового значения, ассоциированного с ключом 'low-L', состояние центральной клетки — с помощью строкового значения, ассоциированного с ключом 'mid-M', и т.д.

Этот словарь и есть структура данных, которая представляет состояние поля для игры в "крестики-нолики". Сохраните его в переменной theBoard. Откройте новое окно файлового редактора, введите в него следующий исходный код и сохраните его в файле ticTacToe.py.

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

Структуре данных, сохраненной в переменной theBoard, соответствует состояние поля, представленное на рис. 5.3.

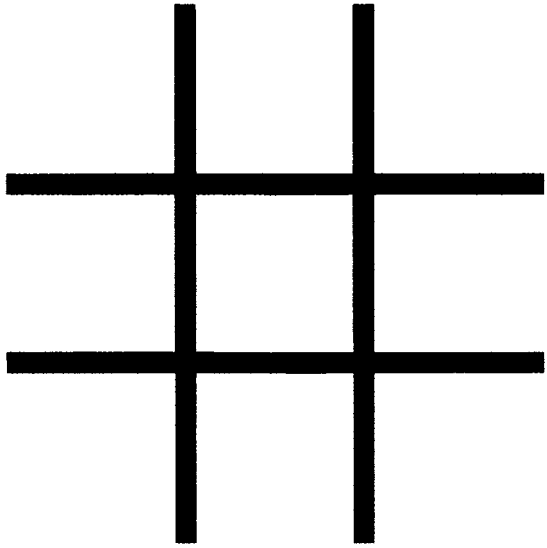


Рис. 5.3. Пустое поле для игры в "крестики-нолики"

Поскольку в словаре theBoard каждому ключу соответствует строка в виде одиночного символа пробела, данное состояние соответствует пустому полю. Если игрок X своим первым ходом выберет центральную клетку, то это состояние поля будет представлено следующим словарем.

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': 'X', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
```

Теперь структуре данных, сохраненной в переменной `theBoard`, соответствует вид игрового поля, представленный на рис. 5.4.

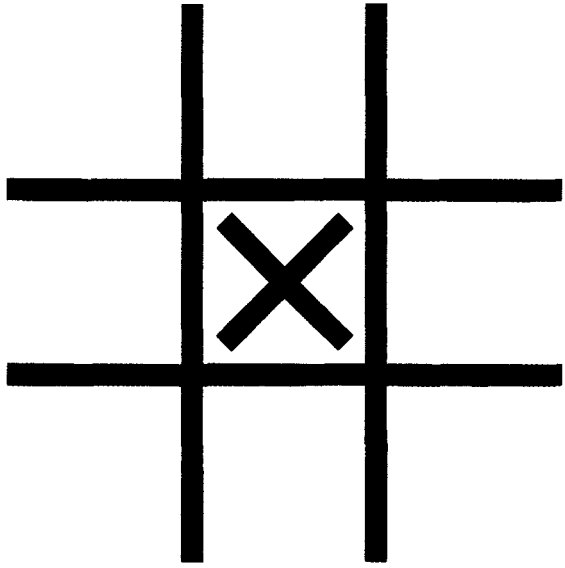


Рис. 5.4. Вид игрового поля после первого хода

Ниже показана структура данных, соответствующая выигрышу игрока O, разместившего три символа *O* в верхних клетках.

```
theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O',
            'mid-L': 'X', 'mid-M': 'X', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}
```

Этой структуре данных соответствует вид поля, представленный на рис. 5.5.

Разумеется, игроки могут видеть только то, что выведено на экран, а не непосредственное содержимое переменных. Создадим функцию, отображающую содержимое словаря на экране. Внесите в файл `ticTacToe.py` следующие изменения (новый код выделен полужирным шрифтом).

```
theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
            'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': ' '}
def printBoard(board):
```

```
    print(board['top-L'] + '|' + board['top-M'] + '|' +
          board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' +
          board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' +
          board['low-R'])
printBoard(theBoard)
```

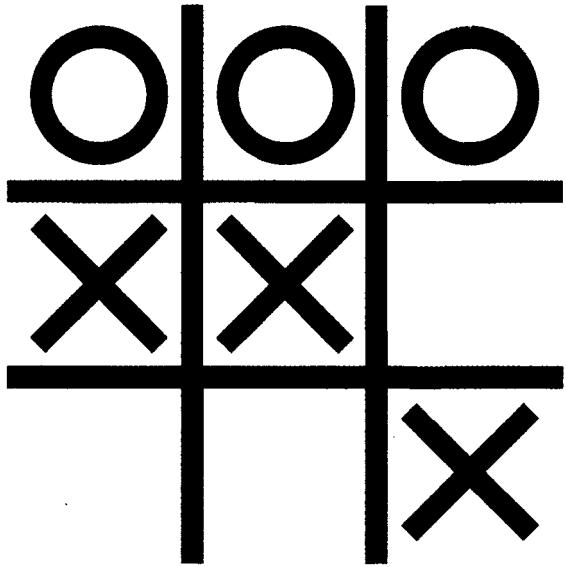


Рис. 5.5. Выиграл игрок O

Когда вы запустите эту программу, функция `printBoard()` выведет на экран пустое игровое поле.

```
 | |
-+-+-
 | |
-+-+-
 | |
```

Функция `printBoard()` может обрабатывать любую структуру "крестиков-ноликов", которую вы ей передадите. Внесите в код следующие изменения.

```
theBoard = {'top-L': 'O', 'top-M': 'O', 'top-R': 'O',
            'mid-L': 'X', 'mid-M': 'X', 'mid-R': ' ',
            'low-L': ' ', 'low-M': ' ', 'low-R': 'X'}
```

```
def printBoard(board):
    print(board['top-L'] + '|' + board['top-M'] + '|' +
          board['top-R'])
    print('-+-+-')
    print(board['mid-L'] + '|' + board['mid-M'] + '|' +
          board['mid-R'])
    print('-+-+-')
    print(board['low-L'] + '|' + board['low-M'] + '|' +
          board['low-R'])
printBoard(theBoard)
```

В результате выполнения этой программы на экран будет выведено следующее состояние игрового поля.

```
O|O|O
-+-+-
X|X|
-+-+-
 | |X
```

Поскольку вы создали структуру данных для представления игрового поля и написали код (функцию `printBoard()`), способный интерпретировать эту структуру, тем самым вы создали программу, которая "моделирует" игру в "крестики-нолики". Вы могли бы организовать свою структуру данных иначе (например, использовать ключи наподобие `'TOP-LEFT'` вместо `'top-L'`), но коль скоро ваш код правильно обрабатывает выбранную структуру данных, программа будет работать корректно.

Например, функция `printBoard()` ожидает, что ей будет передаваться структура данных в виде словаря с ключами для всех девяти клеток. Если, скажем, в передаваемом функции словаре отсутствует ключ `'mid-L'`, то ваша программа работать не будет.

```
O|O|O
-+-+-
Traceback (most recent call last):
  File "ticTacToe.py", line 10, in <module>
    printBoard(theBoard)
  File "ticTacToe.py", line 6, in printBoard
    print(board['mid-L'] + '|' + board['mid-M'] + '|' +
          board['mid-R'])
KeyError: 'mid-L'
```

А сейчас мы добавим код, который позволяет игрокам вводить свои ходы. Внесите изменения в программу *ticTacToe.py*, чтобы она приняла следующий вид.

```
  theBoard = {'top-L': ' ', 'top-M': ' ', 'top-R': ' ',
              'mid-L': ' ', 'mid-M': ' ', 'mid-R': ' ',
              'low-L': ' ', 'low-M': ' ', 'low-R': ' '}

  def printBoard(board):
      print(board['top-L'] + '|' + board['top-M'] + '|' +
            board['top-R'])
      print('-+-+-')
      print(board['mid-L'] + '|' + board['mid-M'] + '|' +
            board['mid-R'])
      print('-+-+-')
      print(board['low-L'] + '|' + board['low-M'] + '|' +
            board['low-R'])
  turn = 'X'
  for i in range(9):
❶     printBoard(theBoard)
      print('Turn for ' + turn + '. Move on which space?')
❷     move = input()
❸     theBoard[move] = turn
❹     if turn == 'X':
          turn = 'O'
      else:
          turn = 'X'
  printBoard(theBoard)
```

Новый код выводит состояние игрового поля в начале каждого очередного хода ❶, получает ход активного игрока ❷, соответствующим образом обновляет игровое поле ❸, а затем передает право хода другому игроку ❹, прежде чем перейти к следующему ходу. Когда вы запустите программу, вывод в процессе игры будет выглядеть примерно так.

```
  | |
-+-+-
  | |
-+-+-
  | |
Turn for X. Move on which space?
mid-M
  | |
-+-+-
 |X|
-+-+-
  | |
Turn for O. Move on which space?
low-L
  | |
-+-+-
 |X|
-+-+-
O| |
```

```
--опущено--

O|O|X
-+-+-
X|X|O
-+-+-
O| |X
Turn for X. Move on which space?

low-M
O|O|X
-+-+-
X|X|O
-+-+-
O|X|X
```

Это еще далеко не полный вариант программы, поскольку в нем, например, вообще не определяется, выиграл ли игрок. Однако и такого кода вполне достаточно для того, чтобы понять, как структуры данных могут использоваться в программах.

Примечание

Любознательные читатели могут ознакомиться с полным исходным кодом программы для игры в "крестики-нолики", посетив сайт `http://nostarch.com/automatestuff/`.

Вложенные словари и списки

Моделировать игру в "крестики-нолики" было сравнительно просто: для описания состояния игрового поля потребовался всего лишь один словарь с девятью парами "ключ–значение". Может оказаться так, что по мере возрастания сложности ваших моделей вам понадобятся словари и списки, содержащие другие списки и словари. Списки удобны для хранения упорядоченных последовательностей значений, а словари — для ассоциирования значений с ключами. Ниже приведена программа, в которой для описания того, что приносит с собой каждый из гостей, приглашенных на пикник, используется словарь, содержащий другой словарь. Функция `totalBrought()` способна читать эту структуру данных и рассчитывать общее количество принесенного гостями.

```
allGuests = {'Alice': {'apples': 5, 'pretzels': 12},
             'Bob': {'ham sandwiches': 3, 'apples': 2},
             'Carol': {'cups': 3, 'apple pies': 1}}
```

```
def totalBrought(guests, item):
    numBrought = 0
❶   for k, v in guests.items():
❷       numBrought = numBrought + v.get(item, 0)
    return numBrought

print('Number of things being brought:')
print(' - Apples         ' + str(totalBrought(allGuests,
                                 'apples')))
print(' - Cups           ' + str(totalBrought(allGuests,
                                 'cups')))
print(' - Cakes          ' + str(totalBrought(allGuests,
                                 'cakes')))
print(' - Ham Sandwiches ' + str(totalBrought(allGuests,
                                 'ham sandwiches')))
print(' - Apple Pies ' + str(totalBrought(allGuests,
                                 'applepies')))
```

В функции `totalBrought()` цикл `for` выполняет итерации по парам "ключ-значение", хранящимся в переменной `guests` ❶. В цикле переменной `k` присваивается строка с именем гостя, а переменной `v` — словарь с информацией о принесенном гостями. Если в словаре имеется ключ, соответствующий тому, что принес гость, то его значение (количество принесенных единиц) прибавляется к переменной `numBrought` ❷. В случае отсутствия такого ключа метод `get()` возвращает значение `0`, которое, будучи прибавленным к значению `numBrought`, не влияет на результат сложения.

Вывод этой программы выглядит так.

```
Number of things being brought:
- Apples 7
- Cups 3
- Cakes 0
- Ham Sandwiches 3
- Apple Pies 1
```

Может показаться, что эта модель слишком простая, чтобы обременять себя написанием специальной программы для нее. Однако задумайтесь над тем, что эта же функция `totalBrought()` позволит легко обрабатывать словари, содержащие тысячи гостей, которые могут приносить тысячи различных блюд. В подобных случаях сохранение такой информации в структуре данных и ее обработка с помощью функции `totalBrought()` сохранит вам массу времени.

Вы можете моделировать различные задачи с помощью структур данных по своему усмотрению при условии, что остальная часть кода корректно обрабатывает выбранную модель. Когда вы только начинаете программировать, не задумывайтесь особенно о выборе "наилучшей модели" для

описания своих данных. Возможно, по мере обретения опыта вам удастся найти более подходящие модели, но самое главное заключается в том, чтобы эта модель удовлетворяла задачам вашей программы.

Резюме

Из этой главы вы узнали все о словарях. Списки и словари могут содержать множество различных значений, включая другие списки и словари. Словари удобны тем, что позволяют отображать одни элементы (ключи) в другие (значения), в отличие от списков, которые просто содержат упорядоченные последовательности значений. Доступ к значениям словаря осуществляется посредством использования квадратных скобок, точно так же, как и в списках. Однако вместо целочисленных индексов в словарях допускается использование ключей в виде самых разных типов данных: целых и вещественных чисел, строк, кортежей. Организуя данные программы в структуры, можно создавать представления реальных объектов. Примером этого может служить игра в "крестики-нолики".

Рассмотренный материал охватывает почти все основные концепции программирования на Python. В оставшейся части книги вам предстоит узнать еще очень много нового, но вам уже известно достаточно для того, чтобы писать полезные программы, автоматизирующие выполнение ряда важных задач. Возможно, вы даже не осознаете, что ваших знаний вполне достаточно для того, чтобы загружать веб-страницы, обновлять электронные таблицы или отправлять текстовые сообщения, и неоценимую помощь в этом вам окажут модули Python! Эти модули, написанные другими программистами, предоставляют функции, которые упрощают для вас выполнение задач такого рода. Поэтому беритесь за написание реальных программ для выполнения полезных автоматизированных задач.

Контрольные вопросы

1. Как выглядит код для пустого словаря?
2. Как выглядит элемент словаря с ключом `'foo'` и значением `42`?
3. Опишите основные различия между словарем и списком.
4. Что произойдет при попытке получения доступа к элементу `spam['foo']`, если `spam` — это `{'bar': 100}`?
5. Если в переменной `spam` хранится словарь, то в чем состоит разница между выражениями `'cat' in spam` и `'cat' in spam.keys()`?
6. Если в переменной `spam` хранится словарь, то в чем состоит разница между выражениями `'cat' in spam` и `'cat' in spam.values()`?
7. В какой сокращенной форме можно записать приведенный ниже код?

```
if 'color' not in spam:
    spam['color'] = 'black'
```

8. Какие модуль и функция могут быть использованы для "красивой печати" значений словаря?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Инвентарь приключенческой игры

Вы создаете приключенческую видеоигру. Структурой данных для инвентаря игрока должен быть словарь, в котором ключи — это строковые значения, описывающие инвентарь, а значения — количество имеющихся у игрока единиц данного инвентаря. Например, в случае словаря

```
{'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}
```

это означает, что у игрока есть 1 веревка, 6 фонарей, 42 золотые монеты, 1 кинжал и 12 стрел.

Напишите функцию `displayInventory()`, которая принимает в качестве аргумента описание любого "инвентаря" и отображает его примерно в следующем виде.

```
Inventory:
12 arrow
42 gold coin
1 rope
6 torch
1 dagger
Total number of items: 62
```

Подсказка. Для просмотра всех ключей словаря можно использовать цикл `for`.

```
# inventory.py
stuff = {'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}

def displayInventory(inventory):
    print("Inventory:")
    item_total = 0
    for k, v in inventory.items():
        print(str(v) + ' ' + k)
```

```
        item_total += v
    print("Total number of items: " + str(item_total))

displayInventory(stuff)
```

Функция преобразования списка в словарь для приключенческой игры

Представьте, что добыча побежденного дракона представлена в виде списка строк наподобие следующего:

```
dragonLoot = ['gold coin', 'dagger', 'gold coin', 'gold coin',
↳ 'ruby']
```

Напишите функцию `addToInventory(inventory, addedItems)`, в которой параметр `inventory` — это словарь, представляющий инвентарь игрока (как в предыдущем проекте), а параметр `addedItems` — это список наподобие `dragonLoot`. Функция `addToInventory()` должна возвращать словарь, представляющий обновленный инвентарь. Обратите внимание на то, что в списке `addedItems` один и тот же элемент может встречаться несколько раз. Ваш код может выглядеть примерно так.

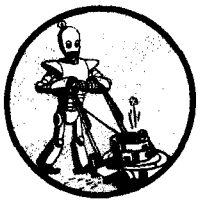
```
def addToInventory(inventory, addedItems):
# сюда должен быть вставлен ваш код

inv = {'gold coin': 42, 'rope': 1}
dragonLoot = ['gold coin', 'dagger', 'gold coin', 'gold coin',
↳ 'ruby']
inv = addToInventory(inv, dragonLoot)
displayInventory(inv)
```

Предыдущая программа (с вашей функцией `displayInventory()` из предыдущего проекта) должна вывести следующее.

```
Inventory:
45 gold coin
1 rope
1 ruby
1 dagger

Total number of items: 48
```

6

МАНИПУЛИРОВАНИЕ СТРОКАМИ



Текст — одна из наиболее распространенных форм данных, которые будут обрабатываться вашими программами. Вы уже знаете, как конкатенировать два строковых значения с помощью оператора +, но ваши возможности гораздо шире. Python позволяет извлекать части строк из строковых значений, добавлять и удалять пробелы, преобразовывать буквы из нижнего регистра в верхний и обратно, а также форматировать строки по своему желанию. Вы даже можете написать код на Python для доступа к буферу обмена, используя его для копирования и вставки текста.

В этой главе мы поработаем над двумя программными проектами: простым менеджером паролей и программой для автоматизации рутинной работы по форматированию текста.

Работа со строками

Приступим к рассмотрению различных способов записи строк, а также их вывода на экран и получения доступа к ним с помощью кода на языке Python.

Строковые литералы

Ввод строк в коде Python не составляет труда: они начинаются и заканчиваются символами апострофа (одинарными кавычками). Но как быть, если кавычки требуются в самой строке? Ввод строки в виде `'That is Alice's cat.'` не сработает, поскольку Python интерпретирует вторую кавычку после слова `Alice` как конец строки и сочтет остальную часть текста (`s cat.'`) недопустимым кодом. К счастью, существует несколько способов ввода строк.

Кавычки

Начало и конец строки можно обозначать не только парой апострофов, но и парой кавычек. Одно из преимуществ использования кавычек — в том, что они позволяют включать символ апострофа в состав строки. Введите в интерактивной оболочке следующий код:

```
>>> spam = "That is Alice's cat."
```

Поскольку строка начинается с кавычки, Python в состоянии идентифицировать апостроф как часть строки. Если же в составе строки требуются как апострофы, так и кавычки, то необходимо прибегать к экранированию символов.

Экранированные символы

Техника экранирования позволяет использовать символы, вставка которых в строку иными способами невозможна. *Экранированный символ* включает символ обратной косой черты (\), за которым следует сам символ, добавляемый в строку. (Несмотря на то что экранированный символ состоит из двух символов, его рассматривают как одиночный.) Вот так, например, выглядит экранированный апостроф: `\'`. Его можно использовать даже в строке, которая начинается и заканчивается апострофом. Чтобы увидеть, как работают экранированные символы, введите в интерактивной оболочке следующий код:

```
>>> spam = 'Say hi to Bob\'s mother.'
```

Поскольку в слове `Bob\'s` апострофу предшествует символ обратной косой черты, Python знает, что в данном случае апостроф не служит маркером конца строки. Экранированные символы `\'` и `\"` позволяют включать в строку соответственно апострофы и кавычки.

Экранированные символы, которые можно использовать, приведены в табл. 6.1.

Таблица 6.1. Экранированные символы

Экранированный символ	Отображаемый символ
`\'`	Апостроф
`\"`	Кавычка
`\t`	Символ табуляции
`\n`	Символ новой строки (разрыва строки)
`\\`	Символ обратной косой черты

Введите в интерактивной оболочке следующую команду.

```
>>> print("Hello there!\nHow are you?\nI\'m doing fine.")
Hello there!
How are you?
I'm doing fine.
```

"Сырые" строки

Поместив символ `r` перед начальной кавычкой, вы обозначаете строку как "сырую". *Сырая* (т.е. необработанная) строка полностью игнорирует механизм экранирования и выводит все символы обратной косой черты, которые встречаются в строке, как обычные символы. Введите, например, в интерактивной оболочке следующую команду.

```
>>> print(r'That is Carol\'s cat.')
That is Carol\'s cat.
```

Поскольку это "сырая" строка, Python считает все символы обратной косой черты ее частью, а не началом экранированного символа. "Сырые" строки полезны в тех случаях, когда вы вводите строковые значения с многочисленными символами обратной косой черты, как это бывает при использовании регулярных выражений, описанных в следующей главе.

Многострочные блоки

Несмотря на то что вы всегда можете включить в строку символ новой строки с помощью экранированного символа `\n`, во многих случаях гораздо проще использовать многострочные блоки. В Python многострочные блоки представляют собой группу строк, заключенных в тройные апострофы или кавычки. Любые апострофы, кавычки или символы новой строки в блоке, ограниченном такими "тройными кавычками", считаются частью строки. Правила Python, регламентирующие форматирование блоков кода с помощью отступов, в отношении многострочных блоков не действуют.

Откройте файловый редактор и введите следующий код.

```
print('''Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and
↪ extortion.

Sincerely,
Bob''')
```

Сохраните эту программу в файле *catnapping.py* и выполните ее. Вы должны получить следующий вывод.

```
Dear Alice,

Eve's cat has been arrested for catnapping, cat burglary, and
extortion.

Sincerely,
Bob
```

Обратите внимание на то, что для символа апострофа в слове `Eve's` экранирование не понадобилось. В "сырых" строках экранирование апострофов и кавычек необязательно. Для вывода текста в таком же виде без использования многострочного блока потребуется следующий вызов функции `print()`.

```
print('Dear Alice,\n\nEve\'s cat has been arrested for catnapping,
↪cat burglary, and extortion.\n\nSincerely,\nBob')
```

Многострочные комментарии

В то время как символом "решетка" (#) обозначается однострочный комментарий, длина которого ограничивается концом строки, многострочные блоки часто используют в качестве многострочных комментариев, охватывающих произвольное количество строк. Приведенный ниже текст абсолютно корректен в Python.

```
"""This is a test Python program.
Written by Al Sweigart al@inventwithpython.com

This program was designed for Python 3, not Python 2.
"""

def spam():
    """This is a multiline comment to help
    explain what the spam() function does."""
    print('Hello!')
```

Индексирование строк и извлечение срезов

В случае строк операции индексирования и извлечения срезов выполняются точно так же, как и в случае списков. Можете рассматривать строку `'Hello world!'` как список, в котором каждый символ строки является элементом списка и имеет соответствующий индекс.

```
'   H   e   l   l   o       w   o   r   l   d   !   ,
    0   1   2   3   4   5   6   7   8   9   10  11
```

Пробелы и восклицательный знак входят в число символов, поэтому фраза `'Hello world!'` содержит 12 символов, от символа `H` с индексом `0` до символа `!` с индексом `11`.

Введите в интерактивной оболочке следующие команды.

```
>>> spam = 'Hello world!'
  spam[0]
'H'
>>> spam[4]
'o'
>>> spam[-1]
'!'
>>> spam[0:5]
'Hello'
>>> spam[:5]
'Hello'
>>> spam[6:]
'world!'
```

Указав индекс, вы получаете символ, находящийся в соответствующей позиции в строке. В случае указания диапазона индексов, т.е. извлечения среза, элемент с начальным индексом включается в срез, тогда как элемент с конечным индексом не включается. Поэтому, если `spam` — это строка `'Hello world!'`, то срез `spam[0:5]` содержит строку `'Hello'`. Подстрока, получаемая с помощью среза `spam[0:5]`, будет включать в себя все символы от `spam[0]` до `spam[4]`, тогда как символ пробела, имеющий индекс `5`, в нее не войдет.

Имейте в виду, что извлечение части строки посредством среза не сопровождается изменением исходной строки. Вы можете сохранить срез, извлеченный из одной переменной, в другой переменной. Введите в интерактивной оболочке следующие команды.

```
>>> spam = 'Hello world!'
>>> fizz = spam[0:5]
>>> fizz
'Hello'
```

Извлекая срез и сохраняя результирующую подстроку в другой переменной, вы получаете быстрый и простой доступ как ко всей строке, так и к ее подстроке.

Использование операторов in и not in со строками

Операторы in и not in применяются к строкам точно так же, как и к спискам. Результатом вычисления выражения в виде двух строк, соединенных любым из этих операторов, является булево значение True или False. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello' in 'Hello World'
True
>>> 'Hello' in 'Hello'
True
>>> 'HELLO' in 'Hello World'
False
>>> '' in 'spam'
True
>>> 'cats' not in 'cats and dogs'
False
```

Эти выражения проверяют, содержится ли первая строка (в строгом соответствии с регистром набора символов) во второй строке.

Полезные методы для работы со строками

Некоторые методы анализируют строки или создают преобразованные строковые значения. В этом разделе описаны те из них, которые вы будете использовать наиболее часто.

Методы upper(), lower(), isupper() и islower()

Методы upper() и lower() возвращают новую строку, в которой все буквы исходной строки преобразованы соответственно в верхний или нижний регистр. Небуквенные символы не испытывают никаких изменений. Введите в интерактивной оболочке следующий код.

```
>>> spam = 'Hello world!'
>>> spam = spam.upper()
>>> spam
'HELLO WORLD!'
>>> spam = spam.lower()
>>> spam
'hello world!'
```

Имейте в виду, что эти методы возвращают новые строковые значения, не изменяя саму исходную строку. Чтобы изменить исходную строку, необходимо вызвать для нее метод upper() или lower() и присвоить результирующее строковое значение той же переменной, в которой хранилась

исходная строка. Именно поэтому для того, чтобы изменить строку, хранящуюся в переменной spam, следует выполнить операцию присваивания spam = spam.upper(), а не просто вызвать функцию spam.upper(). (В качестве аналогии можно привести переменную eggs, содержащую значение 10. Выражение eggs + 3 не изменит значения eggs, но это можно сделать с помощью инструкции eggs = eggs + 3.)

Методы upper() и lower() удобно применять в тех случаях, когда при сравнении строк не должен учитываться регистр букв. Строки 'great' и 'GREat' — это разные строки. Однако в приведенной ниже небольшой программе не имеет значения, в какой форме пользователь введет слово, Great, GREAT или grEAT, поскольку строка предварительно преобразуется в верхний регистр.

```
print('How are you?')
feeling = input()
if feeling.lower() == 'great':
    print('I feel great too.')
else:
    print('I hope the rest of your day is good.')
```

Когда вы запустите эту программу, на экране отобразится вопрос, и даже если вы ответите на него, введя слово GREat, все равно будет выведена строка I feel great too. Добавляя в свою программу код, нивелирующий различия в написании слов и игнорирующий такие ошибки, как неправильное использование строчных и прописных букв, вы упростите работу с ней и уменьшите вероятность ее аварийного завершения из-за ошибок, допущенных пользователем при вводе текста.

```
How are you?
GREat
I feel great too.
```

Методы isupper() и islower() возвращают булево значение True, если в строке имеется хотя бы одна буква и все буквы относятся соответственно к верхнему или нижнему регистру. В противном случае метод возвращает значение False. Введите в интерактивной оболочке следующие команды и обратите внимание на возвращаемые методами значения.

```
>>> spam = 'Hello world!'
>>> spam.islower()
False
>>> spam.isupper()
False
>>> 'HELLO'.isupper()
```

```
True
>>> 'abc12345'.islower()
True
>>> '12345'.islower()
False
>>> '12345'.isupper()
False
```

Поскольку методы `upper()` и `lower()` сами возвращают строки, для этих строк, в свою очередь, также можно вызывать строковые методы. Выражения, с помощью которых это делается, выглядят как цепочки вызовов методов. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello'.upper()
'HELLO'
>>> 'Hello'.upper().lower()
'hello'
>>> 'Hello'.upper().lower().upper()
'HELLO'
>>> 'HELLO'.lower()
'hello'
>>> 'HELLO'.lower().islower()
True
```

Строковые методы `isX()`

Наряду с функциями `islower()` и `isupper()` существует ряд методов для работы со строками, имена которых начинаются со слова `is`. Методы этой категории возвращают булево значение, описывающее природу строки, для которой они вызываются. Ниже приведен перечень часто используемых строковых методов типа `isX()`.

- `isalpha()` возвращает значение `True`, если строка состоит только из букв и не является пустой.
- `isalnum()` возвращает значение `True`, если строка состоит только из буквенно-цифровых символов и не является пустой.
- `isdecimal()` возвращает значение `True`, если строка состоит только из цифровых символов и не является пустой.
- `isspace()` возвращает значение `True`, если строка состоит только из символов пробела, табуляции, новой строки и не является пустой.
- `istitle()` возвращает значение `True`, если строка состоит только из слов, которые начинаются с буквы в верхнем регистре, за которой следуют только буквы в нижнем регистре.

Введите в интерактивной оболочке следующие команды.

```
>>> 'hello'.isalpha()
True
>>> 'hello123'.isalpha()
False
>>> 'hello123'.isalnum()
True
>>> 'hello'.isalnum()
True
>>> '123'.isdecimal()
True
>>> ' '.isspace()
True
>>> 'This Is Title Case'.istitle()
True
>>> 'This Is Title Case 123'.istitle()
True
>>> 'This Is not Title Case'.istitle()
False
>>> 'This Is NOT Title Case Either'.istitle()
False
```

Методы `isX()` удобно применять для проверки допустимости введенных пользователем данных. Например, приведенная ниже программа повторяет запрос ввода пользователем своего возраста и пароля до тех пор, пока эти данные не будут предоставлены в корректном формате. Откройте новое окно в файловом редакторе, введите в нем следующий код программы и сохраните его в файле *validateInput.py*.

```
while True:
    print('Enter your age:')
    age = input()
    if age.isdecimal():
        break
    print('Please enter a number for your age.')

while True:
    print('Select a new password (letters and numbers only):')
    password = input()
    if password.isalnum():
        break
    print('Passwords can only have letters and numbers.')
```

В первом цикле `while` программа запрашивает ввод возраста пользователя и сохраняет введенное значение. Если пользователь ввел возраст в виде допустимого значения (десятичного числа), первый цикл прерывается и управление передается второму циклу `while`, в котором запрашивается пароль. В противном случае программа информирует пользователя о том, что должно быть введено число, и предлагает повторно указать возраст.

Во втором цикле while запрашивается и сохраняется пароль. Если пользователь ввел буквенно-цифровое значение, выполняется выход из цикла. В противном случае программа информирует пользователя о том, что допускаются только пароли, состоящие из букв и цифр, и предлагает ему повторить ввод.

Запустив программу, вы должны получить примерно такой вывод.

```
Enter your age:
forty two
Please enter a number for your age.
Enter your age:
42
Select a new password (letters and numbers only):
secr3t!
Passwords can only have letters and numbers.
Select a new password (letters and numbers only):
secr3t
```

Вызывая методы isdecimal() и isalnum() для переменных, мы можем выяснить, являются ли введенные пользователем значения цифровыми или буквенно-цифровыми. В данном случае эти проверки позволили отвергнуть ввод пользователем строки forty two при указании возраста, но принять ввод числа 42, а также отвергнуть ввод значения secr3t! в качестве пароля, но принять ввод значения secr3t.

Методы startswith() и endswith()

Методы startswith() и endswith() возвращают значение True, если строка, для которой они вызываются, соответственно начинается или заканчивается строкой, переданной методу; в противном случае они возвращают значение False. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello world!'.startswith('Hello')
True
>>> 'Hello world!'.endswith('world!')
True
>>> 'abc123'.startswith('abcdef')
False
>>> 'abc123'.endswith('12')
False
>>> 'Hello world!'.startswith('Hello world!')
True
>>> 'Hello world!'.endswith('Hello world!')
True
```

Эти методы — полезная альтернатива оператору сравнения ==, если сравнение с другой строкой требуется выполнить не для всей исходной строки, а только для первой или последней ее части.

Строковые методы join() и split()

Метод join() удобно использовать в тех случаях, когда ряд строк, представленных в виде списка, необходимо объединить в одно строковое значение. Метод join() вызывается для некоторой строки, принимает список строк в качестве аргумента и возвращает строку. Возвращаемая строка представляет собой результат конкатенации всех строк, переданных в виде списка. Введите в интерактивной оболочке следующие команды.

```
>>> ', '.join(['cats', 'rats', 'bats'])
'cats, rats, bats'
>>> ' '.join(['My', 'name', 'is', 'Simon'])
'My name is Simon'
>>> 'ABC'.join(['My', 'name', 'is', 'Simon'])
'MyABCnameABCisABCSimon'
```

Обратите внимание на то, что строка, для которой вызывается метод join(), вставляется между переданными посредством аргумента строками. Например, если выполнить вызов join(['cats', 'rats', 'bats']) для строки ', ', то будет возвращена строка 'cats, rats, bats'.

Не забывайте о том, что метод join() вызывается для строкового значения и в качестве аргумента принимает список. (Вы можете вызвать метод как-то иначе, сами того не осознавая.) Метод split() делает совершенно противоположное: он вызывается для строкового значения и возвращает список строк. Введите в интерактивной оболочке следующую команду.

```
>>> 'My name is Simon'.split()
['My', 'name', 'is', 'Simon']
```

По умолчанию строка 'My name is Simon' разбивается на отдельные строки в тех местах, где встречаются пробельные символы: пробел, символ табуляции, символ новой строки. Эти пробельные символы не включаются в строки, возвращаемые в виде списка. Вы можете указать другую строку-разделитель, передав ее методу split(). Например, введите в интерактивной оболочке следующие команды.

```
>>> 'MyABCnameABCisABCSimon'.split('ABC')
['My', 'name', 'is', 'Simon']
>>> 'My name is Simon'.split('m')
['My na', 'e is Si', 'on']
```

Обычный способ применения метода `split()` — разбиение многострочного блока по символам новой строки. Введите в интерактивной оболочке следующие команды.

```
>>> spam = '''Dear Alice,
How have you been? I am fine.
There is a container in the fridge
that is labeled "Milk Experiment".

Please do not drink it.
Sincerely,
Bob'''
>>> spam.split('\n')
['Dear Alice,', 'How have you been? I am fine.', 'There is a
container in the fridge', 'that is labeled "Milk Experiment".',
'', 'Please do not drink it.', 'Sincerely,', 'Bob']
```

Передача методу `split()` строки `'\n'` в качестве аргумента позволяет выполнить разбивку многострочного блока, сохраненного в переменной `spam`, в позициях символов новой строки, и возвратить список, каждый элемент которого соответствует одной строке текста.

Выравнивание текста с помощью методов `rjust()`, `ljust()` и `center()`

Строковые методы `rjust()` и `ljust()` возвращают версию строки, для которой они вызываются, выровненную за счет вставки пробелов. В обоих методах первый аргумент — целое число, определяющее длину выровненной строки. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello'.rjust(10)
'  Hello'
>>> 'Hello'.rjust(20)
'               Hello'
>>> 'Hello World'.rjust(20)
'         Hello World'
>>> 'Hello'.ljust(10)
'Hello  '
```

Выражение `'Hello'.rjust(10)` говорит о том, что мы хотим выровнять строку `'Hello'` вправо так, чтобы полная длина строки составляла 10. В слове `'Hello'` насчитывается пять символов, поэтому слева от него будут добавлены пять пробелов, в результате чего полная длина строки составит 10 символов, причем слово `'Hello'` будет выровнено вправо.

Необязательный второй аргумент в обоих методах указывает на символ-заполнитель, отличный от пробела. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello'.rjust(20, '*')
'***************Hello'
>>> 'Hello'.ljust(20, '-')
'Hello---------------'
```

Метод `center()` работает аналогично методам `ljust()` и `rjust()`, но центрирует его, а не выравнивает по правому или левому краю. Введите в интерактивной оболочке следующие команды.

```
>>> 'Hello'.center(20)
'       Hello        '
>>> 'Hello'.center(20, '=')
'=======Hello========'
```

Эти методы особенно полезны в ситуациях, когда необходимо вывести табулированные значения с подходящей разрядкой. Откройте новое окно в файловом редакторе, введите в него следующий код и сохраните его в файле *picnicTable.py*.

```
def printPicnic(itemsDict, leftWidth, rightWidth):
    print('PICNIC ITEMS'.center(leftWidth + rightWidth, '-'))
    for k, v in itemsDict.items():
        print(k.ljust(leftWidth, '.') + str(v).rjust(rightWidth))
picnicItems = {'sandwiches': 4, 'apples': 12, 'cups': 4,
               'cookies': 8000}
printPicnic(picnicItems, 12, 5)
printPicnic(picnicItems, 20, 6)
```

В этой программе мы определяем метод `printPicnic()`, который получает данные в виде словаря и использует методы `center()`, `ljust()` и `rjust()` для отображения этих данных в аккуратном формате в виде выровненной страницы.

Функции `printPicnic()` передается словарь `picnicItems`. В нем содержатся переменные, имеющие следующие значения: `sandwiches` — 4, `apples` — 12, `cups` — 4 и `cookies` — 8000. Мы хотим представить информацию в двух столбцах, причем слева должно отображаться название блюда, а справа — количество.

Для этого нужно решить, какой ширины должны быть левый и правый столбцы. Эти значения будут передаваться методу `printPicnic()` вместе со словарем.

Метод printPicnic() принимает словарь, а также значения ширины для левого (leftWidth) и правого (rightWidth) столбцов. Он выводит заголовок PICNIC ITEMS над таблицей по ее центру. Затем он обрабатывает в цикле элементы словаря, выводя каждую пару "имя-значение" в отдельной строке таблицы, причем ключ выравнивается влево, значение – вправо, а оставшиеся в каждом столбце пустые позиции заполняются пробелами.

Определив метод printPicnic(), мы определяем словарь picnicItems и дважды вызываем метод printPicnic(), передавая ему различные значения ширины левого и правого столбцов.

В процессе выполнения программа дважды выводит данные. В первый раз ширина левой колонки составляет 12 символов, а правой – 5. Во второй раз эти значения составляют соответственно 20 и 6 символов.

```
---PICNIC ITEMS--
sandwiches.. 4
apples...... 12
cups........ 4
cookies..... 8000
-------PICNIC ITEMS-------
sandwiches.......... 4
apples.............. 12
cups................ 4
cookies............. 8000
```

Используя методы rjust(), ljust() и center(), вы можете быть уверены в том, что данные в строках таблицы аккуратно выровнены, даже если точное количество символов, содержащихся в каждой строке, неизвестно.

Удаление пробелов с помощью методов strip(), rstrip() и lstrip()

Иногда возникает необходимость в удалении из строки ее ведущих, замыкающих или и тех, и других пробельных символов (пробелы, символы табуляции и символы новой строки). Метод strip() возвращает новую строку без начальных и конечных пробельных символов. Методы lstrip() и rstrip() удаляют пробельные символы соответственно в начале и конце строки. Введите в интерактивной оболочке следующие команды.

```
>>> spam = '  Hello World  '
>>> spam.strip()
'Hello World'
>>> spam.lstrip()
'Hello World  '
>>> spam.rstrip()
'  Hello World'
```

С помощью необязательного строкового аргумента можно указать, какие символы должны удаляться с обоих концов строки. Введите в интерактивной оболочке следующие команды.

```
>>> spam = 'SpamSpamBaconSpamEggsSpamSpam'
>>> spam.strip('ampS')
'BaconSpamEggs'
```

Передавая методу `strip()` аргумент `'ampS'`, мы сообщаем ему, что в начале и конце строки, сохраненной в переменной `spam`, должны быть удалены все вхождения символов `a`, `m`, `p` и `S`. Порядок указания символов в строке, передаваемой методу `strip()`, несуществен: вызовы `strip('mapS')` или `strip('Spam')` дадут тот же результат, что и вызов `strip('ampS')`.

Копирование и вставка строк с помощью модуля `pyperclip`

В модуле `pyperclip` имеются функции `copy()` и `paste()`, которые могут выполнять операции копирования и вставки текста через буфер обмена вашего компьютера. Пересылка вывода программы в буфер обмена упрощает вставку текста в сообщения электронной почты или документы текстового процессора, а также его передачу другому программному обеспечению.

Модуль `pyperclip` не поставляется вместе с Python. Чтобы его установить, следуйте указаниям по установке модулей сторонних разработчиков, приведенным в приложении А. Установив модуль, введите в интерактивной оболочке следующие команды.

```
>>> import pyperclip
>>> pyperclip.copy('Hello world!')
>>> pyperclip.paste()
'Hello world!'
```

Разумеется, если содержимое буфера будет изменено внешней программой, то именно оно будет возвращено методом `paste()`. *Например, если я скопирую данное предложение в буфер обмена, а затем вызову метод* `paste()`*, то получу следующий вывод:*

```
>>> pyperclip.paste()
'Например, если я скопирую это предложение в буфер обмена, а
затем вызову метод paste(), то получу следующий вывод:'
```

Выполнение сценариев Python вне IDLE

До сих пор вы выполняли свои сценарии на Python с помощью интерактивной оболочки или файлового редактора в IDLE. Однако вряд ли вам понравится открывать IDLE всякий раз, когда потребуется выполнить сценарий. К счастью, существуют более удобные способы, упрощающие запуск сценариев Python. Соответствующие процедуры запуска сценариев для Windows, OS X и Linux немного различаются, но все они описаны по отдельности в приложении Б. Загляните в него, если хотите узнать подробнее об этих способах, а также о том, как передавать сценариям аргументы командной строки. (В IDLE такая возможность отсутствует.)

Проект: парольная защита

Возможно, у вас есть учетные записи на многих веб-сайтах. Использовать для каждой из них один и тот же пароль — плохая привычка, поскольку при наличии брешей в системе безопасности любого из этих сайтов хакерам открывается доступ ко всем вашим учетным записям. Наилучшее решение — использовать менеджер паролей на своем компьютере с одним главным паролем для доступа к нему. Открыв менеджер паролей, вы сможете скопировать нужный пароль в буфер обмена и вставить его в поле **Введите пароль**, предоставляемое при попытке доступа к веб-сайту.

Предлагаемая в этом примере программа менеджера паролей не обеспечивает полной безопасности, но демонстрирует базовые принципы работы программ подобного рода.

Проекты в главах

Это первый из "проектов в главе", которые встречаются в книге. Начиная с этой главы и во всех последующих вам будут предлагаться проекты, демонстрирующие применение на практике понятий, рассмотрению которых была посвящена глава. Особенностью написания всех проектов является то, что каждый из них начинается с "чистого листа" (пустого окна файлового редактора) и заканчивается полностью функциональным вариантом программы. Как и при работе с примерами в интерактивной оболочке, было бы желательно, если бы вы не просто читали эти разделы, а прорабатывали их у себя на компьютере.

Шаг 1. Проектирование программы и структур данных

Мы хотим, чтобы программа принимала аргумент командной строки, которым будет служить имя учетной записи, например учетной записи электронной почты или блога. Пароль для доступа к этой учетной записи будет

копироваться в буфер обмена, откуда пользователь сможет вставить его в поле Password (Пароль). Благодаря этому пользователь не должен держать в памяти длинные пароли, которые надежны, но трудно запоминаются.

Откройте новое окно файлового редактора и сохраните его пустое содержимое в файле *pw.py*. Программа должна начинаться строкой директивы #! (см. приложение Б), за которой следует комментарий, содержащий краткое описание назначения программы. Поскольку вы хотите связывать с именем каждой учетной записи пароль для доступа к ней, целесообразно хранить эти данные в виде строк словаря. Этот словарь и будет той структурой данных, которая обеспечивает организованное хранение имени учетной записи и соответствующего пароля. Введите следующий код.

```
#! python3
# pw.py - Программа для незащищенного хранения паролей.

PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
             'luggage': '12345'}
```

Шаг 2. Обработка аргументов командной строки

Аргументы командной строки будут храниться в переменной sys.argv. (Более подробная информация о том, как использовать аргументы командной строки в программах, приведена в приложении Б.) Первой в списке sys.argv всегда должна быть строка, содержащая имя файла программы ('pw.py'), а вторым — первый из аргументов командной строки. Таким аргументом для нашей программы является имя учетной записи, с которой вы хотите ассоциировать пароль. Поскольку этот аргумент обязательный, вы отображаете сообщение, напоминающее пользователю о необходимости ввода имени учетной записи, если он забыл его ввести (это будет в том случае, если в списке sys.argv содержится менее двух аргументов). Дополните ранее введенный код следующим.

```
#! python3
# pw.py - Программа для незащищенного хранения паролей.

PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
             'luggage': '12345'}
import sys
if len(sys.argv) < 2:
    print('Использование: python pw.py [имя учетной записи] -
↳ копирование пароля учетной записи')
    sys.exit()

account = sys.argv[1] # первый аргумент командной строки — это
                      # имя учетной записи
```

Шаг 3. Копирование пароля

Теперь, когда имя учетной записи сохранено в виде строки в переменной account, вы должны проверить, существует ли оно в словаре PASSWORDS в виде ключа. Если существует, вы копируете значение ключа в буфер обмена, используя вызов pyperclip.copy(). (Поскольку используется модуль pyperclip, его необходимо импортировать.) Заметьте, что на самом деле без переменной account можно было бы обойтись, поскольку везде в программе вместо нее можно было бы использовать выражение sys.argv[1]. Однако лучше использовать переменную account, так как читать программу в этом случае гораздо легче.

Дополните ранее введенный код, как показано ниже.

```
#! python3
# pw.py - Программа для незащищенного хранения паролей.

PASSWORDS = {'email': 'F7minlBDDuvMJuxESSKHFhTxFtjVB6',
             'blog': 'VmALvQyKAxiVH5G8v01if1MLZF3sdt',
             'luggage': '12345'}

import sys, pyperclip
if len(sys.argv) < 2:
    print('Использование: python pw.py [имя учетной записи] -
↳копирование пароля учетной записи')
    sys.exit()

account = sys.argv[1] # первый аргумент командной строки — это
                      # имя учетной записи

if account in PASSWORDS:
    pyperclip.copy(PASSWORDS[account])
    print('Пароль для ' + account + ' скопирован в буфер.')
else:
    print('Учетная запись ' + account + 'отсутствует в списке.')
```

Добавленный код выполняет поиск имени учетной записи в словаре PASSWORDS. Если это имя является ключом в словаре, то мы получаем значение, соответствующее этому ключу, копируем его в буфер обмена и выводим сообщение, которое подтверждает выполнение копирования. В противном случае выводится сообщение о том, что учетная запись с таким именем отсутствует в списке.

Этот сценарий представляет собой полностью завершенную программу. Воспользовавшись приведенными в приложении Б инструкциями по запуску программ из командной строки, вы можете теперь быстро копировать пароли своих учетных записей в буфер обмена. Если вам понадобится обновить программу для изменения или добавления паролей, то достаточно будет внести необходимые изменения в код словаря PASSWORDS.

Разумеется, вряд ли вы захотите хранить все свои пароли в одном месте, откуда любой человек может легко их скопировать. Но можно модифицировать эту программу и использовать ее для быстрого копирования обычного текста в буфер обмена. Предположим, вы отправляете несколько писем электронной почты, в которых содержится много общих абзацев. Вы можете поместить каждый абзац в качестве значения в словарь PASSWORDS (в этом случае вы, вероятно, измените имя словаря), а затем будете иметь возможность быстро выбирать и копировать множество стандартных фрагментов текста в буфер обмена.

Пользователи Windows могут создать пакетный файл (файл с расширением *.bat*) для запуска этой программы из окна **Выполнить**, которое можно легко открыть, нажав комбинацию клавиш <Windows+R>. (Более подробно о пакетных файлах читайте в приложении Б.) Введите в окне файлового редактора и сохраните в файле *pw.bat* в папке *C:\Windows* следующий код.

```
@py.exe C:\Python34\pw.py %*
@pause
```

С использованием только что созданного пакетного файла выполнение программы для хранения паролей сводится к нажатию комбинации клавиш <Windows+R> и вводу команды `pw <имя_учетной_записи>`.

Проект: добавление маркеров в разметку Wiki-документов

Редактируя статьи в Википедии, можно создавать маркированные списки, вводя каждый элемент списка в отдельной строке, которая предваряется маркером в виде звездочки. Предположим, что у вас имеется очень большой список, в который нужно добавить маркеры. Вы могли бы сделать это вручную, вводя символы звездочки в начале каждой строки, и так строка за строкой. Но эту задачу можно легко автоматизировать с помощью короткого сценария Python.

Сценарий *bulletPointAdder.py* будет получать текст из буфера обмена, добавлять звездочку и пробел в начале каждой строки, а затем возвращать этот новый текст в буфер обмена. Например, если бы я скопировал в буфер обмена текст (предназначенный для публикации в Википедии статьи "Список списков списков")

```
Lists of animals
Lists of aquarium life
Lists of biologists by author abbreviation
Lists of cultivars
```

а затем выполнил программу *bulletPointAdder.py*, то в буфере обмена содержался бы следующий текст:

```
* Lists of animals
* Lists of aquarium life
* Lists of biologists by author abbreviation
* Lists of cultivars
```

Этот предваряемый звездочками текст полностью подготовлен к вставке в статью Википедии в качестве маркированного списка.

Шаг 1. Копирование и вставка посредством буфера обмена

Вы хотите, чтобы программа *bulletPointAdder.py* делала следующее:

1) вставляла текст из буфера обмена;
2) выполняла над ним некоторые действия;
3) копировала новый текст в буфер обмена.

С пунктом 2 придется немного повозиться, а вот пункты 1 и 3 достаточно просты: они требуют использования всего лишь двух вызовов — `pyperclip.copy()` и `pyperclip.paste()`. Поэтому на данном этапе мы напишем только ту часть программы, которая реализует шаги 1 и 3. Введите в файловом редакторе приведенный ниже код и сохраните его в файле *bulletPointAdder.py*.

```
#! python3
# bulletPointAdder.py - Добавляет маркеры Википедии в начало
# каждой строки текста, сохраненного в буфере обмена.

import pyperclip
text = pyperclip.paste()

# TODO: разделить строки и добавить звездочки.

pyperclip.copy(text)
```

Комментарий `TODO` (ЧТО СДЕЛАТЬ) — напоминание о том, что эту часть программы еще предстоит написать. Следующий шаг — фактическая реализация данной части программы.

Шаг 2. Разбивка текста на строки и добавление звездочек

Вызов `pyperclip.paste()` возвращает весь текст из буфера в виде одной большой строки. Если бы мы использовали пример со статьей "Список списков", то строка, сохраненная в виде текста, выглядела бы так:

```
'Lists of animals\nLists of aquarium life\nLists of biologists by
author abbreviation\nLists of cultivars'
```

Наличие в этой строке символов новой строки \n приведет к тому, что она будет отображаться в виде многострочного текста при выводе на экран или вставке из буфера обмена. В этом одном строковом значении содержится много "строк". Вам нужно добавить звездочку в начале каждой из этих строк.

Можно было бы написать код, который выполняет поиск всех символов \n и вставляет после каждого из них звездочку. Однако гораздо проще использовать метод split() для возврата списка строк, по одной для каждой строки оригинального текста, а затем добавить звездочку в начале каждой строки, входящей в список. Придайте программе следующий вид.

```
#! python3
# bulletPointAdder.py - Добавляет маркеры Википедии в начало
# каждой строки текста, сохраненного в буфере обмена.

import pyperclip
text = pyperclip.paste()

# Разделяет строки и добавляет звездочки.
lines = text.split('\n')
for i in range(len(lines)):       # цикл по списку "lines"
    lines[i] = '* ' + lines[i]    # добавляет звездочку в каждую
                                  # строку в списке "lines"

pyperclip.copy(text)
```

Выполняя разбиение текста по символам новой строки, мы получаем список, в котором каждый элемент списка располагается в отдельной строке текста. Мы сохраняем этот список в строках, а затем проходим в цикле по всем элементам в этих строках. В начале каждой строки мы добавляем звездочку и пробел. Теперь каждая текстовая строка начинается со звездочки.

Шаг 3. Объединение измененных строк

Теперь строки списка содержат измененные строки, начинающиеся со звездочек. Но метод pyperclip.copy() ожидает значение в виде одиночной строки, а не списка строковых значений. Чтобы создать это одиночное строковое значение, передайте строки методу join() для их объединения в одну строку. Дополните программный код, как показано ниже.

```
#! python3
# bulletPointAdder.py - Добавляет маркеры Википедии в начало
# каждой строки текста, сохраненного в буфере обмена.

import pyperclip
text = pyperclip.paste()
```

```
# Разделяет строки и добавляет звездочки.
lines = text.split('\n')
for i in range(len(lines)):      # цикл по списку "lines"
    lines[i] = '* ' + lines[i]   # добавляет звездочку в каждую
                                 # строку в списке "lines"
text = '\n'.join(lines)
pyperclip.copy(text)
```

В процессе выполнения программа заменяет текст из буфера обмена текстом, в начале каждой строки которого располагаются звездочки. Этот код представляет завершенный вариант программы, и вы можете попытаться выполнить ее в отношении текста, скопированного в буфер обмена.

Даже если вы не испытываете потребности в автоматизации этой довольно специфической задачи, вам могут понадобиться другие виды манипуляций с текстом, такие как удаление замыкающих пробелов в конце строк или преобразование текста в верхний или нижний регистр. Какой бы ни была специфика ваших запросов, вы можете использовать буфер обмена для выполнения операций ввода и вывода.

Резюме

Текст — распространенная форма представления данных, и Python поставляется с множеством полезных методов, предназначенных для обработки текста, сохраненного в строковых значениях. Вы можете использовать индексирование, извлечение срезов и строковые методы практически в любой программе на Python, которую будете писать.

Программы, которые вы сейчас пишете, кажутся не слишком сложными: они не имеют графического интерфейса пользователя с красочными изображениями и цветным текстом. Пока что вы отображаете текст с помощью метода `print()` и предоставляете пользователю возможность вводить текст, используя метод `input()`. Однако пользователь может ускорить ввод больших объемов текста посредством буфера обмена. Эта возможность открывает широкий простор для написания программ, манипулирующих большими объемами текстовых данных. Пусть даже эти программы не содержат окон или графики, но они могут быстро выполнять массу полезной работы.

Другой способ манипулирования большими объемами текста — это чтение и запись текста путем непосредственного обмена данными с жестким диском. О том, как это делается с помощью Python, вы узнаете из следующей главы.

Контрольные вопросы

1. Что такое экранированные символы?
2. Что представляют собой экранированные символы `\n` и `\t`?
3. Как добавить символ обратной косой черты (`\`) в строку?
4. Строковое значение `"Howl's Moving Castle"` — это допустимая строка. Почему она не вызовет ошибку, несмотря на наличие неэкранированного символа апострофа в слове `Howl's`?
5. Если вы не хотите вставлять символ `\n` в свою строку, то как вы напишете строку, содержащую символы новой строки?
6. Каковы будут результаты вычисления приведенных ниже выражений?
   - `'Hello world!'[1]`
   - `'Hello world!'[0:5]`
   - `'Hello world!'[:5]`
   - `'Hello world!'[3:]`
7. Каковы будут результаты вычисления приведенных ниже выражений?
   - `'Hello'.upper()`
   - `'Hello'.upper().isupper()`
   - `'Hello'.upper().lower()`
8. Каковы будут результаты вычисления приведенных ниже выражений?
   - `'Remember, remember, the fifth of November.'.split()`
   - `'-'.join('There can be only one.'.split())`
9. Какие методы используются для выравнивания строки по левому краю, правому краю и центру?
10. Как удалить пробельные символы в начале и конце строки?

Учебный проект

Чтобы закрепить полученные знания на практике, напишите программу для предложенной ниже задачи.

Табличный вывод данных

Напишите функцию `printTable()`, которая принимает список списков строк и отображает его в виде аккуратной таблицы с выравниванием текста по правому краю в каждом столбце. Исходите из предположения, что внутренние списки будут содержать одно и то же количество строк. Например, список мог бы выглядеть так.

```
tableData = [['apples', 'oranges', 'cherries', 'banana'],
             ['Alice', 'Bob', 'Carol', 'David'],
             ['dogs', 'cats', 'moose', 'goose']]
Manipulating Strings 143
```

Вывод функции `printTable()` будет примерно таким.

```
  apples Alice  dogs
 oranges   Bob  cats
cherries Carol moose
  banana David goose
```

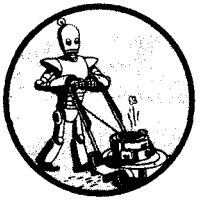
Подсказка. Ваш код должен в первую очередь найти самую длинную строку в каждом из внутренних списков; ширина столбца должна быть достаточно большой для того, чтобы в нем уместилась любая строка. Значения максимальной ширины столбцов могут храниться в виде списка целых чисел. Код функции `printTable()` может начинаться с инструкции `colWidths = [0] * len(tableData)`, которая создает список, содержащий значения `0` в количестве, равном количеству внутренних списков в списке `tableData`. Таким образом, в элементе `colWidths[0]` может храниться ширина самой длинной строки, содержащейся в `tableData[0]`, в элементе `colWidths[1]` — ширина самой длинной строки, содержащейся в `tableData[1]` и т.д. Затем вы можете найти самое большое значение в списке `colWidths`, чтобы определить, какое целочисленное значение ширины следует передать строковому методу `rjust()`.

ЧАСТЬ II

АВТОМАТИЗАЦИЯ ЗАДАЧ

7

ПОИСК ПО ШАБЛОНУ С ПОМОЩЬЮ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ



Вероятно, вам не раз приходилось выполнять текстовый поиск, вводя слова, которые нужно найти, и нажимая клавиши <Ctrl+F>. *Регулярные выражения* представляют собой следующий шаг в этом направлении: они позволяют задавать *шаблон* искомого текста. Вы можете не знать телефонный номер компании, но если вы живете в США или Канаде, то вам известно, что он будет включать три цифры, за которыми следуют дефис и еще четыре цифры (в самом начале номера может указываться необязательный территориальный код). Зная этот шаблон, можно сразу же предположить, что цифры `415-555-1234` означают телефонный номер, а цифры `4,155,551,234` таковым не являются.

Как ни полезны регулярные выражения, лишь немногие из тех, кто не является программистом, знают, что это такое, хотя большинство современных текстовых процессоров, таких как Microsoft Word или OpenOffice, предлагают средства поиска и замены, основанные на использовании регулярных выражений. Регулярные выражения сэкономят массу времени не только пользователям, но и программистам. Более того, по мнению канадского писателя Кори Доктороу, изучение регулярных выражений должно предшествовать изучению самого программирования:

> "Знание [регулярных выражений] может означать разницу между решением задачи за 3 действия или за 3000 действий. Когда ты дока в программировании, то легко забываешь о том, что на задачу, решение которой стоит тебе нажатия какой-то пары клавиш, другие могут потратить несколько дней напряженной работы, чреватой внесением дополнительных ошибок"[1].

---

[1] Cory Doctorow, "Here's what ICT should really teach kids: how to do regular expressions," Guardian, December 4, 2012, `http://www.theguardian.com/technology/2012/dec/04/ict-teach-kids -regular-expressions/`. — *Примеч. ред.*

В этой главе вы сначала напишете программу для нахождения образцов текста без использования регулярных выражений, а затем увидите, как регулярные выражения позволяют сделать то же самое с использованием гораздо более ясного кода меньшего объема. Я продемонстрирую вам, как работают простые шаблоны на основе регулярных выражений, после чего мы займемся более сложными операциями, такими как замена строк и создание собственных символьных классов. Наконец, в завершение главы вы напишете программу для автоматического извлечения телефонных номеров и адресов электронной почты из блока текста.

Поиск образцов текста без использования регулярных выражений

Предположим, вы хотите найти телефонный номер, содержащийся в строке. Исходный текстовый шаблон вам известен: три цифры, дефис, три цифры, дефис, четыре цифры, например `415-555-4242`.

Для проверки соответствия строки данному образцу мы будем использовать функцию `isPhoneNumber()`, возвращающую одно из двух возможных значений: `True` или `False`. Откройте в файловом редакторе новое окно, введите в нем приведенный ниже код и сохраните его в файле *isPhoneNumber.py*.

```
  def isPhoneNumber(text):
❶     if len(text) != 12:
          return False
      for i in range(0, 3):
❷         if not text[i].isdecimal():
              return False
❸     if text[3] != '-':
          return False
      for i in range(4, 7):
❹         if not text[i].isdecimal():
              return False
❺     if text[7] != '-':
          return False
      for i in range(8, 12):
❻         if not text[i].isdecimal():
              return False
❼     return True
  print('415-555-4242 - это телефонный номер:')
  print(isPhoneNumber('415-555-4242'))
  print('Moshi moshi - это телефонный номер:')
  print(isPhoneNumber('Moshi moshi'))
```

Выполнив эту программу, вы должны получить следующий вывод.

```
415-555-4242 - это телефонный номер:
True
Moshi moshi - это телефонный номер:
False
```

Код функции `isPhoneNumber()` выполняет несколько видов тестов, цель которых — проверить, представляет ли содержащаяся в переменной `text` строка телефонный номер в корректном формате. При отрицательном исходе любой из проверок функция возвращает значение `False`. Сначала проверяется, содержит ли строка в точности 12 символов ❶. Затем выполняется проверка того, что территориальный телефонный код (т.е. первые три символа текста) состоит только из цифр ❷. Остальная часть кода функции проверяет соответствие строки формату шаблона телефонного номера: за территориальным кодом следуют первый дефис ❸, еще три цифры ❹, еще один дефис ❺ и, наконец, еще четыре цифры ❻. Если программе удается выполнить все виды проверки, то возвращается значение `True` ❼.

При вызове функции `isPhoneNumber()` с аргументом `'415-555-4242'` возвращается значение `True`. Вызов функции `isPhoneNumber()` с аргументом `'Moshi moshi'` возвратит значение `False`; при этом не будет пройден уже первый тест, поскольку количество символов в строке `'Moshi moshi'` отличается от 12.

Для нахождения приведенного выше текстового образца в строке большего размера вам потребуется написать еще больше кода. Замените последние четыре вызова функции `print()` в файле *isPhoneNumber.py* следующим кодом.

```
message = 'Позвони мне завтра по номеру 415-555-1011.
415-555-9999 - это телефонный номер моего офиса.'
for i in range(len(message)):
❶     chunk = message[i:i+12]
❷     if isPhoneNumber(chunk):
          print('Найденный телефонный номер: ' + chunk)
print('Готово')
```

Выполнив эту программу, вы должны получить следующий вывод.

```
Найденный телефонный номер: 415-555-1011
Найденный телефонный номер: 415-555-9999
Готово
```

На каждой итерации цикла `for` переменной `chunk` присваиваются очередные 12 последовательных символов строки `message` ❶. Например, на первой итерации `i` равно `0`, и переменной `chunk` присваивается срез `message[0:12]`

(т.е. строка 'Позвони мне '). На следующей итерации i равно 1, и переменной chunk присваивается срез message[1:13] (т.е. строка 'озвони мне з').

Вы передаете очередную порцию строки функции isPhoneNumber(), чтобы проверить, соответствует ли она образцу телефонного номера ❷, и если это действительно так, то выводите эту часть строки на экран.

Циклический просмотр строки message продолжается, и в конечном счете будут обнаружены 12 символов, представляющих телефонный номер. В ходе выполнения всего цикла просматривается вся строка, причем каждый раз тестируются части строки длиной 12 символов и выводятся те из них, которые удовлетворяют условиям, проверяемым в функции isPhoneNumber(). По окончании анализа всей строки message будет выведено слово Готово.

В то время как содержащаяся в данном примере в переменной message строка очень короткая, в других случаях она может содержать миллионы символов, и тем не менее программа выполнится менее чем за секунду. Аналогичная программа, выполняющая поиск телефонных номеров с помощью регулярных выражений, также будет выполняться менее секунды, однако регулярные выражения позволяют тратить на написание подобных программ гораздо меньше времени.

Поиск образцов текста с помощью регулярных выражений

Предыдущая программа для нахождения телефонных номеров вполне функциональна, однако, учитывая ее относительно большой размер, предлагает лишь ограниченные возможности: функция isPhoneNumber() включает 17 строк кода, но способна находить телефонные номера, соответствующие только одному виду шаблона. Что если телефонный номер будет записан в строке в формате 415.555.4242 или (415) 555-4242? Что если телефонный номер включает дополнительные цифры и выглядит, например, так: 415-555-4242 x99? Функция isPhoneNumber() не в состоянии идентифицировать подобные номера. Чтобы учесть другие допустимые шаблоны, вы могли бы написать дополнительный код, однако существует гораздо более простой способ.

Компактное описание текстовых шаблонов возможно с помощью регулярных выражений. Например, регулярному выражению \d соответствует любой цифровой символ, т.е. любая одиночная цифра от 0 до 9. Регулярное выражение \d\d\d-\d\d\d-\d\d\d\d используется в Python для нахождения текстовых строк того же формата, что и в случае функции isPhoneNumber(): строка из трех цифр, дефис, еще три цифры, другой дефис и еще четыре цифры. Никакая другая строка не будет соответствовать регулярному выражению \d\d\d-\d\d\d-\d\d \d\d.

Вместе с тем регулярные выражения могут быть гораздо более сложными. Например, указав 3 в фигурных скобках ({3}) после шаблона, мы сообщаем следующее: "Искать троекратное соответствие этому шаблону". Поэтому корректному телефонному номеру будет соответствовать также следующий шаблон: \d{3}-\d{3}-\d{4}.

Создание объектов Regex

В Python все функции, предназначенные для работы с регулярными выражениями, содержатся в модуле re. Введите в интерактивной оболочке следующую команду для того, чтобы импортировать этот модуль:

```
>>> import re
```

Примечание

Для выполнения большинства последующих примеров главы вам потребуется модуль re, *поэтому не забывайте импортировать его в начале любого сценария, который напишете, или при каждом перезапуске IDLE. В противном случае вы получите сообщение об ошибке* NameError, *гласящее о том, что имя* 're' *не определено.*

Вызов метода re.compile() с передачей ему строкового значения, представляющего регулярное выражение, возвращает объект соответствующего регулярного выражения (или, как их принято называть, объект Regex).

Чтобы создать объект Regex, совпадающий с шаблоном телефонного номера, введите в интерактивной оболочке следующую команду (вспомните,

Передача "сырых" строк методу re.compile()

Вспомните, что в Python для экранирования символов используется символ обратной косой черты (\). Строковое значение '\n' представляет одиночный символ новой строки, а не символ обратной косой черты, за которым следует строчная буква n. Чтобы вывести символ обратной косой черты, вам потребуется его экранировать (\\). Таким образом, строка '\\n' представляет символ обратной косой черты, за которым следует строчная буква n. Однако, поместив символ r перед первым апострофом строкового значения, вы можете указать, что это "сырая" строка, в которой символы не экранированы.

Поскольку символы обратной косой черты часто используются в регулярных выражениях, удобно передавать методу re.compile() "сырые" строки, а не вводить дополнительные символы обратной косой черты. Ввести r'\d\d\d-\d\d\d-\d\d\d\d' намного проще, чем '\\d\\d\\d-\\d\\d\\d-\\d\\d\\d\\d'.

что выражение \d означает "цифровой символ", а выражение \d\d\d-\d\d\d-\d\d\d\d — регулярное выражение, соответствующее шаблону корректного телефонного номера):

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
```

Теперь в переменной phoneNumRegex содержится объект Regex.

Поиск соответствий объектам Regex

Метод search() объекта Regex ищет в переданной ему строке любые совпадения с регулярным выражением. В случае отсутствия в строке совпадений с регулярным выражением он возвращает значение None. Если совпадения обнаружены, то метод search() возвращает объект Match. Объекты Match имеют метод group(), который возвращает найденные соответствия шаблону. (О том, что такое группы, будет сказано ниже.) В качестве примера введите в интерактивной оболочке следующие команды.

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('Мой номер: 415-555-4242.')
>>> print('Найденный телефонный номер: ' + mo.group())
Найденный телефонный номер: 415-555-4242
```

Здесь mo — это не более чем произвольное имя, выбранное в качестве типового имени переменной для работы с объектами Match. Возможно, этот пример поначалу покажется вам слишком сложным, но он значительно короче представленной ранее программы *isPhoneNumber.py*, хотя делает то же самое.

В этом примере мы передаем желаемый шаблон методу re.compile() и сохраняем результирующий объект Regex в переменной phoneNumRegex. Затем мы вызываем метод search() для переменной phoneNumRegex и передаем ему строку, в которой хотим найти соответствия шаблону. Результат поиска сохраняется в переменной mo. В данном примере нам известно, что искомый образец будет найден в строке, и поэтому мы знаем, что метод search() вернет объект Match. Зная, что переменная mo содержит объект Match, а не нулевое значение None, мы можем вызвать для нее метод group(), возвращающий найденное соответствие. Передача функции print() выражения mo.group() обеспечивает вывод полученного соответствия шаблону, т.е. отображение телефонного номера 415-555-4242.

Пошаговая процедура поиска соответствий регулярному выражению

Процедура использования регулярных выражений включает несколько шагов, каждый из которых довольно прост.

1. Импортируйте модуль `regex` с помощью инструкции `import re`.
2. Создайте объект `Regex` с помощью функции `re.compile()`. (Не забывайте о том, что необходимо использовать "сырую" строку.)
3. Передайте строку, в которой хотите выполнить поиск, методу `search()` объекта `Regex`. Этот метод возвращает объект `Match`.
4. Вызовите метод `group()` объекта `Match`, который вернет строку, содержащую фактический найденный текст, соответствующий данному регулярному выражению.

Примечание

Всячески поощряя самостоятельное выполнение вами кода примеров в интерактивной оболочке, я рекомендую наряду с этим использовать предлагаемые в Интернете тестеры регулярных выражений, которые дают точную картину соответствия регулярных выражений фрагментам введенного вами текста. Лично я порекомендовал бы вам воспользоваться тестером, который предлагается на сайте `http://regexpal.com/`.

Другие возможные шаблоны регулярных выражений

Теперь, когда основные шаги по созданию и нахождению объектов регулярных выражений с помощью Python вам уже известны, вы вполне готовы испытать более мощные возможности поиска по шаблонам.

Создание групп с помощью круглых скобок

Предположим, вы хотите отделить территориальный код от остальной части телефонного номера. Добавление круглых скобок приводит к созданию групп в регулярных выражениях: `(\d\d\d)-(\d\d\d-\d\d\d\d)`. После этого вы можете использовать метод `group()` объекта совпадения для захвата совпавшего текста, соответствующего только одной группе.

Первый набор круглых скобок в строке регулярного выражения будет группой `1`. Второй набор будет группой `2`. Передавая целые числа 1 или 2 методу `group()` объекта совпадения, вы можете избирательно захватывать различные части совпавшего текста. Если методу `group()` передается `0` или вообще ничего не передается, то он возвращает весь найденный текст, соответствующий шаблону. Введите в интерактивной оболочке следующие команды.

```
>>> phoneNumRegex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('Мой номер: 415-555-4242.')
>>> mo.group(1)
'415'
>>> mo.group(2)
'555-4242'
>>> mo.group(0)
'415-555-4242'
>>> mo.group()
'415-555-4242'
```

Если вам нужно извлечь сразу все группы, используйте метод `groups()` (обратите внимание на показатель множественного числа в имени метода — букву `s`).

```
>>> mo.groups()
('415', '555-4242')
>>> areaCode, mainNumber = mo.groups()
>>> print(areaCode)
415
>>> print(mainNumber)
555-4242
```

Поскольку метод `mo.groups()` возвращает кортеж, состоящий из нескольких значений, вы можете использовать трюк с групповым присваиванием, когда каждое значение присваивается отдельной переменной, как в строке `areaCode, mainNumber = mo.groups()` приведенного выше кода.

Круглые скобки имеют в регулярных выражениях особый смысл, но что если вам нужно находить в тексте сами скобки? Ведь, например, в телефонных номерах круглые скобки часто используются для выделения территориального кода. В подобных случаях символу `(` или `)` должен предшествовать символ обратной косой черты. Введите в интерактивной оболочке следующие команды.

```
>>> phoneNumRegex = re.compile(r'(\(\d\d\d\)) (\d\d\d-\d\d\d\d)')
>>> mo = phoneNumRegex.search('Мой номер:
↳ (415) 555-4242.')
>>> mo.group(1)
'(415)'
>>> mo.group(2)
'555-4242'
```

Экранированные символы `\(` и `\)` в "сырой" строке, передаваемой методу `re.compile()`, означают соответствие фактическим символам круглых скобок.

Выбор альтернативных групп с помощью канала

Символ | называется *каналом*. Вы можете использовать его всегда, когда хотите находить соответствие одному из нескольких альтернативных выражений. Например, регулярному выражению `r'Batman|Tina Fey'` будут соответствовать как строка `'Batman'`, так и строка `'Tina Fey'`.

Если в исследуемой строке содержатся *обе* эти строки, то в объекте `Match` будет возвращена та из них, которая встретится первой. Введите в интерактивной оболочке следующие команды.

```
>>> heroRegex = re.compile (r'Batman|Tina Fey')
>>> mo1 = heroRegex.search('Batman and Tina Fey.')
>>> mo1.group()
'Batman'

>>> mo2 = heroRegex.search('Tina Fey and Batman.')
>>> mo2.group()
'Tina Fey'
```

Примечание

Для поиска всех совпадений с шаблоном можно использовать метод `findall()`, который обсуждается в разделе "Метод `findall()`".

. Для выбора альтернативных вариантов при поиске совпадений можно использовать канал. Предположим, например, что вы заинтересованы в поиске соответствий любой из строк `'Batman'`, `'Batmobile'`, `'Batcopter'` и `'Batbat'`. Поскольку в начале каждой из них содержатся символы `Bat`, было бы неплохо иметь возможность задать этот префикс только один раз. Это можно сделать с помощью круглых скобок. Введите в интерактивной оболочке следующие команды.

```
>>> batRegex = re.compile(r'Bat(man|mobile|copter|bat)')
>>> mo = batRegex.search('Batmobile потерял колесо')
>>> mo.group()
'Batmobile'
>>> mo.group(1)
'mobile'
```

Вызов `mo.group()` возвращает весь совпавший с шаблоном текст, т.е. строку `'Batmobile'`, тогда как вызов `mo.group(1)` возвращает лишь часть совпавшего текста, соответствующую первой группе в круглых скобках, т.е. `'mobile'`. Используя символ канала и группирование с помощью круглых скобок, вы можете указать несколько альтернативных шаблонов для поиска соответствий с помощью регулярного выражения.

Если требуется поиск соответствий самому символу канала, то в регулярном выражении ему должен предшествовать символ обратной косой черты: \|.

Указание необязательной группы символов с помощью вопросительного знака

Иногда возникает необходимость в шаблонах, содержащих необязательные символы или группы символов. Это означает, что регулярное выражение должно найти соответствие, независимо от того, содержит ли оно некоторый фрагмент текста. Символ ? указывает на то, что предшествующая ему группа является необязательной частью поискового шаблона. Введите в интерактивной оболочке следующие команды.

```
>>> batRegex = re.compile(r'Bat(wo)?man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'

>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'
```

Часть (wo)? регулярного выражения означает, что шаблон wo является необязательной группой. Регулярному выражению будет соответствовать текст, в котором подстрока wo либо вообще не встречается, либо встречается только один раз. Именно поэтому данному регулярному выражению соответствует как слово 'Batwoman', так и слово 'Batman'.

Используя ранее приведенный пример телефонного номера, вы можете составить регулярное выражение, находящее номера, которые могут содержать, а могут и не содержать территориальный код. Введите в интерактивной оболочке следующие команды.

```
>>> phoneRegex = re.compile(r'(\d\d\d-)?\d\d\d-\d\d\d\d')
>>> mo1 = phoneRegex.search('Мой номер: 415-555-4242')
>>> mo1.group()
'415-555-4242'
>>> mo2 = phoneRegex.search('Мой номер: 555-4242')
>>> mo2.group()
'555-4242'
```

Считайте, что символ ? имеет следующий смысл: "Искать соответствие тексту, в котором группа, предшествующая вопросительному знаку, встречается нуль или один раз".

Если вам необходимо находить соответствие самому вопросительному знаку, то в регулярном выражении ему должен предшествовать символ обратной косой черты: \?.

Указание соответствия группе символов, повторяющейся нуль или несколько раз, с помощью звездочки

Символ * ("звездочка") означает "совпадение с нулевым или большим количеством экземпляров", т.е. группа, предшествующая звездочке, может встречаться в тексте любое количество раз подряд. Она может либо вообще отсутствовать, либо повторяться снова и снова. Вновь вернемся к примеру с Batman.

```
>>> batRegex = re.compile(r'Bat(wo)*man')
>>> mo1 = batRegex.search('The Adventures of Batman')
>>> mo1.group()
'Batman'

>>> mo2 = batRegex.search('The Adventures of Batwoman')
>>> mo2.group()
'Batwoman'

>>> mo3 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo3.group()
'Batwowowowoman'
```

В случае слова 'Batman' часть (wo)* регулярного выражения соответствует нулевому количеству (т.е. отсутствию) экземпляров группы wo в строке; в случае слова 'Batwoman' часть (wo)* совпадает с одним экземпляром wo; а в случае слова 'Batwowowowoman' часть (wo)* совпадает с четырьмя экземплярами группы wo.

Если вам необходимо находить соответствие самому символу звездочки, то в регулярном выражении ему должен предшествовать символ обратной косой черты: *.

Указание соответствия одному или нескольким повторениям группы с помощью плюса

Если символ * в регулярном выражении означает "совпадение с нулевым или большим количеством экземпляров", то символ + ("плюс") означает "совпадение с одиночным или большим количеством экземпляров". В отличие от символа звездочки, который не требует появления предшествующей группы в исследуемой строке, группа, предшествующая плюсу, должна появиться в строке хотя бы один раз. Такая группа не является необязательной.

Введите в интерактивной оболочке следующие команды и сравните полученные результаты с результатами из предыдущего раздела, относящимися к регулярным выражениям со звездочкой.

```
>>> batRegex = re.compile(r'Bat(wo)+man')
>>> mo1 = batRegex.search('The Adventures of Batwoman')
>>> mo1.group()
'Batwoman'

>>> mo2 = batRegex.search('The Adventures of Batwowowowoman')
>>> mo2.group()
'Batwowowowoman'

>>> mo3 = batRegex.search('The Adventures of Batman')
>>> mo3 == None
True
```

Регулярное выражение `Bat(wo)+man` не найдет соответствия в строке `'The Adventures of Batman'`, поскольку плюс требует наличия по крайней мере одного экземпляра подстроки `wo`.

Если вам необходимо находить соответствие самому символу плюса, то в регулярном выражении ему должен предшествовать символ обратной косой черты: `\+`.

Указание соответствия определенному количеству повторений группы с помощью фигурных скобок

Если у вас имеется группа, которую вы хотите повторить определенное количество раз, укажите за ней число необходимых повторений в фигурных скобках в своем регулярном выражении. Например, регулярному выражению `(Ha){3}` будет соответствовать строка `'HaHaHa'`, но не строка `'HaHa'`, поскольку в последнем случае группа `(Ha)` повторяется всего лишь два раза.

Вместо одного числа вы можете указать диапазон, записав в фигурных скобках значения минимального и максимального числа допустимых повторений, разделенные запятой. Например, регулярному выражению `(Ha){3,5}` будут соответствовать строки `'HaHaHa'`, `'HaHaHaHa'` и `'HaHaHaHaHa'`.

Как первое, так и второе из этих чисел в фигурных скобках могут опускаться, оставляя неограниченными минимальное и максимальное количества повторений. Например, регулярному выражению `(Ha){3,}` будут соответствовать три и более экземпляров группы `(Ha)`, тогда как регулярному выражению `(Ha){,5}` будут соответствовать от нуля до пяти экземпляров. Фигурные скобки позволяют записывать регулярные выражения в более компактном виде. Следующим двум регулярным выражениям соответствуют идентичные шаблоны.

```
(Ha){3}
(Ha)(Ha)(Ha)
```

Этим двум регулярным выражениям также соответствуют идентичные шаблоны.

```
(Ha){3,5}
((Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha))|((Ha)(Ha)(Ha)(Ha)(Ha))
```

Введите в интерактивной оболочке следующие команды.

```
>>> haRegex = re.compile(r'(Ha){3}')
>>> mo1 = haRegex.search('HaHaHa')
>>> mo1.group()
'HaHaHa'

>>> mo2 = haRegex.search('Ha')
>>> mo2 == None
True
```

Здесь регулярное выражение `(Ha){3}` совпадает со строкой `'HaHaHa'`, но не со строкой `'Ha'`. Поскольку оно не соответствует строке `'Ha'`, метод `search()` возвращает значение `None`.

Жадный и нежадный виды поиска

Поскольку регулярному выражению `(Ha){3,5}` могут соответствовать три, четыре или пять вхождений `Ha` в строке `'HaHaHaHaHa'`, вас может удивлять, почему вызов объектом `Match` метода `group()` в предыдущем примере с фигурными скобками возвращает строку `'HaHaHaHaHa'`, а не ее более короткие подстроки. В конце концов, строки `'HaHaHa'` и `'HaHaHaHa'` также представляют собой действительные соответствия регулярному выражению `(Ha){3,5}`.

Регулярные выражения Python по умолчанию являются *жадными* в том смысле, что в неоднозначных ситуациях они будут пытаться соответствовать как можно более длинной строке. *Нежадная* версия фигурных скобок, которая пытается соответствовать самой короткой из возможных строк, характеризуется наличием вопросительного знака сразу за закрывающей фигурной скобкой.

Введите в интерактивной оболочке следующие команды и обратите внимание на различия между жадной и нежадной формами фигурных скобок при выполнении поиска в одной и той же тестируемой строке.

```
>>> greedyHaRegex = re.compile(r'(Ha){3,5}')
>>> mo1 = greedyHaRegex.search('HaHaHaHaHa')
>>> mo1.group()
'HaHaHaHaHa'

>>> nongreedyHaRegex = re.compile(r'(Ha){3,5}?')
>>> mo2 = nongreedyHaRegex.search('HaHaHaHaHa')
>>> mo2.group()
'HaHaHa'
```

Обратите внимание на то, что в регулярных выражениях вопросительный знак может использоваться в двояком смысле: указывать на нежадный поиск и обозначать необязательную группу. Эти две его функции никак между собой не связаны.

Метод `findall()`

В дополнение к методу `search()` объекты `Regex` имеют метод `findall()`. Если метод `search()` возвращает объект `Match` первого совпадения, найденного в тестируемой строке, то метод `findall()` возвращает строки каждого из найденных совпадений. Чтобы увидеть, как метод `search()` возвращает лишь объект `Match` для первого найденного вхождения совпадающего текста, введите в интерактивной оболочке следующие команды.

```
>>> phoneNumRegex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')
>>> mo = phoneNumRegex.search('Сотовый: 415-555-9999
↳ Рабочий: 212-555-0000')
>>> mo.group()
'415-555-9999'
```

С другой стороны, метод `findall()` возвратит не объект `Match`, а список строк, *коль скоро в регулярном выражении отсутствуют группы*. Введите в интерактивной оболочке следующие команды.

```
>>> phoneNumRegex =
↳ re.compile(r'\d\d\d-\d\d\d-\d\d\d\d') # не имеет групп
>>> phoneNumRegex.findall('Сотовый: 415-555-9999
↳ Рабочий: 212-555-0000')
['415-555-9999', '212-555-0000']
```

В случае же *наличия* групп в регулярном выражении метод `findall()` возвратит список кортежей. Каждый кортеж представляет найденное соответствие, а его элементы — совпавшие строки для каждой группы в регулярном выражении. Чтобы увидеть, как работает метод `findall()`, введите в интерактивной оболочке следующие команды (обратите внимание на то, что

теперь в компилируемом регулярном выражении имеются группы, заключенные в круглые скобки).

```
>>> phoneNumRegex =
↳ re.compile(r'(\d\d\d)-(\d\d\d)-(\d\d\d\d)') #имеет группы
>>> phoneNumRegex.findall('Сотовый: 415-555-9999
↳ Рабочий: 212-555-0000')
[('415', '555', '1122'), ('212', '555', '0000')]
```

Подведем краткое резюме работы метода `findall()`.

1. Будучи вызванным для регулярного выражения, не содержащего групп, например `\d\d\d-\d\d\d-\d\d\d\d`, метод `findall()` возвращает список совпавших строк, такой как `['415-555- 9999', '212-555-0000']`.
2. Будучи вызванным для регулярного выражения, содержащего группы, например `(\d\d\d)-(\d\d\d)-(\d\ d\d\d)`, метод `findall()` возвращает список кортежей строк (по одной строке для каждой группы), такой как `[('415', '555', '1122'), ('212', '555', '0000')]`.

Символьные классы

Из предыдущих примеров вам уже известно, что сокращение `\d` может представлять любую цифру. Таким образом, `\d` — это сокращенное обозначение регулярного выражения `(0|1|2|3|4|5|6|7|8|9)`. Аналогичные сокращения существуют для многих других символьных классов (табл. 7.1).

Таблица 7.1. Сокращенные обозначения распространенных символьных классов

Сокращение	Представляемые символы
`\d`	Любая цифра в диапазоне от 0 до 9
`\D`	Любой символ, не являющийся цифрой в диапазоне от 0 до 9
`\w`	Любая буква, цифра или символ подчеркивания (так называемые *словарные* символы)
`\W`	Любой символ, не являющийся буквой, цифрой или символом подчеркивания
`\s`	Символ пробела, табуляции или новой строки (так называемые *пробельные* символы)
`\S`	Любой символ, не являющийся символом пробела, табуляции или новой строки

Символьные классы (классы символов) удобно использовать для компактной записи регулярных выражений. Классу символов `[0-5]` будет соответствовать любая одиночная цифра в диапазоне от `0` до `5`; такая запись гораздо короче, чем запись `(0|1|2|3|4|5)`.

Введите в интерактивной оболочке следующие команды.

```
>>> xmasRegex = re.compile(r'\d+\s\w+')
>>> xmasRegex.findall('12 drummers, 11 pipers, 10 lords, 9 ladies,
```

```
↪8 maids, 7 swans, 6 geese, 5 rings, 4 birds, 3 hens, 2 doves,
↪1 partridge')
['12 drummers', '11 pipers', '10 lords', '9 ladies', '8 maids',
'7 swans', '6 geese', '5 rings', '4 birds', '3 hens', '2 doves',
'1 partridge']
```

Регулярному выражению `\d+\s\w+` будет соответствовать текст, содержащий одну или несколько цифр (`\d+`), за которыми следует пробельный символ (`\s`), за которым следует один или несколько словарных символов — буква, цифра или символ подчеркивания (`\w+`). Метод `findall()` возвращает все совпавшие строки шаблона регулярного выражения в виде списка.

Создание собственных символьных классов

Иногда возникает необходимость в сопоставлении регулярного выражения с символами из определенного набора, для которого сокращенные символьные классы (`\d`, `\w`, `\s` и т.д.) оказываются слишком широкими. Вы можете создать собственный символьный класс, используя квадратные скобки. Например, символьному классу `[aeiouAEIOU]` будет соответствовать любая гласная, будь то в нижнем или верхнем регистре. Введите в интерактивной оболочке следующие команды.

```
>>> vowelRegex = re.compile(r'[aeiouAEIOU]')
>>> vowelRegex.findall('RoboCop eats baby food. BABY FOOD.')
['o', 'o', 'o', 'e', 'a', 'a', 'o', 'o', 'A', 'O', 'O']
```

В классы также можно включать диапазоны букв и цифр, используя дефис. Например, классу `[a-zA-Z0-9]` будут соответствовать все буквы в нижнем и верхнем регистрах, а также цифры.

Обратите внимание на то, что в пределах квадратных скобок обычные символы регулярных выражений не интерпретируются как таковые. Это означает, что перед символами `.`, `*`, `?`, `(` и `)` не следует ставить символ обратной косой черты. Например, классу `[0-5.]` будут соответствовать цифры от `0` до `5` и точка. Вы не должны записывать этот класс как `[0-5\.]`.

Поместив сразу за открывающей квадратной скобкой класса символ "крышка", или циркумфлекс (`^`),можно создать *инвертированный символьный класс*. Инвертированному символьному классу будет соответствовать любой символ, не принадлежащий исходному классу. Например, введите в интерактивной оболочке следующие команды.

```
>>> consonantRegex = re.compile(r'[^aeiouAEIOU]')
>>> consonantRegex.findall('RoboCop eats baby food. BABY FOOD.')
['R', 'b', 'c', 'p', ' ', 't', 's', ' ', 'b', 'b', 'y', ' ', 'f',
'd', '.', ' ', 'B', 'B', 'Y', ' ', 'F', 'D', '.']
```

Теперь вместо гласных регулярному выражению будут соответствовать все символы, не являющиеся гласными.

Символ крышки и знак доллара

Вы также можете использовать символ ^ ("крышка") в начале регулярного выражения для указания того, что соответствие регулярному выражению должно находиться в *начале* тестируемого текста. Аналогичным образом можно поместить знак доллара ($) в конце регулярного выражения для указания того, что строка должна заканчиваться данным шаблоном регулярного выражения. Еще можно использовать символы ^ и $ совместно, для того чтобы показать: данному регулярному выражению должна соответствовать вся строка, т.е. совпадения с ним лишь какой-то подстроки будет недостаточно.

Например, регулярному выражению `r'^Hello'` будут соответствовать строки, начинающиеся со слова `'Hello'`. Введите в интерактивной оболочке следующие команды.

```
>>> beginsWithHello = re.compile(r'^Hello')
>>> beginsWithHello.search('Hello world!')
<_sre.SRE_Match object; span=(0, 5), match='Hello'>
>>> beginsWithHello.search('He said hello.') == None
True
```

Регулярному выражению `r'\d$'` будут соответствовать строки, заканчивающиеся любой цифрой в диапазоне от 0 до 9. Введите в интерактивной оболочке следующие команды.

```
>>> endsWithNumber = re.compile(r'\d$')
>>> endsWithNumber.search('Your number is 42')
<_sre.SRE_Match object; span=(16, 17), match='2'>
>>> endsWithNumber.search('Your number is forty two.') == None
True
```

Регулярному выражению `r'^\d+$'` будут соответствовать строки, которые начинаются и заканчиваются одной или несколькими цифрами. Введите в интерактивной оболочке следующие команды.

```
>>> wholeStringIsNum = re.compile(r'^\d+$')
>>> wholeStringIsNum.search('1234567890')
<_sre.SRE_Match object; span=(0, 10), match='1234567890'>
>>> wholeStringIsNum.search('12345xyz67890') == None
True
>>> wholeStringIsNum.search('12 34567890') == None
True
```

Последние два вызова метода `search()` в предыдущем примере демонстрируют необходимость совпадения с регулярным выражением всей строки, если в нем используются одновременно символы ^ и $.

Вам нужно лишь твердо запомнить, что символ ^ относится к началу строки, а символ $ — к концу.

Групповой символ

Символ . ("точка") в регулярных выражениях называется *групповым символом* и представляет собой сокращенную форму записи символьного класса, совпадающего с любым одиночным символом, за исключением символа новой строки. Например, введите в интерактивной оболочке следующие команды.

```
>>> atRegex = re.compile(r'.at')
>>> atRegex.findall('The cat in the hat sat on the flat mat.')
['cat', 'hat', 'sat', 'lat', 'mat']
```

Вспомните о том, что символу точки соответствует только одиночный произвольный символ, что и объясняет, почему в качестве совпадающего текста в слове `flat` была выбрана только подстрока `lat`. Если вы хотите искать соответствие самой точке, то в регулярном выражении ей должен предшествовать символ обратной косой черты: `\.`.

Указание соответствия любому тексту с помощью комбинации "точка–звездочка"

Иногда желательно, чтобы регулярному выражению соответствовало все и вся. Предположим, например, что вы хотите найти совпадение со строкой `'First Name:'`, за которой следует любой текст, за которым следует строка `'Last Name:'`, а за ней — вновь любой текст. Для представления этого "любого текста" можно использовать комбинацию "точка-звездочка" (`.*`). Вспомните о том, что символ . означает "любой одиночный символ, за исключением символа новой строки", а символ * означает "нуль или более вхождений предыдущего символа".

Введите в интерактивной оболочке следующие команды.

```
>>> nameRegex = re.compile(r'First Name: (.*) Last Name: (.*)')
>>> mo = nameRegex.search('First Name: Al Last Name: Sweigart')
>>> mo.group(1)
'Al'
>>> mo.group(2)
'Sweigart'
```

Комбинация `.*` работает в режиме *жадного поиска*: она всегда будет стремиться захватить как можно больше текста. Чтобы заставить ее работать в нежадной манере, дополните ее вопросительным знаком: `.*?`. Как и в случае фигурных скобок, вопросительный знак указывает Python на необходимость использования *нежадного* поиска.

Чтобы увидеть, чем различаются жадная и нежадная версии регулярного выражения, введите в интерактивной оболочке следующие команды.

```
>>> nongreedyRegex = re.compile(r'<.*?>')
>>> mo = nongreedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man>'

>>> greedyRegex = re.compile(r'<.*>')
>>> mo = greedyRegex.search('<To serve man> for dinner.>')
>>> mo.group()
'<To serve man> for dinner.>'
```

В общих чертах смысл обоих регулярных выражений можно выразить следующими словами: "Найти открывающую угловую скобку, за которой следует произвольный текст, завершающийся закрывающая угловая скобка". Однако в строке `'<To serve man> for dinner.>'` имеются два возможных варианта совпадения для закрывающей угловой скобки. В нежадной версии регулярного выражения Python ограничивается самой короткой из возможных строк: `'<To serve man>'`. В жадной версии Python стремится к тому, чтобы найти соответствие в виде как можно более длинной строки из всех возможных: `'<To serve man> for dinner.>'`.

Указание соответствия символам новой строки с помощью точки

Комбинации "точка-звездочка" будет соответствовать все, за исключением символа новой строки. Передав методу `re.compile()` при его вызове константу `re.DOTALL` в качестве второго аргумента, можно установить режим, при котором точке соответствует также символ новой строки.

Введите в интерактивной оболочке следующие команды.

```
>>> noNewlineRegex = re.compile('.*')
>>> noNewlineRegex.search('Serve the public trust.\nProtect the
↪ innocent.\nUphold the law.').group()
'Serve the public trust.'

>>> newlineRegex = re.compile('.*', re.DOTALL)
>>> newlineRegex.search('Serve the public trust.\nProtect the
↪ innocent.\nUphold the law.').group()
'Serve the public trust.\nProtect the innocent.\nUphold the law.'
```

Регулярному выражению noNewlineRegex, при создании которого методу re.compile() не передавался аргумент re.DOTALL, будет соответствовать весь текст, но только до первого символа новой строки, тогда как регулярному выражению newlineRegex, при создании которого методу re.compile() была передана константа re.DOTALL, будет соответствовать весь текст, включая символ новой строки. Именно поэтому вызов newlineRegex.search() возвращает полную строку вместе с ее символами новой строки.

Сводка символов регулярных выражений

В этой главе вы познакомились со многими элементами нотации регулярных выражений, поэтому имеет смысл привести их полный перечень.

- ? — соответствует отсутствию или одиночному вхождению предшествующей группы.
- * — соответствует отсутствию или произвольному количеству вхождений предшествующей группы.
- + — соответствует одиночному или большему количеству вхождений предшествующей группы.
- {n} — соответствует в точности n вхождениям предшествующей группы.
- {n,} — соответствует n или более вхождениям предшествующей группы.
- {,m} — соответствует отсутствию или вплоть до m вхождений предшествующей группы.
- {n,m} — соответствует не менее чем n и не более чем m вхождениям предшествующей группы.
- {n,m}?, или *?, или +? — выполняет нежадный поиск вхождений предшествующей группы.
- ^spam — означает, что строка должна начинаться символами spam.
- spam$ — означает, что строка должна заканчиваться символами spam.
- . — соответствует любому символу, за исключением символа новой строки.
- \d, \w и \s — совпадают с одиночным цифровым, словарным и пробельным символами соответственно.
- \D, \W и \S — совпадают с произвольным одиночным символом, не являющимся цифровым, словарным и пробельным соответственно.
- [abc] — совпадает с любым одиночным символом из числа тех, которые указаны в квадратных скобках (например, a, b или c).
- [^abc] — совпадает с любым одиночным символом, не относящимся к числу тех, которые указаны в квадратных скобках.

Игнорирование регистра при поиске соответствий

Обычно при установлении соответствия между регулярным выражением и текстом учитывается, в каком регистре записаны буквы. Например, следующим регулярным выражениям будут соответствовать разные строки.

```
>>> regex1 = re.compile('RoboCop')
>>> regex2 = re.compile('ROBOCOP')
>>> regex3 = re.compile('robOcop')
>>> regex4 = re.compile('RobocOp')
```

Однако иногда вас интересует лишь сам факт совпадения букв, независимо от их регистра. Можно установить режим, в котором регистр букв игнорируется, передав методу `re.compile()` при его вызове константу `re.IGNORECASE` или `re.I` в качестве второго аргумента. Введите в интерактивной оболочке следующие команды.

```
>>> robocop = re.compile(r'robocop', re.I)
>>> robocop.search('RoboCop is part man, part machine, all
↳ cop.').group()
'RoboCop'

>>> robocop.search('ROBOCOP protects the innocent.').group()
'ROBOCOP'

>>> robocop.search('Al, why does your programming book talk about
↳ robocop so much?').group()
'robocop'
```

Замена строк с помощью метода sub()

Регулярные выражения могут не только находить заданные образцы текста, но и заменять их новым текстом. Объекты `Regex` имеют метод `sub()`, который принимает два аргумента. Первый аргумент — это строка, которая должна подставляться вместо найденных совпадений. Второй аргумент — это строка регулярного выражения. Метод `sub()` возвращает строку, к которой применены замены.

Введите в интерактивной оболочке следующие команды.

```
>>> namesRegex = re.compile(r'Agent \w+')
>>> namesRegex.sub('CENSORED', 'Agent Alice gave the secret
↳ documents to Agent Bob.')
'CENSORED gave the secret documents to CENSORED.'
```

Иногда возникают ситуации, когда совпавший с регулярным выражением текст сам используется в качестве части подстановочного текста. Для этого можно использовать в первом аргументе при вызове метода `sub()` обратные ссылки `\1`, `\2`, `\3` и т.д., которые имеют следующий смысл: "Вставить в подстановочный текст группы 1, 2, 3 и так далее".

Предположим, например, что вы хотите скрыть имена секретных агентов, отображая в них лишь первые буквы. Для этого можно воспользоваться регулярным выражением `Agent (\w)\w*` и передать методу `sub()` в качестве первого аргумента строку `r'\1****'`. Ссылка `\1` в этой строке будет заменяться тем текстом, который будет захвачен группой `1`, т.е. группой `(\w)` регулярного выражения.

```
>>> agentNamesRegex = re.compile(r'Agent (\w)\w*')
>>> agentNamesRegex.sub(r'\1****', 'Agent Alice told Agent Carol
↳ that Agent Eve knew Agent Bob was a double agent.')
A**** told C**** that E**** knew B**** was a double agent.'
```

Работа со сложными регулярными выражениями

С регулярными выражениями легко работать в случае простых текстовых шаблонов. Однако сопоставление с более сложными текстовыми шаблонами может требовать построения длинных и запутанных регулярных выражений. Один из способов внесения ясности в подобных ситуациях заключается в использовании режима работы метода `re.compile()`, при котором пробелы и комментарии в строке регулярного выражения игнорируются. Чтобы включить этот "многословный режим", следует передать методу `re.compile()` константу `re.VERBOSE` в качестве второго аргумента.

Тогда вместо сложного для восприятия регулярного выражения наподобие

```
phoneRegex = re.compile(r'((\d{3}|\(\d{3}\))?(\s|-|\.)?\d{3}
(\s|-|\.)\d{4}(\s*(ext|x|ext.)\s*\d{2,5})?)')
```

можно использовать развернутое в нескольких строках и снабженное комментариями регулярное выражение следующего вида, понять смысл которого значительно легче:

```
phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?            # территориальный код
    (\s|-|\.)?                    # разделитель
    \d{3}                         # первые 3 цифры
    (\s|-|\.)                     # разделитель
    \d{4}                         # последние 4 цифры
    (\s*(ext|x|ext.)\s*\d{2,5})?  # добавочный номер
    )''', re.VERBOSE)
```

Обратите внимание на использование в предыдущем примере синтаксиса с тройными апострофами (`'''`) для создания многострочного текста, что позволило растянуть определение регулярного выражения на несколько строк, благодаря чему читать его стало намного легче.

Для комментариев в пределах строки регулярного выражения действуют те же правила, что и в случае обычного кода Python: символ # и все содержимое до конца строки игнорируются. Кроме того, дополнительные пробелы в многострочном регулярном выражении не считаются частью текстового шаблона, соответствие которому ищется. Это позволяет записать регулярное выражение таким образом, чтобы оно легче читалось.

Комбинация констант `re.IGNORECASE`, `re.DOTALL` и `re.VERBOSE`

Что если вы захотите использовать режим `re.VERBOSE` для включения комментариев в свое регулярное выражение, но одновременно иметь возможность игнорировать регистр с помощью константы `re.IGNORECASE`? К сожалению, метод `re.compile()` принимает лишь одиночное значение в качестве второго аргумента. Это ограничение можно обойти, объединяя константы `re.IGNORECASE`, `re.DOTALL` и `re.VERBOSE` с помощью символа канала (`|`), который в данном контексте имеет смысл *побитового* оператора ИЛИ.

Следовательно, если вы хотите, чтобы регулярное выражение игнорировало регистр и символам точки в нем соответствовали бы также символы новой строки, вызывайте метод `re.compile()` примерно следующим образом:

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL)
```

А вот пример объединения всех трех опций во втором аргументе:

```
>>> someRegexValue = re.compile('foo', re.IGNORECASE | re.DOTALL |
re.VERBOSE)
```

Этот синтаксис немного старомоден и происходит от ранних версий Python. Детальное рассмотрение работы побитовых операторов выходит за рамки книги, но вы сможете получить более подробную информацию по этому вопросу, посетив сайт `http://nostarch.com/automatestuff/`. Во втором аргументе можно передавать и другие опции; они не используются широко, и при желании вы сможете найти более подробную информацию о них самостоятельно.

Проект: извлечение телефонных номеров и адресов электронной почты

Предположим, вам предстоит заняться рутинной работой — найти все телефонные номера и адреса электронной почты, которые содержатся на длинной веб-странице или в документе большого размера. Если прокручивать страницу вручную, то на такой поиск у вас может уйти много времени. Но если бы у вас была программа, способная выполнять поиск телефонных номеров и электронных адресов в буфере обмена, то можно было бы просто нажать комбинацию клавиш <Ctrl+A> для выделения всего текста и комбинацию клавиши <Ctrl+C> для копирования выделенного текста в буфер, а затем выполнить программу. Такая программа могла бы заменить находящийся в буфере текст найденными телефонными номерами и адресами электронной почты.

Всякий раз, когда вы приступаете к новому проекту, возникает соблазн сразу же браться за написание кода. Однако в подавляющем большинстве случаев лучше не спешить и оценить общую картину. Я рекомендую всегда начинать с составления высокоуровневого плана того, что должна делать программа. На данном этапе не стоит задумываться о фактическом коде — об этом можно будет побеспокоиться позже. Приступая к работе, ограничивайтесь "широкими мазками".

Например, вот что должна делать ваша программа, предназначенная для извлечения телефонных номеров и адресов электронной почты:

- получать текст из буфера обмена;
- находить в тексте все телефонные номера и адреса электронной почты;
- переносить найденный текст в буфер обмена.

Вот теперь уже можно подумать над тем, как реализовать все это в виде кода. Код должен выполнять следующие операции:

- использовать модуль `pyperclip` для копирования и вставки строк;
- создавать два регулярных выражения, первое из которых соответствует телефонным номерам, а второе — адресам электронной почты;
- находить все совпадения, а не только первое, для обоих регулярных выражений;
- аккуратно форматировать найденные строки, преобразуя их в одну строку для вставки в буфер;
- отображать соответствующее сообщение, если искомые соответствия в тексте не были обнаружены.

Этот список служит своего рода дорожной картой проекта. В процессе написания кода вы сможете сконцентрировать внимание на каждом этапе по отдельности. Любой из этих этапов вполне выполним и выражается в терминах того, что вы уже умеете делать с помощью Python.

Шаг 1. Создание регулярного выражения для поиска телефонных номеров

Прежде всего, необходимо создать регулярное выражение для поиска телефонных номеров. Создайте новый файл, введите в него следующий код и сохраните его в файле *phoneAndEmail.py*.

```
#! python3
# phoneAndEmail.py - Находит телефонные номера и
# адреса электронной почты в буфере обмена.

import pyperclip, re

phoneRegex = re.compile(r'''(
    (\d{3}|\(\d{3}\))?              # территориальный код
    (\s|-|\.)?                      # разделитель
    (\d{3})                         # первые 3 цифры
    (\s|-|\.)                       # разделитель
    (\d{4})                         # последние 4 цифры
    (\s*(ext|x|ext.)\s*(\d{2,5}))?  # добавочный номер
    )''', re.VERBOSE)

# TODO: Создать регулярное выражение для адресов электронной
#       почты.

# TODO: Найти соответствия в тексте, содержащемся в
#       буфере обмена.

# TODO: Скопировать результаты в буфер обмена.
```

Комментарии `TODO` (ЧТО_СДЕЛАТЬ) всего лишь обозначают "скелет" программы. Впоследствии на их месте будет находиться фактический код.

Телефонный номер начинается с территориального кода, который не является обязательным, в связи с чем за соответствующей ему группой следует вопросительный знак. Поскольку территориальный код может иметь ровно три цифры (т.е. `\d{3}`) или ровно три цифры в круглых скобках (т.е. `\(\d{3}\)`), эти две альтернативы следует соединить символом канала. Вы также можете добавить в регулярное выражение комментарий `# территориальный код`, напоминающий о том, с чем должно совпадать регулярное выражение `(\d{3}|\(\d{3}\))?`.

В качестве разделителя групп цифр в телефонном номере могут использоваться пробел (`\s`), дефис (`-`) или точка (`.`), поэтому данные компоненты регулярного выражения также должны быть соединены символами канала. В следующих трех компонентах нет ничего сложного: три цифры, за которыми следует другой разделитель, а затем еще четыре цифры. Последняя часть — это необязательный добавочный номер, состоящий из

произвольного количества пробелов, за которыми следует буквенное обозначение `ext`, `x` или `ext.`, а затем — и сам добавочный номер, содержащий от двух до пяти цифр.

Шаг 2. Создание регулярного выражения для поиска адресов электронной почты

Вам также необходимо иметь регулярное выражение для поиска адресов электронной почты. Добавьте в программу новый код, выделенный ниже полужирным шрифтом.

```
#! python3
# phoneAndEmail.py - Находит телефонные номера и
# адреса электронной почты в буфере обмена.

import pyperclip, re

phoneRegex = re.compile(r'''(
--пропущенный код--

# Создание регулярного выражения для адресов электронной почты.
emailRegex = re.compile(r'''(
❶    [a-zA-Z0-9._%+-]+     # имя пользователя
❷    @                     # символ @
❸    [a-zA-Z0-9.-]+        # имя домена
     (\.[a-zA-Z]{2,4})     # остальная часть адреса
     )''', re.VERBOSE)

# TODO: Найти соответствия в тексте, содержащемся в
#       буфере обмена.

# TODO: Скопировать результаты в буфер обмена.
```

Часть адреса, содержащая имя пользователя ❶, включает один или более символов, которыми могут быть любые из следующих символов: буквы в верхнем или нижнем регистре, цифры, точка, символ подчеркивания, знак процента, знак "плюс" и дефис. Все эти символы можно указать в виде символьного класса: `[a-zA-Z0-9._%+-]`.

Имя домена отделяется от имени пользователя символом `@` ❷. Доменное имя ❸ представляется с помощью более узкого класса, включающего только буквы, цифры, точку и дефис: `[a-zA-Z0-9.-]`. А последняя, так называемая часть "dot-com" (с технической точки зрения представляющая *домен верхнего уровня*), фактически может содержать только точку и любой текст. Эта часть может включать от двух до четырех символов.

Формат адресов электронной почты определяется многими, подчас причудливыми, правилами. Данному регулярному выражению будут соответствовать не все корректные адреса электронной почты, но его будет

достаточно почти для всех видов электронных адресов, с которыми вы можете столкнуться.

Шаг 3. Поиск всех совпадений в тексте, скопированном в буфер обмена

Теперь, когда у вас уже есть регулярные выражения для поиска телефонных номеров и адресов электронной почты, можно поручить модулю `re` выполнить утомительную работу по поиску в буфере обмена всех строк, соответствующих составленным регулярным выражениям. Метод `pyperclip.paste()` получит строковое значение текста, хранящегося в буфере обмена, а метод `findall()` вернет список кортежей.

Добавьте в программу новый код, выделенный ниже полужирным шрифтом.

```
#! python3
# phoneAndEmail.py - Находит телефонные номера и
# адреса электронной почты в буфере обмена.

import pyperclip, re

phoneRegex = re.compile(r'''(
--пропущенный код--

# Поиск соответствий в тексте, содержащемся в
# буфере обмена.
text = str(pyperclip.paste())
❶ matches = []
❷ for groups in phoneRegex.findall(text):
    phoneNum = '-'.join([groups[1], groups[3], groups[5]])
    if groups[8] != '':
        phoneNum += ' x' + groups[8]
    matches.append(phoneNum)
❸ for groups in emailRegex.findall(text):
    matches.append(groups[0])

# TODO: Скопировать результаты в буфер обмена.
```

Для каждого совпадения создается один кортеж, и каждый кортеж содержит строки для каждой группы в регулярном выражении. Не забывайте о том, что группе `0` соответствует все регулярное выражение, поэтому то, что вам нужно, — это группа с индексом `0` в кортеже.

Найденные совпадения с регулярным выражениям сохраняются в списке `matches`. Поначалу этот список пуст ❶. Далее следуют два цикла `for`. В случае адресов электронной почты достаточно присоединять к списку `matches` группу `0` каждого найденного совпадения ❸. В случае же телефонных номеров мы не можем ограничиться только этим. Поскольку программа ищет

телефонные номера, формат которых может быть различным, то, прежде чем присоединять их к списку, их нужно привести к единому стандартному формату. В переменной `phoneNum` содержится строка, скомпонованная из групп `1`, `3`, `5` и `8` совпавшего текста ❷. (Этими группами являются территориальный код, первые три цифры, последние четыре цифры и добавочный номер.)

Шаг 4. Объединение совпадений в одну строку для копирования в буфер обмена

Теперь, когда адреса электронной почты и телефонные номера сохранены в переменной `matches`, их необходимо скопировать в буфер обмена. Функция `pyperclip.copy()` принимает одиночное строковое значение, а не список строк, поэтому для переменной `matches` вызывается метод `join()`.

Чтобы упростить проверку работы программы, выведем все найденные совпадения на экран. А если ни телефонных номеров, ни адресов электронной почты в тексте не найдено, будет выведено соответствующее сообщение.

Внесите в программу следующие изменения.

```
#! python3
# phoneAndEmail.py - Находит телефонные номера и
# адреса электронной почты в буфере обмена.

--пропущенный код--
for groups in emailRegex.findall(text):
  matches.append(groups[0])

# Копирование результатов в буфер обмена.
if len(matches) > 0:
    pyperclip.copy('\n'.join(matches))
    print('Скопировано в буфер обмена:')
    print('\n'.join(matches))
else:
    print('Телефонные номера и адреса электронной
↳ почты не обнаружены.')
```

Выполнение программы

В качестве примера откройте свой браузер на странице контактов сайта No Starch Press по адресу `http://www.nostarch.com/contactus.htm`, нажмите комбинацию клавиш <Ctrl+A> для выделения всего текста на странице, а затем комбинацию клавиш <Ctrl+C> для копирования этого текста в буфер. Выполнив программу, вы должны получить примерно следующий вывод.

```
Скопировано в буфер обмена:
800-420-7240
```

```
415-863-9900
415-863-9950
info@nostarch.com
media@nostarch.com
academic@nostarch.com
help@nostarch.com
```

Идеи относительно создания аналогичных программ

Распознавание образцов текста (и их замена с помощью метода `sub()`) имеет множество возможных областей применения, например:

- нахождение URL-адресов веб-сайтов, начинающихся с префикса `http://` или `https://`;
- унификация дат, приведенных в различных форматах (например, `1/21/2015`, `01-21-2015` и `2015/1/21`), путем их замены датами, представленными с помощью выбранного стандартного формата;
- удаление конфиденциальной информации, такой как номера социальных страховок или кредитных карт;
- исправление типичных опечаток, таких как наличие нескольких пробелов между словами, случайное повторение слов или наличие нескольких восклицательных знаков в конце предложения. Это так раздражает!!!

Резюме

Несмотря на то что компьютер способен очень быстро выполнять текстовый поиск, он нуждается в конкретных указаниях относительно того, что именно необходимо найти. Регулярные выражения позволяют точно определить символьные шаблоны для поиска. В действительности некоторые текстовые процессоры и электронные таблицы предоставляют средства поиска и замены, использующие механизм регулярных выражений.

Поставляемый вместе с Python модуль `re` обеспечивает компиляцию объектов регулярных выражений, так называемых объектов `Regex`. В частности, эти объекты имеют следующие методы: `search()` — поиск одиночных совпадений с регулярным выражением, `findall()` — поиск всех экземпляров совпадений и `sub()` — поиск с заменой текста.

В этой главе синтаксис регулярных выражений был рассмотрен далеко не полностью. Более подробную информацию по этой теме вы сможете получить, обратившись к официальной документации Python по адресу `http://docs.python.org/3/library/re.html`. Вам также будет полезно прочитать практическое руководство по работе с регулярными выражениями, доступное по адресу `http://www.regular-expressions.info/`.

Теперь, когда вы уже приобрели определенный опыт манипулирования текстовыми строками и научились работать с текстовыми шаблонами, мы можем перейти к углубленному рассмотрению методов чтения и записи данных в файлы, сохраняемые на жестком диске вашего компьютера.

Контрольные вопросы

1. С помощью какой функции создаются объекты регулярных выражений (объекты `Regex`)?
2. Почему при создании объектов `Regex` часто используются "сырые" строки?
3. Что возвращает метод `search()`?
4. Как получить из объекта `Match` фактические строки, соответствующие шаблону регулярного выражения?
5. Что именно представляет в объекте регулярного выражения, созданном на основе строки `r'(\d\d\d)-(\d\d\d-\d\d\d\d)'`, группа `0`? Группа `1`? Группа `2`?
6. В синтаксисе регулярных выражений круглые скобки и точки имеют особый смысл. Как бы вы указали в регулярном выражении, что символы круглых скобок и точки сами являются объектом поиска?
7. Метод `findall()` возвращает список строк или список кортежей строк. В каких именно случаях возвращается тот или иной тип значений?
8. Что означает символ `|` в регулярных выражениях?
9. Какие две функции выполняет символ `?` в регулярных выражениях?
10. В чем разница между символами `+` и `*` в регулярных выражениях?
11. В чем разница между записями `{3}` и `{3,5}` в регулярных выражениях?
12. Что означают сокращенные символьные классы `\d`, `\w` и `\s` в регулярных выражениях?
13. Что означают сокращенные символьные классы `\D`, `\W` и `\S` в регулярных выражениях?
14. Как сделать регулярные выражения нечувствительными к регистру?
15. Чему обычно соответствует символ `.`? Чему он соответствует, если методу `re.compile()` в качестве второго аргумента передана константа `re.DOTALL`?
16. В чем разница между сочетаниями символов `.*` и `.*??`
17. Как записать символьный класс, которому соответствуют все цифры и буквы в нижнем регистре?
18. Если `numRegex = re.compile(r'\d+')`, то что вернет вызов `numRegex.sub('X', '12 drummers, 11 pipers, five rings, 3 hens')`?

19. Что становится возможным при передаче константы `re.VERBOSE` в качестве второго аргумента при вызове метода `re.compile()`?
20. Как бы вы записали регулярное выражение, которому соответствуют числа с запятой в качестве разделителя между каждыми тремя цифрами? Этому выражению должны соответствовать следующие числа:
    - `'42'`
    - `'1,234'`
    - `'6,368,745'`

    и не должны соответствовать следующие:
    - `'12,34,567'` (только две цифры между запятыми);
    - `'1234'` (отсутствуют запятые).
21. Как бы вы записали регулярное выражение, которому соответствуют все полные имена, включающие фамилию `Nakamoto`? Можно предположить, что имя, предшествующее фамилии, всегда состоит из одного слова, начинающегося с большой буквы. Этому регулярному выражению должны соответствовать следующие имена:
    - `'Satoshi Nakamoto'`
    - `'Alice Nakamoto'`
    - `'RoboCop Nakamoto'`

    и не должны соответствовать следующие:
    - `'satoshi Nakamoto'` (имя не начинается с большой буквы);
    - `'Mr. Nakamoto'` (в предшествующем слове имеется небуквенный символ);
    - `'Nakamoto'` (имя отсутствует);
    - `'Satoshi nakamoto'` (фамилия `Nakamoto` не начинается с большой буквы).
22. Как бы вы записали регулярное выражение, совпадающее с предложениями, которые начинаются с одного из слов `Alice`, `Bob` и `Carol`; вторым словом является одно из слов `eats`, `pets` и `throws`; третьим словом является одно из слов `apples`, `cats` и `baseballs`; и которые заканчиваются точкой? Это регулярное выражение должно быть нечувствительным к регистру. Ему должны соответствовать следующие предложения:
    - `'Alice eats apples.'`
    - `'Bob pets cats.'`
    - `'Carol throws baseballs.'`
    - `'Alice throws Apples.'`
    - `'BOB EATS CATS.'`

и не должны соответствовать следующие:

- `'RoboCop eats apples.'`
- `'ALICE THROWS FOOTBALLS.'`
- `'Carol eats 7 cats.'`

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Обнаружение сильных паролей

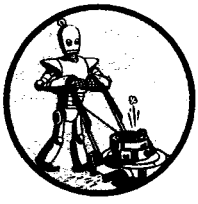
Напишите функцию, которая использует регулярные выражения для проверки того, что переданная ей строка представляет собой сильный пароль. Сильными считаются пароли, которые состоят по крайней мере из восьми символов, содержат символы в верхнем и нижнем регистрах и включают по крайней мере одну цифру.

Версия функции `strip()`, использующая регулярные выражения

Напишите функцию, которая принимает строку и делает то же, что и строковый метод `strip()`. Если ей не переданы никакие другие аргументы, кроме строки, то данная функция должна удалить из строки начальные и конечные пробельные символы. В противном случае из строки должны быть удалены символы, переданные функции в качестве второго аргумента.

8

ЧТЕНИЕ И ЗАПИСЬ ФАЙЛОВ



Переменные — отличное средство хранения данных во время выполнения программы, но если вы хотите, чтобы данные существовали и после ее выполнения, то сохраните их в файле. Можно рассматривать содержимое файла как одну строку, размер которой способен исчисляться гигабайтами. Из этой главы вы узнаете о том, как использовать Python для создания, чтения и сохранения файлов на жестком диске.

Файлы и пути доступа к ним

Файл имеет два ключевых свойства: *имя файла* (обычно записываемое в виде одного слова) и *путь доступа к файлу*. Путь определяет, где именно в компьютере располагается файл. Например, в моем ноутбуке, работающем под управлением Windows 7, есть файл с именем *projects.docx* и путем доступа *C:\Users\asweigart\Documents*. Часть имени файла, которая следует за последней точкой, называется *расширением имени файла* (для краткости часто говорят *расширение файла* или просто *расширение*) и указывает на тип файла. Файл *project.docx* — это документ Word, а *Users*, *asweigart* и *Documents* — это названия папок (также называемых *каталогами*). Папки могут содержать файлы и другие папки. Например, файл *project.docx* содержится в папке *Documents*, которая содержится в папке *asweigart*, которая, в свою очередь, содержится в папке *Users*. Этот способ организации папок проиллюстрирован на рис. 8.1.

Часть *C:* пути к файлу — это *корневая папка*, которая содержит все остальные папки. В Windows корневой является папка *C:*, которая также называется диском *C:*. В OS X и Linux корневой является папка */*. В этой книге для корневой папки используется обозначение *C:* в стиле Windows. Если вы выполняете примеры в интерактивной оболочке OS X или Linux, то используйте вместо этого обозначение */*.

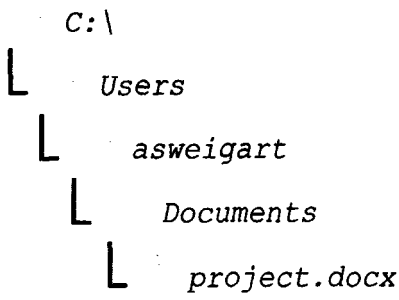


Рис. 8.1. Расположение файла в иерархии папок

Дополнительные тома, такие как диски DVD или USB-флешки, будут отображаться по-разному в разных операционных системах. В Windows они отображаются в виде корневых дисков с другими буквенными обозначениями: *D:*, *E:* и т.д. В OS X они отображаются в виде новых папок в папке */Volumes*, в Linux — в виде новых папок в папке */mnt* (от англ. *mount*). Кроме того, имейте в виду, что, в то время как в Windows и OS X имена файлов и папок нечувствительны к регистру, в Linux они зависят от регистра.

Использование обратной косой черты в Windows и косой черты в OS X и Linux

В Windows пути к файлам записывают, разделяя имена папок обратной косой чертой (\). В то же время в OS X и Linux разделителем служит косая черта (/). Если вы хотите, чтобы ваши программы работали во всех операционных системах, пишите свои сценарии на Python так, чтобы обрабатывались оба случая.

К счастью, это очень просто делается с помощью функции `os.path.join()`. Если вы передадите ей строковые значения имен файлов и папок в своем пути доступа, то вызов `os.path.join()` возвратит строку, в которой путь доступа к файлу указан с использованием корректной версии разделителя. Введите в интерактивной оболочке следующие команды.

```
>>> import os
>>> os.path.join('usr', 'bin', 'spam')
'usr\\bin\\spam'
```

Я выполняю все примеры в интерактивной оболочке Windows, поэтому вызов `os.path.join('usr', 'bin', 'spam')` вернул мне строку `'usr\\bin\\spam'`. (Обратите внимание на то, что символы обратной косой черты продублированы, поскольку каждый из них нуждается в экранировании другим символом обратной косой черты.) Если бы я вызвал эту функцию в OS X или Linux, то была бы возвращена строка `'usr/bin/spam'`.

Функция `os.path.join()` оказывается полезной при создании строк для имен файлов. Эти строки будут передаваться ряду функций, используемых при работе с файлами, с которыми вы познакомитесь в данной главе. Например, в следующем примере имена файлов из заданного списка присоединяются к имени папки.

```
>>> myFiles = ['accounts.txt', 'details.csv', 'invite.docx']
>>> for filename in myFiles:
        print(os.path.join('C:\\Users\\asweigart', filename))
C:\Users\asweigart\accounts.txt
C:\Users\asweigart\details.csv
C:\Users\asweigart\invite.docx
```

Текущий рабочий каталог

Каждая программа, которая выполняется на компьютере, имеет *текущий рабочий каталог* (current working directory – cwd). Предполагается, что любые имена файлов или пути, которые не начинаются с указания корневой папки, заданы относительно текущего рабочего каталога. Для получения значения текущего рабочего каталога в виде строки используется функция `os.getcwd()`, а для его изменения – функция `os.chdir()`. Введите в интерактивной оболочке следующие команды.

```
>>> import os
>>> os.getcwd()
'C:\\Python34'
>>> os.chdir('C:\\Windows\\System32')
>>> os.getcwd()
'C:\\Windows\\System32'
```

Здесь в качестве текущего рабочего каталога устанавливается папка *C:\Python34*, поэтому имя файла *project.docx* ссылается на файл *C:\Python34\project.docx*. Если мы заменим текущий рабочий каталог каталогом *C:\Windows*, то имя файла *project.docx* будет интерпретироваться как полное имя *C:\Windows\project.docx*.

При попытке перейти в несуществующий каталог Python выведет сообщение об ошибке.

```
>>> os.chdir('C:\\ThisFolderDoesNotExist')
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    os.chdir('C:\\ThisFolderDoesNotExist')
FileNotFoundError: [WinError 2] The system cannot find the file
specified:
'C:\\ThisFolderDoesNotExist'
```

Примечание

Несмотря на то что более современным эквивалентом термина каталог является термин "папка", в качестве стандартного используется термин "текущий рабочий каталог" (или просто "рабочий каталог"), а не "текущая рабочая папка".

Абсолютные и относительные пути доступа

Существуют два способа определения пути доступа к файлу:

- *абсолютный путь*, который всегда начинается с имени корневой папки;
- *относительный путь*, который задается относительно текущего рабочего каталога программы.

Существуют также папки, обозначаемые одним (.) или двумя (..) символами точки. Это не реальные папки, а специальные имена, которые могут использоваться при задании путей. Одиночная точка является сокращенным обозначением, имеющим смысл "данная папка". Двойная точка имеет смысл "родительская папка".

На рис. 8.2 показан пример расположения папок и файлов. Если в качестве текущего установлен рабочий каталог *C:\bacon*, то относительные пути к другим папкам и файлам задаются так, как показано на рисунке.

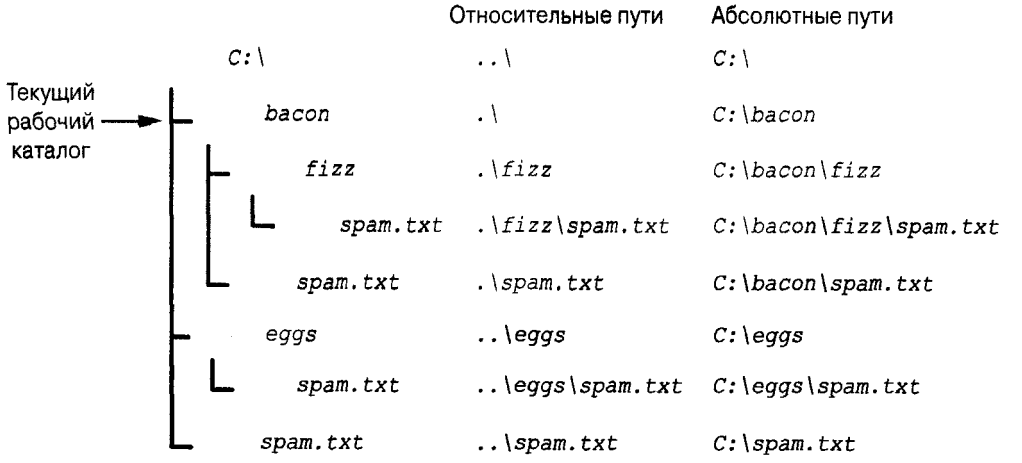


Рис. 8.2. Относительные пути доступа к папкам и файлам в рабочем каталоге C:\bacon

Имя .\ в начале относительного пути является необязательным. Например, пути *.\spam.txt* и *spam.txt* ведут к одному и тому же файлу.

Создание новых папок с помощью функции os.makedirs()

Ваши программы могут создавать новые папки (каталоги) с помощью функции `os.makedirs()`. Введите в интерактивной оболочке следующие команды.

```
>>> import os
>>> os.makedirs('C:\\delicious\\walnut\\waffles')
```

В результате будет создана не только папка *C:\delicious*, но и расположенная в ней папка *walnut*, а также расположенная в папке *C:\delicious\walnut* папка *waffles*. Таким образом, функция `os.makedirs()` будет создавать все необходимые промежуточные папки, гарантируя существование полного пути. Соответствующая иерархия папок показана на рис. 8.3.

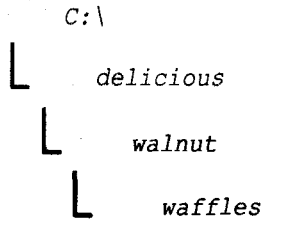


Рис. 8.3. Результат работы функции `os.makedirs('C:\\delicious\\walnut\\waffles')`

Модуль os.path

Модуль `os.path` содержит множество полезных функций для работы с именами файлов и путями доступа. Например, вы уже использовали функцию `os.path.join()`, создающую корректные пути для любой операционной системы. Поскольку `os.path` — это модуль, содержащийся в модуле `os`, для его импортирования достаточно выполнить инструкцию `import os`. Всякий раз, когда вашей программе необходимо работать с файлами, папками или путями доступа, вы можете обращаться для справки к коротким примерам, приведенным в этом разделе. Полная документация модуля `os.path` приведена на сайте Python по адресу `http://docs.python.org/3/library/os.path.html`.

Примечание

Для большинства последующих примеров, приведенных в этом разделе, вам потребуется модуль `os`*, поэтому не забывайте импортировать его в начале каждого сценария и при каждом перезапуске IDLE. В противном случае будет выведено сообщение об ошибке* `NameError: name 'os' is not defined`.

Обработка абсолютных и относительных путей

Модуль `os.path` предоставляет функции для возврата абсолютного пути по заданному относительному пути и проверки того, является ли путь абсолютным.

- Вызов `os.path.abspath(path)` возвращает строку абсолютного пути аргумента. Это простой способ преобразования относительного пути в абсолютный.
- Вызов `os.path.isabs(path)` возвращает значение `True`, если аргумент — абсолютный путь, и `False`, если аргумент — относительный путь.
- Вызов `os.path.relpath(path, start)` возвращает строку относительного пути от *start* к *path*. Если путь *start* не предоставлен, то в качестве него используется текущий рабочий каталог.

Испытайте работу этих функций в интерактивной оболочке.

```
>>> os.path.abspath('.')
'C:\\Python34'
>>> os.path.abspath('.\\Scripts')
'C:\\Python34\\Scripts'
>>> os.path.isabs('.')
False
>>> os.path.isabs(os.path.abspath('.'))
True
```

Поскольку в момент вызова функции `os.path.abspath()` текущим рабочим каталогом был `C:\Python34`, папка `.\` представляет абсолютный путь `'C:\\Python34'`.

Примечание

Так как файлы и папки в вашей системе, вероятнее всего, будут отличаться от моих, вы не сможете воспроизвести все примеры в этой главе в том виде, в каком они приводятся. Тем не менее попытайтесь их выполнить, используя папки на своем компьютере.

Выполните в интерактивной оболочке следующие команды.

```
>>> os.path.relpath('C:\\Windows', 'C:\\')
'Windows'
>>> os.path.relpath('C:\\Windows', 'C:\\spam\\eggs')
'..\\..\\Windows'
>>> os.getcwd()
'C:\\Python34'
```

Вызов `os.path.dirname(path)` возвращает строку, содержащую всю часть пути, которая предшествует последней косой черте в аргументе *path*. Вызов `os.path.basename(path)` возвращает строку, содержащую всю ту часть пути, которая следует за последней косой чертой в аргументе *path*. Имя папки и базовое имя пути показаны на рис. 8.4.

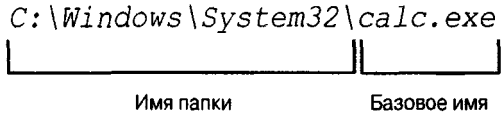


Рис. 8.4. Базовое имя указывается за последней косой чертой в обозначении пути и совпадает с именем файла. Имя папки — это вся часть пути, которая предшествует последней косой черте

Например, введите в интерактивной оболочке следующие команды.

```
>>> path = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.basename(path)
'calc.exe'
>>> os.path.dirname(path)
'C:\\Windows\\System32'
```

Если вам нужны как имя папки, так и базовое имя, достаточно вызвать функцию `os.path.split()`, которая возвращает кортеж, включающий обе эти строки.

```
>>> calcFilePath = 'C:\\Windows\\System32\\calc.exe'
>>> os.path.split(calcFilePath)
('C:\\Windows\\System32', 'calc.exe')
```

Заметьте, что тот же кортеж можно получить, вызвав функции `os.path.dirname()` и `os.path.basename()` и поместив возвращаемые ими значения в кортеж.

```
>>> (os.path.dirname(calcFilePath),
os.path.basename(calcFilePath))
('C:\\Windows\\System32', 'calc.exe')
```

Однако если вам нужны оба значения, то вызов `os.path.split()` является более коротким.

Обратите внимание на то, что возвращаемое значение функции `os.path.split()` *не* представляет собой список строк, соответствующих каждой из папок пути по отдельности. Для получения такого списка следует использовать строковый метод `split()` совместно с разделителем `os.sep`. Вспомните, что переменная `os.sep` содержит ту версию косой черты (прямой или

обратной), используемой в качестве разделителя имен папок, которая соответствует установленной на компьютере операционной системе.

Например, введите в интерактивной оболочке следующие команды.

```
>>> calcFilePath.split(os.path.sep)
['C:', 'Windows', 'System32', 'calc.exe']
```

На компьютерах, работающих под управлением OS X и Linux, в начале возвращенного списка будет указана пустая строка.

```
>>> '/usr/bin'.split(os.path.sep)
['', 'usr', 'bin']
```

Строковый метод `split()` обеспечивает возврат списка всех частей пути. Если вы передадите ему разделитель `os.path.sep`, то он сработает в любой операционной системе.

Определение размеров файлов и содержимого папок

Научившись работать с путями доступа к файлам, можете приступить к сбору информации о конкретных файлах и папках. Модуль `os.path` предоставляет функции, позволяющие находить размеры файлов, выраженные в байтах, и определять, какие файлы и папки содержатся в заданной папке.

- Вызов функции `os.path.getsize(path)` возвращает выраженный в байтах размер файла, указанного в аргументе *path*.
- Вызов `os.listdir(path)` возвращает список строк с именами всех файлов с путем доступа, указанным в аргументе `path`. (Обратите внимание на то, что эта функция содержится в модуле `os`, а не в модуле `os.path`.)

Вот что я получаю, когда выполняю эти функции в интерактивной оболочке.

```
>>> os.path.getsize('C:\\Windows\\System32\\calc.exe')
776192
>>> os.listdir('C:\\Windows\\System32')
['0409', '12520437.cpx', '12520850.cpx', '5U877.ax',
'aaclient.dll',
--пропущенный код--
'xwtpdui.dll', 'xwtpw32.dll', 'zh-CN', 'zh-HK', 'zh-TW',
'zipfldr.dll']
```

Как видите, на моем компьютере программа `calc.exe` имеет размер 776 192 байта, и в папке `C:\Windows\system32` содержится множество

файлов. Если бы потребовалось найти суммарный размер всех файлов, находящихся в этой папке, то для этого я мог бы воспользоваться функциями `os.path.getsize()` и `os.listdir()`.

```
>>> totalSize = 0
>>> for filename in os.listdir('C:\\Windows\\System32'):
      totalSize = totalSize +
      os.path.getsize(os.path.join('C:\\Windows\\System32',
      filename))

>>> print(totalSize)
1117846456
```

По мере того как в цикле перебираются имена всех файлов, содержащихся в папке `C:\Windows\System32`, значение переменной `totalSize` каждый раз увеличивается на размер очередного файла. Заметьте, как я использую функцию `os.path.join()` для присоединения имени папки к текущему имени файла при вызове функции `os.path.getsize()`. Целочисленное значение, возвращаемое функцией `os.path.getsize()`, добавляется к текущему значению переменной `totalSize`. По завершении цикла выводится значение `totalSize`, отображающее суммарный размер содержимого папки `C:\Windows\System32`.

Проверка существования пути

Многие функции Python завершаются аварийно с выдачей сообщения об ошибке в случае, если предоставленный им путь не существует. Модуль `os.path` представляет функции, позволяющие проверить, существует ли заданный путь и соответствует ли он файлу или папке.

- Вызов `os.path.exists(path)` возвращает значение `True`, если файл (или папка), на который ссылается аргумент, существует, и значение `False`, если он не существует.
- Вызов `os.path.isfile(path)` возвращает значение `True`, если заданный аргументом путь существует и является файлом, и значение `False` в противном случае.
- Вызов `os.path.isdir(path)` возвращает значение `True`, если заданный аргументом путь существует и является папкой; иначе — `False`.

Вот что я получаю, когда выполняю эти функции в интерактивной оболочке.

```
>>> os.path.exists('C:\\Windows')
True
>>> os.path.exists('C:\\some_made_up_folder')
```

```
False
>>> os.path.isdir('C:\\Windows\\System32')
True
>>> os.path.isfile('C:\\Windows\\System32')
False
>>> os.path.isdir('C:\\Windows\\System32\\calc.exe')
False
>>> os.path.isfile('C:\\Windows\\System32\\calc.exe')
True
```

Чтобы определить, подключен ли в данный момент к компьютеру DVD или флеш-диск, можно воспользоваться функцией `os.path.exists()`. Например, если бы я хотел проверить, подключен ли флеш-диск `D:\` на моем Windows-компьютере, то я мог бы это сделать с помощью следующей команды.

```
>>> os.path.exists('D:\\')
False
```

Ой! Похоже, я забыл подключить свою флешку!

Чтение и запись файлов

Научившись уверенно работать с папками и относительными путями, вы сможете задавать расположение файлов для операций чтения и записи. Функции, рассматриваемые в нескольких последующих разделах, будут применяться к простым текстовым файлам. *Простые текстовые файлы* содержат лишь базовые текстовые символы, не сопровождающиеся информацией о шрифте, размере и цвете текста. В качестве примера простых текстовых файлов можно привести файлы с расширением *.txt* или файлы сценариев Python с расширением *.py*. Эти файлы могут быть открыты с помощью приложения Notepad в Windows или приложения TextEdit в OSX. Ваши программы могут легко читать содержимое простых текстовых файлов и обрабатывать их как одиночные строковые значения.

Другим типом файлов являются *двоичные* (бинарные) файлы, такие как файлы документов, создаваемых текстовыми процессорами, PDF-файлы, а также файлы изображений, электронных таблиц и выполняемых программ. Открыв двоичный файл в приложении Блокнот или TextEdit, вы увидите бессмысленный набор странных символов (рис. 8.5).

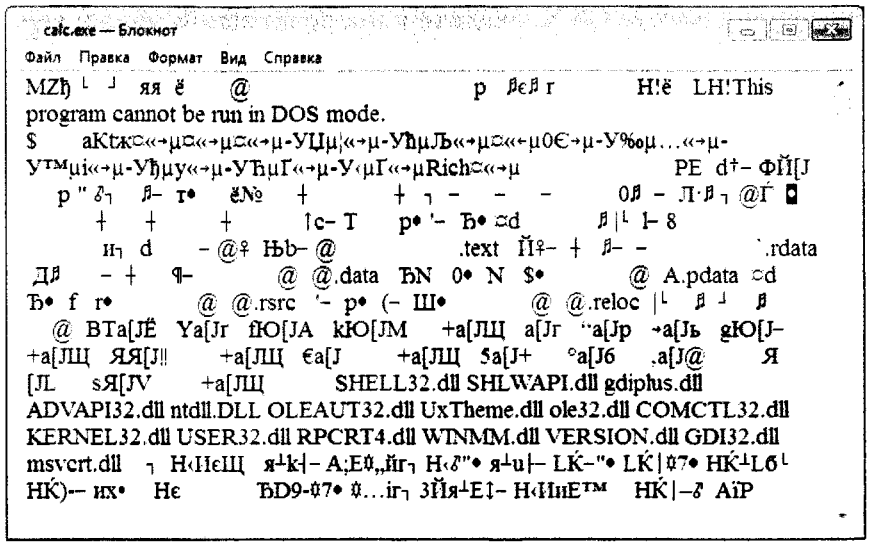


Рис. 8.5. Windows-программа calc.exe, открытая в приложении Блокнот

Поскольку различные типы двоичных файлов должны обрабатываться по-разному, мы не будем пытаться в этой книге непосредственно читать и записывать "сырые" двоичные файлы. К счастью, многие модули упрощают работу с двоичными файлами. С одним из них, модулем `shelve`, вы познакомитесь чуть позже.

В Python операции чтения/записи файлов выполняются в три этапа:

1) вызовите функцию `open()`, которая возвратит объект `File`;
2) вызовите метод `read()` или `write()` для объекта `File`;
3) закройте файл, вызвав метод `close()` для объекта `File`.

Открытие файла с помощью функции `open()`

Чтобы открыть файл с помощью функции `open()`, вы передаете ей строку, содержащую путь к файлу, который хотите открыть, причем это может быть абсолютный или относительный путь. Функция `open()` возвращает объект `File`.

Создайте текстовый файл *hello.txt* с помощью приложения Notepad или TextEdit. Введите `Hello, world!` и сохраните файл в своей пользовательской папке. Затем, если вы используете Windows, введите в интерактивной оболочке следующую команду:

```
>>> helloFile =open('C:\\Users\\ваша_папка_пользователя\\
hello.txt')
```

Если вы используете OS X, введите в интерактивной оболочке следующую команду:

```
>>> helloFile = open('/Users/ваша_папка_пользователя/hello.txt')
```

Подставьте вместо заменителя *ваша_папка_пользователя* реальное имя своей папки. Например, на своем компьютере с Windows я зарегистрирован как пользователь `asweigart` и поэтому использую путь *'C:\\Users\\asweigart\\hello.txt'*.

Обе приведенные выше команды открывают файл в режиме "чтения простого текста" или, для краткости, "в режиме чтения". Если файл открывается в режиме чтения, то Python позволяет лишь читать данные из файла; вы не сможете записать данные в файл или каким-либо образом изменить его содержимое. Режим чтения — это режим по умолчанию для файлов, открываемых в Python. Но если вы не хотите полагаться на установки по умолчанию, то можете явно указать этот режим, передав функции `open()` строковое значение `'r'` в качестве второго аргумента. Поэтому вызовы `open('/Users/asweigart/hello.txt', 'r')` и `open('/Users/asweigart/hello.txt')` делают одно и то же.

Вызов `open()` возвращает объект `File`. Этот объект представляет файл на вашем компьютере и является просто еще одним типом значений в Python, во многом таким же, как списки или словари, с которыми вы уже знакомы. В предыдущем примере вы сохранили объект `File` в переменной `helloFile`. Теперь всякий раз, когда вы захотите прочитать или записать данный файл, вам достаточно будет вызвать соответствующий метод для объекта `File`, сохраненного в переменной `helloFile`.

Чтение содержимого файла

Имея объект `File`, можно приступить к чтению данных из него. Если требуется прочитать все содержимое файла в виде одного строкового значения, используйте метод `read()` объекта `File`. Продолжим работу с объектом `File` файла *hello.txt*, сохраненным в переменной `helloFile`. Введите в интерактивной оболочке следующие команды.

```
>>> helloContent = helloFile.read()
>>> helloContent
'Hello, world!'
```

Если вы хотите рассматривать содержимое файла как одну большую строку, то метод `read()` возвращает строку, хранящуюся в этом файле.

Возможен и другой подход, когда вы используете метод `readlines()` для получения из файла списка строковых значений, по одной строке для каждой строки текста. Например, создайте файл *sonnet29.txt* в той же папке, в которой находится файл *hello.txt*, и введите в него следующий текст.

```
When, in disgrace with fortune and men's eyes,
I all alone beweep my outcast state,
And trouble deaf heaven with my bootless cries,
And look upon myself and curse my fate,
```

В процессе ввода текста не забывайте разрывать строки, нажимая клавишу <Enter>. Затем введите в интерактивной оболочке следующие команды.

```
>>> sonnetFile = open('sonnet29.txt')
>>> sonnetFile.readlines()
[When, in disgrace with fortune and men's eyes,\n', ' I all alone
beweep my outcast state,\n', And trouble deaf heaven with my
bootless cries,\n', And look upon myself and curse my fate,']
```

Обратите внимание на то, что каждое из строковых значений, за исключением последней строки файла, заканчивается символом новой строки, `\n`. Во многих случаях работать со списком строк проще, чем с одним длинным строковым значением.

Запись в файл

Python позволяет записывать содержимое файла аналогично тому, как это делается при "записи" строк на экран с помощью функции `print()`. Однако запись в файл, открытый в режиме чтения, невозможна. Вместо этого файл должен быть открыт в режиме записи простого текста или добавления простого текста (далее для краткости – "в режиме записи" и "в режиме добавления" соответственно).

В режиме записи содержимое существующего файла удаляется, и новые данные записываются "с чистого листа" аналогично тому, как в процессе операции присваивания старое значение переменной заменяется новым. Чтобы открыть файл в режиме записи, следует передать методу `open()` строку `'w'` в качестве второго аргумента. С другой стороны, в режиме добавления новый текст добавляется в конец существующего файла. Эту операцию можно рассматривать как присоединение нового значения к хранящемуся в переменной списку, а не полную перезапись содержимого переменной. Чтобы открыть файл в режиме добавления, следует передать методу `open()` строку `'a'` в качестве второго аргумента.

Если файла с именем, переданным методу `open()`, не существует, то как в режиме записи, так и в режиме добавления будет создан новый, пустой файл. Прежде чем вновь открывать файл после выполнения операций чтения или записи, его предварительно нужно закрыть с помощью метода `close()`.

Сведем все это воедино. Введите в интерактивной оболочке следующие команды.

```
>>> baconFile = open('bacon.txt', 'w')
>>> baconFile.write('Hello, world!\n')
13
>>> baconFile.close()
>>> baconFile = open('bacon.txt', 'a')
>>> baconFile.write('Bacon is not a vegetable.')
25
>>> baconFile.close()
>>> baconFile = open('bacon.txt')
>>> content = baconFile.read()
>>> baconFile.close()
>>> print(content)
Hello, world!
Bacon is not a vegetable.
```

Прежде всего мы открываем файл *bacon.txt* в режиме записи. Поскольку файла *bacon.txt* пока что не существует, Python создает его. В результате вызова метода `write()` для открытого файла и передачи ему строкового аргумента `'Hello, world! /n'` осуществляется запись строки в файл и возвращается количество записанных символов, включая символы новой строки. Затем мы закрываем файл.

Чтобы дополнить новым текстом содержимое существующего файла, а не заменить только что записанную строку, мы открываем файл в режиме добавления текста. Мы записываем в файл строку `'Bacon is not a vegetable.'` и закрываем его. Наконец, чтобы вывести содержимое файла на экран, мы открываем файл в установленном по умолчании режиме чтения, вызываем метод `read()`, сохраняем результирующий объект `File` в переменной `content`, закрываем файл и выводим на экран его содержимое.

Обратите внимание на то, что метод `write()` не записывает автоматически символ новой строки в конце строки, как это делает функция `print()`. Этот символ вы должны добавлять самостоятельно.

Сохранение переменных с помощью модуля `shelve`

Используя модуль `shelve`, можно сохранять переменные в двоичных файлах-хранилищах. Благодаря этому впоследствии программа может восстановить значения переменных, читая данные с жесткого диска.

С помощью модуля `shelve` можно добавить в программу возможности Save (Сохранить) и Open (Открыть). Например, выполнив программу и введя ряд конфигурационных параметров, вы сможете сохранить их в файле хранилища и загрузить при последующем запуске программы.

Введите в интерактивной оболочке следующие команды.

```
>>> import shelve
>>> shelfFile = shelve.open('mydata')
>>> cats = ['Zophie', 'Pooka', 'Simon']
>>> shelfFile['cats'] = cats
>>> shelfFile.close()
```

Чтобы иметь возможность читать и записывать данные с помощью модуля `shelve`, прежде всего его необходимо импортировать. Вызовите метод `shelve.open()` и передайте ему имя файла, а затем сохраните возвращенное значение в переменной. Доступ к хранилищу осуществляется по ключу, как при работе со словарями. Закончив работу, вызовите метод `close()`. В данном случае значение сохраняется в переменной `shelfFile`. Мы создаем список `cats` и записываем его в хранилище с помощью инструкции `shelfFile['cats'] = cats`, которая сохраняет в переменной `shelfFile` список в виде значения, ассоциированного с ключом `'cats'`. Затем мы вызываем метод `close()` для переменной `shelfFile`.

Выполнив предыдущий код на Windows-компьютере, вы увидите в текущем рабочем каталоге три новых файла: *mydata.bak*, *mydata.dat* и *mydata.dir*. На компьютере, работающем под управлением OS X, будет создан только один файл *mydata.db*.

Описанные двоичные файлы содержат данные, которые вы сохранили в хранилище. Точный формат хранения данных в этих двоичных файлах для вас не имеет значения; вам достаточно знать лишь то, что именно делает модуль `shelve`, а не как он это делает. Данный модуль освобождает вас от всех забот, связанных с организацией хранения данных программы в файлах.

Программа может использовать модуль `shelve` для последующего открытия файлов хранилища и извлечения из них данных. Хранилища не нуждаются в открытии в режиме чтения или записи — как только хранилище открыто, вы можете выполнять оба типа операций. Введите в интерактивной оболочке следующие команды.

```
>>> shelfFile = shelve.open('mydata')
>>> type(shelfFile)
<class 'shelve.DbfilenameShelf'>
>>> shelfFile['cats']
['Zophie', 'Pooka', 'Simon']
>>> shelfFile.close()
```

Здесь мы открываем файлы хранилища для проверки того, что данные были корректно сохранены. Команда `shelfFile['cats']` возвращает тот же список, который был сохранен ранее, что подтверждает корректность сохранения данных, а метод `close()` закрывает хранилище.

Как и словари, хранилища имеют методы `keys()` и `values()`, извлекающие из хранилища коллекции ключей и значений, подобные спискам. Поскольку возвращаемые этими методами коллекции лишь подобны спискам, а не являются истинными списками, для того чтобы получать их в виде списков, их следует передавать функции `list()`. Введите в интерактивной оболочке следующие команды.

```
>>> shelfFile = shelve.open('mydata')
>>> list(shelfFile.keys())
['cats']
>>> list(shelfFile.values())
[['Zophie', 'Pooka', 'Simon']]
>>> shelfFile.close()
```

Формат простого текста удобно использовать для создания файлов, которые вы будете читать в текстовом редакторе наподобие Notepad или TextEdit. Если же вы хотите сохранять данные из своих программ на языке Python, то используйте модуль `shelve`.

Сохранение переменных с помощью функции `pprint.pformat()`

Вспомните, как в разделе "Красивая печать" в главе 5 говорилось о том, что функция `pprint.pprint()` обеспечивает "красивый" вывод содержимого списка или словаря на экран, тогда как функция `pprint.pformat()` просто возвращает тот же текст в виде строки. Эта строка не только отформатирована так, что ее удобно читать, но и представляет собой синтаксически правильный код Python. Предположим, у вас есть словарь, сохраненный в переменной, и вы хотите сохранить эту переменную и ее содержимое для будущего использования. Применив функцию `pprint.pformat()`, вы получите строку, которую можно записать в *.py*-файл. Этот файл будет вашим собственным модулем, который вы сможете импортировать всякий раз, когда захотите использовать хранящуюся в нем переменную.

Например, введите в интерактивной оболочке следующие команды.

```
>>> import pprint
>>> cats = [{'name': 'Zophie', 'desc': 'chubby'}, {'name':
↳ 'Pooka', 'desc': 'fluffy'}]
>>> pprint.pformat(cats)
```

```
"[{'desc': 'chubby', 'name': 'Zophie'}, {'desc': 'fluffy', 'name':
'Pooka'}]"
>>> fileObj = open('myCats.py', 'w')
>>> fileObj.write('cats = ' + pprint.pformat(cats) + '\n')
83
>>> fileObj.close()
```

Здесь мы импортируем модуль `pprint`, чтобы иметь возможность использовать функцию `pprint.pformat()`. У нас есть список словарей, сохраненный в переменной `cats`. Чтобы сохранить возможность обращения к списку, хранящемуся в переменной `cats`, даже после того как будет закрыта оболочка, мы используем функцию `pprint.pformat()`, возвращающую список в виде строки. Получив данные, хранящиеся переменной `cats`, в виде строки, мы сможем легко записать их в файл, который мы назовем *myCats.py*.

Модули, импортируемые с помощью инструкции `import`, сами являются не более чем обычными сценариями Python. После того как строка, возвращаемая `pprint.pformat()`, будет сохранена в *.py*-файле, этот файл станет модулем, который может быть импортирован подобно любому другому модулю.

А поскольку сценарии Python сами по себе являются простыми текстовыми файлами с расширением *.py*, ваши программы на Python могут даже генерировать другие Python-программы. Впоследствии эти файлы могут импортироваться в сценарии.

```
>>> import myCats
>>> myCats.cats
[{'name': 'Zophie', 'desc': 'chubby'}, {'name': 'Pooka', 'desc':
'fluffy'}]
>>> myCats.cats[0]
{'name': 'Zophie', 'desc': 'chubby'}
>>> myCats.cats[0]['name']
'Zophie'
```

Преимуществом создания *.py*-файлов (в отличие от сохранения переменных с помощью модуля `shelve`) является то, что, поскольку они представляют собой обычный текстовый файл, его содержимое можно читать и изменять с помощью обычного текстового редактора. Однако для большинства приложений сохранение данных с использованием модуля `shelve` является более предпочтительным способом сохранения переменных в файле. Записывать в файл в виде простого текста можно только данные элементарных типов, такие как целые числа и числа с плавающей точкой, строки, списки и словари. Объекты же `File`, например, не могут быть закодированы в виде текста.

Проект: генерация файлов случайных экзаменационных билетов

Предположим, вы читаете географию группе из 35 студентов и хотите провести контрольную работу на знание столиц штатов в США. Увы, обстановка в вашем классе такая, что вы не можете быть уверены в том, что студенты не будут списывать друг у друга. Вы хотели бы составить экзаменационные билеты таким образом, чтобы вопросы в них располагались в случайном порядке, благодаря чему каждый билет будет отличаться от другого, и это затруднит списывание ответов. Разумеется, составлять такие билеты вручную — задача утомительная и к тому же отнимающая много времени. К счастью, вы уже освоили некоторые возможности Python.

Вот примерный план действий, которые должна выполнять программа:

- создать 35 различных билетов;
- создать по 50 вопросов с множественным выбором для каждого билета, расположив их в случайном порядке;
- предоставить правильный ответ и три случайно выбранных неправильных ответа на каждый вопрос, располагая их в случайном порядке;
- записать билеты в 35 текстовых файлов;
- записать ключи ответов в 35 текстовых файлов.

Это означает, что код должен будет выполнять следующие операции:

- сохранять названия штатов и их столиц в словаре;
- вызывать методы `open()`, `write()` и `close()` для текстовых файлов, в которых хранятся билеты и ключи ответов;
- использовать функцию `random.shuffle()` для рандомизации (расположения в случайном порядке следования) вопросов и вариантов множественного выбора.

Шаг 1. Сохранение данных билетов в словаре

Первый шаг состоит в том, чтобы создать "скелет" сценария и наполнить его данными билета. Создайте файл *randomQuizGenerator.py* и введите в него следующий текст.

```
#! python3
# randomQuizGenerator.py - Создает экзаменационные билеты с
# вопросами и ответами, расположенными в случайном порядке,
# вместе с ключами ответов.

❶ import random
# Данные билета. Ключи - названия штатов, а значения - столицы.
❷ capitals = {'Alabama': 'Montgomery', 'Alaska': 'Juneau',
'Arizona': 'Phoenix', 'Arkansas': 'Little Rock', 'California':
```

```
    'Sacramento', 'Colorado': 'Denver', 'Connecticut': 'Hartford',
    'Delaware': 'Dover', 'Florida': 'Tallahassee', 'Georgia':
    'Atlanta', 'Hawaii': 'Honolulu', 'Idaho': 'Boise', 'Illinois':
    'Springfield', 'Indiana': 'Indianapolis', 'Iowa': 'Des Moines',
    'Kansas': 'Topeka', 'Kentucky': 'Frankfort', 'Louisiana':
    'Baton Rouge', 'Maine': 'Augusta', 'Maryland': 'Annapolis',
    'Massachusetts': 'Boston', 'Michigan': 'Lansing', 'Minnesota':
    'Saint Paul', 'Mississippi': 'Jackson', 'Missouri':
    'Jefferson City', 'Montana': 'Helena', 'Nebraska': 'Lincoln',
    'Nevada': 'Carson City', 'New Hampshire': 'Concord',
    'New Jersey': 'Trenton', 'New  Mexico': 'Santa Fe',
    'New York': 'Albany', 'North Carolina': 'Raleigh',
    'North Dakota': 'Bismarck', 'Ohio': 'Columbus', 'Oklahoma':
    'Oklahoma City', 'Oregon': 'Salem', 'Pennsylvania':
    'Harrisburg', 'Rhode Island': 'Providence', 'South Carolina':
    'Columbia', 'South Dakota': 'Pierre', 'Tennessee': 'Nashville',
    'Texas': 'Austin', 'Utah': 'Salt Lake City', 'Vermont':
    'Montpelier', 'Virginia': 'Richmond', 'Washington': 'Olympia',
    'West Virginia': 'Charleston', 'Wisconsin': 'Madison',
    'Wyoming': 'Cheyenne'}

    # Генерация 35 файлов билетов.
❸   for quizNum in range(35):
        # TODO: Создать файлы билетов и ключей ответов.

        # TODO: Записать заголовок билета.

        # TODO: Перемешать порядок следования штатов.

        # TODO: Организовать цикл по всем 50 штатам,
        # создавая вопрос для каждого из них.
```

Поскольку эта программа должна располагать вопросы и ответы в случайном порядке, вам понадобится импортировать модуль `random` ❶, чтобы использовать его функции. Переменная `capitals` ❷ содержит словарь, в котором штаты США играют роль ключей, а значениями являются названия столиц штатов. А поскольку вы хотите создать 35 билетов, код, который будет фактически генерировать файлы билетов и ключей ответов (на данном этапе отмечен комментариями TODO), должен быть помещен в цикл `for`, выполняющий 35 итераций ❸. (Это число можно изменить для генерации любого заданного количества билетов.)

Шаг 2. Создание файлов билетов и перемешивание вопросов

А теперь настал черед приступить к замене комментариев TODO реальным кодом.

Код в цикле будет повторен 35 раз — по одному разу на каждый билет, в связи с чем вам достаточно сосредоточивать внимание в цикле каждый раз

только на одном билете. Прежде всего, необходимо создать фактический файл билета. Он должен иметь уникальное имя, а также содержать некий стандартный заголовок с пустыми полями для имени, даты и класса, которые будут заполняться студентами. Далее вам нужно будет получить список штатов, расположенных в случайном порядке, который впоследствии можно будет использовать для создания вопросов и ответов к каждому билету.

Добавьте в файл *randomQuizGenerator.py* указанные ниже строки кода.

```
#! python3
# randomQuizGenerator.py - Создает экзаменационные билеты с
# вопросами и ответами, расположенными в случайном порядке,
# вместе с ключами ответов.

--пропущенный код--

# Генерация 35 файлов билетов.
for quizNum in range(35):
# Создание файлов билетов и ключей ответов.
❶ quizFile = open('capitalsquiz%s.txt' % (quizNum + 1), 'w')
❷ answerKeyFile = open('capitalsquiz_answers%s.txt' % (quizNum +
1), 'w')
# Запись заголовка билета.
❸ quizFile.write('Имя:\n\nДата:\n\nКурс:\n\n')
quizFile.write((' ' * 15) +
    'Проверка знания столиц штатов (Билет %s)' % (quizNum + 1))
quizFile.write('\n\n')
# Перемешивание порядка следования столиц штатов.
states = list(capitals.keys())
❹ random.shuffle(states)
# TODO: Организовать цикл по всем 50 штатам,
#       создавая вопрос для каждого из них.
```

Файлы будут иметь имена *capitalsquiz<N>.txt*, где *<N>* — это уникальный номер билета, который берется из переменной цикла `quizNum`. Ключи ответов для файлов *capitalsquiz<N>.txt* будут храниться в текстовых файлах *capitalsquiz_answers<N>.txt*. При каждом прохождении цикла вместо заместителя `%s` в строках `'capitalsquiz%s.txt'` и `'capitalsquiz_answers%s.txt'` будет подставляться значение `(quizNum + 1)`, поэтому файлами первого из создаваемых билетов и ключа ответа будут *capitalsquiz1.txt* и *capitalsquiz_answers1.txt*. Эта файлы будут создаваться вызовами функции `open()` в инструкциях ❶ и ❷ с передачей им строки `'w'` в качестве второго аргумента для открытия файлов в режиме записи.

Инструкции `write()` ❸ создают заголовок билета с полями, которые будут заполняться студентами. Наконец, с помощью функции `random.shuffle()` ❹, которая случайным образом переупорядочивает список любых переданных ей значений, создается рандомизированный список всех штатов США.

Шаг 3. Создание вариантов ответов

Теперь необходимо сгенерировать варианты ответов для каждого вопроса, предоставляя возможность выбора одного из ответов, обозначенных буквами от A до D. Вам понадобится создать еще один цикл `for` — он будет генерировать содержимое для каждого из 50 вопросов билета. Далее будет третий, вложенный цикл `for`, предназначенный для генерации вариантов множественного выбора для каждого вопроса. Дополните имеющийся код, как показано ниже.

```
#! python3
# randomQuizGenerator.py - Создает экзаменационные билеты с
# вопросами и ответами, расположенными в случайном порядке,
# вместе с ключами ответов.

--пропущенный код--

    # Организация цикла по всем 50 штатам
    # с созданием вопроса для каждого из них.
    for questionNum in range(50):

        # Получение правильных и неправильных ответов.
❶       correctAnswer = capitals[states[questionNum]]
❷       wrongAnswers = list(capitals.values())
❸       del wrongAnswers[wrongAnswers.index(correctAnswer)]
❹       wrongAnswers = random.sample(wrongAnswers, 3)
❺       answerOptions = wrongAnswers + [correctAnswer]
❻       random.shuffle(answerOptions)

        # TODO: Записать варианты вопросов и ответов
        # в файл билета.

        # TODO: Записать ключ ответа в файл.
```

Корректный ответ можно легко получить — он хранится в виде значения в словаре `capitals` ❶. Данный цикл итерирует по штатам, содержащимся в перемешанном списке штатов, от `states[0]` до `states[49]`, находит каждый штат в `capitals` и сохраняет название его столицы в переменной `correctAnswer`.

Со списком возможных неправильных ответов дело обстоит несколько сложнее. Вы сможете получить его, продублировав все значения из словаря `capitals` ❷, удалив правильный ответ ❸ и выбрав три случайных значения из этого списка ❹. Функция `random.sample()` упрощает этот выбор. Ее первый аргумент — это список, из которого вы хотите выбирать значения; второй аргумент — это количество значений, которые вы хотите выбрать. Полный список вариантов ответа представляет собой сочетание трех непра-

вильных ответов с правильным ❺. Наконец, ответы следует перемешать ❻, чтобы правильный ответ не всегда соответствовал варианту D.

Шаг 4. Запись содержимого в файлы билетов и ключей ответов

Теперь все, что осталось сделать, — это записать вопрос в файл билета, а ответ — в файл ключей ответов. Дополните код, как показано ниже.

```
#! python3
# randomQuizGenerator.py - Создает экзаменационные билеты с
# вопросами и ответами, расположенными в случайном порядке,
# вместе с ключами ответов.

--пропущенный код--

    # Организация цикла по всем 50 штатам
    # с созданием вопроса для каждого из них.
    for questionNum in range(50):
        --пропущенный код--

        # Запись вариантов вопросов и ответов в файл билета.
        quizFile.write('%s. Выберите столицу штата %s.\n' %
            (questionNum + 1, states[questionNum]))
❶       for i in range(4):
❷           quizFile.write(' %s. %s\n' % ('ABCD'[i],
                           answerOptions[i]))
        quizFile.write('\n')

        # Запись ключа ответа в файл.
❸       answerKeyFile.write('%s. %s\n' % (questionNum + 1,
            'ABCD'[answerOptions.index(correctAnswer)]))
    quizFile.close()
    answerKeyFile.close()
```

Цикл `for`, перебирающий целые числа от `0` до `3`, записывает варианты ответов в список `answerOptions` ❶. В выражении `'ABCD'[i]` ❷ строка `'ABCD'` трактуется как массив с элементами `'A'`, `'B'`, `'C'` и `'D'`, выбираемыми на соответствующей итерации цикла.

В последней строке ❸ выражение `answerOptions.index(correctAnswer)` находит целочисленный индекс правильного ответа среди случайно расположенных вариантов, а вычисление выражения `'ABCD'[answerOptions.index(correctAnswer)]` дает буквенное обозначение правильного варианта ответа, подлежащего записи в файл ключа ответа.

Ниже показан примерный вид содержимого файла *capitalsquiz1.txt*, хотя, разумеется, вопросы и варианты ответов в вашем файле будут выглядеть иначе, в зависимости от результатов вызова функции `random.shuffle()`.

```
Имя:
Дата:
Курс:
                Проверка знания столиц штатов (Билет 1)
1.Выберите столицу штата West Virginia.
    A. Hartford
    B. Santa Fe
    C. Harrisburg
    D. Charleston

2. Выберите столицу штата Colorado.
    A. Raleigh
    B. Harrisburg
    C. Denver
    D. Lincoln

--опущено--
```

Соответствующий текстовый файл *capitalsquiz_answers1.txt* будет выглядеть примерно так.

```
1. D
2. C
3. A
4. C
--опущено--
```

Проект: буфер обмена для работы с несколькими значениями

Предположим, вам предстоит утомительная работа по заполнению ряда форм на веб-странице или многочисленных текстовых полей в программе. Буфер обмена обеспечивает экономию времени, избавляя вас от необходимости повторного ввода одного и того же текста. Вместе с тем в него можно копировать только один фрагмент текста за один раз. Если же у вас имеется несколько различных текстовых фрагментов, подлежащих копированию и вставке, то вам приходится каждый раз заново выделять и копировать одни и те же фрагменты.

Однако можно написать программу на языке Python, которая будет отслеживать различные фрагменты текста. Мы назовем этот "многозарядный" буфер ("multiclipboard") *mcb.pyw* (поскольку "mcb" короче, чем "multiclipboard"). Расширение *.pyw* означает, что Python не будет отображать окно терминала в процессе выполнения программы. (Более подробно об этом читайте в приложении Б.)

Программа будет сохранять каждый фрагмент копируемого в буфер текста с использованием своего ключевого слова. Например, если вы выполните команду `py mcb.pyw save spam`, то текущее содержимое буфера обмена будет сохранено с ключевым словом `spam`. Впоследствии этот текст можно будет вновь загрузить в буфер обмена с помощью команды `py mcb.pyw spam`. А если пользователь забудет, какие ключевые слова соответствуют тем или иным текстовым фрагментам, то он сможет выполнить команду `py mcb.pyw list` для копирования списка всех ключевых слов в буфер обмена.

Вот что делает данная программа:

- проверяет аргумент командной строки для ключевого слова;
- если этот аргумент — `save`, то содержимое буфера обмена сохраняется с данным ключевым словом;
- если этот аргумент — `list`, то все ключевые слова копируются в буфер обмена;
- в противном случае текст, соответствующий ключевому слову, копируется в буфер обмена.

Это означает, что код должен выполнять следующие действия:

- читать аргументы командной строки из переменной `sys.argv`;
- выполнять операции чтения и записи в буфер обмена;
- сохранять и загружать текст в файл хранилища.

Если вы работаете в Windows, то вам будет легко выполнить этот сценарий из окна Run (Выполнить), создав пакетный файл *mcb.bat* со следующим содержимым:

```
@pyw.exe C:\Python34\mcb.pyw %*
```

Шаг 1. Комментарии и настройка хранилища

Начнем с создания каркаса сценария, содержащего некоторые комментарии и базовые настройки. Создайте следующий код.

```
#! python3
# mcb.pyw - Сохраняет и загружает фрагменты текста
# в буфер обмена.
❶ # Использование: py.exe mcb.pyw save <ключевое_слово> -
#                       Сохраняет буфер обмена в ключевое слово.
#                  py.exe mcb.pyw <ключевое_слово> -
#                       Загружает ключевое слово в буфер обмена.
#                  py.exe mcb.pyw list -
#                       Загружает все ключевые слова в буфер
#                       обмена.
```

```
❷ import shelve, pyperclip, sys

❸ mcbShelf = shelve.open('mcb')

   # TODO: Сохранить содержимое буфера обмена.

   # TODO: Сформировать список ключевых слов и загрузить
   #       содержимое.

   mcbShelf.close()
```

Общепринятой практикой является размещение общей информации о порядке использования программы, оформленной в виде комментариев в начале файла ❶. Если вы вдруг забудете, как выполнить сценарий, взгляните на комментарий. Затем импортируются необходимые модули ❷. Для копирования и вставки текста потребуется модуль `pyperclip`, а для чтения аргументов командной строки — модуль `sys`. Также потребуется модуль `shelve`: всякий раз, когда пользователь захочет сохранить новый фрагмент находящегося в буфере обмена текста, вы сохраните его в файле хранилища. Далее, если пользователь захочет поместить текст обратно в буфер, вы откроете файл хранилища и загрузите его в программу. Имя файла хранилища будет содержать префикс `mcb` ❸.

Шаг 2. Создание содержимого буфера обмена, ассоциируемого с ключевым словом

Программа выполняет различные действия в зависимости от того, чего хочет пользователь: сохранить текст, ассоциируя его с ключевым словом, загрузить текст в буфер или вывести список всех имеющихся ключевых слов. Рассмотрим первый случай. Дополните код, как показано ниже.

```
   #! python3
   # mcb.pyw - Сохраняет и загружает фрагменты текста
   # в буфер обмена.
 --пропущенный код--

   # Сохранение содержимого буфера обмена.
❶ if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
❷         mcbShelf[sys.argv[2]] = pyperclip.paste()
   elif len(sys.argv) == 2:
❸     # Сформировать список ключевых слов и загрузить содержимое.

mcbShelf.close()
```

Если первым аргументом командной строки (который всегда будет иметь индекс 1 в списке `sys.argv`) является `'save'` ❶, то вторым аргументом

является ключевое слово для текущего содержимого буфера обмена. Ключевое слово будет использоваться в качестве ключа для хранилища `mcbShelf`, тогда как значением будет текст, находящийся в данный момент в буфере обмена ❷.

Если предоставлен только один аргумент командной строки, то вы предполагаете, что это либо строка `'list'`, либо ключевое слово для загрузки содержимого в буфер обмена. Этот код вы реализуете позднее. А пока что оставьте в этом месте кода соответствующий комментарий TODO ❸.

Шаг 3. Список ключевых слов и загрузка содержимого, ассоциированного с ключевым словом

Наконец, реализуем два оставшихся пункта: загрузка в буфер обмена текста, ассоциированного с определенным ключевым словом, и вывод списка всех доступных ключевых слов. Дополните код, как показано ниже.

```
#! python3
# mcb.pyw - Сохраняет и загружает фрагменты текста
# в буфер обмена.
--пропущенный код--

# Сохранение содержимого буфера обмена.
if len(sys.argv) == 3 and sys.argv[1].lower() == 'save':
        mcbShelf[sys.argv[2]] = pyperclip.paste()
elif len(sys.argv) == 2:
    # Формирование списка ключевых слов и загрузка содержимого.
❶   if sys.argv[1].lower() == 'list':
❷       pyperclip.copy(str(list(mcbShelf.keys())))
    elif sys.argv[1] in mcbShelf:
❸       pyperclip.copy(mcbShelf[sys.argv[1]])

mcbShelf.close()
```

Если имеется только один аргумент командной строки, то прежде всего необходимо проверить, является ли им строка `'list'` ❶. Если это действительно так, то в буфер обмена копируется строковое представление списка ключей хранилища ❷. Пользователь может вставить этот список в окно открытого текстового редактора и прочитать его.

В противном случае можно полагать, что аргумент командной строки является ключевым словом. Если это ключевое слово существует в виде ключа хранилища `mcbShelf`, можно загрузить соответствующее значение в буфер обмена ❸.

Вот и все! В зависимости от установленной на компьютере операционной системы эта программа может запускаться по-разному. Детали запуска программ в различных операционных системах описаны в приложении Б.

Вспомните программу парольной защиты, сохраняющую пароли в словаре, которую мы создали в главе 6. Обновление паролей требовало изменения исходного кода программы. Это далеко не идеальный вариант, поскольку пользователи не могут чувствовать себя комфортно, если для обновления программы им приходится самостоятельно вносить изменения в код. Кроме того, всякий раз, когда приходится изменять исходный код программы, существует риск случайного внесения в нее новых ошибок. Сохраняя данные для программы не в коде, а в другом месте, вы облегчаете использование программы другими людьми и снижаете вероятность появления в ней новых ошибок.

Резюме

Файлы организуются в папки (другое название — каталоги), и их расположение описывается путями доступа. Каждая программа, выполняющаяся на вашем компьютере, имеет текущий рабочий каталог, что позволяет указывать пути относительно текущего расположения вместо того, чтобы всегда задавать полный (или абсолютный) путь. Модуль `os.path` содержит множество функций, предназначенных для манипулирования путями доступа к файлам.

Ваши программы имеют возможность непосредственно взаимодействовать с содержимым текстовых файлов. Функция `open()` позволяет открывать эти файлы для чтения их содержимого в виде одной длинной строки (с помощью метода `read()`) или в виде списка строк (с помощью метода `readlines()`). Функция `open()` может открывать файлы в режиме записи или присоединения для создания новых текстовых файлов или добавления текста в конец существующих файлов соответственно.

В предыдущих главах вы использовали буфер обмена в качестве средства, позволяющего вставлять в программу готовый текст, а не вводить его вручную. Теперь же вы научились читать необходимые программе данные непосредственно с жесткого диска, что является большим достижением, поскольку хранить данные в файлах гораздо надежнее, чем в буфере обмена.

Из следующей главы вы узнаете о том, как обрабатывать сами файлы, т.е. выполнять такие операции, как копирование, удаление, переименование, перемещение файлов и многие другие.

Контрольные вопросы

1. Относительно чего задается относительный путь?
2. С чего начинается абсолютный путь?
3. Каково назначение функций `os.getcwd()` и `os.chdir()`?

4. Что собой представляют папки . и ..?
5. Какие части пути *C:\bacon\eggs\spam.txt* представляют имя папки и базовое имя?
6. Назовите три возможных значения аргумента, задающие режим открытия файла, которые могут передаваться функции `open()`.
7. Что происходит при открытии существующего файла в режиме записи?
8. Чем различаются методы `read()` и `readlines()`?
9. Какую структуру данных напоминает организация хранилища, создаваемого с помощью модуля `shelve`?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Расширение возможностей буфера обмена, рассчитанного на работу с несколькими значениями

Расширьте возможности программы для работы с несколькими значениями через буфер обмена таким образом, чтобы она допускала использование аргумента командной строки вида `delete <ключевое_слово>`, обеспечивающего удаление ключевого слова из хранилища. Затем добавьте аргумент командной строки `delete`, позволяющий удалить *все* ключевые слова.

Программа Mad Libs

Создайте программу Mad Libs, которая читает текстовые файлы и предоставляет пользователю возможность добавлять собственный текст в любом месте файла, где встречаются слова `ADJECTIVE` (прилагательное), `NOUN` (существительное), `ADVERB` (наречие) и `VERB` (глагол). Например, содержимое текстового файла может иметь следующий вид:

```
The ADJECTIVE panda walked to the NOUN and then VERB. A nearby
NOUN was unaffected by these events.
```

Программа найдет вхождения перечисленных слов и предложит пользователю заменить их.

```
Введите имя прилагательное:
silly
Введите имя существительное:
chandelier
```

```
Введите глагол:
screamed
Введите имя существительное:
pickup truck
```

В результате будет создан следующий текстовый файл:

```
The silly panda walked to the chandelier and then screamed. A
nearby pickup truck was unaffected by these events.
```

Результаты должны выводиться на экран и сохраняться в новом текстовом файле.

Поиск с помощью регулярных выражений

Напишите программу, которая открывает все файлы с расширением *.txt*, находящиеся в папке, и выполняет поиск строк, соответствующих предоставленному пользователем регулярному выражению. Результаты должны выводиться на экран.

9

УПРАВЛЕНИЕ ФАЙЛАМИ

В предыдущей главе вы научились создавать и записывать на жесткий диск новые файлы в Python. Но ваши программы могут реорганизовывать и файлы, которые уже существуют на жестком диске. Возможно, вам приходилось работать с папками, хранящими десятки, сотни и даже тысячи файлов, которые нужно было копировать, переименовывать, перемещать или сжимать вручную. Задачи подобного рода могут быть самыми разнообразными, включая, например, следующие:

- создание копий всех PDF-файлов (и *только* PDF-файлов) во всех подпапках заданной папки;
- удаление ведущих нулей из имен всех файлов наподобие *spam001.txt*, *spam002.txt*, *spam003.txt* и так далее, сохраненных в некоторой папке в количестве, исчисляемом сотнями;
- сжатие содержимого нескольких папок в один ZIP-файл (что может быть использовано для создания простейшей системы для создания резервных копий файлов).

Подобные рутинные задачи так и просятся, чтобы их автоматизировали с помощью Python. Программируя свой компьютер для выполнения такого рода задач, вы легко превратите его в расторопного, безошибочно функционирующего офисного клерка.

Начав работать с файлами, вы вскоре поймете, насколько удобно иметь возможность непосредственно видеть, какое расширение (*.txt*, *.pdf*, *.jpg* и др.) имеет тот или иной файл. В операционных системах OS X и Linux обозреватель файлов в большинстве случаев автоматически отображает расширения имен. В случае же Windows расширения имен файлов по умолчанию могут скрываться. Чтобы отобразить их, выберите пункты меню Пуск⇨Панель управления⇨Оформление и персонализация⇨Параметры папок. Перейдите в открывшемся окне на вкладку Вид и снимите флажок Скрывать расширения для зарегистрированных типов файлов в разделе Дополнительные параметры.

Модуль shutil

Модуль shutil (от англ. "shell utilities" — утилиты командной оболочки) содержит функции, позволяющие копировать, перемещать, переименовывать и удалять файлы с помощью программ на Python. Прежде чем использовать эти утилиты, необходимо выполнить инструкцию import shutil.

Копирование файлов и папок

Модуль shutil предоставляет функции для копирования как файлов, так и целых папок.

Вызов shutil.copy(*исходный_путь*, *путь_назначения*) приведет к копированию файла, расположение которого определяется путем *исходный_путь*, в папку, определяемую путем *путь_назначения*. (Параметры *исходный_путь* и *путь_назначения* являются строками.) Если аргумент *путь_назначения* — это имя файла, то оно будет использовано в качестве нового имени скопированного файла. Эта функция возвращает строку, содержащую путь к скопированному файлу.

Чтобы посмотреть, как работает функция shutil.copy(), введите в интерактивной оболочке следующие команды.

```
>>> import shutil, os
>>> os.chdir('C:\\')
❶ >>> shutil.copy('C:\\spam.txt', 'C:\\delicious')
'C:\\delicious\\spam.txt'
❷ >>> shutil.copy('eggs.txt', 'C:\\delicious\\eggs2.txt')
'C:\\delicious\\eggs2.txt'
```

Первый вызов функции shutil.copy() копирует файл *C:\spam.txt* в папку *C:\delicious*. Возвращаемым значением является путь к скопированному файлу. Обратите внимание на то, что в качестве объекта назначения указана папка ❶, а имя копии файла совпадает с именем исходного файла *spam.txt*. Второй вызов shutil.copy() ❷ также выполняет копирование файла *C:\eggs.txt* в папку *C:\delicious*, но копии файла присваивается имя *eggs2.txt*.

В то время как функция shutil.copy() копирует одиночный файл, функция shutil.copytree() копирует папку вместе со всеми папками и файлами, которые в ней содержатся. В результате вызова shutil.copytree(*исходный_путь*, *путь_назначения*) папка, находящаяся в расположении *исходный_путь*, копируется вместе со всеми находящимися в ней файлами и подпапками в расположение *путь_назначения*. Параметры *исходный_путь* и *путь_назначения* являются строками. Функция возвращает строку, представляющую путь к копии папки.

Введите в интерактивной оболочке следующие команды.

```
>>> import shutil, os
>>> os.chdir('C:\\')
>>> shutil.copytree('C:\\bacon', 'C:\\bacon_backup')
'C:\\bacon_backup'
```

В результате вызова `shutil.copytree()` создается новая папка *bacon_backup* с таким же содержимым, как и содержимое исходной папки *bacon*. Теперь у вас есть резервная копия вашей бесценной папки *bacon*.

Перемещение и переименование файлов и папок

Вызов функции `shutil.move(`*исходный_путь*`, `*путь_назначения*`)` перемещает файл или папку из расположения *исходный_путь* в расположение *путь_назначения* и возвращает строку, представляющую абсолютный путь к новому расположению.

Если параметр *путь_назначения* представляет папку, исходный файл перемещается в папку назначения и сохраняет свое текущее имя. Например, введите в интерактивной оболочке следующие команды.

```
>>> import shutil
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs\\bacon.txt'
```

В предположении, что папка *eggs* существует в каталоге *C:*, вызов `shutil.move()` в словесной формулировке означает следующее: "Переместить файл *C:\bacon.txt* в папку *C:\eggs*".

Если в папке *C:\eggs* уже существует файл *bacon.txt*, то он будет заменен. Поскольку таким образом можно очень легко случайно потерять нужный файл, вы должны проявлять определенную осторожность, когда используете функцию `move()`.

Параметр *путь_назначения* также может задавать имя файла. В следующем примере исходный файл перемещается и переименовывается.

```
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs\\new_bacon.txt')
'C:\\eggs\\new_bacon.txt'
```

Эта строка кода имеет следующий смысл: "Переместить файл *C:\bacon.txt* в папку *C:\eggs* и присвоить перемещенному файлу *bacon.txt* новое имя *new_bacon.txt*".

Оба предыдущих примера работают в предположении, что в каталоге *C:* существует папка *eggs*. Но если это не так, то функция `move()` переименует файл *bacon.txt* в файл *eggs*.

```
>>> shutil.move('C:\\bacon.txt', 'C:\\eggs')
'C:\\eggs'
```

Здесь функция `move()`, не найдя папку *eggs* в каталоге *C:*, предполагает, что путь назначения должен обозначать имя файла, а не папки. Поэтому текстовый файл *bacon.txt* переименовывается в файл *eggs* (текстовый файл, но без расширения *.txt*) — вероятно, вразрез с вашими намерениями! Разглядеть подобную ошибку в программе очень трудно, поскольку вызов `move()` может беспрепятственно сделать то, чего вы совершенно не ожидали. Это еще одна из причин, по которым работа с функцией `move()` требует осторожности.

Наконец, задаваемые в качестве пути назначения папки должны существовать, иначе Python сгенерирует исключение. Введите в интерактивной оболочке следующую команду.

```
>>> shutil.move('spam.txt', 'c:\\does_not_exist\\eggs\\ham')
Traceback (most recent call last):
  File "C:\Python34\lib\shutil.py", line 521, in move
    os.rename(src, real_dst)
FileNotFoundError: [WinError 3] The system cannot find the path
specified:
'spam.txt' -> 'c:\\does_not_exist\\eggs\\ham'
```

В процессе обработки вышеприведенного исключения возникает другое исключение.

```
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    shutil.move('spam.txt', 'c:\\does_not_exist\\eggs\\ham')
  File "C:\Python34\lib\shutil.py", line 533, in move
    copy2(src, real_dst)
  File "C:\Python34\lib\shutil.py", line 244, in copy2
    copyfile(src, dst, follow_symlinks=follow_symlinks)
  File "C:\Python34\lib\shutil.py", line 108, in copyfile
    with open(dst, 'wb') as fdst:
FileNotFoundError: [Errno 2] No such file or directory:
'c:\\does_not_exist\\
eggs\\ham'
```

Python ищет *eggs* и *ham* в каталоге *does_not_exist*. Поскольку найти несуществующие каталоги ему не удается, он не может переместить файл *spam.txt* по указанному пути.

Безвозвратное удаление файлов и папок

Если для удаления одиночного файла или одиночной пустой папки можно воспользоваться функциями, содержащимися в модуле `os`, то для удаления папки вместе со всем ее содержимым следует использовать модуль `shutil`.

- Вызов `os.unlink(`*путь*`)` удаляет файл, расположенный по указанному пути.
- Вызов `os.rmdir(`*путь*`)` удаляет папку, расположенную по указанному пути. Эта папка должна быть пуста, т.е. не должна содержать никаких других папок и файлов.
- Вызов `shutil.rmtree(`*путь*`)` удаляет папку, расположенную по указанному пути, вместе со всеми содержащимися в ней другими папками и файлами.

Используя эти функции в своих программах, соблюдайте осторожность! Часто имеет смысл предварительно запустить версию программы, в которой эти вызовы "закомментированы", а контроль того, какие файлы будут удаляться, осуществляется с помощью функции `print()`. Ниже приведен фрагмент программы на языке Python, предназначенный для удаления файлов с расширением `.txt`, но из-за допущенной опечатки (выделена полужирным шрифтом) удаляющий файлы с расширением *.rxt*.

```
import os
for filename in os.listdir():
    if filename.endswith('.rxt'):
        os.unlink(filename)
```

Если бы у вас были важные файлы, имена которых заканчиваются расширением *.rxt*, то все они были бы безвозвратно удалены. Вместо этого следовало бы сначала запустить программу с измененной версией этого фрагмента.

```
import os
for filename in os.listdir():
    if filename.endswith('.rxt'):
        #os.unlink(filename)
        print(filename)
```

Теперь вызов `os.unlink()` не будет выполняться, поскольку он включен в текст комментария ("закомментирован"), и в силу этого Python проигнорирует его. В таком случае вместо фактического удаления файла будет лишь выведено имя файла, подлежащего удалению. Предварительно выполнив эту версию программы, вы сразу же обнаружите, что в программе имеется

ошибка, из-за которой она будет ошибочно удалять не текстовые файлы с расширением *.txt*, а файлы с расширением *.rxt*.

Убедившись в том, что программа работает так, как запланировано, удалите строку `print(`*имя_файла*`)`, а также символ комментария в строке с инструкцией `os.unlink(`*имя_файла*`)`. После этого вновь запустите программу для удаления файлов.

Сохраняйте резервные копии удаленных файлов и папок с помощью модуля `send2trash`

Поскольку встроенная функция Python `shutil.rmtree()` необратимо удаляет файлы и папки, ее использование связано с определенным риском. Намного лучший способ удаления файлов и папок предлагает модуль `send2trash` от независимых разработчиков. Этот модуль можно установить, выполнив команду `pip install send2trash` в окне терминала. (Более подробная информация о порядке установки модулей, разработанных сторонними компаниями, приведена в приложении А.)

Модуль `send2trash` намного безопаснее в использовании, чем обычные функции Python, выполняющие операцию удаления, поскольку он отправляет удаляемые файлы и папки в корзину компьютера или в специальную корзину, а не удаляет их безвозвратно. Если из-за ошибок в программе будут удалены файлы, которые вы не собирались удалять, но при этом использовался модуль `send2trash`, то впоследствии у вас будет возможность восстановить их из специальной корзины.

После того как вы установите модуль `send2trash`, введите в интерактивной оболочке следующие команды.

```
>>> import send2trash
>>> baconFile = open('bacon.txt', 'a') # создает файл
>>> baconFile.write('Bacon is not a vegetable.')
25
>>> baconFile.close()
>>> send2trash.send2trash('bacon.txt')
```

Вообще говоря, целесообразно всегда использовать функцию `send2trash.send2trash()` для удаления файлов и папок. Однако, несмотря на то что отправка файлов в специальную корзину оставляет вам возможность их последующего восстановления, размер свободного дискового пространства при этом не увеличивается, как в случае безвозвратного удаления файлов. Если у вас возникает потребность в освобождении дискового пространства, используйте для удаления файлов и папок функции модулей `os` и `shutil`. Обратите внимание на то, что функция `send2trash()` может лишь отправлять файлы в корзину, но не извлекать их из нее.

Обход дерева каталогов

Предположим, вы хотите переименовать все файлы, находящиеся в некоторой папке, а также во всех ее подпапках. Следовательно, вам необходимо выполнить обход всего дерева каталогов, обрабатывая при этом каждый файл. К сожалению, написание соответствующей программы — задача нетривиальная, и потому Python предоставляет функцию, организующую этот процесс вместо вас.

Рассмотрим папку *C:\delicious* и все ее содержимое (рис. 9.1).

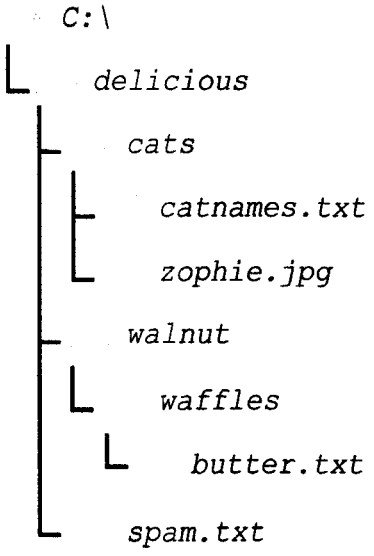


Рис. 9.1. Пример папки, содержащей три другие папки и четыре файла

Ниже приведен пример программы, в которой для обхода дерева каталогов, представленного на рис. 9.1, используется функция `os.walk()`.

```
import os

for folderName, subfolders, filenames in os.walk('C:\\delicious'):
    print('Текущая папка - ' + folderName)

    for subfolder in subfolders:
        print('ПОДПАПКА ПАПКИ ' + folderName + ': ' + subfolder)

    for filename in filenames:
        print('ФАЙЛ В ПАПКЕ ' + folderName + ': '+ filename)

    print('')
```

Функции `os.walk()` передается единственное строковое значение — путь к папке. Вы можете использовать функцию `os.walk()` в цикле `for` для обхода дерева каталогов во многом примерно так, как функцию `range()` можно использовать для перебора всех целых чисел, принадлежащих некоторому диапазону. Однако, в отличие от функции `range()`, функция `os.walk()` возвращает три значения на каждой итерации цикла:

1) строку, содержащую текущее имя папки;
2) список строк, представляющих имена папок, которые содержатся в текущей папке;
3) список строк, представляющих имена файлов, которые содержатся в текущей папке.

(Под текущей папкой я подразумеваю папку, используемую в текущей итерации цикла. Применение функции `os.walk()` не приводит к смене текущего рабочего каталога программы.)

Подобно тому как допускается выбрать имя переменной `i` в коде `for i in range(10):`, можно выбирать имена переменных для трех вышеперечисленных значений. Обычно в качестве таковых я использую соответственно имена *foldername*, *subfolders* и *filenames*.

Если вы выполните эту программу, то ее вывод будет выглядеть так.

```
'Текущая папка - C:\delicious
ПОДПАПКА ПАПКИ C:\delicious: cats
ПОДПАПКА ПАПКИ C:\delicious: walnut
ФАЙЛ В ПАПКЕ C:\delicious: spam.txt

'Текущая папка - C:\delicious\cats
ФАЙЛ В ПАПКЕ C:\delicious\cats: catnames.txt
ФАЙЛ В ПАПКЕ C:\delicious\cats: zophie.jpg

'Текущая папка - C:\delicious\walnut
ПОДПАПКА ПАПКИ C:\delicious\walnut: waffles

'Текущая папка - C:\delicious\walnut\waffles
ФАЙЛ В ПАПКЕ C:\delicious\walnut\waffles: butter.txt.
```

Поскольку функция `os.walk()` возвращает списки строк для переменных `subfolder` и `filename`, можно использовать эти списки в их собственных циклах `for`. Замените вызовы функции `print()` собственным пользовательским кодом. (Или же удалите циклы `for`, если вам не нужен какой-то один из этих циклов или оба цикла.)

Сжатие файлов с помощью модуля zipfile

Вероятно, вы знакомы с ZIP-файлами (имеющими расширение *.zip*), в которых в сжатом виде может храниться содержимое многих других файлов. В результате сжатия файла его размер уменьшается, что немаловажно при передаче файлов через Интернет. А поскольку один ZIP-файл может содержать множество файлов и папок, эта возможность очень удобна для упаковки нескольких файлов в один. Этот единственный файл (так называемый *архивный файл*) можно, например, присоединить к сообщению электронной почты.

Ваши Python-программы могут как создавать, так и открывать (или *распаковывать*) ZIP-файлы с помощью функций из модуля `zipfile`. Предположим, у вас имеется ZIP-файл *example.zip*, содержимое которого представлено на рис. 9.2.

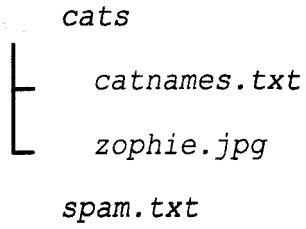


Рис. 9.2. Содержимое файла example.zip

Можете скачать этот файл по адресу `http://nostarch.com/automatestuff/` или просто использовать один из ZIP-файлов, которые уже имеются на вашем компьютере.

Чтение ZIP-файлов

Чтобы прочитать содержимое ZIP-файла, прежде всего необходимо создать объект `ZipFile` (обратите внимание на использование прописных букв "Z" и "F" в имени объекта). Объекты `ZipFile` концептуально напоминают объекты `File`, возвращаемые функцией `open()`, с которой вы познакомились в предыдущей главе: они представляют собой значения, посредством которых программа взаимодействует с файлами. Для создания объекта `ZipFile` следует вызвать функцию `zipfile.ZipFile()`, передав ей строку с именем *.zip*-файла. Обратите внимание на то, что `zipfile` — это имя модуля Python, а `ZipFile()` — имя функции.

Введите в интерактивной оболочке следующие команды.

```
>>> import zipfile, os
>>> os.chdir('C:\\') # перейти в папку, содержащую example.zip
```

```
>>> exampleZip = zipfile.ZipFile('example.zip')
>>> exampleZip.namelist()
['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']
>>> spamInfo = exampleZip.getinfo('spam.txt')
>>> spamInfo.file_size
13908
>>> spamInfo.compress_size
3828
❶ >>> 'Сжатый файл в %s раза меньше!' % (round(spamInfo.file_size
↳/ spamInfo.compress_size, 2))
'Сжатый файл в 3.63 раза меньше!'
>>> exampleZip.close()
```

Объект `ZipFile` имеет метод `namelist()`, который возвращает список строк для всех файлов и папок, содержащихся в ZIP-файле. Эти строки можно передать методу `getinfo()` объекта `ZipFile`, который возвратит объект `ZipInfo`, содержащий информацию о данном файле. Объекты `ZipInfo` имеют собственные атрибуты, такие как `file_size` и `compress_size`, которые содержат соответственно целочисленные значения размера исходного и сжатого файлов, выраженные в байтах. В то время как объект `ZipFile` представляет весь архивный файл, объект `ZipInfo` хранит полезную информацию относительно отдельного файла в сжатом архиве.

Команда ❶ рассчитывает степень сжатия файла `example.zip` путем деления размера исходного файла на размер сжатого файла и выводит эту информацию, используя строку форматирования с описателем формата `%s`.

Извлечение файлов из ZIP-архива

Метод `extractall()` объектов `ZipFile` извлекает все файлы и папки из ZIP-файла в текущий рабочий каталог.

```
>>> import zipfile, os
>>> os.chdir('C:\\') # перейти в папку, содержащую example.zip
>>> exampleZip = zipfile.ZipFile('example.zip')
❶ >>> exampleZip.extractall()
>>> exampleZip.close()
```

После выполнения этого кода содержимое файла *example.zip* будет извлечено в каталог *C:*. Методу `extractall()` можно передать имя папки в качестве необязательного параметра, позволяющего извлекать файлы в папку, не являющуюся текущим рабочим каталогом. Если переданная методу `extractall()` папка не существует, то она будет создана. Например, если вызов ❶ заменить вызовом `exampleZip.extractall('C:\\delicious')`, код извлечет файлы из файла *example.zip* во вновь созданную папку *C:\delicious*.

Метод `extract()` объектов `ZipFile` извлекает одиночный файл из ZIP-файла. Продолжите выполнение примера в интерактивной оболочке.

```
>>> exampleZip.extract('spam.txt')
'C:\\spam.txt'
>>> exampleZip.extract('spam.txt', 'C:\\some\\new\\folders')
'C:\\some\\new\\folders\\spam.txt'
>>> exampleZip.close()
```

Передаваемая методу `extract()` строка должна соответствовать одной из строк в списке, возвращаемом методом `namelist()`. Методу `extract()` также можно передавать необязательный второй параметр, обеспечивающий возможность извлечения файлов в папку, не являющуюся текущим рабочим каталогом. Если заданная вторым параметром папка не существует, Python создаст ее. Метод `extract()` возвращает абсолютный путь, по которому был распакован данный файл.

Создание ZIP-файлов и добавление в них новых файлов

Чтобы создать собственный ZIP-файл, необходимо открыть объект `ZipFile` в *режиме записи* посредством передачи строки `'w'` в качестве второго аргумента. (Это аналогично открытию текстового файла в режиме записи путем передачи строки `'w'` методу `open()` в качестве второго параметра.)

Когда вы передаете путь методу `write()` объекта `ZipFile`, Python сжимает файл, расположенный по указанному пути, и добавляет его в ZIP-файл. Первый аргумент метода `write()` — это строка, содержащая имя добавляемого файла. Второй аргумент — это параметр *типа сжатия*, сообщающий компьютеру о том, какой алгоритм следует использовать для сжатия файлов; вы всегда можете установить для этого параметра значение `zipfile.ZIP_DEFLATED`. (Этому значению соответствует алгоритм сжатия *без потерь*, который хорошо работает со всеми типами данных.) Введите в интерактивной оболочке следующие команды.

```
>>> import zipfile
>>> newZip = zipfile.ZipFile('new.zip', 'w')
>>> newZip.write('spam.txt', compress_type=zipfile.ZIP_DEFLATED)
>>> newZip.close()
```

Данный код создает новый ZIP-файл *new.zip*, включающий сжатое содержимое файла *spam.txt*.

Имейте в виду, что, как и в случае обычной записи файлов, все существующее содержимое ZIP-файла в режиме записи удаляется. Если вы хотите просто добавить файлы в существующий ZIP-файл, передайте строку `'a'` в качестве второго параметра методу `zipfile.ZipFile()`, чтобы открыть ZIP-файл в режиме присоединения.

Проект: переименование файлов с заменой американского формата дат европейским

Предположим, начальник перебросил вам по электронной почте тысячи файлов с просьбой переименовать их с заменой американского формата дат (ММ-ДД-ГГГГ) в их именах европейским (ДД-ММ-ГГГГ). Выполняя это поручение вручную, вы можете потратить на него целый день! Не лучше ли написать программу, которая сделает всю работу за вас?

Вот что должна делать эта программа:

- выполнять в текущем рабочем каталоге поиск всех файлов, имена которых содержат дату в американском формате;
- при нахождении каждого такого файла переименовывать его, меняя местами обозначения даты и месяца, чтобы привести стиль указания даты в соответствие с европейским форматом.

Это означает, что код должен выполнить следующие действия:

- создать регулярное выражение для распознавания образцов текста, соответствующих дате, заданной в американском стиле;
- вызвать функцию `os.listdir()` для создания списка всех файлов, содержащихся в рабочем каталоге;
- организовать просмотр всех имен файлов в цикле, определяя, содержат ли они даты, с помощью соответствующего регулярного выражения;
- если в имя файла входит дата, изменить его с помощью функции `shutil.move()`.

Приступая к работе над данным проектом, откройте новое окно в файловом редакторе и сохраните его в файле *renameDates.py*.

Шаг 1. Создание регулярного выражения для поиска дат, указанных в американском формате

В первой части программы потребуется импортировать необходимые модули и создать регулярное выражение, способное распознавать даты в формате ММ-ДД-ГГГГ. Комментарии TODO будут напоминать о программном коде, который еще предстоит написать. Использование общепринятого

обозначения TODO (ЧТО_СДЕЛАТЬ) для этих комментариев упрощает их поиск путем нажатия комбинации клавиш <Ctrl+F> при работе в IDLE. Введите в файл следующий код.

```
#! python3
# renameDates.py - Переименовывает файлы, имена которых включают
# даты, указанные в американском формате (ММ-ДД-ГГГГ), приводя
# их в соответствие с европейским форматом дат (ДД-ММ-ГГГГ).

❶ import shutil, os, re

# Создание регулярного выражения, которому соответствуют имена
# файлов, содержащие даты в американском формате.
❷ datePattern = re.compile(r"""^(.*?) # весь текст перед датой
    ((0|1)?\d)-                       # одна или две цифры месяца
    ((0|1|2|3)?\d)-                   # одна или две цифры числа
    ((19|20)\d\d)                     # четыре цифры года
    (.*?)$                            # весь текст после даты
❸   """, re.VERBOSE)

# TODO: Организовать цикл по файлам в рабочем каталоге.

# TODO: Пропустить файлы с именами, не содержащими дат.

# TODO: Получить отдельные части имен файлов.

# TODO: Сформировать имена, соответствующие европейскому стилю
#       указания дат.

# TODO: Получить полные абсолютные пути к файлам.

# TODO: Переименовать файлы.
```

Из этой главы вы уже знаете о том, что для переименования файлов можно использовать функцию `shutil.move()`, аргументами которой служат исходное и новое имена файла. Поскольку эта функция содержится в модуле `shutil`, его необходимо импортировать ❶.

Однако, прежде чем переименовывать файлы, нужно идентифицировать те из них, которые подлежат переименованию. Переименовывать следует файлы, в именах которых содержатся даты, например *spam4-4-1984.txt* или *01-03-2014eggs.zip*, тогда как имена таких файлов, как *littlebrother.epub*, не содержащие дат, можно игнорировать.

Для распознавания этого шаблона можно использовать регулярные выражения. Импортировав модуль `re` в начале файла, вызовите функцию `re.compile()` для создания объекта `Regex` ❷. Передача этой функции константы `re.VERBOSE` в качестве второго аргумента ❸ разрешает использовать пробелы и комментарии в строке регулярного выражения для повышения ее удобочитаемости.

Строка регулярного выражения начинается символами `^(.*?)`, которым соответствует любой текст в имени файла, предшествующий дате. Группе `((0|1)?\d)` соответствует цифровое обозначение месяца. Первой цифрой может быть как `0`, так и `1`, так что регулярное выражение совпадет как с обозначением `12` в случае декабря, так и с обозначением `02` в случае февраля. Кроме того, эта цифра задана как необязательная, поэтому, например, апрель будет распознан независимо от того, как он обозначен: `04` или `4`. Обозначениям дней соответствует группа `((0|1|2|3)?\d)`, которая следует аналогичной логике; `3`, `03` и `31` — каждый из этих вариантов является допустимым для обозначения дней. (Если вы отличаетесь наблюдательностью, то заметите, что данному регулярному выражению будут соответствовать и некоторые недопустимые даты, такие как `4-31-2014`, `2-29-2013` или `0-15-2014`. При работе с датами следует учитывать множество тонких моментов, которые могут легко ускользнуть от вашего внимания. Однако для большинства простых случаев это регулярное выражение может считаться вполне приемлемым для нашей программы.) Несмотря на то что `1885` — корректное обозначение года, вы можете ограничиться годами, относящимися к XX и XXI столетиям. Тем самым вы избавитесь от случайного переименования тех файлов, в именах которых встречаются цифровые обозначения, лишь похожие на даты, такие как *10-10-1000.txt*.

Части `(.*?)$` регулярного выражения соответствует любой текст, который следует за датой в имени файла.

Шаг 2. Идентификация частей имен файлов, соответствующих датам

После этого программа должна просмотреть в цикле строки имен файлов из списка, возвращенного функцией `os.listdir()`, для их сравнения с регулярным выражением. Любые файлы, имена которых не включают дату, должны игнорироваться. Для имен, содержащих дату, совпавший с шаблоном текст должен быть сохранен в нескольких переменных. Замените первые три комментария TODO в вашей программе следующим кодом.

```
#! python3
# renameDates.py - Переименовывает файлы, имена которых включают
# даты, указанные в американском формате (ММ-ДД-ГГГГ), приводя
# их в соответствие с европейским форматом дат (ДД-ММ-ГГГГ).

--пропущенный код--

# Организация цикла по файлам в рабочем каталоге.
for amerFilename in os.listdir('.'):
    mo = datePattern.search(amerFilename)
```

```
        # Пропуск файлов с именами, не содержащими дат.
❶       if mo == None:
❷           continue

❸       # Получение отдельных частей имен файлов.
        beforePart = mo.group(1)
        monthPart = mo.group(2)
        dayPart = mo.group(4)
        yearPart = mo.group(6)
        afterPart = mo.group(8)

    --пропущенный код--
```

Если метод `search()` возвращает значение `None` ❶, значит, строка имени файла, содержащаяся в переменной `amerFilename`, не соответствует регулярному выражению. Инструкция `continue` ❷ игнорирует оставшуюся часть цикла и осуществляет переход к следующему имени файла.

В противном случае строки, соответствующие отдельным группам в регулярном выражении, сохраняются в переменных `beforePart`, `monthPart`, `dayPart`, `yearPart` и `afterPart` ❸. Эти строки будут использоваться при выполнении следующего шага для формирования имен файлов с датами в европейском формате.

Для поддержания сквозной нумерации групп попробуйте прочитать регулярное выражение с самого начала, ведя отсчет посредством прибавления единицы всякий раз, когда вам встречается открывающая круглая скобка. Не думайте о коде и просто наметьте каркас регулярного выражения. Возможно, так вам будет легче визуализировать группы.

```
datePattern = re.compile(r"""^(1) # весь текст перед датой
    (2 (3) )-                     # одна или две цифры месяца
    (4 (5) )-                     # одна или две цифры числа
    (6 (7) )                      # четыре цифры года
    (8)$                          # весь текст после даты
    """, re.VERBOSE)
```

Здесь числа от `1` до `8` представляют группы в регулярном выражении, которое вы написали. Представление структуры регулярного выражения с использованием лишь круглых скобок и номеров групп поможет вам понять его смысл, прежде чем вы перейдете к созданию остальной части программы.

Шаг 3. Формирование нового имени файла и переименование файлов

Последнее, что осталось сделать, — это конкатенировать строки, сохраненные в переменных на предыдущем шаге, для приведения даты к

европейскому формату, в соответствии с которым число предшествует месяцу. Замените три оставшихся комментария TODO кодом, как показано ниже.

```
#! python3
# renameDates.py - Переименовывает файлы, имена которых включают
# даты, указанные в американском формате (ММ-ДД-ГГГГ), приводя
# их в соответствие с европейским форматом дат (ДД-ММ-ГГГГ).

--пропущенный код--

    # Формирование имен, соответствующих европейскому стилю
    # указания дат.
❶   euroFilename = beforePart + dayPart + '-' + monthPart +
                   '-' + yearPart + afterPart

    # Получение полных абсолютных путей к файлам.
    absWorkingDir = os.path.abspath('.')
    amerFilename = os.path.join(absWorkingDir, amerFilename)
    euroFilename = os.path.join(absWorkingDir, euroFilename)

    # Переименование файлов.
❷   print('Заменяем имя "%s" именем "%s"...'
          % (amerFilename, euroFilename))
❸   #shutil.move(amerFilename, euroFilename) # раскомментировать
                                             # после выполнения
                                             # тестирования
```

Конкатенированная строка сохраняется в переменной `euroFilename` ❶. Затем исходное и новое имена файлов, сохраненные в переменных `amerFilename` и `euroFilename`, передаются функции `shutil.move()` для окончательного переименования файла ❸.

В данной версии программы вызов `shutil.move()` отключен с помощью комментария и имена файлов, подлежащих переименованию, просто выводятся на экран ❷. Запуск программы в таком режиме позволяет еще раз убедиться в корректном переименовании файлов. После этого можно удалить символ комментария в строке с вызовом `shutil.move()` и вновь выполнить программу для фактического переименования файлов.

Идеи относительно создания аналогичных программ

Необходимость в переименовании большого количества файлов может возникать по целому ряду других причин:

- для добавления стандартного префикса в начале имен файлов; например, файл *eggs.txt* переименовывается в файл *spam_eggs.txt*;

- для преобразования дат в именах файлов из европейского формата в американский;
- для удаления ведущих нулей из имен таких файлов, как *spam0042.txt*.

Проект: создание резервной копии папки в виде ZIP-файла

Предположим, вы работаете над проектом, файлы которого хранятся в папке *C:\AlsPythonBook*. Вас волнует сохранность результатов вашей работы, и вам хотелось бы периодически создавать "моментальные снимки" проекта, сохраняя всю папку в одном ZIP-файле. Для вас было бы желательно хранить различные версии проекта в файлах с именами, содержащими номер резервной копии, который увеличивается всякий раз, когда создается новый ZIP-файл, например *AlsPythonBook_1.zip*, *AlsPythonBook_2.zip*, *AlsPythonBook_3.zip* и т.д. Это можно было бы делать и вручную, но такой подход чреват тем, что номера ZIP-файлов могут быть случайно перепутаны. Гораздо проще написать программу, которая будет выполнять эту рутинную работу вместо вас.

Приступая к работе над данным проектом, откройте новое окно в файловом редакторе и сохраните его в файле *backupToZip.py*.

Шаг 1. Определение имени, которое следует присвоить ZIP-файлу

Код этой программы будет помещен в функцию `backupToZip()`. Это упростит его копирование и вставку в другие программы на Python, нуждающиеся в этой функциональности. В конце программы эта функция будет вызываться для создания резервной копии содержимого папки. Введите следующий код.

```
#! python3
# backupToZip.py - Копирует папку вместе со всем ее содержимым
# в ZIP-файл с инкрементируемым номером копии в имени файла.

❶ import zipfile, os

def backupToZip(folder):
    # Создание резервной копии всего содержимого папки "folder"
    # в виде ZIP-файла.

    folder = os.path.abspath(folder) # убедиться в том, что
                                     # задан абсолютный путь
                                     # к файлу

    # Определить, какое имя файла должен использовать этот код,
    # исходя из имен уже существующих файлов.
```

```
❷     number = 1
❸     while True:
          zipFilename = os.path.basename(folder) + '_' +
                        str(number) + '.zip'
          if not os.path.exists(zipFilename):
              break
          number = number + 1

❹     # TODO: Создать ZIP-файл.

      # TODO: Обойти все дерево папки и сжать файлы, содержащиеся
      #       в каждой папке.
      print('Готово.')

backupToZip('C:\\delicious')
```

Мы начинаем с элементарных вещей: добавляем "магическую" строку запуска Python (с символами #!), описываем назначение программы и импортируем модули `zipfile` и `os` ❶.

Далее определяется функция `backupToZip()`, принимающая всего один параметр — `folder`. Этот параметр — строка, содержащая путь к папке, резервную копию содержимого которой следует создать. Сначала функция определяет имя, которое следует присвоить создаваемому ZIP-файлу; затем она создает сам файл, совершает обход содержимого папки `folder` и добавляет все ее подпапки и файлы в ZIP-файл. Включите в исходный код комментарии TODO для этих шагов как напоминание о том, что необходимо сделать в дальнейшем ❹.

В первой части программы, ответственной за присвоение имени ZIP-файлу, используется базовое имя абсолютного пути к папке. Если резервируется содержимое папки *C:\delicious*, то именем ZIP-файла будет *delicious_N.zip*, где N = 1 при первом запуске программы, N = 2 — при втором и т.д.

Вы можете определить, каким должно быть значение N, проверив, существуют ли уже файлы *delicious_1.zip*, *delicious_2.zip* и т.д. Значение N хранится в переменной `number` ❷ и инкрементируется в цикле, в котором с помощью функции `os.path.exists()` проверяется существование файлов ❸. Как только обнаружен несуществующий файл, цикл прерывается, поскольку нам уже известно, какое имя следует присвоить новому ZIP-файлу.

Шаг 2. Создание нового ZIP-файла

Следующим шагом является создание ZIP-файла. Дополните программу новым кодом, как показано ниже.

```
#! python3
# backupToZip.py - Копирует папку вместе со всем ее содержимым
# в ZIP-файл с инкрементируемым номером копии в имени файла.
```

```
--пропущенный код--
    while True:
        zipFilename = os.path.basename(folder) + '_' +
                      str(number) + '.zip'
        if not os.path.exists(zipFilename):
            break
        number = number + 1

    # Создание ZIP-файла.
    print('Создается файл %s...' % (zipFilename))
❶   backupZip = zipfile.ZipFile(zipFilename, 'w')

    # TODO: Обойти все дерево папки и сжать файлы, содержащиеся
    #       в каждой папке.
    print('Готово.')

backupToZip('C:\\delicious')
```

Теперь, когда имя нового ZIP-файла сохранено в переменной `zipFilename`, можно вызвать функцию `zipfile.ZipFile()` для фактического создания ZIP-файла ❶. Не забудьте передать ей строку `'w'` в качестве второго аргумента, чтобы открыть ZIP-файл в режиме записи.

Шаг 3. Обход дерева каталогов и добавление содержимого в ZIP-файл

Наконец, для обхода всех файлов и подпапок, содержащихся в данной папке, используется функция `os.walk()`. Дополните программу новым кодом, выделенным ниже полужирным шрифтом.

```
#! python3
# backupToZip.py - Копирует папку вместе со всем ее содержимым
# в ZIP-файл с инкрементируемым номером копии в имени файла.

--пропущенный код--

    # Обход всего дерева папки и сжатие файлов, содержащихся
    # в каждой папке.
❶   for foldername, subfolders, filenames in os.walk(folder):
        print('Добавление файлов из папки %s...' % (foldername))
        # Добавить в ZIP-файл текущую папку.
❷       backupZip.write(foldername)
        # Добавить в ZIP-файл все файлы из данной папки.
❸       for filename in filenames:
            newBase / os.path.basename(folder) + '_'
            if filename.startswith(newBase) and
            filename.endswith('.zip')
                continue # не создавать резервные копии
                         # ZIP-файлов
            backupZip.write(os.path.join(foldername, filename))
```

```
        backupZip.close()
        print('Готово.')

backupToZip('C:\\delicious')
```

Функцию `os.walk()` можно использовать в цикле `for` ❶, и на каждой итерации цикла она будет возвращать имя текущей папки, а также имена всех подпапок и файлов, содержащихся в данной папке.

В теле цикла `for` папка добавляется в ZIP-файл ❷. Обход всех файлов, имена которых содержатся в списке `filenames` ❸, осуществляется во вложенном цикле `for`. Каждый из них, за исключением ранее созданных ZIP-архивов, добавляется в ZIP-файл.

Выполнив эту программу, вы получите примерно следующий вывод.

```
Создается файл delicious_1.zip...
Добавление файлов из папки C:\delicious...
Добавление файлов из папки C:\delicious\cats...
Добавление файлов из папки C:\delicious\waffles...
Добавление файлов из папки C:\delicious\walnut...
Добавление файлов из папки C:\delicious\walnut\waffles...
Готово.
```

В результате второго вызова программы все файлы, содержащиеся в папке *C:\delicious*, будут архивированы в ZIP-файле *delicious_2.zip* и т.д.

Идеи относительно создания аналогичных программ

Вы сможете использовать обход дерева каталогов и добавление файлов в сжатые ZIP-архивы в целом ряде других программ. Например, можно написать программы для выполнения следующих задач:

- обход дерева каталогов и архивирование лишь файлов с определенными расширениями, например *.txt* или *.py*, и никакими другими;
- обход дерева каталогов и архивирование всех файлов, за исключением тех, которые имеют расширение *.txt* или *.py*;
- поиск в дереве каталогов папки, содержащей наибольшее количество файлов, или папки, занимающей наибольший объем дискового пространства.

Резюме

Даже если вы опытный пользователь, вы, вероятнее всего, работаете с файлами вручную с помощью мыши и клавиатуры. Современные файловые менеджеры упрощают работу с небольшим количеством файлов. Но ино-

гда возникают задачи, для выполнения которых с помощью проводника потребуется несколько часов работы.

Модули `os` и `shutil` предлагают функции, позволяющие осуществлять копирование, перемещение, переименование и удаление файлов. Удаляя файлы, вы, возможно, захотите воспользоваться модулем `send2trash`, который предоставляет возможность не удалять файлы безвозвратно, а перемещать их в корзину. При написании программ, предназначенных для обработки файлов, целесообразно использовать символы комментария для отключения кода, выполняющего непосредственное копирование (перемещение, переименование, удаление) файлов, добавляя вместо него вызовы функции `print()`, которые позволяют убедиться в том, что программа делает именно то, что вам необходимо.

Операции подобного рода приходится выполнять не только над файлами, хранящимися в данной папке, но и над файлами, хранящимися во вложенных папках, а также в папках второго и всех последующих уровней вложенности. Функция `os.walk()` может выполнить обход всей структуры папок вместо вас, позволяя сосредоточить все внимание на том, что должна сделать программа в отношении хранящихся в этих папках файлов.

С помощью модуля `zipfile` можно сжимать и извлекать файлы, хранящиеся в *.zip*-архивах. В сочетании с предоставляемыми модулями `os` и `shutil` функциями для обработки файлов модуль `zipfile` упрощает упаковку нескольких файлов, хранящихся в любой папке на вашем жестком диске. Такие *.zip*-файлы гораздо легче выгружать на веб-сайты или пересылать в виде вложений в сообщения электронной почты, чем множество отдельных файлов.

В предыдущих главах вам предлагался готовый код, который достаточно было просто копировать. Однако при разработке собственных программ вы столкнетесь с тем, что они не всегда правильно работают с первого раза. Следующая глава посвящена некоторым модулям Python, облегчающим анализ и отладку программ, которые помогут вам быстрее добиться их правильной работы.

Контрольные вопросы

1. Чем отличаются функции `shutil.copy()` и `shutil.copytree()`?
2. Какая функция используется для переименования файлов?
3. Чем отличаются функции для удаления файлов, предлагаемые модулями `send2trash` и `shutil`?
4. Метод `close()` имеют как объекты `ZipFile`, так и объекты `File`. Какой метод объектов `ZipFile` эквивалентен методу `open()` объектов `File`?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Выборочное копирование

Напишите программу, выполняющую обход дерева каталогов с целью отбора файлов с заданным расширением (например, `.pdf` или `.jpg`). Скопируйте эти файлы из их текущего расположения в новую папку.

Удаление ненужных файлов

Не так уж редки ситуации, когда несколько ненужных файлов или папок огромного размера занимают существенную часть дискового пространства. Для освобождения места на жестком диске наилучший эффект будет достигнут в том случае, если удалить самые крупные из ненужных файлов. Однако сначала их необходимо найти.

Напишите программу, которая обходит дерево папок, выполняя поиск исключительно больших по своим размерам папок и файлов, — скажем, таких, размеры которых превышают 100 Мбайт. (Вспомните, что размер файла можно определить с помощью функции `os.path.getsize()` из модуля `os`.) Выведите абсолютные пути доступа к этим файлам на экран.

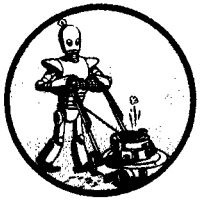
Заполнение пропусков в нумерации файлов

Напишите программу, которая ищет в папке все файлы с именами, содержащими заданный префикс, такими как *spam001.txt*, *spam002.txt* и т.д., и обнаруживает любые пропуски в нумерации файлов (например, имеются файлы *spam001.txt* и *spam003.txt*, но отсутствует файл *spam002.txt*). Программа должна изменять имена файлов с большими номерами таким образом, чтобы ликвидировать имеющиеся пропуски.

В качестве дополнительного задания напишите другую программу, способную создавать пропуски в нумерации файлов, тем самым создавая свободные позиции для добавления новых файлов.

10

ОТЛАДКА



Тех знаний, которые вы к этому времени успели приобрести, вполне достаточно для того, чтобы приступить к написанию более сложных программ. Однако вы должны быть готовы к тому, что в вашем программном коде будут встречаться ошибки, найти которые не так уж и просто. В этой главе обсуждаются инструменты и методики, позволяющие экономить время и усилия, затрачиваемые на обнаружение и устранение логических ошибок в программах.

Перефразируя расхожую шутку программистов, я сказал бы так: "Программирование на 90 процентов состоит из написания кода. Остальные 90 процентов приходятся на отладку".

Ваш компьютер сделает лишь то, что вы ему прикажете; он не способен исполнять ваши *намерения*, читая мысли, и нуждается в соответствующем программном коде, ответственность за подготовку которого возлагается на программистов. Однако даже профессиональные программисты иногда допускают серьезные программные ошибки, поэтому не стоит падать духом, если в работе вашей программы обнаруживаются неполадки.

К счастью, существует целых ряд инструментов и методик, с помощью которых вы сможете точно определить, что именно делает ваш код и в каком месте программы что-то пошло не так, как надо. Во-первых, мы рассмотрим протоколирование операций и утверждения — два средства, облегчающие раннее обнаружение ошибок. Вообще говоря, чем раньше обнаружена ошибка, тем проще ее исправить.

Во-вторых, вы познакомитесь с отладчиком. Отладчик — это средство IDLE, обеспечивающее пошаговое выполнение программы, по одной инструкции за раз, что дает возможность инспектировать значения переменных и отслеживать их изменения в ходе выполнения программы. При этом программа выполняется значительно медленнее, чем обычно, но зато вы получаете возможность наблюдать за реальными значениями переменных, а не оценивать их на основе умозаключений, анализируя исходный код.

Возбуждение исключений

Python возбуждает (генерирует) исключение всякий раз, когда предпринимается попытка выполнить недопустимый код. Из главы 3 вы узнали о том, как обрабатывать исключения Python с помощью инструкций `try` и `except`, позволяющих избежать преждевременного прекращения работы программы при возникновении исключительных ситуаций, которые вы предвидели. Но у вас есть возможность возбуждать в своем коде собственные исключения. Возбуждение исключения означает для компьютера следующее: "Прекратить выполнение кода данной функции и передать управление инструкции `except`".

Исключения возбуждаются с помощью инструкции `raise`, синтаксис которой включает следующие элементы:

- ключевое слово `raise`;
- вызов функции `Exception()`;
- строка с описательным сообщением об ошибке, передаваемая функции `Exception()`.

Например, введите в интерактивной оболочке следующую команду.

```
>>> raise Exception('Это сообщение об ошибке.')
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    raise Exception('Это сообщение об ошибке.')
Exception: Это сообщение об ошибке.
```

В отсутствие инструкций `try` и `except`, охватывающих инструкцию `raise`, которая генерирует исключение, выполнение программы завершается аварийно с выводом соответствующего сообщения.

Часто обработка исключения, которое может возникнуть в функции, выполняется не самой функцией, а вызывающим ее кодом. Поэтому инструкция `raise` нередко встречается в теле функции, а связанные с ней инструкции `try` и `except` — в коде, вызывающем эту функцию. Например, откройте новое окно в файловом редакторе, введите в него следующий код и сохраните программу в файле *boxPrint.py*.

```
  def boxPrint(symbol, width, height):
      if len(symbol) != 1:
❶         raise Exception('Переменная symbol должна быть
  ↳односимвольной строкой.')
      if width <= 2:
❷         raise Exception('Значение width должно превышать 2.')
      if height <= 2:
❸         raise Exception('Значение height должно превышать 2.')
      print(symbol * width)
```

```
        for i in range(height - 2):
            print(symbol + (' ' * (width - 2)) + symbol)
        print(symbol * width)

    for sym, w, h in (('*', 4, 4), ('O', 20, 5), ('x', 1, 3), ('ZZ',
                      3, 3)):
        try:
            boxPrint(sym, w, h)
❹       except Exception as err:
❺           print('Возникло исключение: ' + str(err))
```

Здесь мы определили функцию `boxPrint()`, которая принимает параметр `symbol`, определяющий символ или группу символов, а также параметры `width` и `height`, и использует заданный символ (символы) для создания небольшого изображения, ширина и высота которого определяются остальными двумя параметрами. Полученная прямоугольная фигура выводится на консоль.

Предположим, мы хотим, чтобы параметр `symbol` мог быть только одиночным символом, а значения параметров `width` и `height` превышали `2`. Для этого мы добавляем инструкции `if`, которые возбуждают исключения, если эти требования не удовлетворяются. Впоследствии, когда вызывается функция `boxPrint()` с различными аргументами, конструкция `try/except` обработает недопустимые аргументы.

В этой программе в инструкции `except` используется исключение в форме `Exception as err` ❹. В случае возврата объекта `Exception` функцией `boxPrint()` ❶ ❷ ❸ инструкция `except` сохранит его в переменной `err`. Затем объект `Exception` можно преобразовать в строку посредством передачи его функции `str()` для вывода сообщения об ошибке в понятной для пользователя форме ❺. В результате выполнения программы `boxPrint.py` вы должны получить следующий вывод.

```
****
*  *
*  *
****
OOOOOOOOOOOOOOOOOOOO
O                  O
O                  O
O                  O
OOOOOOOOOOOOOOOOOOOO
Возникло исключение: 'Значение width должно превышать 2.
Возникло исключение: Переменная symbol должна быть односимвольной строкой.
```

Применение инструкций `try` и `except` обеспечивает элегантную обработку ошибок, исключающую возможность неконтролируемого аварийного завершения программы.

Получение обратной трассировки стека вызовов в виде строки

Когда возникает ошибка, Python предоставляет весьма ценную информацию о ее природе в виде так называемой *обратной трассировки стека вызовов*. Эта информация включает текст сообщения об ошибке, номер строки в исходном коде, являющейся причиной неполадок, и последовательность вызовов функций, приведшую к ошибке. Эта последовательность называется *стеком вызовов*.

Откройте новое окно в IDLE, введите приведенный ниже код программы и сохраните его в файле *errorExample.py*.

```
def spam():
    bacon()

def bacon():
    raise Exception('Это сообщение об ошибке.')

spam()
```

Выполнив программу *errorExample.py*, вы получите следующий вывод.

```
Traceback (most recent call last):
  File "errorExample.py", line 7, in <module>
    spam()
  File "errorExample.py", line 2, in spam
    bacon()
  File "errorExample.py", line 5, in bacon
    raise Exception('Это сообщение об ошибке.')
Exception: Это сообщение об ошибке.
```

На основании информации о стеке вызовов можно утверждать, что ошибка возникла в строке 5, т.е. в коде функции `bacon()`. Эта функция была вызвана в строке 2, т.е. в коде функции `spam()`, которая, в свою очередь, была вызвана в строке 7. В тех случаях, когда одна и та же функция может вызываться в нескольких местах программы, стек вызовов позволяет определить, какой именно вызов приводит к ошибке.

Python отображает стек вызовов всякий раз, когда возбужденное исключение остается необработанным. Но его также можно получить в виде строки, вызвав функцию `traceback.format_exc()`. Эта функция пригодится вам в тех случаях, когда вы хотите обработать исключение, но одновременно с этим получить информацию о стеке вызовов. Прежде чем вызывать эту функцию, необходимо импортировать модуль `traceback` Python.

Например, вместо того чтобы просто позволить программе завершиться аварийно сразу же после возникновения исключения, можно записать информацию о стеке вызовов в журнал отчетов и предоставить программе возможность дальнейшего выполнения. Впоследствии, когда вы будете готовы приступить к отладке, вы сможете обратиться к записанной в журнале информации. Введите в интерактивной оболочке следующий код.

```
>>> import traceback
>>> try:
        raise Exception('Это сообщение об ошибке.')
except:
        errorFile = open('errorInfo.txt', 'w')
        errorFile.write(traceback.format_exc())
        errorFile.close()
        print('Информация о стеке вызовов была записана в файл
↳errorInfo.txt.')

113
Информация о стеке вызовов записана в файл errorInfo.txt.
```

Число 113 — это значение, возвращенное методом `write()`, которое равно количеству символов, записанных в файл (включая символы новой строки).

Записанная в файл *errorInfo.txt* информация должна выглядеть примерно так.

```
Traceback (most recent call last):
  File "<pyshell#11>", line 2, in <module>
Exception: Это сообщение об ошибке.
```

Утверждения

Утверждение — это профилактический механизм, позволяющий убедиться в правильности выполнения кода программы. Этот механизм основан на выполнении проверок с помощью инструкций `assert`. Если критерий проверки не выдерживается, то возникает исключение `AssertionError`. Инструкция `assert` имеет следующий синтаксис:

- ключевое слово `assert`;
- условие (т.е. выражение, вычисление которого дает значение `True` или `False`);
- запятая;
- строка, которая отображается в том случае, если условие оказывается ложным.

В качестве примера введите в интерактивной оболочке следующий код.

```
>>> podBayDoorStatus = 'открыто'
>>> assert podBayDoorStatus == 'открыто', 'Дверь грузового отсека
↳должна находиться в состоянии "открыто".'
>>> podBayDoorStatus = 'Мне жаль, Дейв. Боюсь, я ничего не
↳могу сделать.''
>>> assert podBayDoorStatus == 'открыто', 'Дверь грузового отсека
↳должна находиться в состоянии "открыто".'
Traceback (most recent call last):
  File "<pyshell#10>", line 1, in <module>
    assert podBayDoorStatus == 'открыто', 'Дверь грузового отсека
    должна находиться в состоянии "открыто".'
AssertionError: Дверь грузового отсека должна находиться в состоянии "от-
крыто".
```

Здесь мы устанавливаем для переменной `podBayDoorStatus` значение `'открыто'` и рассчитываем на то, что данная переменная всегда будет иметь именно это значение. Предположим, что выполнение этого условия имеет существенное значение для того, чтобы код работал так, как ожидается. Поэтому мы добавляем утверждение, которое позволяет удостовериться в том, что в переменной `podBayDoorStatus` действительно хранится значение `'открыто'`. В данном случае для того чтобы сразу увидеть, где возникает сбой, если утверждение не выполняется, мы включаем в операцию присваивания строку сообщения `'Дверь грузового отсека должна находиться в состоянии "открыто".'`.

Допустим, что впоследствии мы совершаем досадную ошибку, присваивая переменной `podBayDoorStatus` другое значение, но не замечаем этого огреха среди множества строк кода. Утверждение обнаружит эту ошибку и недвусмысленно информирует нас о том, что именно произошло.

В переводе на обычный язык утверждение сообщает нам следующее: "Я убеждаюсь в том, что данное условие выполняется; в противном случае можно утверждать, что где-то в программе скрывается ошибка". В отличие от исключений ваш код *не* должен обрабатывать инструкции `assert` с помощью конструкции `try/except`. Если утверждение оказывается ложным, то ваша программа *должна* завершиться аварийно. Такой способ раннего выявления потенциальных ошибок сокращает промежуток времени, отделяющий момент возникновения первопричины ошибки от момента, когда вам впервые становится известно о наличии ошибки в программе. Тем самым сокращается объем кода, который пришлось бы написать для выявления фрагментов основного кода, порождающих ошибку.

Утверждения предназначены для нахождения ошибок программиста, а не пользователя. Для обнаружения ошибок, которые допускают элегантное (или, как говорят, амортизированное) восстановление работоспособного

состояния программы, таких как ввод пользователем некорректных данных или невозможность обнаружить указанный файл, используйте возбуждение исключений, а не инструкции `assert`.

Использование утверждений в программе, имитирующей работу светофора

Предположим, требуется создать программу, имитирующую работу светофора. В качестве структуры данных, представляющей сигналы светофора на перекрестке, выбираем словарь с ключами `'ns'` и `'ew'` для ламп, ориентированных вдоль направлений "север–юг" и "восток–запад" соответственно. Каждому из этих ключей могут соответствовать значения в виде строк `'green'`, `'yellow'` и `'red'` (красный). Соответствующий код будет выглядеть так.

```
market_2nd = {'ns': 'green', 'ew': 'red'}
mission_16th = {'ns': 'red', 'ew': 'green'}
```

Эти две переменные отвечают за перекрестки, образуемые парами улиц "Market Street–2nd Street" и "Mission Street–16th Street". Приступая к проекту, целесообразно написать функцию `switchLights()`, которая будет переключать светофор, принимая указанный словарь в качестве аргумента.

. Первое, что может прийти вам на ум, — это то, что функция `switchLights()` всего лишь должна последовательно переключать цвета сигналов с одного на другой: за любым значением `'green'` (зеленый) должно следовать значение `'yellow'`(желтый), которое будет заменяться значением `'red'` (красный), в свою очередь сменяющимся значением `'green'` и т.д. Код, реализующий эту идею, может выглядеть примерно так.

```
def switchLights(stoplight):
    for key in stoplight.keys():
        if stoplight[key] == 'green':
            stoplight[key] = 'yellow'
        elif stoplight[key] == 'yellow':
            stoplight[key] = 'red'
        elif stoplight[key] == 'red':
            stoplight[key] = 'green'

switchLights(market_2nd)
```

Возможно, вам уже удалось заметить проблему, связанную с этим кодом, но все же давайте представим, что вы уже написали остальную часть кода программы имитации светофора, насчитывающую тысячи строк, и указанная проблема ускользнула от вашего внимания. Когда вы запустите

программу, она не потерпит крах, чего нельзя будет сказать о ваших виртуальных автомобилях!

Поскольку, как мы предположили, остальная часть программы уже написана, вы не имеете ни малейшего представления о том, в каком ее разделе может скрываться ошибка. Возможно, она затаилась в коде, имитирующем движение машин, или, может быть, в коде, имитирующем действия виртуальных водителей. Прежде чем вы догадаетесь, что следы ошибки ведут к функции `switchLights()`, может пройти немало времени.

Но если бы в процессе написания функции `switchLights()` вы добавили утверждение, проверяющее, что в любой момент времени *по крайней мере один сигнал светофора красный*, то поместили бы в конце функции следующий код.

```
assert 'red' in stoplight.values(), 'Ни один из сигналов не
↳ является красным! ' + str(stoplight)
```

Программа, содержащая это утверждение, завершится аварийно с выдачей следующего сообщения об ошибке.

```
  Traceback (most recent call last):
    File "carSim.py", line 14, in <module>
      switchLights(market_2nd)
    File "carSim.py", line 13, in switchLights
      assert 'red' in stoplight.values(), 'Ни один из сигналов не
      является красным! ' + str(stoplight)
❶ AssertionError:Ни один из сигналов светофора не является
  красным! {'ns': 'yellow', 'ew':'green'}
```

Здесь очень важна строка `AssertionError` ❶. Несмотря то что решение, приводящее к аварийному завершению работы программы, далеко не идеально, оно позволяет незамедлительно известить вас о нарушении проверяемого условия: ни в одном из направлений не горит красный сигнал, а это означает, что движение разрешено одновременно по двум пересекающимся проезжим частям. Заблаговременно обнаруживая этот факт уже на ранних стадиях выполнения программы, вы обеспечиваете значительную экономию своих усилий по ее отладке в будущем.

Отключение утверждений

Механизм утверждений можно отключить, указав в командной строке флаг `-O` при запуске Python. Эта мера вполне уместна, когда этап написания и отладки программы завершен и вы не хотите, чтобы работа программы замедлялась за счет выполнения профилактических проверок с помощью утверждений (хотя в большинстве случаев наличие утверждений не

оказывает существенного влияния на скорость выполнения программы). Утверждения предназначены для программ, находящихся на стадии разработки, но не для готового продукта. К тому времени, когда вы передадите программу в эксплуатацию потребителям, она должна быть свободна от ошибок и не должна нуждаться в проверке соблюдения ограничений. Более подробно о том, как запустить с флагом `-O` программу, которая, как ожидается, уже отлажена, речь идет в приложении Б.

Протоколирование

Если вы уже использовали инструкции `print()` для вывода значений интересующих вас переменных в ходе выполнения программы, то можете считать, что с одной из форм *протоколирования* вы уже знакомы. Средства протоколирования позволяют получать ценнейшую информацию о том, что именно и в какой последовательности происходит в программе. Модуль `logging` в Python упрощает создание и ведение журнала отчетов на основе подготовленных вами сообщений. Сообщения протоколирования включают описание того, когда именно программа достигла вызова функции протоколирования, а также список значений указанных вами переменных в данный момент времени. С другой стороны, отсутствие сообщения свидетельствует о том, что данная часть кода была пропущена и не выполнялась.

Использование модуля `logging`

Чтобы активизировать модуль `logging` для вывода сообщений протоколирования на экран в процессе выполнения программы, скопируйте приведенный ниже код в начало своей программы (но после "магической" строки запуска Python, начинающейся символами `#!`).

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -
↳%(levelname)s - %(message)s')
```

Детали того, как все это работает, для вас несущественны, но вам будет полезно знать, что Python каждый раз, когда протоколирует событие, создает объект `LogRecord`, в котором хранится информация о данном событии. Функция `basicConfig()` модуля `logging` позволяет конкретизировать, какую именно информацию об объекте `LogRecord` и в какой именно форме вы хотите видеть.

Предположим вы написали функцию для расчета факториала числа. Например, факториал числа `4` равен произведению `1 × 2 × 3 × 4`, т.е. `24`. Факториал числа `7` равен `1 × 2 × 3 × 4 × 5 × 6 × 7`, или `5040`. Откройте новое окно в

файловом редакторе и введите приведенный ниже код. В этом коде содержится ошибка, но вы дополнительно предусмотрите несколько сообщений, предназначенных для записи в журнал, которые помогут вам выяснить, где именно что-то идет не так, как нужно. Сохраните программу в файле *factorialLog.py*.

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -
%(levelname)s - %(message)s')
logging.debug('Начало программы')

def factorial(n):
    logging.debug('Начало factorial(%s%%)' % (n))
    total = 1
    for i in range(n + 1):
        total *= i
        logging.debug('i = ' + str(i) + ', total = ' +
str(total))
    logging.debug('Конец factorial(%s%%)' % (n))
    return total

print(factorial(5))
logging.debug('Конец программы')
```

В этом коде для вывода протоколируемой информации используется функция `logging.debug()`. Она вызывает функцию `basicConfig()`, которая выводит строку информации. Формат вывода будет соответствовать тому формату, который задан в функции `basicConfig()`, и включать сообщения, переданные функции `debug()`. Вызов `print(factorial(5))` является частью исходной программы, поэтому результат будет отображаться даже в том случае, если средства протоколирования отключены.

Результирующий вывод будет иметь примерно следующий вид.

```
2015-05-23 16:20:12,664 - DEBUG - Начало программы
2015-05-23 16:20:12,664 - DEBUG - Начало factorial(5)
2015-05-23 16:20:12,665 - DEBUG - i = 0, total = 0
2015-05-23 16:20:12,668 - DEBUG - i = 1, total = 0
2015-05-23 16:20:12,670 - DEBUG - i = 2, total = 0
2015-05-23 16:20:12,673 - DEBUG - i = 3, total = 0
2015-05-23 16:20:12,675 - DEBUG - i = 4, total = 0
2015-05-23 16:20:12,678 - DEBUG - i = 5, total = 0
2015-05-23 16:20:12,680 - DEBUG - Конец factorial(5)
0
2015-05-23 16:20:12,684 - DEBUG - Конец программы
```

Функция `factorial()` возвращает значение `0` для факториала числа `5`, что неверно. В цикле `for` значение переменной `total` должно умножаться

на числа от 1 до 5. Однако сообщения, отображаемые функцией `logging.debug()`, указывают на то, что начальным значением переменной `i` является 0, а не 1. Поскольку результатом умножения любого числа на 0 всегда будет 0, оставшаяся часть итераций также дает неверный результат для переменной `total`. Сообщения журнала отчетов являются теми самыми "хлебными крошками", идя по которым, вам будет легче выяснить, в каком месте программы что-то пошло не так, как надо.

Замените строку `for i in range(n + 1):` строкой `for i in range(1, n + 1):` и вновь выполните программу. Результат должен выглядеть примерно так.

```
2015-05-23 17:13:40,650 - DEBUG - Начало программы
2015-05-23 17:13:40,651 - DEBUG - Начало factorial(5)
2015-05-23 17:13:40,651 - DEBUG - i = 1, total = 1
2015-05-23 17:13:40,654 - DEBUG - i = 2, total = 2
2015-05-23 17:13:40,656 - DEBUG - i = 3, total = 6
2015-05-23 17:13:40,659 - DEBUG - i = 4, total = 24
2015-05-23 17:13:40,661 - DEBUG - i = 5, total = 120
2015-05-23 17:13:40,661 - DEBUG - Конец factorial(5)
120
2015-05-23 17:13:40,666 - DEBUG - Конец программы
```

Теперь вызов `factorial(5)` возвращает корректное значение 120. Сообщения протоколирования показывают: именно то, что происходит в цикле, непосредственно приводит к ошибке.

Как видите, вызовы `logging.debug()` выводят не только передаваемые им строки, но и временные метки, а также слово *DEBUG*.

Не выполняйте отладку с помощью инструкции `print()`

Подход, основанный на импортировании модуля `logging` и вызове функции `logging.basicConfig(level=logging.DEBUG, format = '%(asctime)s - %(levelname)s - %(message)s')`, может выглядеть несколько громоздким. Не исключено, что у вас возникнет желание использовать вместо него обычные вызовы `print()`, однако не поддавайтесь этому искушению! Закончив отладку, вы потратите массу времени на удаление из кода вызовов `print()` для каждого сообщения. Более того, при этом не исключено, что вы случайно удалите полезные вызовы `print()`, никак не связанные с сообщениями протоколирования. Работа с сообщениями протоколирования удобна тем, что вы можете предусмотреть в своей программе столько сообщений, сколько пожелаете, а впоследствии в любой момент сможете отключить их, добавив единственный вызов `logging.disable(logging.CRITICAL)`. В отличие от функции `print()`, модуль `logging` упрощает переключение между режимами отображения и сокрытия сообщений протоколирования.

Сообщения протоколирования предназначены для программистов, а не для пользователей. Пользователя не интересует содержимое словаря, за которым вы хотите наблюдать в процессе отладки; для подобных целей используйте сообщения протоколирования. Для вывода сообщений, предназначенных для пользователя, таких как "Файл не найден" или "Недопустимый ввод; пожалуйста, введите число", следует использовать вызовы `print()`. Ведь не захотите же вы лишить пользователя доступа к полезной информации после того, как отключите вывод сообщений журнала отчетов.

Уровень критичности сообщений

В системе протоколирования так называемые *уровни критичности* (уровни журналирования) сообщений позволяют классифицировать сообщения по степени важности. Существует пять уровней критичности сообщений протоколирования, описанных в табл. 10.1 в порядке возрастания их важности. На каждом уровне сообщения могут выводиться с использованием различных функций протоколирования.

Таблица 10.1. Уровни критичности сообщений в Python

Уровень	Функция протоколирования	Описание
`DEBUG`	`logging.debug()`	Самый низкий уровень. Используется для вывода подробных сведений технического характера. Обычно вы заинтересованы в этих сообщениях в процессе диагностирования проблем
`INFO`	`logging.info()`	Используется для записи информации об обычных событиях, происходящих в программе, или для подтверждения нормального хода работы программы
`WARNING`	`logging.warning()`	Используется для индикации потенциально опасных ситуаций, которые не препятствуют работе программы, но могут привести к этому в будущем
`ERROR`	`logging.error()`	Используется для записи информации об ошибке, которая привела к тому, что программа не смогла выполнить некоторые действия
`CRITICAL`	`logging.critical()`	Наивысший уровень. Используется для индикации фатальных ошибок, которые привели или могут привести к полному прекращению работы программы

Ваши сообщения передаются этим функциям в виде строк. Приведенные описания — это не более чем рекомендации. В конечном счете лишь вы решаете, к какой категории следует отнести то или иное предупреждающее сообщение. Введите в интерактивной оболочке следующие команды.

```
>>> import logging
>>> logging.basicConfig(level=logging.DEBUG,
        format=' %(asctime)s - %(levelname)s - %(message)s')
>>> logging.debug('Отладочная информация.')
2015-05-18 19:04:26,901 - DEBUG - Отладочная информация.
>>> logging.info('Работает модуль logging.')
2015-05-18 19:04:35,569 - INFO - Работает модуль logging.
>>> logging.warning('Риск получения сообщения об ошибке.')
2015-05-18 19:04:56,843 - WARNING - Риск получения сообщения об ошибке.
>>> logging.error('Произошла ошибка.')
2015-05-18 19:05:07,737 - ERROR - Произошла ошибка.
>>> logging.critical('Восстановление работоспособности программы
↳ невозможно')
2015-05-18 19:05:45,794 - CRITICAL - Восстановление работоспособности про-
граммы невозможно
```

Преимуществом использования уровней критичности сообщений является то, что они позволяют задавать граничный приоритет сообщений, подлежащих отслеживанию. Передача значения `logging.DEBUG` в качестве именованного аргумента функции `basicConfig()` приведет к тому, что отображаться будут сообщения всех уровней критичности (из которых `DEBUG` — самый низкий уровень). После того как разработка программы пройдет определенные этапы, вас будет интересовать лишь вывод информации об ошибках. В этом случае вы сможете задать для функции `basicConfig()` в качестве аргумента, определяющего уровень критичности сообщений, значение `logging.ERROR`. В результате будут отображаться лишь сообщения категорий `ERROR` и `CRITICAL`, а сообщения категорий `DEBUG`, `INFO` и `WARNING` будут игнорироваться.

Отключение протоколирования

Завершив отладку программы, вы, вероятно, захотите избавиться от назойливых сообщений, захламляющих экран. Вам не придется удалять все вызовы функций протоколирования вручную, поскольку вместо вас это может сделать функция `logging.disable()`. Вам остается лишь передать ей желаемый уровень критичности ошибок, и она подавит вывод сообщений, относящихся к этому и более низким уровням. Следовательно, если вы захотите полностью отключить средства протоколирования, вам будет достаточно добавить в свою программу вызов `logging.disable(logging.CRITICAL)`. Например, введите в интерактивной оболочке следующие команды.

```
>>> import logging
>>> logging.basicConfig(level=logging.INFO,
↳ format=' %(asctime)s - %(levelname)s - %(message)s')
>>> logging.critical('Критическая ошибка! Критическая ошибка!')
```

```
2015-05-22 11:10:48,054 - CRITICAL - Критическая ошибка!
Критическая ошибка!
>>> logging.disable(logging.CRITICAL)
>>> logging.critical('Критическая ошибка! Критическая ошибка!')
>>> logging.error('Ошибка! Ошибка!')
```

Поскольку вызов `logging.disable()` отключает все следующие за ним инструкции протоколирования, вы, вероятно захотите добавить его в свою программу за строкой `import logging`. Благодаря этому вы сможете быстро находить этот вызов, чтобы при необходимости "закомментировать" или "раскомментировать" его для активизации или отключения средств протоколирования.

Запись сообщений протоколирования в файл журнала

Вместо того чтобы отображать сообщения протоколирования на экране, их можно записывать в текстовый файл журнала отчетов (жарг. *лог-файл*). Для этого следует воспользоваться функцией `logging.basicConfig()`, которая принимает именованный аргумент `filename`.

```
import logging
logging.basicConfig(filename='myProgramLog.txt',
↳ level=logging.DEBUG,
↳ format='%(asctime)s - %(levelname)s - %(message)s')
```

Здесь сообщения протоколирования сохраняются в файле *myProgramLog.txt*. Какими бы полезными ни были сообщения протоколирования, они могут захламлять экран и затруднять чтение вывода программы. Запись этих сообщений в файл журнала позволяет освободить экран, а ознакомиться с текстом сообщений можно будет уже после завершения выполнения программы. Для открытия файла журнала отчетов можно воспользоваться любым текстовым редактором, таким как Блокнот или TextEdit.

Отладчик IDLE

Отладчик — это средство IDLE, которое предоставляет возможность пошагового выполнения программы по одной строке за раз. При этом отладчик выполняет одну строку кода, а затем переходит в состояние ожидания, пока не получит команду продолжить выполнение. Выполнение программы под управлением отладчика позволяет наблюдать за значениями переменных в любом заданном месте программы на протяжении ее срока существования. Отладчик — незаменимое средство для обнаружения логических ошибок в программе.

Чтобы активизировать отладчик IDLE, щелкните на пунктах меню Debug⇨Debugger (Отладка⇨Отладчик) в окне интерактивной оболочки для открытия окна Debug Control (Управление отладкой), как показано на рис. 10.1.

Когда откроется окно Debug Control, установите все четыре флажка, Stack (Стек), Locals (Локальные переменные), Source (Исходный код) и Globals (Глобальные переменные), для отображения полного набора разделов отладочной информации. Всякий раз, когда вы будете запускать программу из окна файлового редактора при открытом окне Debug Control, отладчик будет приостанавливать выполнение программы перед первой инструкцией и отображать следующую информацию:

- строку кода, которая будет выполнена следующей;
- список всех локальных переменных и их значения;
- список всех глобальных переменных и их значения.

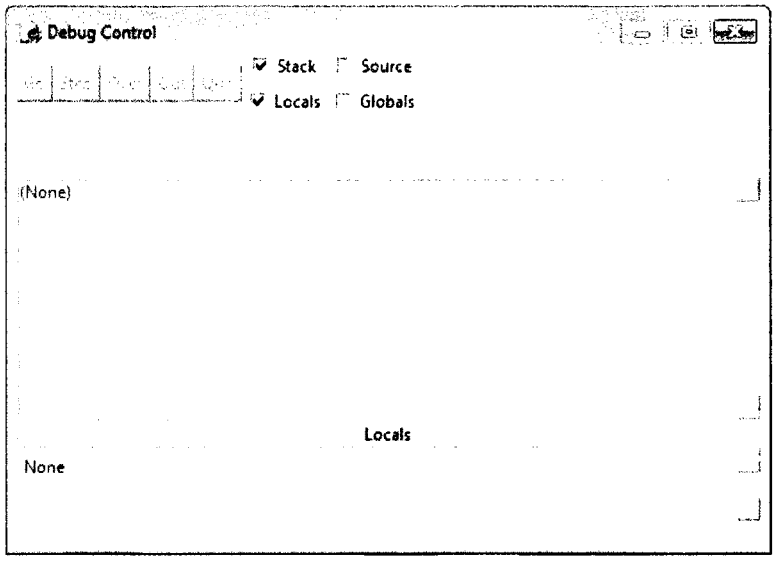


Рис. 10.1. Окно Debug Control

Позже вы увидите, что в списке глобальных переменных присутствуют такие имена, как as `__builtins__`, `__doc__`, `__file__` и другие, которые вы не определяли. Эти переменные Python автоматически устанавливает всякий раз, когда вы запускаете программу. Их рассмотрение выходит за рамки данной книги, и вы можете смело их игнорировать.

Программа будет оставаться в состоянии ожидания до тех пор, пока вы не щелкнете на одной из следующих пяти кнопок в окне Debug Control: Go, Step, Over, Out или Quit.

Кнопка Go

Щелчок на кнопке Go (Выполнить) запускает обычный процесс выполнения программы до ее полного завершения или до тех пор, когда будет достигнута *точка останова*. (О точках останова речь пойдет далее.) Если вы завершили отладку и хотите, чтобы программа продолжила выполнение в обычном режиме, щелкните на кнопке Go.

Кнопка Step

Щелчок на кнопке Step (Шаг с заходом) приводит к выполнению следующей строки кода и приостановке программы. Если значения локальных и глобальных переменных изменились, то их список в окне Debug Control обновится. Если следующей строкой кода является вызов функции, то отладчик выполнит "шаг с заходом", т.е. выполнит вызов функции и приостановится на первой строке ее кода.

Кнопка Over

Щелчок на кнопке Over (Шаг с обходом) приводит к выполнению следующей строки кода, как и при щелчке на кнопке Step. Но если следующей строкой кода является вызов функции, то отладчик выполнит "шаг с обходом", т.е. будет выполнен не только вызов функции, но и весь ее код, и после возврата из функции дальнейшее выполнение приостановится. Например, если следующей строкой кода является вызов функции `print()`, то вас интересует лишь вывод переданной ей строки на экран, а не сам код функции. Поэтому обычно кнопкой Over пользуются чаще, чем кнопкой Step.

Кнопка Out

Щелчок на кнопке Out (Шаг с выходом) приводит к выполнению строк кода в обычном режиме до тех пор, пока не будет выполнен возврат из текущей функции. Если перед этим вы выполнили шаг с заходом в функцию с помощью кнопки Step, а теперь просто хотите продолжить выполнение инструкций вплоть до возврата из функции, щелкните на кнопке Out для выполнения "шага с выходом" из текущего вызова.

Кнопка Quit

Если вы хотите полностью прекратить отладку без дальнейшего выполнения остальной части программы, щелкните на кнопке Quit (Выход), что приведет к немедленному прекращению выполнения программы. Если же вы хотите возобновить работу программы в обычном режиме, то отключите отладчик, повторно выбрав пункты меню Debug⇨Debugger.

Отладка программы для сложения чисел

Откройте новое окно в файловом редакторе и введите следующий код.

```
print('Введите первое слагаемое:')
first = input()
print('Введите второе слагаемое:')
second = input()
print('Введите третье слагаемое:')
third = input()
print('Сумма равна ' + first + second + third)
```

Сохраните текст программы в файле *buggyAddingProgram.py* и выполните ее сначала при отключенном отладчике. Вы должны получить следующий вывод.

```
Введите первое слагаемое:
5
Введите второе слагаемое:
3
Введите третье слагаемое::
42
Сумма равна 5342
```

Программа не завершилась аварийно, однако результат явно неверный. Активизируйте окно Debug Control и вновь выполните программу, на этот раз под управлением отладчика.

Нажмите клавишу <F5> или выберите пункты меню Run⇨Run Module (предварительно активизировав отладчик с помощью пунктов меню Debug⇨Debugger и установив все четыре флажка в окне Debug Control), что приведет к запуску программы и ее приостановке на строке 1. Отладчик всегда приостанавливается на той строке кода, которая будет выполняться следующей. Окно Debug Control примет вид, показанный на рис. 10.2.

Щелкните на кнопке Over один раз для выполнения первого вызова функции `print()`. В данном случае необходимо использовать кнопку Over, а не Step, поскольку в "заходе" в код функции `print()` нет необходимости. Содержимое окна Debug Control обновится, и теперь оно будет отображать информацию, относящуюся к строке 2, а сама строка 2 в окне файлового редактора будет подсвечена (рис. 10.3). Тем самым вы сможете легко определить, какая именно строка кода выполняется в данный момент.

Вновь щелкните на кнопке Over, чтобы выполнился вызов функции `input()`; при этом на все то время, пока IDLE будет ожидать вашего ввода в окне интерактивной оболочки в ответ на вызов функции `input()`, кнопки в окне Debug Control деактивизируются. Введите 5 и нажмите клавишу <Enter>. После этого кнопки в окне Debug Control вновь активизируются.

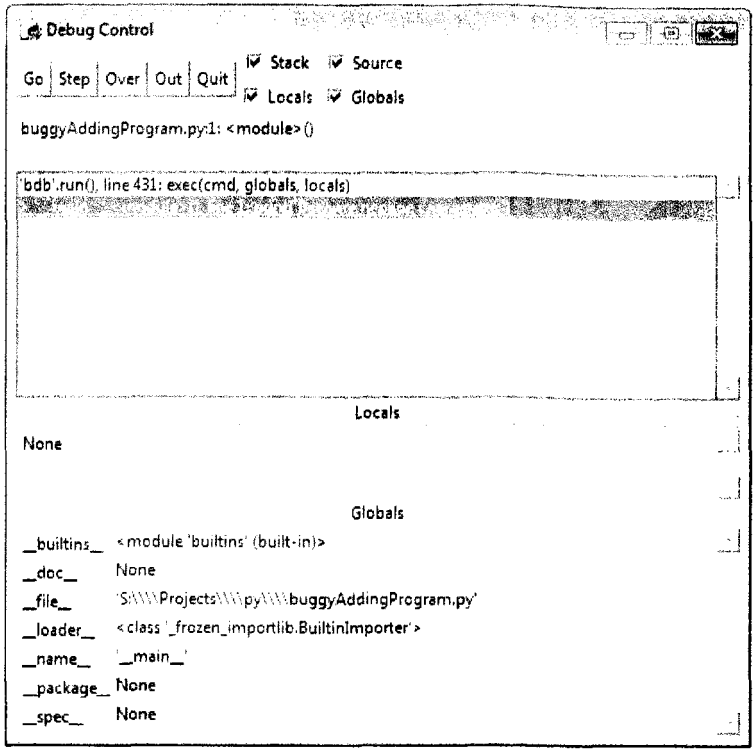


Рис. 10.2. Вид окна Debug Control при первоначальном запуске программы под управлением отладчика

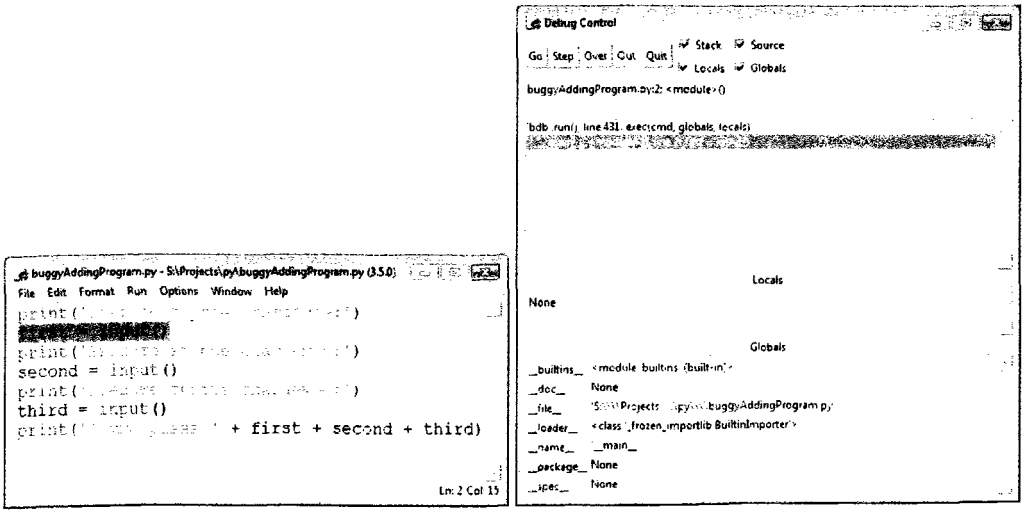


Рис. 10.3. Вид окна Debug Control после щелчка на кнопке Over

Продолжая щелкать на кнопке **Over**, введите **3** и **42** в качестве следующих двух слагаемых, пока отладчик не достигнет строки 7, т.е. последнего вызова функции `print()` в программе. В этот момент окно **Debug Control** должно выглядеть так, как показано на рис. 10.4. Как можно увидеть в разделе **Globals**, первая, вторая и третья переменные содержат строковые значения `'5'`, `'3'` и `'42'`, а не целочисленные значения `5`, `3` и `42`. При достижении последней строки кода вместо сложения трех чисел выполняется конкатенация указанных строк, что приводит к ошибке.

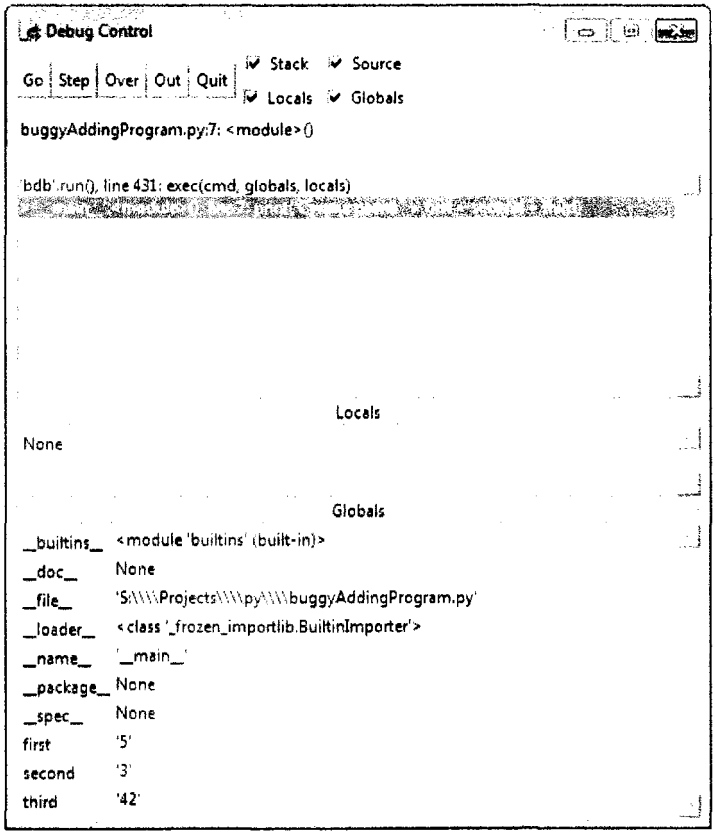


Рис. 10.4. Вид окна Debug Control при достижении последней строки кода. Переменные принимают строковые значения, что приводит к ошибке

Пошаговое выполнение программы с помощью отладчика чрезвычайно информативно, однако замедляет работу программы. Во многих случаях вам хотелось бы, чтобы программа выполнялась в обычном режиме, пока не достигнет определенной строки кода. Можно сконфигурировать отладчик для такого режима работы, используя точки останова.

Точки останова

Точка останова, установленная для определенной строки кода, сообщает отладчику, что по достижении данной строки выполнение программы должно приостановиться. Откройте новое окно файлового редактора и введите следующую программу, имитирующую подбрасывание монеты 1000 раз. Сохраните программу в файле *coinFlip.py*.

```
import random
heads = 0
for i in range(1, 1001):
❶     if random.randint(0, 1) == 1:
          heads = heads + 1
      if i == 500:
❷ print('Полпути пройдено!')
print('Орел выпал ' + str(heads) + ' раз.')
```

Вызов `random.randint(0, 1)` ❶ в половине случаев будет возвращать значение `0`, а в половине — `1`. Этот факт можно использовать для имитации подбрасывания монеты с равными вероятностями выпадения "орла" и "решки", причем в нашем случае "орлу" соответствует значение `1`. Выполнив программу без отладчика, вы очень быстро получите примерно следующий вывод.

```
Полпути пройдено!
Орел выпал 490 раз.
```

Если вы будете выполнять программу под управлением отладчика, то к тому моменту, когда программа закончит выполнение, вам придется щелкнуть на кнопке Over не одну тысячу раз. Если вас интересует, сколько раз выпала решка, когда программа выполнилась наполовину, т.е. при совершении 500 "подбрасываний" монеты из 1000 возможных, можете просто задать точку останова в строке `print('Полпути пройдено!')` ❷. Чтобы задать точку останова, щелкните правой кнопкой мыши на этой строке в окне файлового редактора и выберите во всплывающем меню пункт Set Breakpoint (Задать точку останова), как показано на рис. 10.5.

Задавать точку останова для строки с инструкцией `if` не следует, поскольку данная инструкция выполняется на каждой итерации цикла. Если задать точку останова в коде инструкции `if`, то отладчик будет прерывать выполнение только в тех случаях, когда поток управления будет входить в тело этой инструкции.

Строка, для которой задана точка останова, подсвечивается в файловом редакторе желтым цветом. Когда вы запускаете программу под управлением отладчика, ее выполнение начинается с состояния ожидания в первой

строке. Но если вы щелкнете на кнопке Go, то программа станет выполнять инструкцию за инструкцией до тех пор, пока не достигнет строки, для которой задана точка останова. Далее можете щелкать на кнопках Go, Over, Step и Out, чтобы продолжать выполнение в соответствующем режиме.

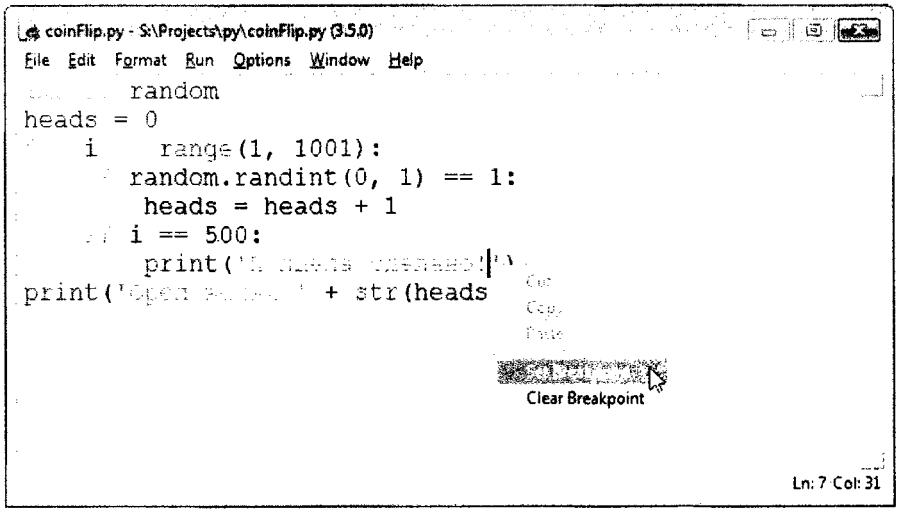


Рис. 10.5. Задание точки останова

Чтобы удалить точку останова, щелкните правой кнопкой мыши на соответствующей строке в окне файлового редактора и выберите в открывшемся меню пункт Clear Breakpoint (Отменить точку останова). Желтая подсветка строки исчезнет, и в будущем выполнение программы на этой строке не будет прерываться.

Резюме

Утверждения, исключения, протоколирование и отладка — незаменимые инструменты, предназначенные для обнаружения и устранения логических ошибок в программах. Механизм утверждений, обеспечиваемый инструкцией `assert` в Python, — отличное средство реализации "профилактических проверок", обеспечивающее раннее обнаружение потенциальных ошибок, если не выполняются необходимые условия. Утверждения предназначены для выявления только тех ошибок, при возникновении которых программа не будет пытаться восстановить работоспособное состояние и должна лишь быстро завершиться аварийно. В противном случае следует использовать исключения.

Исключения можно перехватывать и обрабатывать с помощью инструкций `try` и `except`. Модуль `logging`, обеспечивающий возможность учета уровней критичности ошибок при записи сообщений о них в текстовый

файл журнала, предоставляет неплохие возможности для наблюдения за состоянием программы во время ее выполнения и гораздо более удобен в использовании, чем функция `print()`.

Отладчик обеспечивает пошаговое выполнение программы по одной инструкции за раз. Альтернативной возможностью является выполнение программы с обычной скоростью с возможностью ее прерывания отладчиком при достижении строк кода, для которых заданы точки останова. Используя отладчик, можно наблюдать за значениями любых переменных во время выполнения программы на протяжении всего срока ее существования.

Использование перечисленных средств и методик отладки облегчит вам написание правильно работающих программ. Непреднамеренное внесение логических ошибок в код – неотъемлемый атрибут работы программиста, от которого не помогает полностью избавиться даже многолетний опыт программирования.

Контрольные вопросы

1. Напишите инструкцию `assert`, которая возбуждает исключение `AssertionError`, если значение переменной `spam` представляет собой целое число, меньшее 10.
2. Напишите инструкцию `assert`, которая возбуждает исключение `AssertionError`, если переменные `eggs` и `bacon` содержат одинаковые строки, при этом регистр букв не учитывается (например, они содержат строки `'hello'` и `'hello'` или же строки `'goodbye'` и `'GOODbye'`, которые также считаются одинаковыми).
3. Напишите инструкцию `assert`, которая всегда возбуждает исключение `AssertionError`.
4. Какие две строки кода должна иметь ваша программа для того, чтобы вызывать функцию `logging.debug()`?
5. Какие две строки кода должна иметь ваша программа для того, чтобы вызывать функцию `logging.debug()`, записывающую сообщение в файл журнала *programLog.txt*?
6. Назовите пять уровней критичности сообщений.
7. Какую строку кода вы можете добавить для того, чтобы отключить все сообщения протоколирования в своей программе?
8. Почему для отображения одного и того же сообщения лучше использовать сообщения протоколирования, чем вызов функции `print()`?
9. Чем различаются команды, выполняемые после щелчков на кнопках **Step**, **Over** и **Out** в окне `Debug Control`?

10. Когда отладчик прервет выполнение программы, если вы щелкнете на кнопке Go в окне Debug Control?
11. Что такое точка останова?
12. Как задать точку останова для строки кода в IDLE?

Учебный проект

Чтобы закрепить полученные знания на практике, реализуйте предложенную ниже задачу.

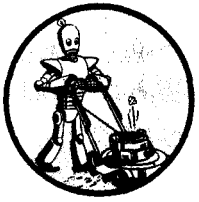
Отладка программы, имитирующей подбрасывание монеты

Приведенная ниже программа предназначена для имитации игры в угадывание того, какой стороной будет обращена вверх упавшая после подбрасывания монета. В распоряжении игрока имеются два варианта ответа (это простая игра). Однако в программе содержатся логические ошибки. Запустите программу несколько раз для того, чтобы обнаружить ошибки, препятствующие ее правильной работе.

```
import random
guess = ''
while guess not in ('орел', 'решка'):
    print('Угадайте результат подбрасывания монеты!
↳Введите орел или решка:')
    guess = input()
toss = random.randint(0, 1) # 0 - решка, 1 - орел
if toss == guess:
    print('Есть!')
else:
    print('Увы! Попробуйте снова!')
    guesss = input()
    if toss == guess:
        print('Есть!')
    else:
        print('Нет. Вам действительно не везет в этой игре.')
```

11

АВТОМАТИЧЕСКИЙ СБОР ДАННЫХ В ИНТЕРНЕТЕ



В те редкие пугающие минуты, когда остаешься без Wi-Fi, отчетливо осознаешь, насколько то, что ты делаешь с помощью компьютера, связано с Интернетом. Я часто замечаю, что по устоявшейся привычке меня так и тянет проверить электронную почту, прочитать сообщения друзей в Твиттере или ответить на вопросы типа "Были ли у Кертвуда Смита главные роли до того, как его пригласили сняться в оригинальном фильме *Робокоп*, выпущенном в 1987 году?"[1]

Поскольку значительная часть работ, выполняемых с помощью компьютера, связана с выходом в Интернет, было бы замечательно, если бы ваши программы могли обеспечивать это подключение. *Автоматический сбор данных в Интернете* (жарг. *веб-скрапинг*) — это термин, обозначающий использование программы для загрузки и обработки содержимого из Интернета. К примеру, Google выполняет множество таких программ, индексируя веб-страницы для своей поисковой системы. В этой главе вы познакомитесь с несколькими модулями, которые облегчают автоматический сбор данных с веб-страниц с помощью Python. Ниже приведен перечень этих модулей.

- **webbrowser**. Поставляется вместе с Python и предназначен для открытия браузера на определенной веб-странице.
- **Requests**. Загружает файлы и веб-страницы из Интернета.
- **Beautiful Soup**. Предназначен для синтаксического анализа кода HTML — языка, на котором написаны веб-страницы.
- **Selenium.** Запускает веб-браузер и управляет его работой; способен заполнять формы и имитировать щелчки мышью в браузере.

[1] Ответ — нет.

Проект: программа *mapIt.py* с модулем webbrowser

Функция `open()` модуля `webbrowser` запускает браузер с использованием указанного URL-адреса. Введите в интерактивной оболочке следующие команды:

```
>>> import webbrowser
>>> webbrowser.open('http://inventwithpython.com/')
```

В браузере откроется веб-страница, расположенная по URL-адресу `http://inventwithpython.com/`. Это почти единственное, что может делать модуль `webbrowser`. Тем не менее функция `open()` позволяет делать интересные вещи. Например, вставка почтового адреса в приложение Google Maps через буфер обмена — довольно хлопотная задача. Вы можете сэкономить на этом некоторое время, написав простой сценарий, который будет автоматически запускать карту в вашем браузере, используя содержимое буфера обмена. Благодаря этому вам останется лишь скопировать адрес в буфер обмена и запустить сценарий, который и загрузит карту.

Вот что должна делать такая программа:

- получать почтовый адрес из командной строки или буфера обмена;
- открывать браузер на странице Google Maps, соответствующей указанному адресу.

Это означает, что ваш код должен выполнять следующие операции:

- читать аргументы командной строки из списка `sys.argv`;
- читать содержимое буфера обмена;
- вызывать функцию `webbrowser.open()` для открытия браузера.

Откройте новое окно файлового редактора и сохраните его в файле *mapIt.py*.

Шаг 1. Определение URL-адреса

Руководствуясь инструкциями, приведенными в приложении Б, настройте файл *mapIt.py* таким образом, чтобы при его запуске, например, с помощью команды

```
C:\> mapit 870 Valencia St, San Francisco, CA 94110
```

сценарий использовал в качестве почтового адреса аргументы командной строки, а не содержимое буфера обмена. Если же аргументы командной строки не заданы, то программа будет знать, что в этом случае нужно использовать буфер обмена.

Прежде всего необходимо определить, какой URL-адрес следует использовать для указанного почтового адреса. Если вы загрузите в браузер страницу `http://maps.google.com/` и выполните поиск по интересующему вас почтовому адресу, то URL-адрес, отображаемый в адресной строке браузера, будет выглядеть примерно так:

```
https://www.google.com/maps/place/870+Valencia+St/@37.7590311,
122.4215096,17z/data=!3m1!4b1!4m2!3m1!1s0x808f7e3dadc07a37:
0xc86b0b2bb93b73d8
```

URL-адрес содержит почтовый адрес, но, кроме него, включает также дополнительный текст. Веб-сайты часто вставляют дополнительные данные в URL-адреса, которые используются для отслеживания привычек посетителей или адаптации сайта к запросам пользователей. Если вы исключите эти данные и попытаетесь выполнить в браузере переход по упрощенному адресу `https://www.google.com/maps/place/870+Valencia+St+San+Francisco+CA/`, то обнаружите, что вам по-прежнему доставляется нужная страница. Следовательно, нам достаточно настроить программу на страницу `'https://www.google.com/maps/place/`*ваша_адресная_строка*`'`, где *ваша_адресная_строка* — это почтовый адрес, в соответствии с которым должна быть открыта карта.

Шаг 2. Обработка аргументов командной строки

Введите следующий код.

```
#! python3
# mapIt.py - Запускает карту в браузере, извлекая почтовый адрес
#            из командной строки или буфера обмена.

import webbrowser, sys
if len(sys.argv) > 1:
    # Получение почтового адреса из командной строки.
    address = ' '.join(sys.argv[1:])

# TODO: Получить почтовый адрес из буфера обмена.
```

Первая строка сценария — "магическая" (начинается символами `#!`). Далее мы импортируем модули `webbrowser` (необходим для запуска сценария) и `sys` (необходим для чтения возможных аргументов командной строки). В переменной `sys.argv` хранится список, включающий имя файла программы и аргументы командной строки. Если этот список содержит не только имя файла программы, то вызов `len(sys.argv)` возвратит значение, большее `1`, указывающее на то, что в командной строке действительно предоставлены аргументы.

Обычно аргументы командной строки разделяются пробелами, но в данном случае мы желаем интерпретировать все аргументы как одну строку. Поскольку `sys.argv` — это список строк, его можно передать методу `join()`, который возвратит результат в виде одной строки. Имя программы в этой строке для нас не представляет интереса, поэтому мы передаем методу `join()` не весь список `sys.argv`, а его срез `sys.argv[1:]`, тем самым исключая ненужный нам элемент списка. Результирующая строка сохраняется в переменной `address`.

Если для запуска программы использовать команду такого вида:

```
mapit 870 Valencia St, San Francisco, CA 94110
```

то в переменной `sys.argv` будет содержаться следующий список:

```
['mapIt.py', '870', 'Valencia', 'St, ', 'San', 'Francisco, ',
'CA', '94110']
```

в то время как в переменной `address` будет содержаться строка `'870 Valencia St, San Francisco, CA 94110'`.

Шаг 3. Обработка содержимого буфера обмена и запуск браузера

Добавьте в программу код, выделенный ниже полужирным шрифтом.

```
#! python3
# mapIt.py - Запускает карту в браузере, извлекая почтовый адрес
#            из командной строки или буфера обмена.

import webbrowser, sys, pyperclip
if len(sys.argv) > 1:
    # Получение почтового адреса из командной строки.
    address = ' '.join(sys.argv[1:])
else:
    # Получение почтового адреса из буфера обмена.
    address = pyperclip.paste()

webbrowser.open('https://www.google.com/maps/place/' + address)
```

Если аргументы командной строки не предоставлены, программа предполагает, что адрес хранится в буфере обмена. Мы можем получить содержимое буфера обмена с помощью функции `pyperclip.paste()` и сохранить его в переменной `address`. Наконец, для запуска браузера с URL-адресом Google Maps вызывается функция `webbrowser.open()`.

Несмотря на то что некоторые из программ, которые вы напишете, сэкономят вам много часов, выполняя вместо вас трудоемкие задачи, не стоит

недооценивать программы, позволяющие сэкономить хотя бы несколько секунд вашего времени каждый раз, когда вы выполняете даже такие простые задачи, как отображение интересующего вас места на карте Google. В табл. 11.1 сравниваются шаги, которые приходится выполнять для отображения карты с помощью и без помощи программы *mapIt.py*.

Таблица 11.1. Получение карты местности с помощью и без помощи программы *mapIt.py*

Получение карты вручную	Использование программы *mapIt.py*
Выделение адреса	Выделение адреса
Копирование адреса	Копирование адреса
Открытие браузера	Запуск программы `mapIt.py`
Переход по адресу `http://maps.google.com/`	—
Щелчок в поле для ввода адреса	—
Вставка адреса	—
Нажатие клавиши <Enter>	—

Вы сами можете оценить, насколько автоматизация этой задачи позволяет сэкономить ваши усилия и время.

Идеи относительно создания аналогичных программ

Коль скоро известен нужный URL-адрес, модуль `webbrowser` избавит от необходимости самостоятельно открывать браузер и переходить на нужный веб-сайт. Вы можете встроить эту функциональность в другие программы для решения следующих задач:

- открытия всех ссылок, приведенных на странице, в отдельных вкладках браузера;
- открытия браузера на странице с прогнозом погоды по вашему региону;
- открытия нескольких сайтов социальных сетей, которые вы регулярно посещаете.

Загрузка файлов из Интернета с помощью модуля Requests

Модуль Requests упрощает загрузку файлов из Интернета, позволяя не задумываться о таких сложных вопросах, как ошибки сети, проблемы подключения и сжатие данных. Этот модуль не поставляется вместе с Python и нуждается в предварительной установке. Выполните в командной строке команду `pip install requests`. (Подробное описание процесса установки модулей, разработанных третьими сторонами, приведено в приложении А.)

Необходимость в написании модуля Requests была продиктована тем обстоятельством, что модуль Python `urllib2` слишком сложен в использовании. Фактически вам следует наглухо вымарать маркером весь этот абзац. Забудьте вообще о том, что я когда-либо упоминал о модуле `urllib2`. Если вам нужно загрузить информацию из Интернета, используйте для этого исключительно модуль Requests.

Вашим следующим шагом должно быть выполнение простого теста, который позволит убедиться в корректности установки модуля Requests:

```
>>> import requests
```

Отсутствие сообщений об ошибке будет свидетельствовать о том, что установка модуля Requests была успешной.

Загрузка веб-страницы посредством функции `requests.get()`

Функция `requests.get()` принимает строку с URL-адресом, по которому должна осуществляться загрузка. Вызвав функцию `type()` для значения, возвращенного вызовом `requests.get()`, вы увидите, что этот вызов возвращает объект `Response`, в котором содержится ответ сервера на ваш запрос. Мы еще поговорим об объекте `Response`, а пока что введите в интерактивной оболочке следующие команды, предварительно убедившись в том, что компьютер подключен к Интернету.

```
  >>> import requests
❶ >>> res = requests.get('http://www.gutenberg.org/
  ↳ cache/epub/1112/pg1112.txt')
  >>> type(res)
  <class 'requests.models.Response'>
❷ >>> res.status_code == requests.codes.ok
  True
  >>> len(res.text)
  178981
  >>> print(res.text[:250])

  The Project Gutenberg EBook of Romeo and Juliet, by William
  Shakespeare. This eBook is for the use of anyone anywhere at no
  cost and with almost no restrictions whatsoever. You may copy
  it, give it away or re-use it under the terms of the Proje
```

По указанному URL-адресу находится веб-страница, содержащая полный текст пьесы "Ромео и Джульетта", который предоставляется бесплатно в рамках проекта "Гутенберг" ❶. Проверить успешность выполнения запроса можно с помощью атрибута `status_code` объекта `Response`.

Значение `requests.codes.ok` этого атрибута говорит о том, что запрос был успешно выполнен ❷. (Кстати, в протоколе HTTP успешному запросу ("OK") соответствует код состояния 200.) Возможно, вам уже приходилось сталкиваться с кодом состояния 404 ("Not Found" – "Не найдено"). В случае успешного запроса загруженная страница сохраняется в виде строки в переменной `text` объекта `Response`. В этой переменной хранится полный текст пьесы в виде одной длинной строки; результат вызова `len(res.text)` говорит о том, что эта строка содержит более 178 000 символов. Наконец, вызов `print(res.text[:250])` отображает лишь первые 250 символов.

Проверка ошибок

Как вам уже известно, объект `Response` имеет атрибут `status_code`, сравнение которого со значением `requests.codes.ok` позволяет судить о том, была ли загрузка успешной. Однако для такой проверки существует более простой способ, который состоит в том, чтобы вызвать метод `raise_for_status()` для объекта `Response`. Данный метод возбуждает исключение, если в процессе загрузки файла произошла ошибка, и не совершает никаких действий в случае успешной загрузки. Введите в интерактивной оболочке следующие команды.

```
>>> res = requests.get('http://inventwithpython.com/
↳несуществующая_страница')
>>> res.raise_for_status()
Traceback (most recent call last):
  File "<pyshell#138>", line 1, in <module>
    res.raise_for_status()
  File "C:\Python34\lib\site-packages\requests\models.py", line 773, in
raise_for_status
    raise HTTPError(http_error_msg, response=self)
requests.exceptions.HTTPError: 404 Client Error: Not Found
```

Метод `raise_for_status()` – это эффективный способ гарантированной остановки программы в случае неудачной загрузки. Конечно же, это неплохо, ведь вы сами заинтересованы в том, чтобы при возникновении неожиданных ошибок выполнение программы было прекращено. Если же неудачная загрузка не является критическим препятствием для дальнейшего выполнения программы, можно обернуть строку кода `raise_for_status()` конструкцией `try/except`, чтобы обработать эту ошибку, не допуская аварийной остановки программы.

```
import requests
res = requests.get('http://inventwithpython.com/
↳несуществующая_страница')
try:
```

```
    res.raise_for_status()
except Exception as exc:
    print('Возникла проблема: %s' % (exc))
```

В результате данного вызова метода `raise_for_status()` программа выведет следующую информацию:

```
Возникла проблема: 404 Client Error: Not Found
```

Целесообразно всегда вызывать метод `raise_for_status()` после функции `requests.get()`. Это позволяет убедиться в том, что загрузка действительно была осуществлена, прежде чем предоставить программе возможность дальнейшего выполнения.

Сохранение загруженных файлов на жестком диске

В этом разделе будет рассказано о том, как сохранить веб-страницу в файле на жестком диске с помощью стандартной функции `open()` и метода `write()`. Однако есть некоторые нюансы. Прежде всего, необходимо открыть файл в режиме двоичной записи, передав функции `open()` строку `'wb'` в качестве второго аргумента. Даже если страница представляет собой простой текст (как, скажем, загруженный ранее текст "Ромео и Джульетты"), для поддержки кодировки Unicode необходимо записывать двоичные данные, а не текстовые.

Кодировка Unicode

Рассмотрение кодировки Unicode выходит за рамки данной книги. Для получения более подробной информации по этому вопросу можете воспользоваться следующими ресурсами в Интернете:

- *Joel on Software: The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)*: `http://www.joelonsoftware.com/articles/Unicode.html`
- *Pragmatic Unicode*: `http://nedbatchelder.com/text/unipain.html`

Записать веб-страницу в файл можно с помощью цикла `for` по возвращаемому значению метода `iter_content()` объекта `Response`.

```
>>> import requests
>>> res = requests.get('http://www.gutenberg.org/
↳cache/epub/1112/pg1112.txt')
>>> res.raise_for_status()
```

```
>>> playFile = open('RomeoAndJuliet.txt', 'wb')
>>> for chunk in res.iter_content(100000):
        playFile.write(chunk)

100000
78983
>>> playFile.close()
```

Метод `iter_content()` возвращает порции содержимого на каждой стадии цикла. Каждая порция данных — это данные байтового типа, и вам предоставляется возможность указать, сколько байтов должна содержать каждая порция. В общем случае одна сотня тысяч байтов — это вполне подходящий размер, поэтому мы передаем функции `iter_content()` значение `100000` в качестве аргумента.

Теперь в рабочем каталоге существует файл *RomeoAndJuliet.txt*. Обратите внимание на то, что это имя, под которым файл сохраняется на жестком диске, отличается от имени файла, используемого веб-страницей (*pg1112.txt*). Загрузкой содержимого веб-страниц управляет модуль Requests. Как только страница загружена, она становится простыми данными в вашей программе. Даже если соединение с Интернетом нарушится после загрузки страницы, ее содержимое останется в вашем компьютере.

Метод `write()` возвращает количество байтов, записанных в файл. В предыдущем примере первая порция включала `100000` байт, так что для оставшейся части файла понадобилось только `78983` байта.

Ниже приведен перечень отдельных стадий процесса загрузки и сохранения файла.

1. Вызов функции `requests.get()` для загрузки файла.
2. Вызов функции `open()` с аргументом `'wb'` для создания нового файла в режиме записи двоичных данных.
3. Цикл по возвращаемому значению метода `iter_content()` объекта `Response`.
4. Вызов метода `write()` на каждой итерации для записи содержимого файла.
5. Вызов метода `close()` для закрытия файла.

Это все, что вам следует знать о модуле `requests`! Возможно, использование цикла `for` и метода `iter_content()` покажется вам несколько более сложным по сравнению с технологией, основанной на применении цепочки вызовов `open()/write()/close()`, которую мы использовали для записи текстовых файлов. Однако такой подход гарантирует, что модуль Requests не будет потреблять слишком много памяти даже в случае загрузки крупных файлов. Более подробную информацию относительно других возможностей модуля Requests можно найти на сайте `http://requests.readthedocs.org/`.

HTML

Прежде чем приступить к анализу веб-страниц, следует познакомиться с основами HTML. Вы также узнаете о том, как воспользоваться мощными средствами разработки, предоставляемыми браузером, которые значительно упростят процесс веб-скрапинга.

Ресурсы для изучения HTML

Язык гипертекстовой разметки (Hypertext Markup Language – HTML) – это формат, используемый для записи веб-страниц. В этой главе предполагается, что вы уже владеете некоторыми навыками работы с HTML, но если вы нуждаетесь в практическом руководстве для начинающих, можно порекомендовать следующие сайты:

- `http://htmldog.com/guides/html/beginner/`
- `http://www.codecademy.com/tracks/web/`
- `https://developer.mozilla.org/en-US/learn/html/`

Краткие сведения по HTML

Если до этого вы сталкивались с HTML-документами лишь эпизодически, то вам будет полезно ознакомиться с приведенным ниже кратким обзором основ HTML. HTML-файл – это обычный текстовый файл с расширением *.html*. Текст в таких файлах окружается *тегами* (дескрипторами). Каждый тег представляет собой слово, заключенное в угловые скобки. Теги сообщают браузеру, как следует форматировать веб-страницу. Начальный и конечный (открывающий и закрывающий) теги могут окружать некоторый текст, вместе с которым они образуют *элемент*.

Текст (или *внутреннее HTML-содержимое*) – это содержимое, заключенное между открывающим и закрывающим тегами. Например, следующий HTML-документ отображает в браузере фразу `Здравствуй, мир!`, в которой слово `Здравствуй` выделено полужирным шрифтом:

```
<strong>Здравствуй</strong>, мир!
```

Вид этой HTML-разметки в браузере показан на рис. 11.1.

Открывающий тег `<strong>` указывает на то, что заключенный между этими тегами текст должен отображаться полужирным шрифтом. Закрывающий тег `</strong>` обозначает, где заканчивается полужирный текст.

В HTML существует множество тегов. Некоторые из них имеют дополнительные свойства в виде *атрибутов*, записываемых в угловых скобках. Например, тег `<a>` содержит текст, создающий гиперссылку. URL-адрес, на

который ссылается этот текст, определяется атрибутом `href`. Вот соответствующий пример:

```
<a href="http://inventwithpython.com">Книги по Python</a>,
бесплатно предоставляемые на сайте Эла.
```

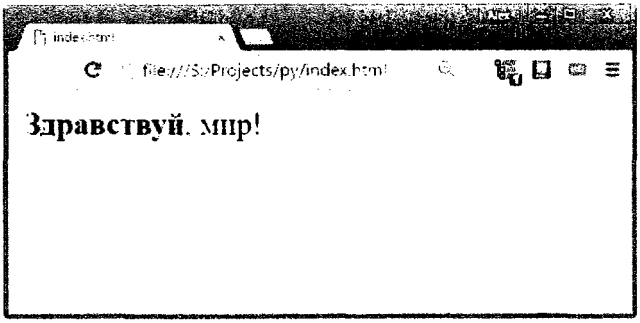


Рис. 11.1. Фраза `Здравствуй, мир!`, визуализированная в браузере

Вид этой HTML-разметки, визуализированной в браузере, показан на рис. 11.2.

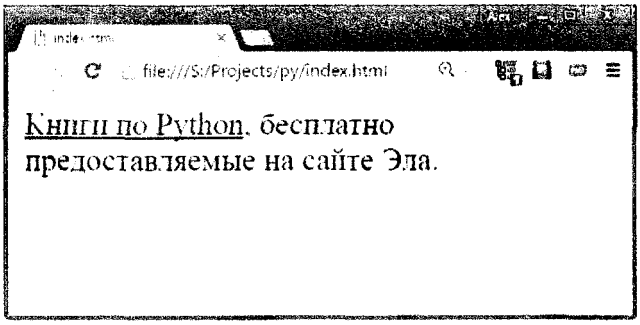


Рис. 11.2. Ссылка, визуализированная в браузере

Некоторые элементы имеют атрибут `id`, который используется в качестве уникального идентификатора элемента на странице. В своих программах вам придется часто выполнять поиск элемента по его атрибуту `id`, поэтому определение данного атрибута с использованием инструментов разработчика, предоставляемых браузером, является одной из самых распространенных задач при написании программ для веб-скрапинга.

Просмотр исходного HTML-кода веб-страницы

Так или иначе вам понадобится просматривать исходный HTML-код веб-страниц, с которыми будут работать ваши программы. Чтобы это сделать, щелкните правой кнопкой мыши (или выполните щелчок мышью

при нажатой клавише <Ctrl> в OS X) на любой веб-странице в своем браузере и выберите в открывшемся контекстном меню пункт **Просмотреть код** (View Source) или **Просмотр кода страницы** (View page source), как показано на рис. 11.3. Таким образом вы сможете просмотреть текст, который в действительности получает ваш браузер. Браузеру известно, как отображать, или, как говорят, *визуализировать*, веб-страницу на основе HTML-кода.

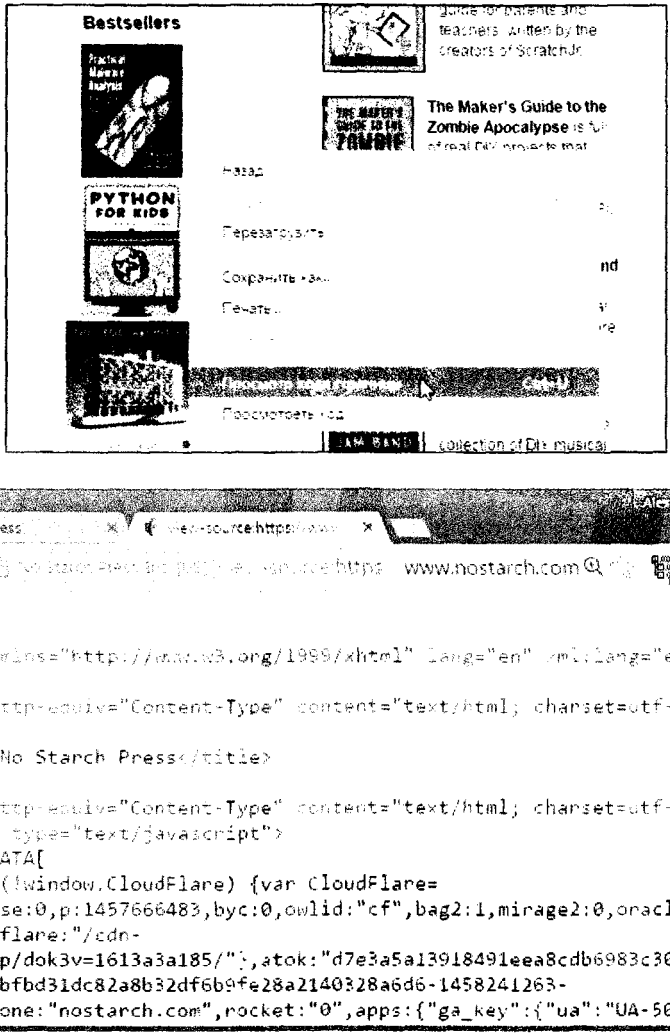


Рис. 11.3. Просмотр исходного кода веб-страницы

Я настоятельно рекомендую вам исследовать HTML-код понравившихся сайтов. Если при просмотре HTML-кода вам не все будет понятно, не огорчайтесь. Чтобы писать простые программы для веб-скрапинга, не нужно быть искусным мастером HTML – в конце концов, речь не идет о создании

собственного веб-сайта. Вам достаточно лишь знать, как организовать сбор данных с существующих сайтов.

Открытие окна инструментов разработчика в браузере

Кроме исходного кода веб-страницы, можно просматривать HTML-разметку страницы с помощью инструментов разработки, предоставляемых браузером. В браузерах Chrome и Internet Explorer для Windows инструменты разработчика уже установлены, и для начала работы с ними вам остается лишь нажать клавишу <F12> (рис. 11.4). Повторное нажатие клавиши <F12> приводит к закрытию окна инструментов разработчика. В браузере Chrome вам предоставляется дополнительная возможность доступа к инструментам разработчика путем выбора пунктов меню **Настройка и управление Google Chrome** (кнопка в конце адресной строки)⇨**Дополнительные инструменты**⇨**Инструменты разработчика**. В OS X для этого следует нажать комбинацию клавиш <⌘+Option+I>.

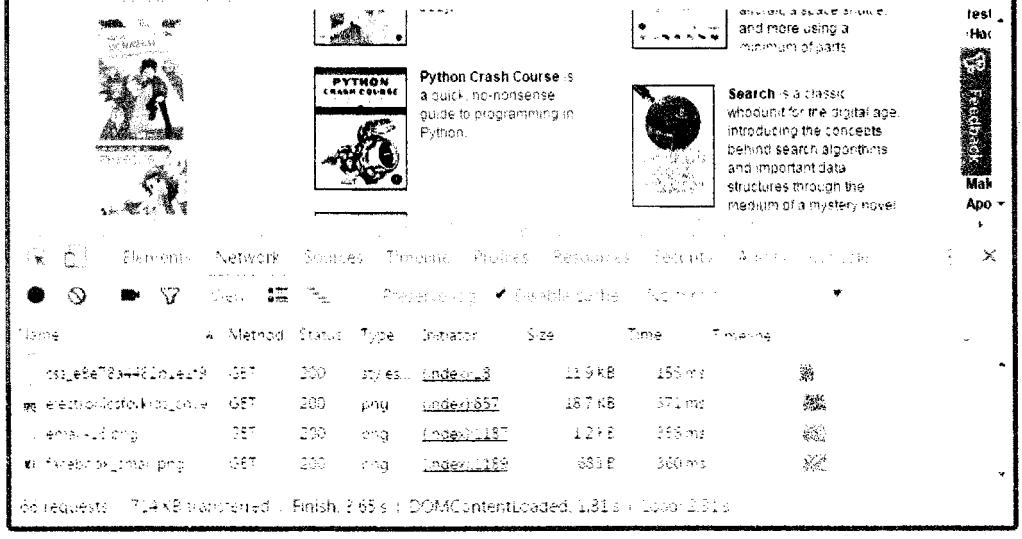


Рис. 11.4. Окно инструментов разработчика в браузере Chrome

В браузере Mozilla Firefox для открытия окна Инспектор можно нажать комбинацию клавиш <Ctrl+Shift+C> в Windows и Linux или комбинацию клавиш <⌘+Option+C> в OS X. В обоих случаях окна инструментов разработчика будут выглядеть почти так же, как в Chrome.

В браузере Safari следует открыть окно Preferences (Установки) и установить флажок **Show Develop menu in the menu bar** (Показывать меню инструментов

разработчика в строке меню) панели Advanced (Дополнительно). После этого можно вызвать окно инструментов разработчика, нажав комбинацию клавиш <⌘+Option+I>.

Активизировав или установив инструменты разработчика в своем браузере, можете щелкнуть правой кнопкой мыши в любой части веб-страницы и выбрать в открывшемся контекстном меню пункт Inspect Element (Исследовать элемент), чтобы выделить HTML-разметку, ответственную за данную часть страницы. Такая возможность придется вам кстати, когда вы приступите к синтаксическому анализу (парсингу) HTML-документов в своих программах для автоматического сбора данных в Интернете.

Не пытайтесь использовать регулярные выражения для HTML-парсинга

Казалось бы, что может быть эффективнее применения регулярных выражений, если речь идет о нахождении определенных HTML-элементов в строке. Однако я категорический противник такого подхода. HTML-разметка может осуществляться множеством самых различных способов, каждому из которых будет соответствовать действительный HTML-документ, и любые попытки охватить все возможные варианты с помощью регулярных выражений потребуют больших усилий и будут чреваты ошибками. Вероятность появления ошибок уменьшится, если использовать модуль Beautiful Soup, специально разработанный для HTML-парсинга.

Дополнительную аргументацию против использования регулярных выражений для парсинга HTML-разметки можно найти по адресу `http://stackoverflow.com/a/1732454/1893164/`.

Использование инструментов разработчика для поиска HTML-элементов

Как только ваша программа загрузит веб-страницу с помощью модуля Requests, вы будете иметь в своем распоряжении ее HTML-содержимое в виде одной строки. Ваши дальнейшие действия заключаются в том, чтобы определить, какая часть HTML-содержимого соответствует объекту вашего интереса.

И здесь вам на помощь придут инструменты разработчика, предоставляемые браузером. Предположим, вы хотите написать программу, осуществляющую сбор данных о прогнозах погоды на сайте `http://weather.gov/`. Прежде чем браться за написание кода, проведите небольшой эксперимент. Если вы посетите этот сайт и выполните поиск по ZIP-коду (почтовому индексу) `94105`, то сайт откроется на странице, представляющей информацию о погоде для этой местности.

Что если вас интересует сбор температурных данных по местности с данным ZIP-кодом? Щелкните правой кнопкой мыши в соответствующем месте страницы (или щелкните мышью при нажатой клавише <Control>, если работаете в OS X) и выберите в открывшемся контекстном меню пункт Inspect Element (Исследовать элемент). Это приведет к открытию окна инструментов разработчика, в котором будет отображаться HTML-код, ответственный за создание данной части страницы. На рис. 11.5 показано окно инструментов разработчика, открытое на HTML-коде, отображающем температуру.

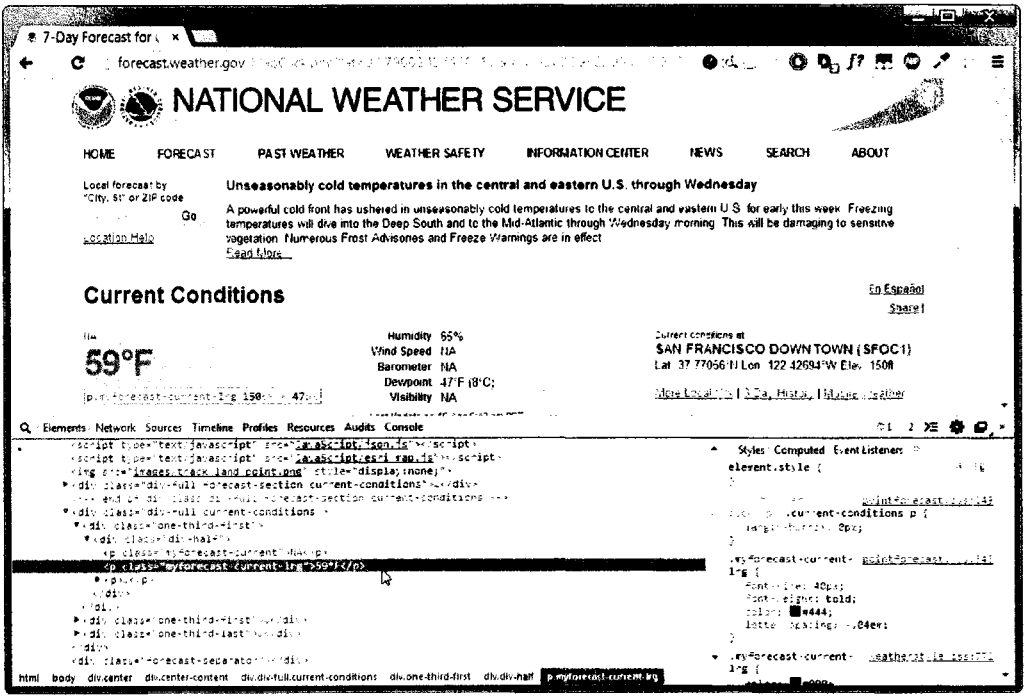


Рис. 11.5. Инспектирование элемента, в котором хранится код, ответственный за отображение температуры, с помощью инструментов разработчика

Окно инструментов разработчика позволяет увидеть, что за отображение температуры отвечает следующий код: `<p class="myforecast-current-lrg">59°F</p>`. Это именно то, что вы искали! Нетрудно заметить, что информация о температуре содержится в элементе `<p>`, которому присвоен класс `myforecast-current-lrg`. Теперь, когда вы уже знаете, что именно необходимо искать, модуль `BeautifulSoup` поможет вам найти искомый элемент в строке.

Синтаксический анализ HTML с помощью Beautiful Soup

Beautiful Soup — это модуль, предназначенный для извлечения информации из HTML-страницы (причем гораздо более приспособленный для этой цели, чем регулярные выражения). Имя модуля Beautiful Soup — `bs4` (от "Beautiful Soup, версия 4"). Для установки этого модуля необходимо выполнить команду `pip install beautifulsoup4` в командной строке. (Более подробные инструкции относительно установки модулей, разработанных третьими сторонами, приведены в приложении А.) Несмотря на то что при установке этого модуля используется имя `beautifulsoup4`, он импортируется с помощью инструкции `import bs4`.

Приведенные в этой главе примеры использования модуля Beautiful Soup охватывают синтаксический анализ (т.е. исследование и идентификацию отдельных элементов) HTML-файла, хранящегося на жестком диске. Откройте новое окно в файловом редакторе IDLE, введите приведенный ниже текст и сохраните его в файле *example.html*. Этот файл также доступен на сайте `http://nostarch.com/automatestuff/`.

```
<!-- Это файл примера example.html. -->

<html><head><title>Заголовок веб-сайта</title></head>
<body>
<p>Загрузите мои книги по <strong>Python</strong> на моем сайте <a
href="http://inventwithpython.com"></a>.</p>
<p class="slogan">Простой подход к изучению Python!</p>
<p>Автор <span id="author">Эл Свейгарт</span></p>
</body></html>
```

Как можно видеть, даже простой HTML-файл включает много различных тегов и атрибутов, количество которых быстро увеличивается при переходе к сложным веб-сайтам. К счастью, модуль Beautiful Soup значительно упрощает работу с HTML-разметкой.

Создание объекта `BeautifulSoup` на основе HTML

Вам потребуется вызвать функцию `bs4.BeautifulSoup()`, передав ей строку, которая содержит HTML-код, подлежащий анализу. Функция `bs4.BeautifulSoup()` возвращает объект `BeautifulSoup`. Введите в интерактивной оболочке следующие команды, предварительно убедившись в том, что ваш компьютер подключен к Интернету.

```
>>> import requests, bs4
>>> res = requests.get('http://nostarch.com')
>>> res.raise_for_status()
```

```
>>> noStarchSoup = bs4.BeautifulSoup(res.text)
>>> type(noStarchSoup)
<class 'bs4.BeautifulSoup'>
```

Этот код сначала загружает основную страницу веб-сайта No Starch Press с помощью вызова `requests.get()`, а затем передает атрибут `text` ответа функции `bs4.BeautifulSoup()`. Возвращенный этой функцией объект `BeautifulSoup` сохраняется в переменной `noStarchSoup`.

Вы также можете загрузить HTML-файл со своего жесткого диска, передав функции `bs4.BeautifulSoup()` объект `File`. Введите в интерактивной оболочке следующие команды (предварительно убедившись в том, что в рабочем каталоге находится файл *example.html*).

```
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile)
>>> type(exampleSoup)
<class 'bs4.BeautifulSoup'>
```

Получив объект `BeautifulSoup`, вы можете использовать его методы для поиска отдельных частей HTML-документа.

Поиск элемента с помощью метода `select()`

Для получения элемента веб-страницы из объекта `BeautifulSoup` можно вызвать метод `select()` и передать ему строку CSS-селектора, выбирающего искомый элемент. Селекторы напоминают регулярные выражения: в обоих случаях задается шаблон поиска, но в случае селекторов поиск осуществляется в HTML-страницах, а не в обычных текстовых строках.

Полное обсуждение синтаксиса CSS-селекторов выходит за рамки данной книги (хорошее практическое руководство по использованию селекторов вы найдете на сайте `http://www.nostarch.com/automatestuffresources`), поэтому ниже мы ограничимся лишь кратким их обзором. Примеры наиболее часто используемых селекторов приведены в табл. 11.2.

Таблица 11.2. Примеры CSS-селекторов

Селектор, передаваемый методу `select()`	Соответствие
`'div'`	Все элементы `<div>`
`'#author'`	Элемент, атрибут `id` которого имеет значение `author`
`'.notice'`	Все элементы, атрибут `class` которых имеет значение `notice`
`'div span'`	Все элементы `<span>`, вложенные в элементы `<div>`
`'div > span'`	Все элементы `<span>`, вложенные *непосредственно* в элементы `<div>`, без каких бы то ни было элементов между ними

Окончание табл. 11.2

Селектор, передаваемый методу `select()`	Соответствие
`'input[name]'`	Все элементы `<input>`, имеющие атрибут *name*, независимо от его значения
`'input[type="button"]'`	Все элементы `<input>`, имеющие атрибут `type` со значением `button`

Различные селекторы могут сочетаться, образуя сложные шаблоны. Например, вызову `soup.select('p #author')` будет соответствовать любой элемент, атрибут `id` которого имеет значение `author`, при условии, что этот элемент вложен в элемент `<p>`. Метод `select()` возвращает список объектов `Tag`, представляющих в Beautiful Soup HTML-элементы. Этот список будет представлять по одному объекту `Tag` для каждого найденного совпадения в HTML-коде объекта `BeautifulSoup`. Объекты `Tag` можно передавать функции `str()` для отображения тегов HTML, которые они представляют.

Значения типа `Tag` имеют атрибут `attrs`, представляющий все HTML-атрибуты данного тега в виде словаря. Используя предыдущий файл *example.html*, введите в интерактивной оболочке следующие команды.

```
>>> import bs4
>>> exampleFile = open('example.html')
>>> exampleSoup = bs4.BeautifulSoup(exampleFile.read())
>>> elems = exampleSoup.select('#author')
>>> type(elems)
<class 'list'>
>>> len(elems)
1
>>> type(elems[0])
<class 'bs4.element.Tag'>
>>> elems[0].getText()
'Эл Свейгарт'
>>> str(elems[0])
'<span id="author">Эл Свергайт</span>'
>>> elems[0].attrs
{'id': 'author'}
```

Этот код извлекает элемент с `id="author"` из нашего примера HTML-разметки. Вызов `select('#author')` возвращает список всех элементов, удовлетворяющих данному условию. Мы сохраняем этот список объектов `Tag` в переменной `elems`, и значение `len(elems)` указывает на то, что в списке имеется только один такой элемент. Вызов метода `getText()` для этого элемента возвращает содержащийся в нем текст, т.е. внутреннюю HTML-разметку. Текст элемента — это содержимое, заключенное между его открывающим и закрывающим тегами. В данном случае это строка `'Эл Свейгарт'`.

При передаче функции `str()` элемента она возвращает строку, включающую открывающий и закрывающий теги вместе с текстом элемента. Наконец, переменная `attrs` предоставляет словарь, содержащий имя атрибута, `'id'`, и его значение, `'author'`.

Также можно извлечь из объекта `BeautifulSoup` все элементы `<p>`. Введите в интерактивной оболочке следующие команды.

```
>>> pElems = exampleSoup.select('p')
>>> str(pElems[0])
'<p>Загрузите мои книги по <strong>Python</strong> на сайте <a
href="http://inventwithpython.com"></a>.</p>'
>>> pElems[0].getText()
'Загрузите мои книги по Python на моем сайте.'
>>> str(pElems[1])
'<p class="slogan">Простой подход к изучению Python!</p>'
>>> pElems[1].getText()
'Простой подход к изучению Python!'
>>> str(pElems[2])
'<p>Автор <span id="author">Эл Свейгарт</span></p>'
>>> pElems[2].getText()
'Автор Эл Свейгарт'
```

На этот раз вызов метода `select()` предоставляет список с тремя совпадениями, который мы сохраняем в переменной `pElems`. Последовательно передавая функции `str()` элементы `pElems[0]`, `pElems[1]` и `pElems[2]`, мы отображаем каждый из этих элементов в виде строки, а вызов метода `getText()` для элементов позволяет отобразить их текстовое содержимое.

Получение данных из атрибутов элемента

Метод `get()` объекта `Tag` упрощает доступ к значениям атрибутов соответствующего элемента. Метод принимает строку с именем атрибута и возвращает его значение. Используя файл *example.html*, введите в интерактивной оболочке следующие команды.

```
>>> import bs4
>>> soup = bs4.BeautifulSoup(open('example.html'))
>>> spanElem = soup.select('span')[0]
>>> str(spanElem)
'<span id="author">Эл Свейгарт</span>'
>>> spanElem.get('id')
'author'
>>> spanElem.get('несуществующий_адрес') == None
True
>>> spanElem.attrs
{'id': 'author'}
```

Здесь мы используем метод `select()` для нахождения элементов `<span>` и сохранения первого из них в переменной `spanElem`. Передав методу `get()` имя атрибута, `'id'`, мы получаем соответствующее значение, `'author'`.

Проект: кнопка "Мне повезет" поисковика Google

Выполнив поиск в Google, я никогда не начинаю сразу же просматривать все полученные ссылки одна за другой. Вместо этого я щелкаю средней кнопкой мыши (или левой кнопкой мыши при нажатой клавише <Ctrl>) на нескольких первых ссылках для открытия каждой из них на новой вкладке, чтобы просмотреть их позже. Поскольку поисковиком Google я пользуюсь довольно часто, описанный порядок поиска – открытие браузера, поиск по заданному поисковому термину и последующие щелчки средней кнопкой мыши на нескольких ссылках – оказывается достаточно трудоемким. Было бы неплохо, если бы я мог просто ввести поисковый термин в командной строке, а компьютер автоматически открыл бы браузер с первыми несколькими результатами поиска, размещенными на отдельных вкладках. Давайте напишем сценарий, реализующий это пожелание.

Вот что должна делать такая программа:

- получать из аргументов командной строки ключевые слова, по которым должен быть выполнен поиск;
- извлекать страницу с результатами поиска;
- размещать каждый результат на отдельной вкладке браузера.

Это означает, что ваш код должен выполнять следующие действия:

- читать аргументы командной строки из списка `sys.argv`;
- извлекать страницу с результатами поиска с помощью модуля Requests;
- находить ссылки на каждый результат поиска;
- вызывать функцию `webbrowser.open()` для открытия браузера.

Откройте новое окно в файловом редакторе и сохраните его в файле `lucky.py`.

Шаг 1. Получение аргументов командной строки и запрос поисковой страницы

Прежде чем приступить к написанию кода, необходимо определить URL-адрес страницы с результатами поиска. Взглянув на адресную строку браузера после выполнения поиска в Google, вы увидите там URL-адрес вида `https://www.google.com/search?q=`*ПОИСКОВЫЙ_ТЕРМИН*. Модуль Requests может загрузить эту страницу, после чего вы сможете использовать Beautiful Soup для нахождения ссылок на результаты поиска в HTML. Наконец, вы

используете модуль webbrowser для открытия этих ссылок в отдельных вкладках браузера.

Введите в созданный файл следующий код.

```
#! python3
# lucky.py - Открывает несколько результатов поиска с помощью
# Google.

import requests, sys, webbrowser, bs4

print('Гуглим...') # отображается при загрузке страницы Google
res = requests.get('http://google.com/search?q=' +
                   ' '.join(sys.argv[1:]))
res.raise_for_status()

# TODO: Извлечь первые несколько найденных ссылок.

# TODO: Открыть отдельную вкладку для каждого результата.
```

Поисковые термины будут предоставляться пользователем с помощью аргументов командной строки при запуске программы. Эти аргументы будут сохраняться в виде строк в списке sys.argv.

Шаг 2. Поиск всех результатов

А теперь настал черед использовать модуль Beautiful Soup для извлечения нескольких первых ссылок на результаты поиска из загруженного HTML-документа. Однако как определить, какой селектор следует использовать для выполнения этой работы? Например, вы не можете просто отобрать все теги <a>, поскольку в полученном HTML-документе будет присутствовать множество ссылок, не представляющих для вас интереса. Вместо этого вы должны инспектировать страницу с результатами поиска с помощью инструментов разработчика, предоставляемых браузером, чтобы определить селектор, который отберет лишь нужные вам ссылки.

Используя для поиска в Google строку *Beautiful Soup* в качестве поискового термина, вы можете открыть окно инструментов разработчика и исследовать некоторые из элементов, содержащих ссылки. Эти элементы могут выглядеть невероятно устрашающе, например так.

```
<a href="/url?sa=t&amp;rct=j&amp;q=&amp;esrc=s&amp;source=web&amp;
cd=1&amp;cad=rja&amp;uact=8&amp;ved=0CCgQFjAA&amp;url=http%3A%2F
%2Fwww.crummy.com%2Fsoftware%2FBeautifulSoup%2F&amp;ei=
LHBVU_XDD9KVyAShmYDwCw&amp;usg=AFQjCNHAxwplurFOBqg5cehWQEVKi-TuLQ&a
mp;sig2=sdZu6WV1BlVSDrwhtworMA" onmousedown="return rwt(this,'','',
'','1','AFQjCNHAxwplurFOBqg5cehWQEVKiTuLQ','sdZu6WV1BlVSDrwhtworMA',
'0CCgQFjAA','','',event)" datahref="http://www.crummy.com/software/
BeautifulSoup/"><em>Beautiful-Soup</em>: We called him Tortoise
because he taught us.</a>
```

Пусть вас не смущает вид элемента. Вам достаточно определить лишь шаблон, который является общим для всех ссылок. Однако в этом элементе <a> нет ничего, что позволило бы легко отличить его от элементов <a>, не имеющих никакого отношения к поиску.

Добавьте в программу код, выделенный ниже полужирным шрифтом.

```
#! python3
# lucky.py - Открывает несколько результатов поиска с помощью
# Google.

import requests, sys, webbrowser, bs4

--пропущенный код--

# Извлечение первых нескольких найденных ссылок.
soup = bs4.BeautifulSoup(res.text)

# Открытие отдельной вкладки для каждого результата.
linkElems = soup.select('.r a')
```

Однако, оглядевшись в окрестностях элемента <a>, вы заметите элемент наподобие <h3 class="r">. Просмотр оставшейся части HTML-кода позволяет предположить, что класс r используется исключительно для ссылок на результаты поиска. Вам необязательно знать, что собой представляет CSS-класс r и что он делает. Вы просто будете использовать его в качестве маркера элемента <a>, который ищете. Вы можете создать объект BeautifulSoup из HTML-текста загруженной страницы, а затем использовать селектор '.r a' для нахождения всех элементов <a>, которые вложены в элемент, имеющий CSS-класс r.

Шаг 3. Открытие браузера для каждого из результатов поиска

Наконец, нам необходимо, чтобы программа открыла в браузере отдельные вкладки для каждого из результатов поиска. Добавьте в конце программы следующий код.

```
#! python3
# lucky.py - Открывает несколько результатов поиска с помощью
# Google.

import requests, sys, webbrowser, bs4

--пропущенный код--
# Открытие отдельной вкладки для каждого результата.
linkElems = soup.select('.r a')
numOpen = min(5, len(linkElems))
```

```
for i in range(numOpen):
    webbrowser.open('http://google.com' + linkElems[i].get('href'))
```

По умолчанию вы открываете с помощью модуля `webbrowser` новые вкладки для пяти результатов поиска. Однако пользователь мог выполнить поиск, дающий менее пяти результатов. Вызов `soup.select()` возвращает список всех элементов, соответствующих селектору `'.r a'`, поэтому количество открываемых вкладок либо будет равно 5, либо будет определяться длиной указанного списка (в зависимости от того, что меньше).

Встроенная функция Python `min()` возвращает наименьшее из переданных ей целых или вещественных чисел. (Также существует встроенная функция `max()`, которая возвращает наибольшее из чисел, которые были ей переданы.) Можно использовать функцию `min()` для того, чтобы выяснить, содержит ли список менее пяти ссылок, и сохранить количество ссылок, подлежащих открытию, в переменной `numOpen`. После этого можно выполнить цикл `for`, вызвав функцию `range(numOpen)`.

На каждой итерации цикла вы открываете новую вкладку в браузере с помощью вызова `webbrowser.open()`. Обратите внимание на то, что значение атрибута `href` в возвращенных элементах `<a>` не содержит начальную часть URL-адреса `http://google.com`, поэтому вы должны конкатенировать ее со строкой значения атрибута `href`.

Теперь вы можете сразу открыть первые пять результатов поиска, выполненного, скажем, для поискового термина *Python programming tutorials*, введя в командной строке команду `lucky python programming tutorials`! (Более подробная информация о простом способе запуска программ в различных операционных системах приведена в приложении Б.)

Идеи относительно создания аналогичных программ

Преимуществом описанного подхода к выполнению поиска является то, что он позволяет легко открывать ссылки в новых вкладках для того, чтобы просмотреть их позднее. Программа, автоматически открывающая сразу несколько ссылок, поможет вам сэкономить время, выполняя следующие операции:

- открытие страниц всех заинтересовавших вас товаров после выполнения поиска на сайте какого-либо интернет-магазина, например Amazon;
- открытие ссылок на все обзоры, посвященные одному и тому же продукту;
- открытие результирующих ссылок на фотографии после выполнения поиска на каком-либо фотосайте, таком как Flickr или Imgur.

Проект: загрузка всех комиксов на сайте XKCD

В блогах и на других регулярно обновляемых веб-сайтах часто есть заглавная страница с последней публикацией и кнопка Previous (Предыдущая), которая приводит к предыдущей публикации. Предпоследний пост также снабжен аналогичной кнопкой Previous и т.д., что обеспечивает возможность последовательного просмотра публикаций на данном сайте в направлении от последней к первой. Если вы хотите скопировать содержимое сайта, чтобы прочесть его позднее в автономном режиме, то можете вручную пройтись по всем страницам и сохранить их. Но это занятие довольно скучное, поэтому поручим программе выполнить эту работу вместо вас.

XKCD — это популярный веб-комикс с сайтом, структура которого вписывается в эту схему (рис. 11.6). На заглавной странице по адресу `http://xkcd.com/` есть кнопка Prev (Предыдущий), щелкая на которой пользователь может переходить к предыдущим комиксам. Загрузка всех комиксов вручную длилась бы целую вечность, но вы можете написать сценарий, который выполнит эту работу за пару минут.

Вот какие действия должна выполнять ваша программа:

- загружать главную страницу XKCD;
- сохранять изображение комикса, отображаемого на данной странице;
- выполнять переходы по ссылке Previous Comic (Предыдущий комикс);
- повторять описанные действия, пока не будет достигнут первый комикс.

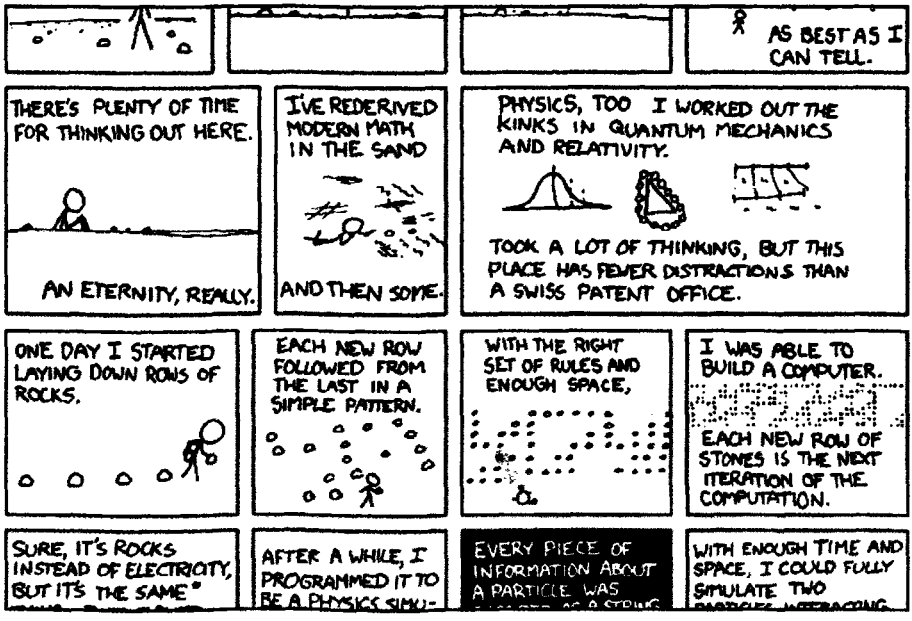


Рис. 11.6. XKCD — "веб-комикс о романтике, сарказме, математике и языке"

Это означает, что ваш код должен выполнять следующие операции:

- загружать страницы с помощью модуля `requests`;
- находить URL-адрес изображения комикса для страницы, используя возможности Beautiful Soup;
- загружать и сохранять изображение комикса на жестком диске, используя метод `iter_content()`.
- находить URL-адрес ссылки Previous Comic и повторять действия.

Откройте новое окно в файловом редакторе и сохраните его в файле *downloadXkcd.py*.

Шаг 1. Проектирование программы

Открыв в браузере окно инструментов разработчика и проинспектировав элементы страницы, вы обнаружите следующее:

- URL-адрес файла изображения комикса предоставляется атрибутом `href` элемента `<img>`;
- элемент `<img>` вложен в элемент `<div id="comic">`;
- кнопка Prev имеет HTML-атрибут `rel` со значением `prev`.
- с кнопкой Prev первого комикса связан URL-адрес `http://xkcd.com/#`, указывающий на отсутствие других предыдущих страниц.

Введите в подготовленный ранее файл следующий код.

```
#! python3
# downloadXkcd.py - Загружает все комиксы XKCD.

import requests, os, bs4

url = 'http://xkcd.com' # начальный URL-адрес
os.makedirs('xkcd', exist_ok=True) # сохраняем комикс в
                                   # папке ./xkcd
while not url.endswith('#'):
    # TODO: Загрузить страницу.
    # TODO: Найти URL-адрес изображения комикса.
    # TODO: Загрузить изображение.
    # TODO: Сохранить изображение в папке ./xkcd.
    # TODO: Получить URL-адрес кнопки Prev.
print('Готово.')
```

У вас будет переменная `url` с начальным значением `'http://xkcd.com'`, которое будет регулярно обновляться (в цикле `for`) значением URL-адреса ссылки на кнопку Prev текущей страницы. На каждом шаге цикла вы загружаете комикс, находящийся по адресу, который хранится в переменной `url`. Вы будете знать, что цикл следует завершить, когда встретится значение `url`, заканчивающееся символом `'#'`.

Файлы изображений будут загружаться в папку *xkcd*, находящуюся в текущем рабочем каталоге. Вызов `os.makedirs()` гарантирует существование этой папки, а именованный аргумент `exist_ok=True` предотвращает возбуждение исключения в том случае, если эта папка уже существует. Остальная часть кода — это просто комментарии, описывающие оставшуюся часть программы.

Шаг 2. Загрузка веб-страницы

Приступим к реализации кода для загрузки страницы. Добавьте в программу код, выделенный ниже полужирным шрифтом.

```
#! python3
# downloadXkcd.py - Загружает все комиксы XKCD.

import requests, os, bs4

url = 'http://xkcd.com' # начальный URL-адрес
os.makedirs('xkcd', exist_ok=True) # сохраняем комикс в
                                   # папке ./xkcd
while not url.endswith('#'):
    # Загрузка страницы.
    print('Загружается страница %s...' % url)
    res = requests.get(url)
    res.raise_for_status()

    soup = bs4.BeautifulSoup(res.text)

    # TODO: Найти URL-адрес изображения комикса.

    # TODO: Загрузить изображение.

    # TODO: Сохранить изображение в папке ./xkcd.

    # TODO: Получить URL-адрес кнопки Prev.

print('Готово.')
```

Здесь прежде всего выводится значение `url`, информирующее пользователя о том, по какому URL-адресу сейчас будет осуществляться загрузка; для последующей загрузки изображения используется функция `request.get()` модуля `requests`. Как всегда, если в процессе загрузки что-то пойдет не так, вы немедленно сможете вызвать метод `raise_for_status()` объекта `Response` для возбуждения исключения и прекращения работы программы. В противном случае нужно будет создать объект `BeautifulSoup` на основе текста загруженной страницы.

Шаг 3. Поиск и загрузка изображения комикса

Добавьте в программу код, выделенный ниже полужирным шрифтом.

```
#! python3
# downloadXkcd.py - Загружает все комиксы XKCD.

import requests, os, bs4

--пропущенный код--

    # Поиск URL-адреса изображения комикса.
    comicElem = soup.select('#comic img')
    if comicElem == []:
        print('Не удалось найти изображение комикса.')
    else:
        comicUrl = comicElem[0].get('src')
        # Загрузить изображение.
        print('Загружается изображение %s...' % (comicUrl))
        res = requests.get(comicUrl)
        res.raise_for_status()

    # TODO: Сохранить изображение в папке ./xkcd.

    # TODO: Получить URL-адрес кнопки Prev.

print('Готово.')
```

Как показывают результаты инспектирования главной страницы XKCD с помощью инструментов разработчика, элемент `<img>` для изображения комикса помещен в элемент `<div>`, атрибут `id` которого имеет значение `comic`. Поэтому для извлечения нужного элемента `<img>` из объекта `BeautifulSoup` следует использовать селектор `'#comic img'`.

Некоторые страницы XKCD имеют специальное содержимое, не являющееся простым файлом изображения. Никаких сложностей это не представляет — вы просто пропускаете эти страницы. Если селектор вообще не найдет никаких элементов, то вызов `soup.select('#comic img')` вернет пустой список. В этом случае программа может просто вывести сообщение об ошибке и продолжить выполнение, опустив загрузку изображения.

В противном случае селектор возвратит список, содержащий один элемент `<img>`. Вы можете получить значение атрибута `src` этого элемента `<img>` и передать его функции `requests.get()` для загрузки файла изображения комикса.

Шаг 4. Сохранение изображения и поиск предыдущего комикса

Добавьте в программу код, выделенный ниже полужирным шрифтом.

```
#! python3
# downloadXkcd.py - Загружает все комиксы XKCD.

import requests, os, bs4

--пропущенный код--

        # Сохранение изображения в папке ./xkcd.
        imageFile = open(os.path.join('xkcd',os.path.
                         basename(comicUrl)), 'wb')
        for chunk in res.iter_content(100000):
            imageFile.write(chunk)
        imageFile.close()

    # Получение URL-адреса кнопки Prev.
    prevLink = soup.select('a[rel="prev"]')[0]
    url = 'http://xkcd.com' + prevLink.get('href')

print('Готово.')
```

К этому моменту файл изображения комикса хранится в переменной `res`. Вам следует записать данные изображения в файл на жестком диске.

Функции `open()` необходимо передать имя локального файла изображения. Переменная `comicUrl` будет иметь значение наподобие `'http://imgs.xkcd.com/comics/heartbleed_explanation.png'`, которое, как вы, наверное, заметили, во многом напоминает путь к файлу. И действительно, вы можете вызвать функцию `os.path.basename()` с `comicUrl` в качестве аргумента, и она возвратит лишь последнюю часть URL, `'heartbleed_explanation.png'`. Эту часть можно использовать в качестве имени файла при сохранении изображения на жестком диске. Полученное имя можно соединить с именем вашей папки `xkcd` с помощью функции `os.path.join()` (ваша программа использует в обозначениях пути символы обратной косой черты при работе в Windows и косой черты при работе в OS X и Linux). Получив имя файла, вы можете вызвать функцию `open()` для открытия нового файла в режиме `'wb'` (запись двоичного файла).

Как уже говорилось, для сохранения загруженных файлов с помощью модуля `Requests` следует организовать цикл по возвращаемому значению метода `iter_content()`. Содержащийся в цикле код записывает порции данных изображения (не более `100000` байт в каждой порции) в файл, после чего вы закрываете файл. Теперь изображение сохранено на вашем жестком диске.

Затем селектор `'a[rel="prev"]'` выбирает элемент `<a>`, атрибут `rel` которого имеет значение `prev`, и атрибут `href` этого элемента используется для получения URL-адреса предыдущего комикса, который сохраняется в переменной `url`. После этого цикл `while` вновь начинает весь процесс загрузки для данного комикса.

Вывод программы выглядит так.

```
Загружается страница http://xkcd.com...
Загружается изображение http://imgs.xkcd.com/comics/phone_alarm.png...
Загружается страница http://xkcd.com/1358/...
Загружается изображение http://imgs.xkcd.com/comics/nro.png...
Загружается страница http://xkcd.com/1357/...
Загружается изображение http://imgs.xkcd.com/comics/free_speech.png...
Загружается страница http://xkcd.com/1356/...
Загружается изображение http://imgs.xkcd.com/comics/
orbital_mechanics.png...
Загружается страница http://xkcd.com/1355/...
Загружается изображение http://imgs.xkcd.com/comics/
airplane_message.png...
Загружается страница http://xkcd.com/1354/...
Загружается изображение http://imgs.xkcd.com/comics/
heartbleed_explanation.png...
--пропущенный код--
```

Этот проект представляет собой неплохой пример программы, которая может автоматически выполнять переходы по ссылкам для сбора больших объемов данных в Интернете. О других возможностях модуля Beautiful Soup можно узнать из его документации, которая находится по адресу `http://www.crummy.com/software/BeautifulSoup/bs4/doc/`.

Идеи относительно создания аналогичных программ

Загрузка страниц и переходы по ссылкам составляют основу многих программ, собирающих данные в Интернете. Аналогичные им программы могут выполнять также следующие задачи:

- резервное копирование всего сайта путем обхода всех его ссылок;
- копирование всех сообщений, опубликованных на форуме;
- дублирование каталога товаров для продажи через интернет-магазин.

Модули `requests` и `BeautifulSoup` сослужат вам отличную службу при условии, что вы можете определить URL-адрес, который необходимо передать функции `requests.get()`. Однако иногда сделать это довольно нелегко. Кроме того, может оказаться так, что сайт, навигацию по которому вы хотите выполнить с помощью своей программы, требует предварительного выполнения процедуры входа. Средством, которое предоставит вашим

программам больше возможностей для решения таких сложных задач, является модуль Selenium.

Управление браузером с помощью модуля Selenium

Модуль Selenium предоставляет Python возможность непосредственно управлять браузером путем программной имитации щелчков на ссылках и прохождения процедуры входа, как если бы сам пользователь взаимодействовал со страницей. Этот модуль обеспечивает гораздо более гибкие методы взаимодействия с веб-страницами, чем модули Requests и Beautiful Soup. Но поскольку модуль Selenium запускает браузер, он работает немного медленнее, и организация фоновой обработки с его помощью требует больших хлопот, если, скажем, вам нужно всего лишь загрузить некоторые файлы из Интернета.

Процедура установки модулей, разработанных сторонними лицами, описана в приложении А.

Запуск браузера, управляемого модулем Selenium

Для работы с примерами в этом разделе вам понадобится браузер Firefox. Это и есть браузер, которым вы будете управлять. Если Firefox у вас еще не установлен, можете бесплатно загрузить его, посетив сайт `http://getfirefox.com/`.

Импортирование модулей для Selenium выполняется не совсем стандартным способом. Вместо команды `import selenium` необходимо выполнить команду `from selenium import webdriver`. (Объяснение истинных причин того, почему для установки модуля Selenium выбран подобный способ, выходит за рамки книги.) После этого вы сможете запустить браузер Firefox с помощью Selenium. Введите в интерактивной оболочке следующие команды.

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> type(browser)
<class 'selenium.webdriver.firefox.webdriver.WebDriver'>
>>> browser.get('http://inventwithpython.com')
```

Вы увидите, что вызов `webdriver.Firefox()` приводит к запуску браузера Firefox. Вызов функции `type()` с передачей ей значения, возвращенного вызовом `webdriver.Firefox()`, показывает, что типом данных этого значения является `WebDriver`. А последующий вызов метода `browser.get('http://inventwithpython.com')` перенаправляет браузер на страницу `http://inventwithpython.com/`. Окно вашего браузера должно выглядеть примерно так, как показано на рис. 11.7.

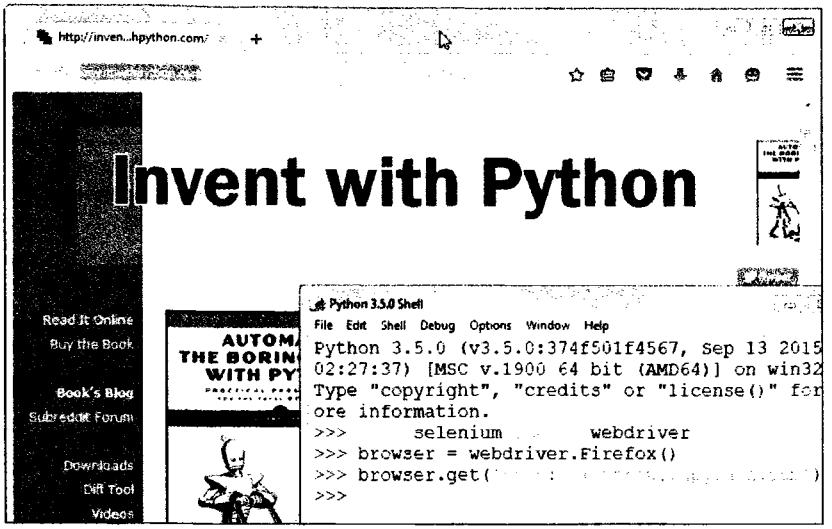


Рис. 11.7. Выполнение вызовов `webdriver.Firefox()` *и* `get()` *в IDLE приводит к запуску браузера Firefox*

Поиск элементов на странице

Объекты `WebDriver` имеют довольно большое количество методов, предназначенных для поиска элементов на странице. Эти методы условно делятся на две группы: `find_element_*()` и `find_elements_*()`. Методы группы `find_element_*()` возвращают одиночный объект `WebElement`, который представляет первый из найденных на странице элементов, соответствующих запросу. Методы группы `find_elements_*()` возвращают список объектов `WebElement_*`, который представляет все элементы, соответствующие запросу. В табл. 11.3 показано несколько примеров вызова методов `find_element_*()` и `find_elements_*()` для объекта `WebDriver`, сохраненного в переменной `browser`.

Таблица 11.3. Примеры использования методов объекта `WebDriver` для поиска элементов

Имя метода	Возвращаемый объект (список объектов) WebElement
`browser.find_element_by_class_name(`*имя*`)` `browser.find_elements_by_class_name(`*имя*`)`	Элементы, имеющие CSS-класс с указанным именем
`browser.find_element_by_css_selector(`*селектор*`)` `browser.find_elements_by_css_selector(`*селектор*`)`	Элементы, соответствующие указанному селектору CSS
`browser.find_element_by_id(`*id*`)` `browser.find_elements_by_id(`*id*`)`	Элементы, соответствующие указанному значению атрибута *id*

Окончание табл. 11.3

Имя метода	Возвращаемый объект (список объектов) WebElement
`browser.find_element_by_link_text(`*текст*`)` `browser.find_elements_by_link_text(`*текст*`)`	Элементы <a>, полностью совпадающие с указанным текстом
`browser.find_element_by_partial_link_text(`*текст*`)` `browser.find_elements_by_partial_link_text(`*текст*`)`	Элементы <a>, содержащие указанный текст
`browser.find_element_by_name(`*имя*`)` `browser.find_elements_by_name(`*имя*`)`	Элементы, содержащие атрибут с указанным именем
`browser.find_element_by_tag_name(`*имя*`)` `browser.find_elements_by_tag_name(`*имя*`)`	Элементы с указанным именем тега (регистр не учитывается); именам 'a' и 'A' будет соответствовать тег <a>

За исключением группы методов `*_by_tag_name()`, аргументы всех методов нечувствительны к регистру. Если элементы, являющиеся объектом поиска метода, отсутствуют на странице, модуль `selenium` возбуждает исключение `NoSuchElement`. Если аварийное завершение работы программы в результате возбуждения исключения не входит в ваши планы, добавьте в код инструкции `try` и `except`.

Получив объект `WebElement`, вы сможете найти более полную информацию о нем, читая атрибуты или вызывая методы, приведенные в табл. 11.4.

Таблица 11.4. Атрибуты и методы объекта `WebElement`

Атрибут или метод	Описание
`tag_name`	Имя тега; например, 'a' в случае элемента <a>
`get_attribute(`*имя*`)`	Значение атрибута с указанным именем для данного элемента
`text`	Текст, содержащийся в элементе; например, 'hello' в случае элемента hello
`clear()`	Удаляет текст, введенный в текстовом поле или текстовой области
`is_displayed()`	Возвращает значение `True`, если элемент видимый, иначе — `False`
`is_enabled()`	Возвращает значение `True` для элементов ввода, если элемент активизирован, иначе — `False`
`is_selected()`	Возвращает значение `True` для флажка или переключателя, если элемент выбран, иначе — `False`
`location`	Словарь с ключами 'x' и 'y' для позиции элемента на странице

Например, откройте новое окно в файловом редакторе и введите следующий код.

```
from selenium import webdriver
browser = webdriver.Firefox()
browser.get('http://inventwithpython.com')

try:
    elem = browser.find_element_by_class_name('bookcover')
    print('Найден элемент <%s> с данным именем класса!' %
          (elem.tag_name))
except:
    print('Не удалось найти элемент с данным именем класса.')
```

Этот код открывает Firefox и перенаправляет его по заданному URL-адресу. На открывшейся странице выполняется поиск элементов с классом `'bookcover'`, и если такой элемент обнаружен, то на экран выводится имя тега, определяемое атрибутом `tag_name`. В противном случае выводится другое сообщение.

Для данной программы вывод будет таким:

```
Найден элемент <img> с данным именем класса!
```

. Мы нашли элемент с именем класса `'bookcover'` и именем тега `'img'`.

Щелчок на странице

Объекты `WebElement`, возвращаемые методами `find_element_*()` и `find_elements_*()`, имеют метод `click()`, имитирующий щелчок мышью на элементе. Этот метод можно использовать для перехода по ссылке, выбора с помощью переключателя, щелчка на кнопке Submit (Отправить) или инициирования любого другого действия, которое может быть запущено щелчком на элементе. Например, введите в интерактивной оболочке следующие команды.

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://inventwithpython.com')
>>> linkElem = browser.find_element_by_link_text('Читать онлайн')
>>> type(linkElem)
<class 'selenium.webdriver.remote.webelement.WebElement'>
>>> linkElem.click() # перейти по ссылке "Читать онлайн"
```

В данном случае мы открываем Firefox на странице `http://inventwithpython.com/`, получаем объект `WebElement` для элемента `<a>` с текстом *Читать онлайн*, а затем имитируем щелчок на этом элементе. Все происходит так, как если бы вы сами щелкнули на ссылке, заставляя браузер открыть соответствующую страницу.

Заполнение и отправка форм

Отправка нажатий клавиш при нахождении фокуса ввода в текстовом поле сводится к нахождению на странице элемента `<input>` или `<textarea>`, соответствующего данному полю, и последующему вызову метода `send_keys()`. Например, введите в интерактивной оболочке следующий код.

```
>>> from selenium import webdriver
>>> browser = webdriver.Firefox()
>>> browser.get('http://gmail.com')
>>> emailElem = browser.find_element_by_id('Email')
>>> emailElem.send_keys('фиктивный_адрес_email@gmail.com')
>>> passwordElem = browser.find_element_by_id('Passwd')
>>> passwordElem.send_keys('12345')
>>> passwordElem.submit()
```

Если только служба Gmail не изменила идентификаторы `id` текстовых полей Username (Имя пользователя) и Password (Пароль) с момента выхода в свет данной книги, этот код заполнит указанные текстовые поля предоставленным текстом. (Для проверки значения `id` вы всегда можете воспользоваться средством инспектирования элементов, предоставляемым браузером.) Вызов метода `submit()` для любого элемента будет иметь тот же эффект, что и щелчок на кнопке Submit (Отправить) формы, содержащей данный элемент. (Вы также могли вызвать `emailElem.submit()`, и код выполнил бы то же самое.)

Отправка кодов специальных клавиш

Selenium включает модуль для обработки нажатий клавиш, которые нельзя ввести в виде строки, функционирующий во многом подобно экранированным символам. Соответствующие значения хранятся в модуле `selenium.webdriver.common.keys`. Поскольку это довольно длинное имя, гораздо проще выполнить в начале программы инструкцию `from selenium.webdriver.common.keys import Keys`. После этого вы сможете использовать имя `Keys` везде, где обычно вы использовали бы запись `selenium.webdriver.common.keys`.

Таблица 11.5. Часто используемые переменные модуля `selenium.webdriver.common.keys`

Атрибут	Описание
`Keys.DOWN, Keys.UP, Keys.LEFT, Keys.RIGHT`	Клавиши со стрелками
`Keys.ENTER, Keys.RETURN`	Клавиши <Enter> и <Return>
`Keys.HOME, Keys.END, Keys.PAGE_DOWN, Keys.PAGE_UP`	Клавиши <Home>, <End>, <PageDown> и <PageUp>
`Keys.ESCAPE, Keys.BACK_SPACE, Keys.DELETE`	Клавиши <Esc>, <Backspace> и <Delete>
`Keys.F1, Keys.F2, . . Keys.F12`	Клавиши от <F1> до <F2>
`Keys.TAB`	Клавиша <Tab>

Например, если курсор в данный момент не находится в текстовом поле, нажатие клавиш <Home> и <End> выполняет прокрутку в начало или конец страницы соответственно. Введите в интерактивной оболочке следующий код и обратите внимание на то, как вызовы метода `send_keys()` приводят к прокрутке страницы.

```
>>> from selenium import webdriver
>>> from selenium.webdriver.common.keys import Keys
>>> browser = webdriver.Firefox()
>>> browser.get('http://nostarch.com')
>>> htmlElem = browser.find_element_by_tag_name('html')
>>> htmlElem.send_keys(Keys.END) # прокрутка в конец
>>> htmlElem.send_keys(Keys.HOME) # прокрутка в начало
```

Тег `<html>` — основной в HTML-файлах: все содержимое HTML-файла заключено между тегами `<html>` и `</html>`. Вызов `browser.find_element_by_tag_name('html')` обеспечивает удобную возможность отправки кодов клавиш целой веб-странице. Это пригодится вам, если, например, новое содержимое загружается сразу же, как только вы прокручиваете страницу до конца.

Щелчки на кнопках браузера

Модуль Selenium также может имитировать щелчки на различных кнопках браузера с помощью следующих методов:

- **browser.back()** — щелчок на кнопке Back (Назад);
- **browser.forward()** — щелчок на кнопке Forward (Вперед);
- **browser.refresh()** — щелчок на кнопке Refresh (Обновить)/Reload (Перезагрузить);
- **browser.quit()** — щелчок на кнопке Close Window (Закрыть окно).

Получение дополнительной информации о модуле Selenium

Модуль Selenium способен делать гораздо больше, чем здесь описано. Он может изменять *cookie*-файлы браузера, получать моментальные снимки веб-страниц и выполнять пользовательские сценарии JavaScript. Чтобы узнать больше об этих возможностях, посетите сайт с документацией по Selenium — `http://selenium-python.readthedocs.org/`.

Резюме

Большинство рутинных задач связано не только с обработкой файлов, хранящихся на вашем компьютере. Возможность программной загрузки веб-страниц выводит вашу программу на просторы Интернета. Модуль Requests упрощает процесс загрузки веб-страниц, а с помощью модуля BeautifulSoup вы, имея лишь самые поверхностные знания базовых понятий HTML и селекторов, сможете выполнять синтаксический анализ загруженных страниц.

Однако для полной автоматизации задач, связанных с Интернетом, вам необходимы возможности непосредственного управления браузером, которые обеспечивает модуль Sellenium. Этот модуль позволяет автоматизировать процедуру входа на сайт и заполнение форм. Поскольку браузеры являются самым распространенным средством отправки и получения информации через Интернет, описанные возможности обязательно нужно включить в свой арсенал программиста.

Контрольные вопросы

1. Вкратце опишите различия между модулями `webbrowser`, Requests, Beautiful Soup и Selenium.
2. Объект какого типа возвращает функция `requests.get()`? Каким образом можно получить доступ к загруженному содержимому в виде строкового значения?
3. Какой метод модуля Requests позволяет проверить успешность загрузки?
4. Как получить код состояния HTTP из ответа на запрос модуля Requests?
5. Как сохранить в файле ответ на запрос модуля Requests?
6. Какая комбинация клавиш предназначена для открытия окна инструментов разработчика, предоставляемых браузером?
7. Как можно просмотреть (в окне инструментов разработчика) HTML-код конкретного элемента на веб-странице?
8. Какая строка CSS-селектора найдет элемент, атрибут `id` которого имеет значение `main`?

9. Какая строка CSS-селектора найдет элементы, относящиеся к CSS-классу `highlight`?
10. Какая строка CSS-селектора найдет все элементы `<div>`, каждый из которых вложен в другой элемент `<div>`?
11. Какая строка CSS-селектора найдет элемент `<button>`, атрибут `value` которого имеет значение `favorite`?
12. Предположим, у вас имеется сохраненный в переменной `spam` объект `Tag` Beautiful Soup для элемента `<div>Hello world!</div>`. Как получить строку `'Hello world!'` из объекта `Tag`?
13. Как сохранить все атрибуты объекта `Tag` Beautiful Soup в переменной `linkElem`?
14. Инструкция `import selenium` не работает. Как правильно импортировать модуль Selenium?
15. В чем суть различия между методами `find_element_*()` и `find_elements_*()`?
16. Какие методы объектов `WebElement` модуля Selenium имитируют щелчки мышью и нажатия клавиш?
17. Чтобы отправить с помощью модуля Selenium форму, можно вызвать метод `send_keys(Keys.ENTER)` для объекта `WebElement` кнопки `Submit`, но какой для этого существует более простой способ?
18. Как сымитировать щелчки на кнопках браузера Forward, Back и Refresh с помощью модуля Selenium?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Программа для отправки электронной почты из командной строки

Напишите программу, которая принимает адрес электронной почты и строку текста в командной строке, а затем, используя модуль Selenium, входит в вашу учетную запись электронной почты и отправляет строку сообщения по указанному адресу. (Возможно, для этой программы целесообразно завести отдельную учетную запись электронной почты.)

Для вас это был бы отличный шанс дополнить свои программы возможностью рассылки уведомлений. Вы также могли бы написать аналогичную программу для отправки сообщений из учетных записей Facebook или Твиттера.

Загрузчик изображений из Интернета

Напишите программу, которая перенаправляет браузер на какой-либо фотосайт, такой как Flickr или Imgur, осуществляет поиск фотографий определенной категории, а затем загружает все результирующие изображения. Вы могли бы написать программу, способную работать с любым фотосайтом, предоставляющим средства поиска.

"2048"

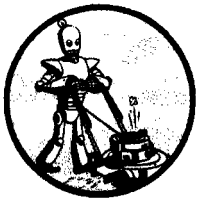
"2048" — это название простой игры, в которой игрок перемещает плитки с помощью клавиш управления курсором на клавиатуре. Когда две плитки с одинаковыми цифрами соприкасаются, они сливаются в одну. Напишите программу, которая открывает игру на сайте `https://gabrielecirulli.github.io/2048/`, а затем отправляет коды клавиш управления курсором, соответствующих перемещениям в направлениях вверх, вправо, вниз и влево, автоматически поддерживая игровой процесс.

Верификация ссылок

Напишите программу, которая принимает URL-адрес веб-страницы, а затем пытается загрузить каждую страницу, на которую на данной странице имеется ссылка. Программа должна помечать флагом те страницы, при попытке открытия которых получен код состояния 404 "Страница не найдена", и выводить информацию о разорванных связях на экран.

12

РАБОТА С ЭЛЕКТРОННЫМИ ТАБЛИЦАМИ EXCEL



Excel — мощная программа для работы с электронными таблицами в операционной системе Windows, пользующаяся широкой популярностью. Задействуя модуль `openpyxl`, ваши программы на Python могут читать и изменять файлы электронных таблиц Excel. Необходимость в этом может возникать сплошь и рядом. В качестве примера можно привести копирование данных из одной таблицы в другую, просмотр тысяч строк таблицы с целью выбора лишь ограниченного их числа и внесения изменений на основании определенных критериев или же анализ сотен таблиц бюджета отделов, чтобы найти отделы с дефицитом баланса. Все это примеры тех рутинных задач, неминуемо возникающих при работе с электронными таблицами, которые Python может выполнять вместо вас.

Excel распространяется компанией Microsoft как коммерческий продукт, однако существуют аналогичные бесплатные программы, предлагаемые для операционных систем Windows, OS X и Linux. Такие приложения, как LibreOffice Calc и OpenOffice Calc, способны работать с принятым в Excel форматом `.xlsx` для файлов электронных таблиц, а это означает, что модуль `openpyxl` может работать и с электронными таблицами указанных приложений. Эти программы доступны для загрузки на сайтах `https://www.libreoffice.org/` и `http://www.openoffice.org/` соответственно. Даже если приложение Excel установлено на вашем компьютере, может оказаться так, что с упомянутыми программами вам будет легче работать. При этом следует заметить, что все снимки экрана, приведенные в этой главе, соответствуют программе Excel 2010, работающей под управлением Windows 7.

Документы Excel

Прежде всего обратимся к некоторым базовым определениям. Документ электронной таблицы Excel называется *рабочей книгой*. Рабочая книга хранится в файле с расширением *.xlsx*. Каждая книга может содержать любое количество *листов* (также называемых *рабочими листами*). Лист, просматриваемый пользователем в данный момент (или последний из тех, которые просматривались, прежде чем была закрыта программа Excel), называется *активным листом*.

Лист состоит из *столбцов* (адресуемых с помощью букв, начиная с A) и *строк* (адресуемых с помощью чисел, начиная с 1). Прямоугольная область, образуемая пересечением столбца и строки, называется *ячейкой*. Каждая ячейка таблицы может содержать числовое или текстовое значение. Совокупность ячеек вместе с содержащимися в них данными образует рабочий лист.

Установка модуля openpyxl

Модуль OpenPyXL не поставляется вместе с Python, поэтому его нужно установить самостоятельно. Следуйте приведенным в приложении A инструкциям по установке модулей, разработанных сторонними компаниями; имя модуля — `openpyxl`. Чтобы протестировать корректность установки, введите в интерактивной оболочке следующую команду:

```
>>> import openpyxl
```

В случае корректной установки модуля выполнение этой инструкции не должно сопровождаться выводом сообщения об ошибке. Также не забывайте импортировать модуль `openpyxl` перед началом работы с примерами в этой главе, иначе вы получите сообщение об ошибке `NameError: name 'openpyxl' is not defined`.

При подготовке этой книги использовалась версия модуля OpenPyXL 2.1.4, однако команда разработчиков регулярно выпускает новые версии модуля. Вы не должны беспокоиться по этому поводу, поскольку новые версии будут поддерживать обратную совместимость с кодом на Python, используемым в примерах[1]. Если у вас установлена более новая версия и вы хотели бы знать, какие новые возможности в ней доступны, обратитесь к полной документации к модулю OpenPyXL, которую можно найти на сайте `http://openpyxl.readthedocs.org/`.

[1] В процессе подготовки перевода книги использовалась версия модуля OpenPyXL 2.1.7, несовместимая с используемой автором версией в отношении построения диаграмм (см. далее). — *Примеч. ред.*

Чтение документов Excel

В этой главе мы будем использовать в примерах электронную таблицу *example.xlsx*, сохраненную в корневой папке. Вы можете либо самостоятельно создать этот файл, либо загрузить его на сайте `http://nostarch.com/automatestuff/`. На рис. 12.1 показаны вкладки трех заданных по умолчанию листов с именами `Лист1`, `Лист2` и `Лист3`, которые Excel автоматически предоставляет во вновь создаваемых рабочих книгах. (Количество открываемых по умолчанию листов может быть различным в зависимости от используемой операционной системы и приложения электронной таблицы.)

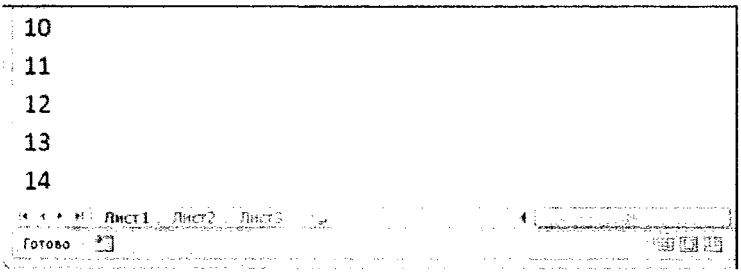


Рис. 12.1. Вкладки листов рабочей книги, расположенные в левом нижнем углу окна Excel

Содержимое листа `Лист1` из файла примера представлено в табл. 12.1.

Таблица 12.1. Рабочий лист `example.xlsx`

	A	B	C
1	05.04.2015 13:34	Яблоки	73
2	05.04.2015 3:41	Вишни	85
3	06.04.2015 12:46	Груши	14
4	08.04.2015 8:59	Апельсины	52
5	10.04.2015 2:07	Яблоки	152
6	10.04.2015 18:10	Бананы	23
7	10.04.2015 2:40	Земляника	98

Имея в своем распоряжении файл с примером, рассмотрим способы манипулирования электронными таблицами с помощью модуля `openpyxl`.

Открытие документов Excel с помощью модуля OpenPyXL

Импортировав модуль `openpyxl`, можно использовать функцию `openpyxl.load_workbook()`. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> type(wb)
<class 'openpyxl.workbook.workbook.Workbook'>
```

Функция `openpyxl.load_workbook()` принимает имя файла в качестве аргумента и возвращает значение типа `Workbook`. Объект `Workbook` представляет файл Excel, подобно тому как объект `File` представляет открытый текстовый файл.

Вспомните: для того чтобы можно было работать с файлом *example.xlsx*, он должен находиться в текущем рабочем каталоге. Вы можете определить, какая именно папка является текущим рабочим каталогом, импортировав модуль `os` и использовав функцию `os.getcwd()`, тогда как для смены рабочего каталога можно использовать функцию `os.chdir()`.

Получение списка листов рабочей книги

Для получения списка листов, входящих в состав рабочей книги, следует вызвать метод `get_sheet_names()`. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> wb.get_sheet_names()
['Лист1', 'Лист2', 'Лист3']
>>> sheet = wb.get_sheet_by_name('Лист3')
>>> sheet
<Worksheet "Лист3">
>>> type(sheet)
<class 'openpyxl.worksheet.worksheet.Worksheet'>
>>> sheet.title
'Лист3'
>>> anotherSheet = wb.get_active_sheet()
>>> anotherSheet
<Worksheet "Лист1">
```

Каждый лист представляется объектом `Worksheet`, который можно получить, передав строку с именем листа методу `get_sheet_by_name()` объекта `workbook`. Наконец, вызвав метод `get_active_sheet()` объекта `Workbook`, можно получить активный лист[2]. Активный лист — это лист, находящийся в начале списка, когда открывается рабочая книга Excel. Коль скоро у вас есть объект `Worksheet`, вы сможете получить его имя из атрибута `title`.

[2] В настоящее время функция `get_active_sheet()` считается устаревшей, и вместо нее рекомендуется использовать атрибут `active` объекта `Workbook`. — *Примеч. ред.*

Получение ячеек рабочих листов

Имея в своем распоряжении объект `Worksheet`, можно получить доступ к объекту `Cell` по его имени. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Лист1')
>>> sheet['A1']
<Cell Лист1.A1>
>>> sheet['A1'].value
datetime.datetime(2015, 4, 5, 13, 34, 2)
>>> c = sheet['B1']
>>> c.value
'Яблоки'
>>> 'Строка ' + str(c.row) + ', Столбец ' + c.column + ': ' +
↳ c.value
'Строка 1, Столбец B : Яблоки'
>>> 'Ячейка ' + c.coordinate + ' : ' + c.value
'Ячейка B1 : Яблоки'
>>> sheet['C1'].value
73
```

Объект `Cell` имеет атрибут `value`, который содержит значение, хранящееся в ячейке. Объекты `Cell` также имеют атрибуты `row`, `column` и `coordinate`, которые предоставляют информацию о расположении данной ячейки в таблице.

В данном примере обращение к значению атрибута `value` объекта `Cell` для ячейки B1 дает строку `Яблоки'`. Атрибут `row` дает целочисленное значение `1`, атрибут `column` — значение `'B'`, а атрибут `coordinate` — значение `'B1'`.

Модуль OpenPyXL автоматически интерпретирует даты в столбце A и возвращает их в виде значений типа `datetime`, а не в виде строк. Более подробно о типе данных `datetime` рассказывается в главе 16.

Программное задание столбца с помощью буквенного обозначения поначалу может вызывать затруднения, особенно при переходе через столбец Z, после которого обозначения становятся двухбуквенными: AA, AB, AC и т.д. В качестве альтернативного варианта можно обращаться к ячейке, используя метод `cell()` объекта `Sheet` с передачей ему целочисленных значений его именованных аргументов `row` и `column`. Первому столбцу или первой строке соответствует целое число `1`, а не `0`. Продолжите выполнение примера в интерактивной оболочке, введя следующие команды.

```
>>> sheet.cell(row=1, column=2)
<Cell Лист1.B1>
```

```
>>> sheet.cell(row=1, column=2).value
'Яблоки'
>>> for i in range(1, 8, 2):
        print(i, sheet.cell(row=i, column=2).value)

1 Яблоки
3 Груши
5 Яблоки
7 Земляника
```

Как можно видеть, вызывая метод `cell()` объекта `Sheet` с передачей ему аргументов `row=1` и `column=2`, мы получаем объект `Cell` для ячейки B1, что до этого было сделано с помощью метода `sheet['B1']`. Затем мы используем вызовы метода `cell()` с передачей ему именованных аргументов в цикле `for` для вывода значений последовательности ячеек.

Предположим, вы хотите смещаться вниз по столбцу B и выводить значения, содержащиеся в ячейках с нечетными номерами строк. Передав значение `2` параметру "step" функции `range()`, можно получить ячейки из каждой второй строки (в данном случае – из каждой нечетной). Именованному аргументу `row` метода `cell()` передается переменная `i` цикла `for`, тогда как именованному аргументу `column` всегда передается значение `2`. Заметьте, что передается именно целое число `2`, а не строка `'B'`.

Размер листа можно определить с помощью методов `get_highest_row()` и `get_highest_column()` объекта `Worksheet`. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Лист1')
>>> sheet.get_highest_row()
7
>>> sheet.get_highest_column()
3
```

Обратите внимание на то, что метод `get_highest_column()` возвращает целое число, а не букву, которая отображается в Excel.

Выполнение преобразований между буквенными и цифровыми обозначениями столбцов

Чтобы преобразовать буквенное обозначение столбца в цифровое, следует вызвать функцию `openpyxl.cell.column_index_from_string()`. Чтобы преобразовать цифровое обозначение столбца в буквенное, следует вызвать функцию `openpyxl.cell.get_column_letter()`. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> from openpyxl.cell import get_column_letter, column_index_from_string
>>> get_column_letter(1)
'A'
>>> get_column_letter(2)
'B'
>>> get_column_letter(27)
'AA'
>>> get_column_letter(900)
'AHP'
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Лист1')
>>> get_column_letter(sheet.get_highest_column())
'C'
>>> column_index_from_string('A')
1
>>> column_index_from_string('AA')
27
```

Импортировав указанные две функции из модуля `openpyxl.cell`, мы можем вызвать функцию `get_column_letter()` с передачей ей целочисленного значения, например 27, чтобы выяснить, какое буквенное обозначение соответствует столбцу 27. Функция `column_index_string()` решает обратную задачу: вы передаете ей буквенное имя столбца, и она возвращает вам его номер. Чтобы использовать эти функции, загружать рабочую книгу вовсе необязательно. Однако при желании можете загрузить рабочую книгу, получить объект `Worksheet` и вызвать один из его методов, например `get_highest_column()`, для получения целочисленного результата. Далее это целое число можно передать функции `get_column_letter()`.

Получение строк и столбцов рабочих листов

Используя срезы объектов `Worksheet`, можно получать все объекты `Cell`, принадлежащие определенной строке, столбцу или прямоугольной области электронной таблицы. После этого можно организовать цикл по всем ячейкам среза. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_sheet_by_name('Лист1')
>>> tuple(sheet['A1':'C3'])
((<Cell Лист1.A1>, <Cell Лист1.B1>, <Cell Лист1.C1>),
(<Cell Лист1.A2>, <Cell Лист1.B2>, <Cell Лист1.C2>),
(<Cell Лист1.A3>, <Cell Лист1.B3>, <Cell Лист1.C3>))
❶ >>> for rowOfCellObjects in sheet['A1':'C3']:
❷         for cellObj in rowOfCellObjects:
              print(cellObj.coordinate, cellObj.value)
          print('--- КОНЕЦ СТРОКИ ---')
```

```
A1 2015-04-05 13:34:02
B1 Яблоки
C1 73
--- КОНЕЦ СТРОКИ ---
A2 2015-04-05 03:41:23
B2 Вишни
C2 85
--- КОНЕЦ СТРОКИ ---
A3 2015-04-06 12:46:51
B3 Груши
C3 14
--- КОНЕЦ СТРОКИ ---
```

Здесь мы указываем, что нас интересуют ячейки прямоугольной области таблицы, левый верхний и правый нижний углы которой определяются ячейками A1 и C3, и получаем объект `Generator`, который содержит объекты `Cell`, принадлежащие указанной области. Чтобы было легче оценить, что именно представляет собой данный объект `Generator`, можно воспользоваться функцией `tuple()` и отобразить ячейки в виде кортежа.

Данный кортеж сам состоит из трех кортежей: по одному для каждой строки интересующей нас области в порядке следования сверху вниз. Каждый из трех внутренних кортежей содержит объекты `Cell`, принадлежащие одной строке, в порядке следования слева направо. Таким образом, в целом срез листа содержит все ячейки прямоугольной области таблицы, углы которой определяются ячейками A1 и C3.

Для вывода значений всех ячеек данной области используются два цикла `for`. Внешний цикл перебирает все строки в срезе ❶, тогда как вложенный цикл перебирает все ячейки текущей строки ❷.

Для доступа к ячейкам конкретной строки или столбца также можно воспользоваться атрибутами `rows` и `columns` объекта `Worksheet`. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_active_sheet()
>>> sheet.columns[1]
(<Cell Лист1.B1>, <Cell Лист1.B2>, <Cell Лист1.B3>, <Cell
Лист1.B4>, <Cell Лист1.B5>, <Cell Лист1.B6>, <Cell Лист1.B7>)
>>> for cellObj in sheet.columns[1]:
        print(cellObj.value)

Яблоки
Вишни
Груши
Апельсины
Яблоки
Бананы
Земляника
```

Используя атрибут `rows` объекта `Worksheet`, мы получаем кортеж кортежей. Каждый из внутренних кортежей представляет собой строку электронной таблицы и содержит ее ячейки в виде объектов `Cell`. Атрибут `columns` также дает кортеж кортежей, причем каждый из внутренних кортежей содержит объекты `Cell`, принадлежащие определенному столбцу. В случае таблицы *example.xlsx*, имеющей 7 строк и 3 столбца, атрибут `rows` дает кортеж, состоящий из 7 кортежей (каждый из которых содержит по 3 объекта `Cell`), а атрибут `columns` — кортеж, состоящий из 3 кортежей (каждый из которых содержит по 7 объектов `Cell`).

Чтобы получить доступ к определенному кортежу, можно сослаться на него по его индексу в охватывающем кортеже. Например, для получения кортежа, представляющего столбец B, следует использовать элемент `sheet.columns[1]`. Чтобы получить кортеж, содержащий объекты `Cell` столбца A, следует использовать элемент `sheet.columns[0]`. Как только в вашем распоряжении появляется кортеж, представляющий строку или столбец таблицы, можете организовать цикл по содержащимся в нем объектам `Cell` и вывести их значения.

Рабочие книги, листы и ячейки

В качестве краткого резюме ниже описан процесс чтения содержимого ячейки электронной таблицы с упоминанием всех вовлеченных в него функций, методов и типов данных.

1. Импортируйте модуль `openpyxl`.
2. Вызовите функцию `openpyxl.load_workbook()`.
3. Получите объект `Workbook`.
4. Вызовите метод `get_active_sheet()` или `get_sheet_by_name()` объекта `Workbook`.
5. Получите объект `Worksheet`.
6. Используйте индексирование или метод `cell()` объекта `Sheet`, передав ему именованные аргументы `row` и `column`.
7. Получите объект `Cell`.
8. Прочитайте значение атрибута `Cell` объекта `value`.

Проект: чтение данных электронной таблицы

Предположим, у вас имеется электронная таблица, содержащая данные переписи населения США в 2010 году, и вам предстоит утомительный просмотр тысяч строк для определения общей численности населения и количества переписных районов по округам. (Переписной район — это просто географическая территория, определенная для целей переписи

населения.) Каждая строка таблицы представляет один переписной район. Наш файл электронной таблицы будет называться *censuspopdata.xlsx*, и его можно загрузить на сайте `http://nostarch.com/automatestuff/`. Содержимое файла выглядит примерно так, как показано на рис. 12.2.

	A	B	C	D	E
1	**Переписной район**	**Штат**	**Округ**	**POP2010**	
9841	06075010500	CA	San Francisco	2685	
9842	06075010600	CA	San Francisco	3894	
9843	06075010700	CA	San Francisco	5592	
9844	06075010800	CA	San Francisco	4578	
9845	06075010900	CA	San Francisco	4320	
9846	06075011000	CA	San Francisco	4827	
9847	06075011100	CA	San Francisco	5164	

Региональная численность населе

Готово

Рис. 12.2. Электронная таблица censuspopdata.xlsx

Конечно, программа Excel способна рассчитать сумму нескольких выделенных ячеек, но при этом вам все еще необходимо самостоятельно выбирать ячейки для каждого из более чем 3000 с лишним округов. Даже если на расчет численности населения округа вручную у вас уйдет лишь несколько секунд, то вычисление всей таблицы потребует многих часов работы.

В этом проекте вы напишете сценарий, который будет читать файл электронной таблицы с данными переписи населения и рассчитывать статистику для всех округов за несколько секунд.

Вот что должна делать данная программа:

- читать данные из электронной таблицы Excel;
- подсчитывать количество переписных районов в каждом округе;
- подсчитывать численность населения, проживающего в каждом округе;
- выводить результаты.

Это означает, что ваш код должен выполнять следующие операции:

- открывать документ Excel и читать содержимое ячеек электронной таблицы с помощью модуля `openpyxl`;
- рассчитывать данные, касающиеся количества переписных районов и численности населения, и сохранять их в структуре данных;
- записывать структуру данных в текстовый файл с расширением *.py* с помощью модуля `pprint`.

Шаг 1. Чтение данных электронной таблицы

В электронной таблице *censuspopdata.xlsx* существует только один рабочий лист с именем `'Региональная численность населения'`, в каждой строке которого хранятся данные, относящиеся к одному переписному району. Столбцами таблицы являются номер переписного района (A), сокращенное название штата (B), название округа (C) и численность населения в переписном районе (D).

Откройте новое окно в файловом редакторе, введите в него следующий код и сохраните его в файле *readCensusExcel.py*.

```
#! python3
# readCensusExcel.py - Формирует таблицу данных о численности
# населения и количестве переписных районов в каждом округе.

❶ import openpyxl, pprint
print('Открытие рабочей книги...')
❷ wb = openpyxl.load_workbook('censuspopdata.xlsx')
❸ sheet = wb.get_sheet_by_name('Региональная численность
↪ населения')
countyData = {}

# TODO: Заполнить словарь countyData данными о численности
# населения и переписных районах округов.

print('Чтение строк...')
❹ for row in range(2, sheet.get_highest_row() + 1):
    # В каждой строке электронной таблицы содержатся данные для
    # одного переписного района.
    state = sheet['B' + str(row)].value
    county = sheet['C' + str(row)].value
    pop = sheet['D' + str(row)].value

# TODO: Открыть новый текстовый файл и записать в него
# содержимое словаря countyData.
```

Этот код импортирует модуль `openpyxl`, а также модуль `pprint`, который вы используете для вывода окончательных данных по кругу ❶. Далее мы открываем файл *censuspopdata.xlsx* ❷, получаем лист с данными переписи ❸ и итерируем по его строкам ❹.

Обратите внимание на создание переменной `countyData`, которая будет содержать данные по численности населения и количеству переписных районов, рассчитанные для каждого округа. Однако, прежде чем сохранить в ней что-либо, необходимо точно определить, как должны быть структурированы хранящиеся в ней данные.

Шаг 2. Заполнение структуры данных

В качестве структуры данных, сохраняемых в переменной countyData, мы выбираем словарь с сокращенными названиями штатов в качестве ключей. Каждое такое обозначение будет отображаться на другой словарь, ключами которого являются строки с названиями округов данного штата. В свою очередь, каждое название округа будет отображаться на словарь, имеющий всего два ключа — 'tracts' и 'pop'. Этим ключам соответствуют количество переписных районов и численность населения округа. Словарь верхнего уровня будет выглядеть примерно так.

```
{'AK': {'Aleutians East': {'pop': 3141, 'tracts': 1},
        'Aleutians West': {'pop': 5561, 'tracts': 2},
        'Anchorage': {'pop': 291826, 'tracts': 55},
        'Bethel': {'pop': 17013, 'tracts': 3},
        'Bristol Bay': {'pop': 997, 'tracts': 1},
--пропущенный код--
```

Если бы в переменной countyData был сохранен предыдущий словарь, то мы получили бы следующие результаты.

```
>>> countyData['AK']['Anchorage']['pop']
291826
>>> countyData['AK']['Anchorage']['tracts']
55
```

Ключи словаря countyData будут иметь следующий общий вид.

```
countyData[state abbrev][округ]['tracts']
countyData[state abbrev][округ]['pop']
```

Теперь, когда вам уже известно, как будут структурированы данные в переменной countyData, вы можете написать код, который заполняет эту структуру данными округа. Добавьте в конце программы код, выделенный ниже полужирным шрифтом.

```
#! python 3
# readCensusExcel.py - Формирует таблицу данных о численности
# населения и количестве переписных районов в каждом округе.

--пропущенный код--

for row in range(2, sheet.get_highest_row() + 1):
    # В каждой строке электронной таблицы содержатся данные для
    # одного переписного района.
    state = sheet['B' + str(row)].value
```

```
        county = sheet['C' + str(row)].value
        pop = sheet['D' + str(row)].value

        # Гарантия существования ключа для данного штата.
❶       countyData.setdefault(state, {})
        # Гарантия существования ключа для данного округа
        # данного штата.
❷       countyData[state].setdefault(county, {'tracts': 0,
                                              'pop': 0})
        # Каждая строка представляет один переписной район,
        # поэтому инкрементировать на единицу.
❸       countyData[state][county]['tracts'] += 1
        # Увеличить численность населения округа на численность
        # населения переписного района.
❹       countyData[state][county]['pop'] += int(pop)

        # TODO: Открыть новый текстовый файл и записать в него
        # содержимое словаря countyData.
```

Последние две строки кода выполняют всю фактическую вычислительную работу, инкрементируя значение ключа `tracts` ❸ и увеличивая значение ключа `pop` ❹ для текущего округа на каждой итерации цикла `for`.

Необходимость остальной части нового кода обусловлена тем, что вы не можете добавить словарь округа в качестве значения для ключа сокращенного названия штата до тех пор, пока сам ключ не будет существовать в переменной `countyData`. (Иначе говоря, инструкция `countyData['AK']['Anchorage']['tracts'] += 1` вызовет ошибку, если ключ `'AK'` еще не существует.) Чтобы гарантировать существование ключа сокращенного названия штата в структуре данных, следует вызвать метод `setdefault()` и установить значение ключа, если для данного штата оно еще не существует ❶.

Подобно тому как словарю `countyData` нужен другой словарь в качестве значения каждого из ключей сокращенных названий штатов, каждый из *этих* словарей также будет нуждаться в собственном словаре в качестве значения ключа для каждого округа ❷. В свою очередь, каждому из *этих* словарей также требуются ключи `'tracts'` и `'pop'`, значения которых последовательно увеличиваются, начиная с `0`. (Если вы чувствуете, что потеряли нить рассуждений, вернитесь к рассмотрению примера словаря в начале этого раздела.)

Поскольку, если ключ уже существует, метод `setdefault()` не будет выполнять никаких действий, вы можете свободно вызывать его на каждой итерации цикла `for`.

Шаг 3. Запись результатов в файл

Когда цикл `for` закончит свою работу, словарь `countyData` будет содержать всю информацию о численности населения и количестве переписных

районов, структурированную с помощью ключей по округам и штатам. На данном этапе вы уже могли бы написать программный код для записи этой информации в текстовый файл или другую электронную таблицу Excel. Мы же ограничимся использованием функции `pprint.pformat()` для записи словаря `countyData` в виде массивной строки в файл *census2010.py*. Добавьте в конец программы код, выделенный ниже полужирным шрифтом (этот код должен быть вне цикла, поэтому проследите за тем, чтобы он был введен без отступов).

```
#! python 3
# readCensusExcel.py - Формирует таблицу данных о численности
# населения и количестве переписных районов в каждом округе.

--пропущенный код--

for row in range(2, sheet.get_highest_row() + 1):
--пропущенный код--

# Открытие нового текстового файла и запись в него
# содержимого словаря countyData.
print('Запись результатов...')
resultFile = open('census2010.py', 'w')
resultFile.write('allData = ' + pprint.pformat(countyData))
resultFile.close()
print('Готово.')
```

Функция `pprint.pformat()` создает строку, отформатированную в виде действительного кода на языке Python. Выводя ее в текстовый файл *census2010.py*, вы генерируете программу на языке Python из своей Python-программы! Это может показаться ненужным усложнением, но теперь вы можете импортировать файл *census2010.py* подобно любому другому модулю Python. В интерактивной оболочке выполните инструкцию перехода из текущего рабочего каталога в папку, в которой находится вновь созданный файл *census2010.py* (на моем ноутбуке это папка *C:\Python34*), и импортируйте этот файл как модуль.

```
>>> import os
>>> os.chdir('C:\\Python34')
>>> import census2010
>>> census2010.allData['AK']['Anchorage']
{'pop': 291826, 'tracts': 55}
>>> anchoragePop = census2010.allData['AK']['Anchorage']['pop']
>>> print('В 2010 году численность населения округа Anchorage
↪ составляла ' + str(anchoragePop))
В 2010 году численность населения округа Anchorage составляла
291826
```

Программа *readCensusExcel.py* относится к категории одноразовых: как только вы получите результаты ее работы, сохраненные в файле *census2010.py*, вам никогда не придется запускать ее повторно. Всякий раз, когда вам понадобятся данные по округам, вы сможете просто импортировать модуль `census2010`.

Расчет этих данных вручную занял бы у вас несколько часов, тогда как программа управилась с этим за несколько секунд. Используя модуль OpenPyXL, вы избавитесь от проблем с извлечением данных, сохраненных в электронных таблицах Excel, и выполнением вычислений над ними. Готовый код этой программы доступен для загрузки на сайте `http://nostarch.com/automatestuff/`.

Идеи относительно создания аналогичных программ

Excel используется многими предприятиями и организациями для хранения различных типов данных, и нередко электронные таблицы разрастаются настолько, что работать с ними становится трудно. Любая программа, работающая с данными в формате Excel, подобна только что рассмотренной: она должна загрузить файл электронной таблицы, подготовить соответствующие переменные или структуры данных, а затем организовать обработку строк таблицы в цикле. Такие программы могут использоваться для решения следующих задач:

- сравнение данных, хранящихся в нескольких строках электронной таблицы;
- открытие нескольких файлов Excel и сравнение данных, хранящихся в различных таблицах;
- поиск пустых строк или недопустимых данных в ячейках электронной таблицы и вывод предупреждающих сообщений в случае их обнаружения;
- чтение данных из электронной таблицы и их использование в качестве входных данных программ на языке Python.

Запись документов Excel

Модуль OpenPyXL также предоставляет возможность записывать данные, а это означает, что ваши программы могут создавать и изменять файлы электронных таблиц. С помощью Python создание электронных таблиц, насчитывающих тысячи строк данных, не составляет никакого труда.

Создание и сохранение документов Excel

Чтобы создать новый пустой объект `Workbook`, следует вызвать функцию `openpyxl.Workbook()`. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> wb.get_sheet_names()
['Sheet']
>>> sheet = wb.get_active_sheet()
>>> sheet.title
'Sheet'
>>> sheet.title = 'Spam Bacon Eggs Sheet'
>>> wb.get_sheet_names()
['Spam Bacon Eggs Sheet']
```

Рабочая книга открывается с одним рабочим листом Sheet. Имя листа можно изменить, сохранив в его атрибуте `title` новую строку.

Любые изменения объекта `Workbook` или его листов и ячеек не будут сохранены в файле электронной таблицы до тех пор, пока вы не вызовете метод `save()` для этого объекта. Введите в интерактивной оболочке следующие команды (предполагается, что файл *example.xlsx* находится в текущем рабочем каталоге).

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('example.xlsx')
>>> sheet = wb.get_active_sheet()
>>> sheet.title = 'Spam Spam Spam'
>>> wb.save('example_copy.xlsx')
```

Здесь мы переименовываем рабочий лист. Чтобы сохранить это изменение, передаем методу `save()` строку с новым именем файла. Передача имени файла, отличного от первоначального, например `'example_copy.xlsx'`, приводит к сохранению копии электронной таблицы.

Всякий раз, когда вы вносите изменения в электронную таблицу, загруженную из файла, сохраняйте ее в файле, имя которого отличается от первоначального. Это будет гарантией того, что в случае записи в новый файл некорректных данных или их повреждения из-за ошибок в программе вы всегда сможете продолжить работу, вернувшись к исходному файлу.

Создание и удаление рабочих листов

Для добавления и удаления листов из рабочей книги используются соответственно методы `create_sheet()` и `remove_sheet()`. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> wb.get_sheet_names()
['Sheet']
>>> wb.create_sheet()
<Worksheet "Sheet1">
>>> wb.get_sheet_names()
['Sheet', 'Sheet1']
>>> wb.create_sheet(index=0, title='Первый лист')
<Worksheet "Первый лист">
>>> wb.get_sheet_names()
['Первый лист', 'Sheet', 'Sheet1']
>>> wb.create_sheet(index=2, title='Средний лист')
<Worksheet "Средний лист">
>>> wb.get_sheet_names()
['Первый лист', 'Sheet', 'Средний лист', 'Sheet1']
```

Метод `create_sheet()` возвращает новый объект `Worksheet` с именем `ЛистX`, который по умолчанию становится последним листом книги. При желании можно задать с помощью именованных аргументов `index` и `title` не только имя, но и индекс вновь создаваемого листа.

Продолжите предыдущий пример, введя следующие команды.

```
>>> wb.get_sheet_names()
['Первый лист', 'Sheet', 'Средний лист', 'Sheet1']
>>> wb.remove_sheet(wb.get_sheet_by_name('Средний лист'))
>>> wb.remove_sheet(wb.get_sheet_by_name('Sheet1'))
>>> wb.get_sheet_names()
['Первый лист', 'Sheet']
```

Метод `remove_sheet()` принимает в качестве аргумента не строку с именем листа, а объект `Worksheet`. Если вам известно лишь имя листа, который вы хотите удалить, вызовите метод `get_sheet_by_name()` и передайте возвращенное им значение методу `remove_sheet()`.

Не забывайте вызывать метод `save()`, чтобы сохранить изменения после того, как добавили или удалили лист из рабочей книги.

Запись значений в ячейки

Запись значений в ячейки во многом напоминает запись значений в ключи словаря. Введите в интерактивной оболочке следующий код.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_sheet_by_name('Sheet')
>>> sheet['A1'] = 'Здравствуй, мир!'
>>> sheet['A1'].value
'Здравствуй, мир!'
```

Если у вас имеются координаты ячейки в виде строки, можете использовать эту строку в качестве ключа словаря для задания ячейки, значение которой нужно изменить.

Проект: обновление электронной таблицы

В этом проекте мы напишем программу, которая обновляет ячейки электронной таблицы, содержащей данные об объеме продаж. Программа будет просматривать электронную таблицу, находить конкретные виды продукции и обновлять их цены. Электронная таблица *produceSales.xlsx*, которую мы будем использовать (рис. 12.3), доступна для загрузки на сайте `http://nostarch.com/automatestuff/`.

	A	B	C	D	E
1	НАИМЕНОВАНИЕ	ЦЕНА (за 1 фунт)	ПРОДАНО (фунт)	ВЫРУЧКА	
2	Картофель	0,86	21,6	18,58	
3	Бамия	2,26	38,6	87,24	
4	Бобы	2,69	32,8	88,23	
5	Арбузы	0,66	27,3	18,02	
6	Чеснок	1,19	4,9	5,83	
7	Пастернак	2,27	1,1	2,5	
8	Спаржа	2,49	37,9	94,37	
9	Авокадо	3,23	9,2	29,72	
10	Сельдерей	3,07	28,9	88,72	
11	Бамия	2,26	40	90,4	

Лист2 Лист3

Готово 125%

Рис. 12.3. Электронная таблица данных об объемах продаж

Каждая строка представляет отдельную операцию продажи. Столбцами являются вид проданного продукта (A), стоимость одного фунта продукта (B), проданное количество в фунтах (C) и сумма выручки от продажи (D). Для столбца ВЫРУЧКА задана формула Excel `=ROUND(B3*C3, 2)`, в соответствии с которой стоимость одного фунта продукта умножается на количество проданного товара и результат округляется с точностью до ближайших сотых. Благодаря этому значения ячеек в столбце ИТОГО будут автоматически обновляться при изменении значений ячеек в столбцах B и C.

А теперь представьте, что цены на чеснок, сельдерей и лимоны были введены неправильно, и вам предстоит утомительный просмотр тысяч строк этой электронной таблицы для исправления данных о цене во всех строках, относящихся к чесноку, сельдерею и лимонам. Воспользоваться операцией поиска и замены с использованием значения цены в данном случае нельзя, поскольку цена какого-либо другого продукта может случайно

оказаться такой же, в результате чего она будет ошибочно распознана как неправильная. На выполнение этой работы вручную у вас уйдет много часов. Однако вы можете написать программу, которая справится с этим за несколько секунд.

Ваша программа должна делать следующее:

- просматривать все строки в цикле;
- изменять показатели цены для чеснока, сельдерея и лимонов.

Это означает, что ваш код должен выполнять следующие операции:

- открывать файл электронной таблицы;
- проверять для каждой строки, не содержит ли столбец A значение Лимон, Сельдерей или Чеснок.
- в случае положительного результата проверки изменять значение цены в столбце B.
- сохранять электронную таблицу в новом файле (в качестве подстраховки, чтобы не потерять таблицу с прежними значениями).

Шаг 1. Создание структуры, содержащей данные для обновления

Ниже приведены новые цены, которые должны быть внесены в электронную таблицу:

Лимон	1.27
Сельдерей	1.19
Чеснок	3.07

Соответственно, мы могли бы написать следующий код.

```
if produceName == 'Лимон':
    cellObj = 1.27
if produceName == 'Сельдерей':
    cellObj = 1.19
if produceName == 'Чеснок':
    cellObj = 3.07
```

Однако подход, основанный на жестко запрограммированных данных обновления цен на продукты, не отличается элегантностью. Если придется вновь обновлять цены, и при этом необязательно для тех же продуктов, то это потребует повторного изменения кода. Но каждый раз, когда вы изменяете код, существует риск внесения в него ошибок в процессе ввода.

Более гибкий подход состоит в том, чтобы сохранить информацию о скорректированных ценах в виде словаря и написать код, использую-

щий эту структуру данных. Откройте новое окно в файловом редакторе и введите следующий код.

```
#! python3
# updateProduce.py - Корректирует цены в электронной таблице
# данных об объемах продаж.

import openpyxl

wb = openpyxl.load_workbook('produceSales.xlsx')
sheet = wb.get_sheet_by_name('Лист')

# Типы продукции и их обновленные цены.
PRICE_UPDATES = {'Лимон': 3.07,
                 'Сельдерей': 1.19,
                 'Чеснок': 1.27}

# TODO: Создать цикл по строкам и обновить цены.
```

Сохраните этот файл, присвоив ему имя *updateProduce.py*. Если вновь потребуется обновить электронную таблицу, нужно будет обновить только словарь `PRICE_UPDATES` и никакой другой код.

Шаг 2. Проверка всех строк и обновление некорректных цен

В следующей части программы организуется цикл по всем строкам электронной таблицы. Добавьте в конце файла *updateProduce.py* код, выделенный ниже полужирным шрифтом.

```
#! python3
# updateProduce.py - Корректирует цены в электронной таблице
# данных об объемах продаж.

--пропущенный код--

# Создание цикла по строкам и обновление цен.
❶ for rowNum in range(2, sheet.get_highest_row()): # пропустить
                                                   # первую
                                                   # строку
❷     produceName = sheet.cell(row=rowNum, column=1).value
❸         if produceName in PRICE_UPDATES:
              sheet.cell(row=rowNum, column=2).value =
  ↳ PRICE_UPDATES[produceName]

❹ wb.save('updatedProduceSales.xlsx')
```

В этом коде цикл по строкам электронной таблицы начинается со строки 2, поскольку строка 1 — это заголовок ❶. Ячейка из столбца 1 (т.е. столбца A) сохраняется в переменной `produceName` ❷. Если ключ `produceName` существует в словаре `PRICE_UPDATES`❸, значит, это и есть строка, цена в которой подлежит исправлению. Правильное значение цены хранится в элементе словаря `PRICE_UPDATES[produceName]`.

Обратите внимание на то, насколько аккуратнее стал выглядеть код благодаря использованию словаря `PRICE_UPDATES`. Для обновления всех цен, нуждающихся в исправлении, потребовалась всего лишь одна инструкция `if`, а не три инструкции наподобие `if produceName == 'Чеснок':`. А поскольку вместо жесткого программирования названий и обновленных цен продуктов используется словарь `PRICE_UPDATES`, то при необходимости внести дополнительные изменения в электронную таблицу нужно будет изменить лишь словарь, но не код программы.

По завершении просмотра всей таблицы и внесения изменений код сохраняет объект `Workbook` в файле *updatedProduceSales.xlsx* ❹. Тем самым мы избегаем затирания старого файла электронной таблицы, который может понадобиться нам, если из-за ошибок в программе внесенные в таблицу исправления оказались неверными. Впоследствии, когда вы убедитесь в корректности обновленного варианта электронной таблицы, прежний файл можно будет удалить.

Готовый код этой программы доступен для загрузки на сайте `http://nostarch.com/automatestuff/`.

Идеи относительно создания аналогичных программ

Поскольку многим офисным сотрудникам приходится постоянно работать с электронными таблицами Excel, любая программа, способная автоматизировать процесс редактирования и записи Excel-документов, может принести реальную пользу. Такая программа может быть использована для решения следующих задач:

- чтение данных из одной электронной таблицы и их запись в другую таблицу;
- чтение данных с веб-сайтов, из текстовых файлов или буфера обмена и их запись в электронную таблицу;
- автоматическое "приведение в порядок" данных, хранящихся в электронной таблице; например, программа может использовать регулярные выражения для чтения телефонных номеров в различных форматах и приведения их к единому стандартному формату.

Настройка типов шрифтов, используемых в ячейках таблицы

Стилевое оформление определенных ячеек, строк или столбцов можно использовать для выделения важных областей электронной таблицы. Например, в нашей электронной таблице программа может выделить полужирным шрифтом строки, содержащие данные, касающиеся картофеля, чеснока и пастернака. Возможно, вы захотите выделить курсивом все строки, в которых цена продукта превышает 5 долларов. Стилевое оформление большой электронной таблицы очень трудоемко, но ваша программа может это сделать в считанные секунды.

Для настройки шрифтов, используемых в ячейках, необходимо импортировать функции `Font()` и `Style()` из модуля `openpyxl.styles`.

```
from openpyxl.styles import Font, Style
```

Это позволит использовать в коде запись `Font()` вместо более длинной записи `openpyxl.styles.Font()`. (Более подробно об этой форме инструкции `import` читайте в разделе "Импортирование модулей" главы 2.)

Ниже приведен пример создания новой рабочей книги, в которой для шрифта, используемого в ячейке A1, устанавливаются курсивное начертание и размер 24 пункта. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> from openpyxl.styles import Font, Style
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_sheet_by_name('Sheet')
❶ >>> italic24Font = Font(size=24, italic=True)
❷ >>> styleObj = Style(font=italic24Font)
❸ >>> sheet['A'].style/styleObj
>>> sheet['A1'] = 'Здравствуй мир!'
>>> wb.save('styled.xlsx')
```

Модуль OpenPyXL предоставляет коллекцию стилевых настроек ячейки с помощью объекта `Style`, который хранится в атрибуте `style` объекта `Cell`. Чтобы задать стиль ячейки, следует присвоить атрибуту `style` соответствующий объект `Style`.

В данном примере вызов функции `Font(size=24, italic=True)` возвращает объект `Font`, который сохраняется в переменной `italic24Font` ❶. Именованные аргументы функции `Font()`, `size` и `italic` позволяют сконфигурировать стилевые атрибуты объекта `Font`. Затем этот объект `Font` передается вызову `Style(font=italic24Font)`, возвращаемое значение которого

сохраняется в переменной `styleObj` ❷. А когда значение `styleObj` присваивается атрибуту `style` ячейки ❸, вся информация о шрифте применяется к ячейке A1.

Объекты Font

Атрибуты `style` объектов `Font` оказывают влияние на то, как текст отображается в ячейках. Атрибуты стиля шрифта устанавливаются путем передачи именованных аргументов функции `Font()`. Их перечень приведен в табл. 12.2.

Таблица 12.2. Именованные аргументы, определяющие атрибуты `style` объектов `Font`

Именованный аргумент	Тип данных	Описание
`name`	`String`	Название шрифта, например `'Calibri'` или `'Times New Roman'`
`size`	`Integer`	Размер шрифта
`bold`	`Boolean`	`True` для полужирного начертания
`italic`	`Boolean`	`True` для курсивного начертания

Вы можете вызвать функцию `Font()` для создания объекта `Font` и сохранить этот объект в переменной. Затем эту переменную можно передать функции `Style()`, сохранить результирующий объект `Style` в переменной и присвоить эту переменную атрибуту `style` объекта `Cell`. Вот пример кода, создающего различные варианты шрифтовых стилей.

```
>>> import openpyxl
>>> from openpyxl.styles import Font, Style
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_sheet_by_name('Sheet')

>>> fontObj1 = Font(name='Times New Roman', bold=True)
>>> styleObj1 = Style(font=fontObj1)
>>> sheet['A1'].style = styleObj1
>>> sheet['A1'] = 'Bold Times New Roman'

>>> fontObj2 = Font(size=24, italic=True)
>>> styleObj2 = Style(font=fontObj2)
>>> sheet['B3'].style = styleObj2
>>> sheet['B3'] = '24 pt Italic'

>>> wb.save('styles.xlsx')
```

Здесь мы сохраняем объект `Font` в переменной `fontObj1` и используем ее для создания объекта `Style`, который сохраняем в переменной `styleObj1`,

после чего присваиваем значение этой переменной атрибуту `style` объекта `Cell` ячейки A1. Затем этот процесс повторяется с использованием других объектов `Font` и `Style` для задания стиля второй ячейки. В результате выполнения этого кода для ячеек A1 и B3 электронной таблицы будут установлены заданные нами стили шрифта, как показано на рис. 12.4.

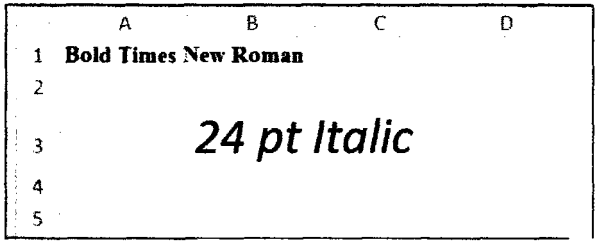


Рис. 12.4. Электронная таблица с модифицированными стилями шрифтов

В случае ячейки A1 мы устанавливаем для названия шрифта `name` значение `'Times New Roman'`, а для начертания шрифта `bold` — значение `true`, поэтому для вывода текста с названием шрифта "Times New Roman" используется полужирное начертание. Поскольку размер шрифта не был нами указан, для него используется значение `11`, установленное модулем `openpyxl` по умолчанию. В случае ячейки B3 наш текст выводится с использованием курсивного начертания и с размером шрифта, равным `24`. Поскольку название шрифта не было нами указано, используется шрифт Calibri, установленный модулем `openpyxl` по умолчанию.

Формулы

Формулы, начинающиеся со знака равенства, позволяют устанавливать для ячеек значения, рассчитанные на основании значений, хранящихся в других ячейках. В этом разделе вы будете использовать модуль `openpyxl` для программного добавления формул в ячейки тем же способом, которым ячейкам присваиваются обычные значения, например:

```
>>> sheet['B9'] = '=SUM(B1:B8)'
```

Эта инструкция сохранит `=SUM(B1:B8)` в качестве значения в ячейке B9. Тем самым для ячейки B9 задается формула, которая суммирует значения, хранящиеся в ячейках от B1 до B8. Результат показан на рис. 12.5.

Формула задается подобно любому другому текстовому значению в ячейке. Введите в интерактивной оболочке следующие команды.

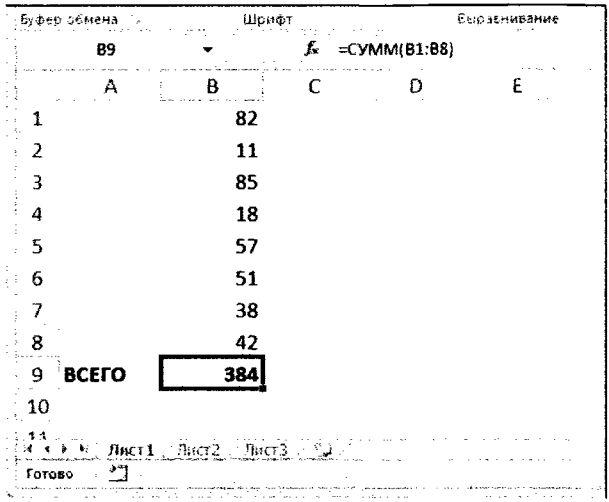


Рис. 12.5. В ячейке B9 содержится формула =SUM(B1:B8), которая складывает содержимое ячеек от B1 до B8.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
>>> sheet['A1'] = 200
>>> sheet['A2'] = 300
>>> sheet['A3'] = '=SUM(A1:A2)'
>>> wb.save('writeFormula.xlsx')
```

Для ячеек A1 и A2 устанавливаются значения `200` и `300` соответственно. Значение в ячейке A3 определяется формулой, складывающей значения ячеек A1 и A2. Если открыть эту электронную таблицу в приложении Excel, то в качестве значения ячейки A3 отобразится `500`.

Хранящуюся в ячейке формулу можно читать, как любое другое значение. Однако, если вы хотите увидеть *результат расчета по формуле*, а не саму формулу, то при вызове функции `load_workbook()` ей следует передать именованный аргумент `data_only` со значением `True`. Это означает, что объект `Workbook` может отображать либо формулы, либо результаты вычисления формул, но не то и другое. (Однако ничто не мешает вам иметь несколько загруженных объектов `Workbook` для одной и той же электронной таблицы.) Чтобы увидеть разницу между загрузкой рабочей книги с использованием и без использования именованного аргумента `data_only`, введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wbFormulas = openpyxl.load_workbook('writeFormula.xlsx')
>>> sheet = wbFormulas.get_active_sheet()
```

```
>>> sheet['A3'].value
'=SUM(A1:A2)'

>>> wbDataOnly = openpyxl.load_workbook('writeFormula.xlsx',
                 data_only=True)
>>> sheet = wbDataOnly.get_active_sheet()
>>> sheet['A3'].value
500
```

Здесь при вызове функции `load_workbook()` с передачей ей именованного аргумента `data_only=True` в ячейке A3 отображается значение `500`, которое является результатом вычисления формулы `=SUM(A1:A2)`, а не текстом самой формулы.

Формулы Excel привносят определенный уровень программируемости в электронные таблицы, но по мере усложнения задач работать с ними становится все труднее. Например, даже при отличном владении инструментом формул Excel попытки понять смысл формулы `=IFERROR(TRIM(IF(LEN(VLOOKUP(F7, Sheet2!$A$1:$B$10000, 2,-FALSE))>0,SUBSTITUTE(VLOOKUP(F7, Sheet2!$A$1:$B$10000, 2, FALSE),-" ", ""),""))`, `"")` могут довести до головной боли. Читать код на языке Python гораздо легче.

Настройка строк и столбцов

В Excel изменить размер строки и столбца не составляет никакого труда. Для этого достаточно перетащить мышью границу заголовка строки или столбца в желаемую позицию. Но если вам необходимо устанавливать размеры строк и столбцов в зависимости от содержимого ячеек или задавать их размеры для большой совокупности файлов электронных таблиц, то будет гораздо быстрее написать программу на Python, которая сделает это вместо вас.

Кроме того, строки и столбцы можно полностью скрывать из виду. Их также можно закреплять на месте, чтобы они всегда были видны на экране и появлялись на каждой странице при выводе электронной таблицы на печать (эту возможность очень удобно использовать в отношении заголовков).

Настройка высоты строк и ширины столбцов

Объекты `Worksheet` имеют атрибуты `row_dimensions` и `column_dimensions`, которые управляют высотой строк и шириной столбцов. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
```

```
>>> sheet['A1'] = 'Высокая строка'
>>> sheet['B2'] = 'Широкий столбец'
>>> sheet.row_dimensions[1].height = 70
>>> sheet.column_dimensions['B'].width = 20
>>> wb.save('dimensions.xlsx')
```

Атрибуты `row_dimensions` и `column_dimensions` рабочего листа представляют собой значения, подобные словарю. Атрибут `row_dimensions` содержит объекты `RowDimension`, а атрибут `column_dimensions` содержит объекты `ColumnDimension`. Доступ к объектам в `row_dimensions` осуществляется с использованием номера строки (в данном случае — 1 и 2), а доступ к объектам в `column_dimensions` — с использованием буквы столбца (в данном случае — A и B).

Внешний вид электронной таблицы `dimensions.xlsx` представлен на рис. 12.6.

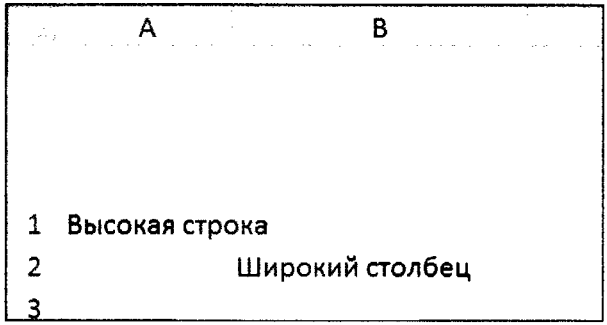


Рис. 12.6. Для строки 1 и столбца B установлены большие значения высоты и ширины

Получив доступ к объекту `RowDimension`, вы можете задать высоту строки, тогда как получение доступа к объекту `ColumnDimension` позволяет установить ширину столбца. Для указания высоты строки разрешено использование целочисленных или вещественных значений в диапазоне от `0` до `409`. Единицами измерения служат *пункты* (1 пункт равен 1/72 дюйма). По умолчанию высота строк устанавливается равной `12.75`. Для указания ширины столбца разрешено использовать целочисленные или вещественные значения в диапазоне от `0` до `255`. Это значение определяет допустимое количество символов с заданным по умолчанию размером шрифта (11 пунктов), отображаемых в ячейке. По умолчанию ширина столбцов составляет `8.43` символа. Столбцы с нулевой шириной и строки с нулевой высотой невидимы для пользователя.

Слияние и отмена слияния ячеек

Ячейки, занимающие прямоугольную область, могут быть объединены в одну ячейку с помощью метода `merge_cells()` рабочего листа. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.Workbook()
>>> sheet = wb.get_active_sheet()
>>> sheet.merge_cells('A1:D3')
>>> sheet['A1'] = 'Объединены двенадцать ячеек.'
>>> sheet.merge_cells('C5:E5')
>>> sheet['C5'] = 'Объединены три ячейки.'
>>> wb.save('merged.xlsx')
```

Аргументом метода `merge_cells()` служит строка, используемая для указания верхней левой и нижней правой ячеек прямоугольной области. Принадлежащие ей ячейки должны быть объединены в одну: строке `'A1:D3'` соответствует слияние 12 ячеек. Чтобы установить значение для этой единственной ячейки, достаточно установить значение для верхней левой ячейки исходной группы.

Выполнив этот код, вы получите результат, представленный на рис. 12.7.

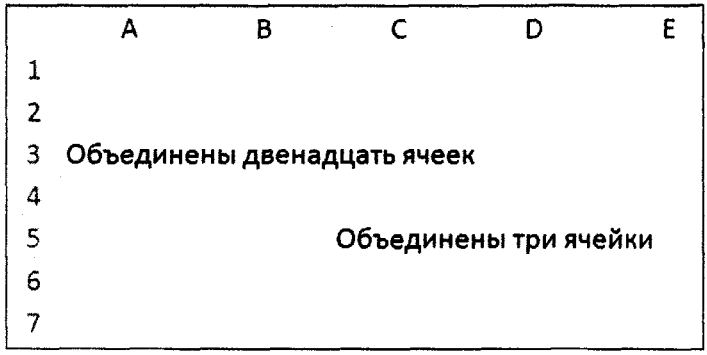


Рис. 12.7. Объединение ячеек в электронной таблице

Чтобы отменить слияние ячеек, вызовите метод `unmerge_cells()` рабочего листа. Введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('merged.xlsx')
>>> sheet = wb.get_active_sheet()
>>> sheet.unmerge_cells('A1:D3')
>>> sheet.unmerge_cells('C5:E5')
>>> wb.save('merged.xlsx')
```

Если вы сохраните изменения и взглянете на таблицу, то увидите, что слившиеся ячейки вновь разделились.

Закрепление областей

Если размер электронной таблицы настолько велик, что ее нельзя увидеть целиком, можно заблокировать несколько верхних строк или крайних слева столбцов в их позициях на экране. В этом случае, например, пользователь всегда будет видеть заблокированные заголовки столбцов или строк, даже если он прокручивает электронную таблицу на экране. Такие заблокированные в своих позициях ячейки называют *закрепленными областями* (freeze panes). В модуле OpenPyXL у каждого объекта `Worksheet` имеется атрибут `freeze_panes`, значением которого может служить объект `Cell` или строка с координатами ячеек. Обратите внимание на то, что все строки и столбцы, расположенные соответственно выше и левее, будут заблокированы, но строка и столбец, в которых расположена сама ячейка, останутся свободными.

Чтобы отменить блокирование всех закрепленных областей, достаточно установить значение атрибута `freeze_panes` равным `None` или `'A1'`. В табл. 12.3 показано, какие строки и столбцы будут заблокированы при некоторых значениях атрибута `freeze_panes`, выбранных в качестве примера.

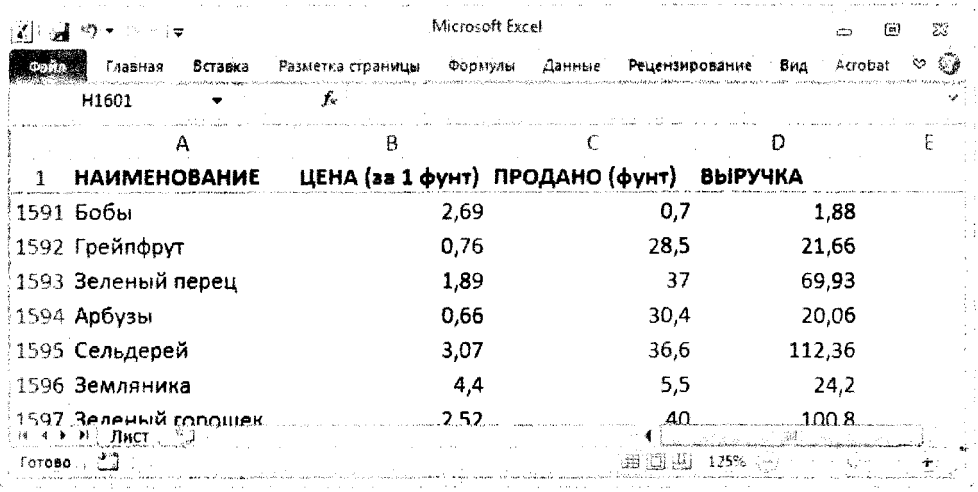
Таблица 12.3. Примеры закрепленных областей

Настройка атрибута `freeze_panes`	Заблокированные строки и столбцы
`sheet.freeze_panes = 'A2'`	Строка 1
`sheet.freeze_panes = 'B1'`	Столбец A
`sheet.freeze_panes = 'C1'`	Столбцы A и B
`sheet.freeze_panes = 'C2'`	Строка 1 и столбцы A и B
`sheet.freeze_panes = 'A1'` или `sheet.freeze_panes = None`	Закрепленные области отсутствуют

Убедившись в том, что в текущем рабочем каталоге находится файл электронной таблицы с данными о продажах, загруженный вами ранее на сайте `http://nostarch.com/automatestuff/`, введите в интерактивной оболочке следующие команды.

```
>>> import openpyxl
>>> wb = openpyxl.load_workbook('produceSales.xlsx')
>>> sheet = wb.get_active_sheet()
>>> sheet.freeze_panes = 'A2'
>>> wb.save('freezeExample.xlsx')
```

Если установить для атрибута `freeze_panes` значение `'A2'`, то строка 1 будет всегда оставаться видимой, независимо от прокрутки электронной таблицы. Результат представлен на рис. 12.8.



	A	B	C	D	E
1	НАИМЕНОВАНИЕ	ЦЕНА (за 1 фунт)	ПРОДАНО (фунт)	ВЫРУЧКА	
1591	Бобы	2,69	0,7	1,88	
1592	Грейпфрут	0,76	28,5	21,66	
1593	Зеленый перец	1,89	37	69,93	
1594	Арбузы	0,66	30,4	20,06	
1595	Сельдерей	3,07	36,6	112,36	
1596	Земляника	4,4	5,5	24,2	
1597	Зеленый горошек	2,52	40	100,8	

Рис. 12.8. При установке значения атрибута `freeze_panes` равным `'A2'` строка 1 всегда будет оставаться видимой, независимо от прокрутки электронной таблицы на экране

Диаграммы

Модуль OpenPyXL поддерживает создание гистограмм, графиков, а также точечных и круговых диаграмм с использованием данных, хранящихся в электронной таблице. Чтобы создать диаграмму, необходимо выполнить следующие действия:

1) создать объект `Reference` на основе ячеек в пределах выделенной прямоугольной области;
2) создать объект `Series`, передав функции `Series()` объект `Reference`;
3) создать объект `Chart`;
4) присоединить объект `Series` к объекту `Chart`;
5) дополнительно можно установить значения переменных `drawing.top`, `drawing.left`, `drawing.width` и `drawing.height` объекта `Chart`, определяющих положение и размеры диаграммы;
6) добавить объект `Chart` в объект `Worksheet`.

Следует сделать несколько пояснений относительно объекта `Reference`. Объекты `Reference` создаются путем вызова функции `openpyxl.charts.Reference()`, принимающей три аргумента, которые описаны ниже.

1. Объект `Worksheet`, содержащий данные вашей диаграммы.
2. Кортеж, который состоит из двух целых чисел, представляющих верхнюю левую ячейку выделенной прямоугольной области ячеек, в которых содержатся данные вашей диаграммы: первое целое число задает строку, второе — столбец. Обратите внимание на то, что первой строке соответствует `1`, а не `0`.
3. Кортеж, который состоит из двух целых чисел, представляющих нижнюю правую ячейку выделенной прямоугольной области ячеек, в которых содержатся данные вашей диаграммы: первое целое число задает строку, второе — столбец.

Некоторые иллюстративные примеры задания координат с помощью аргументов приведены на рис. 12.9.

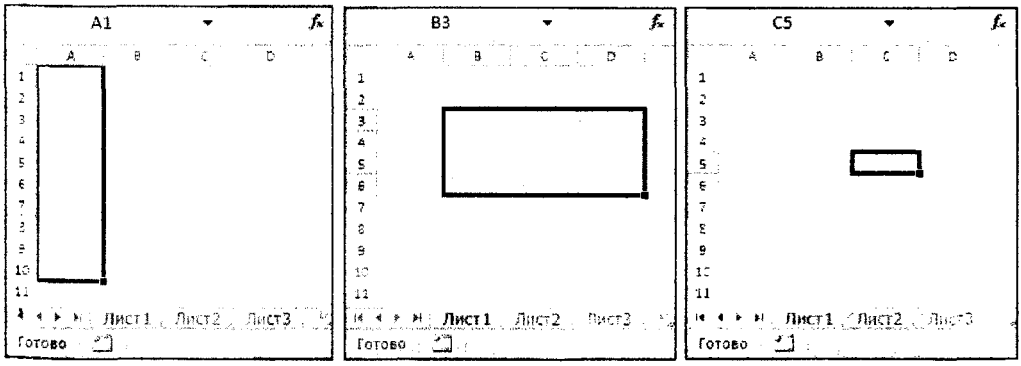


Рис. 12.9. Слева направо: (1, 1), (10, 1); (3, 2), (6, 4); (5, 3), (5, 3)

Создайте гистограмму и добавьте ее в электронную таблицу, введя в интерактивной оболочке следующие команды.

```
>>> from openpyxl import Workbook
>>> from openpyxl.chart import BarChart, Reference
>>> wb = Workbook()
>>> sheet = wb.active
>>> sheet['A1'] = "Серия 1"
>>> for i in range(1,11):
>>>     sheet.append([i])
>>> chart = BarChart()
>>> chart.title = "Первая серия данных"
>>> data = Reference(sheet, min_col=1, min_row=1,
↪ max_col=1, max_row=11)
>>> chart.add_data(data, titles_from_data=True)
>>> sheet.add_chart(chart, "C2")
>>> wb.save("sampleChart1.xlsx")
```

Вид полученной электронной таблицы представлен на рис. 12.10.

Рис. 12.10. Электронная таблица с добавленной диаграммой

Мы создали гистограмму с помощью метода `openpyxl.chart.BarChart()`. Точно так можно создавать графики, а также точечные и круговые диаграммы, вызывая методы `openpyxl.chart.LineChart()`, `openpyxl.chart.ScatterChart()` и `openpyxl.chart.PieChart()`.

К сожалению, в текущей версии модуля OpenPyXL (2.3.3) функция `load_workbook()` не позволяет загружать диаграммы, хранящиеся в файлах Excel. Даже если в файле Excel содержатся диаграммы, загруженный объект `Workbook` не будет их включать. Если вы загрузите объект `Workbook` и сразу же сохраните его в том же *.xlsx*-файле, то это приведет к удалению из него диаграмм.

Резюме

Зачастую наиболее сложной частью обработки информации является не сам процесс обработки, а получение данных в формате, который подходит для вашей программы. Но как только ваша электронная таблица будет загружена в Python, вы сможете извлекать данные и манипулировать ими гораздо быстрее, чем если бы делали это вручную.

Кроме того, электронные таблицы могут генерироваться вашей программой в качестве выходных данных. Поэтому, если вашим коллегам понадобится, чтобы ваш текстовый файл (или PDF-документ), содержащий данные о тысячах операций по продажам, был преобразован в файл электронной таблицы, вам не придется кропотливо копировать и переносить их вручную в Excel.

Теперь, когда вы имеете в своем распоряжении модуль `openpyxl` и обладаете определенными навыками программирования, обработка даже очень больших электронных таблиц покажется вам сущим пустяком.

Контрольные вопросы

Отвечая на приведенные ниже вопросы, представьте, что у вас имеются следующие объекты, хранящиеся в указанных переменных: объект `Workbook` — переменная `wb`, объект `Worksheet` — переменная `sheet`, объект `Cell` — переменная `cell`, объект `Comment` — переменная `comm` и объект `Image` — переменная `img`.

1. Что возвращает функция `openpyxl.load_workbook()`?
2. Что возвращает метод `get_sheet_names()` объекта `Workbook`?
3. Как получить объект `Worksheet` для рабочего листа с именем `'Sheet1'`?
4. Как получить объект `Worksheet` для активного листа рабочей книги?
5. Как получить значение ячейки C5?
6. Как установить значение `"Hello"` для ячейки C5?
7. Как получить номера строки и столбца ячейки в виде целых чисел?
8. Что возвращают методы `get_highest_column()` и `get_highest_row()` рабочего листа и к какому типу данных относятся эти возвращаемые значения?
9. Какую функцию следует вызвать, чтобы получить целочисленный индекс столбца `'M'`?
10. Какую функцию следует вызвать, чтобы получить строковое имя столбца `14`?
11. Как получить кортеж всех объектов `Cell` для ячеек от A1 до F1?
12. Как сохранить рабочую книгу в файле *example.xlsx*?
13. Как задать формулу в ячейке?
14. Что вы должны сделать в первую очередь, если хотите получить результат вычисления формулы, заданной в ячейке, а не саму формулу?
15. Как изменить значение высоты строки с `5` на `100`?
16. Как скрыть столбец C?
17. Назовите несколько средств, которые модуль OpenPyXL 2.1.4 не загружает из файла электронной таблицы.
18. Что такое закрепленная область?
19. Какие пять функций и методов вы должны вызвать, чтобы создать гистограмму?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Генератор таблиц умножения

Создайте программу `multiplicationTable.py`, которая принимает число N из командной строки и создает таблицу умножения размером N×N в электронной таблице Excel. Например, если запустить программу следующим образом:

```
py multiplicationTable.py 6
```

то она должна создать электронную таблицу, которая выглядит так, как показано на рис. 12.11.

	A	B	C	D	E	F	G
1		**1**	**2**	**3**	**4**	**5**	**6**
2	**1**	1	2	3	4	5	6
3	**2**	2	4	6	8	10	12
4	**3**	3	6	9	12	15	18
5	**4**	4	8	12	16	20	24
6	**5**	5	10	15	20	25	30
7	**6**	6	12	18	24	30	36

Рис. 12.11. Таблица умножения, сгенерированная в электронной таблице

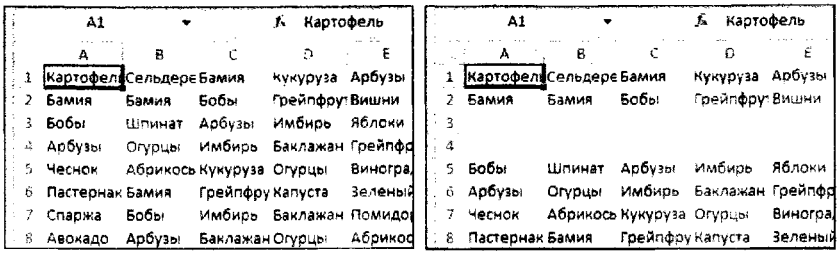
Для разметки таблицы умножения должны использоваться значения в строке 1 и столбце A, отображаемые полужирным шрифтом.

Программа для вставки пустых строк

Создайте программу `blankRowInserter.py`, которая принимает два целых числа и строку с именем файла в качестве аргументов командной строки. Обозначим первое число буквой N, а второе – буквой M. Программа должна вставлять в электронную таблицу M пустых строк, начиная со строки N. Например, если вызвать программу следующим образом:

```
python blankRowInserter.py 3 2 myProduce.xlsx
```

то электронная таблица будет выглядеть до и после работы программы так, как показано на рис. 12.12.



	A	B	C	D	E
1	Картофель	Сельдере	Бамия	Кукуруза	Арбузы
2	Бамия	Бамия	Бобы	Грейпфрут	Вишни
3	Бобы	Шпинат	Арбузы	Имбирь	Яблоки
4	Арбузы	Огурцы	Имбирь	Баклажан	Грейпфр
5	Чеснок	Абрикось	Кукуруза	Огурцы	Виногра
6	Пастернак	Бамия	Грейпфру	Капуста	Зеленый
7	Спаржа	Бобы	Имбирь	Баклажан	Помидо
8	Авокадо	Арбузы	Баклажан	Огурцы	Абрикос

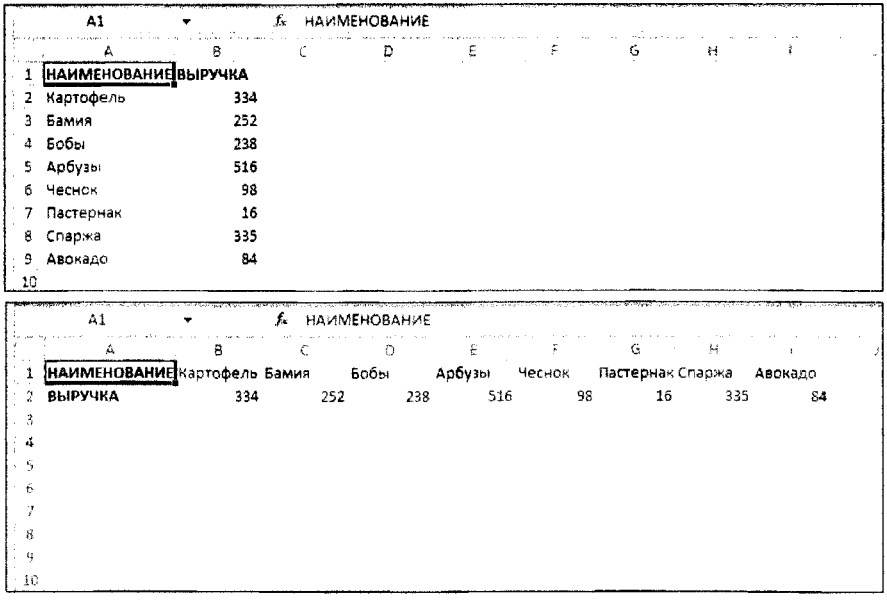
	A	B	C	D	E
1	Картофель	Сельдере	Бамия	Кукуруза	Арбузы
2	Бамия	Бамия	Бобы	Грейпфрут	Вишни
3					
4					
5	Бобы	Шпинат	Арбузы	Имбирь	Яблоки
6	Арбузы	Огурцы	Имбирь	Баклажан	Грейпфр
7	Чеснок	Абрикось	Кукуруза	Огурцы	Виногра
8	Пастернак	Бамия	Грейпфру	Капуста	Зеленый

Рис. 12.12. Вид электронной таблицы до (слева) и после (справа) вставки двух пустых строк в строке 3

Прежде всего ваша программа должна читать содержимое электронной таблицы. Затем в процессе записи новой таблицы программа использует цикл `for` для копирования первых N строк. Для каждой из оставшихся строк программа будет прибавлять M к номеру строки в выходной электронной таблице.

Отражение электронной таблицы относительно диагонали

Напишите программу, меняющую строки и столбцы электронной таблицы местами. Например, она должна переводить значение ячейки из столбца 3 строки в столбец 5 строки 3 (и обратно). Аналогичным образом должны быть инвертированы все ячейки электронной таблицы. Например, электронная таблица до и после такой инверсии будет выглядеть так, как показано на рис. 12.13.



	A	B
1	НАИМЕНОВАНИЕ	ВЫРУЧКА
2	Картофель	334
3	Бамия	252
4	Бобы	238
5	Арбузы	516
6	Чеснок	98
7	Пастернак	16
8	Спаржа	335
9	Авокадо	84
10		

	A	B	C	D	E	F	G	H	I
1	НАИМЕНОВАНИЕ	Картофель	Бамия	Бобы	Арбузы	Чеснок	Пастернак	Спаржа	Авокадо
2	ВЫРУЧКА	334	252	238	516	98	16	335	84
3									
4									
5									
6									
7									
8									
9									
10									

Рис. 12.13. Вид электронной таблицы до (вверху) и после (внизу) инверсии

При написании этой программы вы можете использовать вложенные циклы для чтения данных электронной таблицы и их переноса в структуру данных типа списка списков. Для хранения содержимого ячейки из столбца y строки x электронной таблицы вы можете использовать элемент `sheetData[x][y]` этой структуры данных. Затем, когда вы будете записывать данные в электронную таблицу, используйте для ячейки в столбце x строки y значение элемента `sheetData[y][x]`.

Преобразование текстовых файлов в электронную таблицу

Напишите программу, которая читала бы содержимое нескольких текстовых файлов (возможно, созданных вами) и вставляла его в электронную таблицу по одной текстовой строке в одну строку таблицы. Строки первого текстового файла будут заполнять ячейки столбца A, второго — ячейки столбца B и т.д.

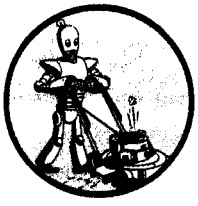
Используйте метод `readlines()` объекта `File` для возврата списка строк, по одной строке на одну текстовую строку файла. В случае первого файла выведите первую текстовую строку в столбец 1 строки 1. Вторую текстовую строку следует записать в столбец 1 строки 2 и т.д. Содержимое следующего файла, прочитанное с помощью метода `readlines()`, будет записываться в столбец 2, дальнейшего файла – в столбец 3 и т.д.

Преобразование электронной таблицы в текстовые файлы

Напишите программу, которая выполняла бы операции предыдущей программы в обратном порядке. Программа должна открывать электронную таблицу и записывать содержимое ячеек столбца A в один текстовый файл, содержимое ячеек столбца B — в другой текстовый файл и т.д.

13

РАБОТА С ДОКУМЕНТАМИ В ФОРМАТАХ PDF И WORD



Документы в форматах PDF и Word представляют собой бинарные файлы, и поэтому работать с ними гораздо сложнее, чем с простыми текстовыми файлами. Помимо текста в них содержится масса дополнительной информации, описывающей шрифты, используемые цвета и компоновку документа. Если вы хотите, чтобы ваши программы читали и записывали PDF- или Word-документы, вам потребуется нечто большее, нежели просто передача имен файлов функции `open()`.

К счастью, в Python предусмотрены модули, упрощающие взаимодействие с документами в форматах PDF и Word. В этой главе будут рассмотрены два таких модуля: `PyPDF2` и `Python-Docx`.

PDF-документы

PDF — это сокращенное название формата *Portable Document Format* (формат переносимых документов), для которого используются файлы с расширением *.pdf*. Несмотря на то что формат PDF поддерживает множество возможностей, в этой главе мы сосредоточимся на двух из них, с которыми вы будете чаще всего иметь дело: чтение текстового содержимого PDF-файлов и создание новых PDF-документов на основе существующих.

Для работы с PDF-документами вы будете использовать модуль PyPDF2. Чтобы установить его, выполните команду `pip install PyPDF2` в командной строке. Указанное здесь имя модуля чувствительно к регистру, поэтому лишь буква y должна вводиться в нижнем регистре; все остальные буквы в нем вводятся в верхнем регистре. (Более подробно процедура установки модулей, разработанных третьими сторонами, описана в приложении A.) Признаком того, что модуль установлен корректно, является отсутствие сообщений об ошибке при выполнении команды `import PyPDF2` в интерактивной оболочке.

Проблематичность формата PDF

PDF-файлы являются замечательным средством для передачи документов в электронном виде, которые пользователям будет легко читать и выводить на печать, однако выполнить их синтаксический анализ с целью преобразования в простой текст с помощью других программ — задача нетривиальная. А раз так, модуль PyPDF2 может вносить ошибки при извлечении текста из PDF-файла или вообще оказаться не в состоянии открыть некоторые документы в PDF-формате. К сожалению, с этим почти ничего нельзя поделать. В подобных ситуациях модуль PyPDF2 просто не сможет работать с некоторыми из ваших PDF-файлов. Тем не менее за свою практику я еще ни разу не сталкивался с тем, чтобы мне не удалось открыть PDF-файл с помощью модуля PyPDF2.

Извлечение текста из PDF-файлов

Модуль PyPDF2 не располагает возможностями извлечения изображений, диаграмм и других мультимедийных данных из PDF-документов, но способен извлекать текст и возвращать его в виде строки Python. Приступая к изучению работы модуля PyPDF2, мы применим его к примеру PDF-документа, представленному на рис. 13.1.

Загрузите этот PDF-документ с сайта `http://nostarch.com/automatestuff/` и введите в интерактивной оболочке следующие команды.

```
>>> import PyPDF2
>>> pdfFileObj = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
❶ >>> pdfReader.numPages
19
❷ >>> pageObj = pdfReader.getPage(0)
❸ >>> pageObj.extractText()
'OOFFFFIICCIIAALL BBOOAARRDD MMIINNUUTTEESS Meeting of March 7,
2015 \n      The Board of Elementary and Secondary Education
shall provide leadership and create policies for education that
expand opportunities for children, empower families and
communities, and advance Louisiana in an increasingly
competitive global market. BOARD of ELEMENTARY and SECONDARY
EDUCATION '
```

Прежде всего, импортируйте модуль PyPDF2. Затем откройте файл *meetingminutes.pdf* в режиме чтения двоичных данных и сохраните его в переменной `pdfFileObj`. Получите объект `PdfFileReader`, представляющий этот PDF-документ, вызовите метод `PyPDF2.PdfFileReader()` и передайте ему объект `pdfFileObj`. Сохраните этот объект `PdfFileReader` в переменной `pdfReader`.

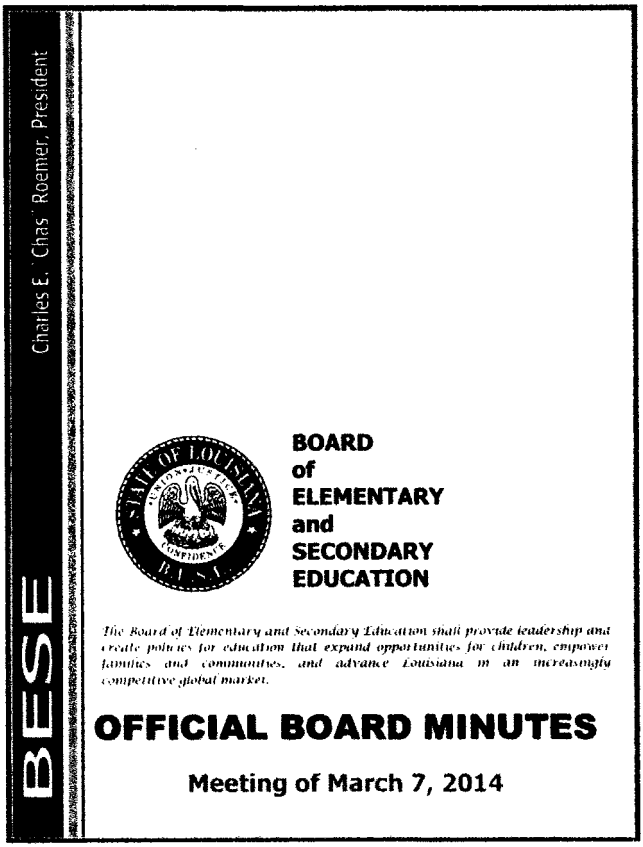


Рис. 13.1. Страница PDF-документа, из которой мы будем извлекать текст

Полное количество страниц в этом документе хранится в атрибуте `numPages` объекта `PdfFileReader`❶. Документ из этого примера содержит 19 страниц, но мы извлечем из него только текст первой страницы.

Чтобы извлечь текст страницы, необходимо получить объект `Page`, который представляет одиночную PDF-страницу из объекта `PdfFileReader`. Объект `Page` можно получить, вызвав метод `getPage()` ❷ для объекта `PdfFileReader` с передачей ему номера интересующей вас страницы, в данном случае — `0`.

В модуле PyPDF2 используется индексация страниц *с отсчетом от нуля*: первая страница — это страница 0, вторая — страница 1 и т.д. Такой порядок нумерации соблюдается всегда, даже в тех случаях, когда нумерация страниц в документе иная. Например, предположим, что ваш документ PDF представляет собой трехстраничную выдержку из длинного отчета и его страницы имеют номера 42, 43 и 44. Чтобы получить первую страницу этого документа, следует делать вызов `pdfReader.getPage(0)`, а не `getPage(42)` или `getPage(1)`.

Получив объект `Page`, вызовите его метод `extractText()`, который возвратит строку, содержащую текст страницы ❸. Извлечение текста не проходит идеально: PDF-текст *Charles E. "Chas" Roemer, President* выпал из строки, возвращенной методом `extractText()`, а в некоторых местах нарушена разрядка. И все же даже это неидеальное текстовое содержимое может быть достаточно подходящим для вашей программы.

Дешифрование PDF-документов

Некоторые PDF-документы могут быть зашифрованы, что препятствует их прочтению теми, кто будет открывать документ, не предоставляя пароль. Введите в интерактивной оболочке следующие команды, пытаясь открыть загруженный вами PDF-документ, который зашифрован паролем *rosebud*.

```
>>> import PyPDF2
>>> pdfReader = PyPDF2.PdfFileReader(open('encrypted.pdf',
                                          'rb'))
❶ >>> pdfReader.isEncrypted
True
>>> pdfReader.getPage(0)
❷ Traceback (most recent call last):
  File "<pyshell#173>", line 1, in <module>
    pdfReader.getPage()
  --пропущенный код--
  File "C:\Python34\lib\site-packages\PyPDF2\pdf.py", line 1173,
      in getObject
    raise utils.PdfReadError("file has not been decrypted")
PyPDF2.utils.PdfReadError: file has not been decrypted
❸ >>> pdfReader.decrypt('rosebud')
1
>>> pageObj = pdfReader.getPage(0)
```

У всех объектов `PdfFileReader` есть атрибут `isEncrypted`, который имеет значение `True`, если PDF-документ зашифрован, и `False` — в противном случае ❶. Любая попытка вызвать функцию, пытающуюся прочесть файл, прежде чем он будет дешифрован с использованием правильного пароля, вызовет ошибку ❷.

Чтобы прочитать зашифрованный PDF-документ, вызовите функцию `decrypt()`, передав ей пароль в виде строки ❸. Вызвав функцию `decrypt()` с правильным паролем, вы увидите, что вызов метода `getPage()` больше не сопровождается выводом сообщения об ошибке. В случае предоставления неверного пароля функция `decrypt()` возвратит `0`, и метод `getPage()` не сможет быть выполнен. Обратите внимание на то, что метод `decrypt()` расшифровывает объект `PdfFileReader`, но не фактический PDF-файл. После того как ваша программа завершит работу, файл на вашем жестком диске

останется зашифрованным. При следующем запуске ваша программа должна будет снова вызвать функцию `decrypt()`.

Создание PDF-документов

В модуле PyPDF2 объекты `PdfFileReader` дополняются объектами `PdfFileWriter`, которые могут создавать новые PDF-файлы. Но модуль PyPDF2 не поддерживает запись произвольного текста в формат PDF, как это делает Python с простыми текстовыми файлами. Возможности PyPDF2 в отношении записи PDF-документов ограничены копированием страниц из других PDF-документов, поворотом и наложением страниц, а также шифрованием файлов.

Модуль PyPDF2 не обеспечивает непосредственное редактирование PDF-документов. Вместо этого нужно создать новый PDF-файл, а затем скопировать содержимое из существующего документа. Выполняя примеры в этом разделе, мы будем действовать в соответствии со следующей общей процедурой.

1) открыть один или несколько существующих PDF-файлов (исходных PDF-документов) в объектах `PdfFileReader`;
2) создать новый объект `PdfFileWriter`;
3) скопировать страницы из объектов `PdfFileReader` в объект `PdfFileWriter`;
4) использовать объект `PdfFileWriter` для записи выходного PDF-документа.

Создавая объект `PdfFileWriter`, вы лишь создаете значение, которое представляет PDF-документ в Python. При этом фактический PDF-файл не создается. Для этого необходимо вызвать метод `write()` объекта `PdfFileWriter`.

Метод `write()` принимает в качестве аргумента обычный объект `File`, открытый в режиме записи двоичных данных. Такой объект можно получить, вызвав функцию Python `open()` с двумя аргументами: строкой, содержащей желаемое имя PDF-файла, и строкой `'wb'`, указывающей на то, что файл должен быть открыт в режиме записи двоичных данных.

Если не все из сказанного вам понятно, не огорчайтесь: вы увидите, как все это работает, в приведенных далее примерах кода.

Копирование страниц

Модуль PyPDF2 можно использовать для копирования страниц из одного PDF-документа в другой. Это позволит вам объединять несколько PDF-файлов, переупорядочивать страницы или вырезать ненужные.

Загрузите файлы *meetingminutes.pdf* и *meetingminutes2.pdf* с сайта `http://nostarch.com/automatestuff/` и поместите их в текущий рабочий каталог. Введите в интерактивной оболочке следующие команды.

```
   >>> import PyPDF2
   >>> pdf1File = open('meetingminutes.pdf', 'rb')
   >>> pdf2File = open('meetingminutes2.pdf', 'rb')
❶  >>> pdf1Reader = PyPDF2.PdfFileReader(pdf1File)
❷  >>> pdf2Reader = PyPDF2.PdfFileReader(pdf2File)
❸  >>> pdfWriter = PyPDF2.PdfFileWriter()

   >>> for pageNum in range(pdf1Reader.numPages):
❹      pageObj = pdf1Reader.getPage(pageNum)
❺      pdfWriter.addPage(pageObj)

   >>> for pageNum in range(pdf2Reader.numPages):
❹      pageObj = pdf2Reader.getPage(pageNum)
❺      pdfWriter.addPage(pageObj)

❻  >>> pdfOutputFile = open('combinedminutes.pdf', 'wb')
   >>> pdfWriter.write(pdfOutputFile)
   >>> pdfOutputFile.close()
   >>> pdf1File.close()
   >>> pdf2File.close()
```

Откройте оба PDF-файла в режиме чтения двоичных данных и сохраните результирующие объекты `File` в переменных `pdf1File` и `pdf2File`. Получите объект `PdfFileReader` для файла *meetingminutes.pdf*, вызвав функцию `PyPDF2.PdfFileReader()` и передав ей переменную `pdf1File` ❶. Вызовите эту же функцию повторно с передачей ей переменной `pdf2File`, чтобы получить объект `PdfFileReader` для файла *meetingminutes2.pdf* ❷. Затем создайте новый объект `PdfFileWriter`, представляющий пустой PDF-документ ❸.

Далее скопируйте все страницы из двух исходных PDF-файлов и добавьте их в объект `PdfFileWriter`. Получите объект `Page`, вызвав метод `getPage()` для объекта `PdfFileReader` ❹. После этого передайте этот объект `Page` методу `addPage()` своего объекта `PdfFileWriter` ❺. Эти действия выполняются сначала для объекта `pdf1Reader`, а затем для объекта `pdf2Reader`. Закончив с копированием страниц, запишите новый PDF-документ *combinedminutes.pdf*, передав объект `File` методу `write()` объекта `PdfFileWriter` ❻.

Примечание

Модуль PyPDF2 не обеспечивает вставку страниц посередине документа, представляемого объектом `PdfFileWriter`; метод `addPage()` способен лишь присоединять страницы в конце документа.

Вы создали новый PDF-файл, объединяющий страницы из файлов *meetingminutes.pdf* и *meetingminutes2.pdf* в один документ. Не забывайте о том, что объект `File`, передаваемый функции `PyPDF2.PdfFileReader()`, должен быть открыт в режиме чтения двоичных данных путем передачи строки `'rb'` в качестве второго аргумента функции `open()`. Аналогичным образом объект `File`, передаваемый функции `PyPDF2.PdfFileWriter()`, должен быть открыт в режиме записи двоичных данных с помощью строки `'wb'`.

Поворот страниц

Страницы PDF-документа можно поворачивать на углы, кратные 90°, в направлении по часовой стрелке и против часовой стрелки с помощью методов `rotateClockwise()` и `rotateCounterClockwise()` соответственно. В качестве аргументов эти методы принимают целые числа 90, 180 и 270. Убедившись в наличии файла *meetingminutes.pdf* в текущем рабочем каталоге, введите в интерактивной оболочке следующие команды.

```
>>> import PyPDF2
>>> minutesFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
❶ >>> page = pdfReader.getPage(0)
❷ >>> page.rotateClockwise(90)
{'/Contents': [IndirectObject(961, 0), IndirectObject(962, 0),
--пропущенный код--
}
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> pdfWriter.addPage(page)
❸ >>> resultPdfFile = open('rotatedPage.pdf', 'wb')
>>> pdfWriter.write(resultPdfFile)
>>> resultPdfFile.close()
>>> minutesFile.close()
```

Здесь мы вызываем метод `getPage(0)` для выбора первой страницы PDF-документа ❶, а затем поворачиваем эту страницу на 90° по часовой стрелке с помощью вызова `rotateClockwise(90)` ❷. Мы записываем повернутую страницу в новый PDF-документ и сохраняем его в файле *rotatedPage.pdf* ❸.

Результирующий документ будет содержать одну исходную страницу, повернутую на 90° по часовой стрелке (рис. 13.2). Значения, возвращаемые методами `rotateClockwise()` и `rotateCounterClockwise()`, содержат массу дополнительной информации, которую можно игнорировать.

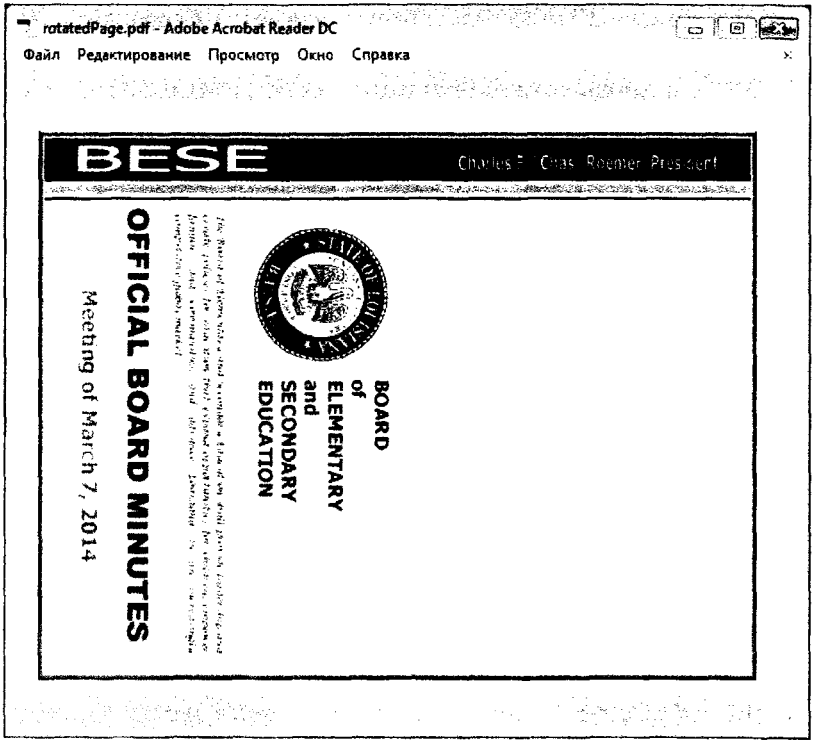


Рис. 13.2. Сохраненная в файле rotatedPage.pdf страница, повернутая на 90° по часовой стрелке

Наложение страниц

Модуль PyPDF2 также предоставляет возможность наложения содержимого одной страницы поверх содержимого другой, что может быть использовано для добавления в документ логотипа, временной метки или водяного знака. С помощью Python можно легко добавить водяные знаки в несколько файлов и только на те страницы, которые будут указаны в программе.

Загрузите файл *watermark.pdf* на сайте `http://nostarch.com/automatestuff/` и поместите этот PDF-документ в текущий рабочий каталог вместе с файлом *meetingminutes.pdf*. После этого введите в интерактивной оболочке следующие команды.

```
>>> import PyPDF2
>>> minutesFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(minutesFile)
>>> minutesFirstPage = pdfReader.getPage(0)
>>> pdfWatermarkReader =
        PyPDF2.PdfFileReader(open('watermark.pdf', 'rb'))
>>> minutesFirstPage.mergePage(pdfWatermarkReader.getPage(0))
>>> pdfWriter = PyPDF2.PdfFileWriter()
```

```
❻ >>> pdfWriter.addPage(minutesFirstPage)

❼ >>> for pageNum in range(1, pdfReader.numPages):
          pageObj = pdfReader.getPage(pageNum)
          pdfWriter.addPage(pageObj)

>>> resultPdfFile = open('watermarkedCover.pdf', 'wb')
>>> pdfWriter.write(resultPdfFile)
>>> minutesFile.close()
>>> resultPdfFile.close()
```

Здесь мы создаем объект `PdfFileReader` на основе файла *meetingminutes.pdf* ❶. Чтобы получить объект `Page` для первой страницы и сохранить его в переменной `minutesFirstPage`, мы вызываем метод `getPage(0)` ❷. Затем создается объект `PdfFileReader` для файла *watermark.pdf* ❸ и вызывается метод `mergePage()` для объекта, сохраненного в переменной `minutesFirstPage` ❹. Аргументом, который мы передаем методу `mergePage()`, является объект `Page` для первой страницы файла *watermark.pdf*.

Теперь, когда для переменной `minutesFirstPage` был выполнен вызов метода `mergePage()`, она представляет первую страницу файла с добавленным водяным знаком. В следующей строке мы создаем объект `PdfFileWriter` ❺ и добавляем в него эту страницу ❻. Затем мы выполняем цикл по оставшимся страницам файла *meetingminutes.pdf* и добавляем их в объект `PdfFileWriter` ❼. Наконец, мы открываем новый PDF-файл *watermarkedCover.pdf* и записываем в него содержимое объекта `PdfFileWriter`.

Результат представлен на рис. 13.3. В нашем новом PDF-документе *watermarkedCover.pdf* представлено все содержимое документа *meetingminutes.pdf* с первой страницей, помеченной водяным знаком.

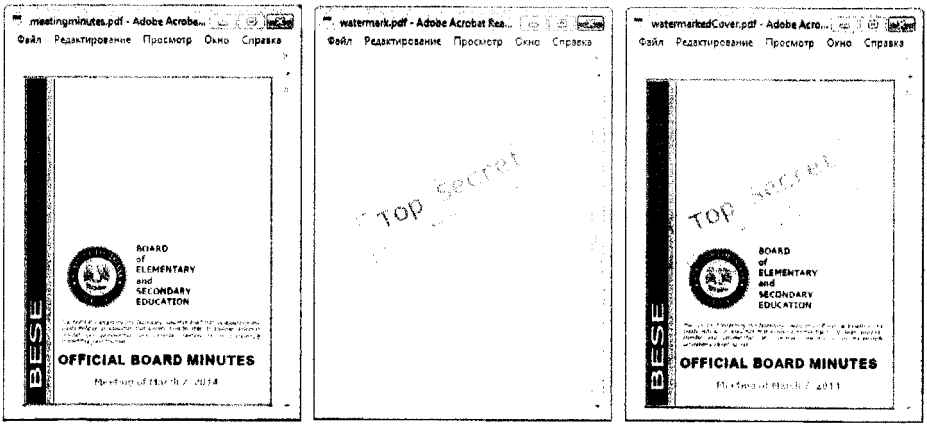


Рис. 13.3. Оригинальный PDF-документ (слева), водяной знак (в центре) и объединенный PDF-документ (справа)

Шифрование PDF-документов

Объект `PdfFileWriter` также позволяет шифровать PDF-документы. Введите в интерактивной оболочке следующие команды.

```
>>> import PyPDF2
>>> pdfFile = open('meetingminutes.pdf', 'rb')
>>> pdfReader = PyPDF2.PdfFileReader(pdfFile)
>>> pdfWriter = PyPDF2.PdfFileWriter()
>>> for pageNum in range(pdfReader.numPages):
        pdfWriter.addPage(pdfReader.getPage(pageNum))

❶ >>> pdfWriter.encrypt('swordfish')
>>> resultPdf = open('encryptedminutes.pdf', 'wb')
>>> pdfWriter.write(resultPdf)
>>> resultPdf.close()
```

Прежде чем вызывать метод `write()` для сохранения файла, вызовите метод `encrypt()` и передайте ему строку пароля ❶. PDF-документы могут иметь *пароль пользователя* (позволяющий просматривать документ) и *пароль владельца* (позволяющий устанавливать разрешения для вывода документа на печать, снабжения его комментариями и извлечения текста, а также предоставляющий другие права). Пароли пользователя и владельца задаются соответственно в качестве первого и второго аргументов метода `encrypt()`. Если передать методу `encrypt()` только одну строку, то она будет использована для обоих паролей.

В этом примере мы скопировали страницы документа *meetingminutes.pdf* в объект `PdfFileWriter`. Далее мы зашифровали объект `PdfFileWriter` с помощью пароля *swordfish*, открыли новый PDF-файл *encryptedminutes.pdf* и записали в него содержимое объекта `PdfFileWriter`. Прежде чем кто-либо сможет просмотреть содержимое файла *encryptedminutes.pdf*, он должен будет ввести этот пароль. После того как вы убедитесь в том, что копия зашифрована корректно, оригинальный незашифрованный файл *meetingminutes.pdf* при желании можно будет удалить.

Проект: объединение выбранных страниц из многих PDF-документов

Предположим, вам предстоит выполнить утомительную работу по слиянию нескольких PDF-документов в один PDF-файл. Каждый из документов начинается с титульного листа на первой странице, но вы не хотите, чтобы первые страницы попадали в окончательный документ. Несмотря на то что существует много бесплатных программ, позволяющих объединять PDF-документы, все, на что способны многие из них, — это просто слияние

исходных файлов в единый файл. Давайте напишем программу, позволяющую выбирать страницы, которые должны включаться в результирующий PDF-документ.

На верхнем уровне планирования программа должна выполнять следующие действия:

- находить все PDF-файлы в текущем рабочем каталоге;
- сортировать файлы по именам, чтобы файлы добавлялись в определенном порядке;
- записывать каждую страницу исходных PDF-файлов, за исключением их первых страниц, в выходной файл;

В терминах реализации ваш код должен выполнять следующие операции:

- вызывать функцию `os.listdir()` для нахождения всех файлов в текущем рабочем каталоге и удалять все файлы, кроме файлов в формате PDF;
- вызывать метод Python `sort()` для списка с целью расположения имен файлов в алфавитном порядке;
- создавать объект `PdfFileWriter` для выходного PDF-файла;
- организовывать цикл по всем PDF-файлам с созданием объекта `PdfFileReader` для каждого из них;
- организовывать цикл по всем страницам (за исключением первой) каждого PDF-файла;
- добавлять страницы в выходной PDF-файл;
- записывать выходной PDF-файл в файл *allminutes.pdf*.

Откройте для этого проекта новое окно в файловом редакторе и сохраните его в файле *combinePdfs.py*.

Шаг 1. Поиск всех PDF-файлов

Прежде всего ваша программа должна получить список всех файлов с расширением *.pdf* в текущем рабочем каталоге и выполнить их сортировку. Введите в файл следующий код.

```
#! python3
# combinePdfs.py - Объединяет все PDF-документы, находящиеся
# в текущем рабочем каталоге, в единый PDF-документ.

❶ import PyPDF2, os

# Получение списка имен всех PDF-файлов.
pdfFiles = []
for filename in os.listdir('.'):
    if filename.endswith('.pdf'):
❷       pdfFiles.append(filename)
```

```
❸ pdfFiles.sort(key=str.lower)

❹ pdfWriter = PyPDF2.PdfFileWriter()

   # TODO: Организовать цикл по всем PDF-файлам.

   # TODO: Организовать цикл по всем страницам (за исключением
   # первой) с их добавлением в результирующий документ.

   # TODO: Сохранить результирующий PDF-документ в файле.
```

Вслед за "магической строкой" и комментарием, описывающим назначение программы, в этом коде импортируются модули `os` и `PyPDF2` ❶. Вызов `os.listdir('.')` возвращает список всех файлов, находящихся в текущем рабочем каталоге. Затем код просматривает этот список в цикле и добавляет в новый список `pdfFiles` лишь файлы с расширением *.pdf* ❷. После этого список `pdfFiles` сортируется в алфавитном порядке, который задается именованным аргументом `key=str.lower` при вызове метода `sort()` ❸.

Для хранения объединенных PDF-страниц создается объект `PdfFileWriter` ❹. Наконец, несколькими комментариями описывается остальная часть программы.

Шаг 2. Открытие PDF-файлов

Теперь программа должна прочитать каждый из PDF-файлов, входящих в список `pdfFiles`. Добавьте в программу код, выделенный ниже полужирным шрифтом.

```
#! python3
# combinePdfs.py - Объединяет все PDF-документы, находящиеся
# в текущем рабочем каталоге, в единый PDF-документ.

import PyPDF2, os

# Получение списка имен всех PDF-файлов.
pdfFiles = []
--пропущенный код--

# Организация цикла по всем PDF-файлам.
for filename in pdfFiles:
    pdfFileObj = open(filename, 'rb')
    pdfReader = PyPDF2.PdfFileReader(pdfFileObj)
    # TODO: Организовать цикл по всем страницам (за исключением
    # первой) с их добавлением в результирующий документ.

# TODO: Сохранить результирующий PDF-документ в файле.
```

В цикле каждый PDF-файл открывается в режиме чтения двоичных данных путем вызова метода `open()` с передачей ему строки `'rb'` в качестве второго аргумента. Метод `open()` возвращает объект `File`, который передается функции `PyPDF2.PdfFileReader()`, создающей объект `PdfFileReader` для данного PDF-файла.

Шаг 3. Добавление страниц

Для каждого из PDF-файлов необходимо отобрать в цикле все страницы, за исключением первой. Добавьте в программу код, выделенный ниже полужирным шрифтом.

```
#! python3
# combinePdfs.py - Объединяет все PDF-документы, находящиеся
# в текущем рабочем каталоге, в единый PDF-документ.

  import PyPDF2, os

--пропущенный код--

  # Организация цикла по всем PDF-файлам.
  for filename in pdfFiles:
  --пропущенный код--
      # Организация цикла по всем страницам (за исключением
      # первой) с их добавлением в результирующий документ.
❶     for pageNum in range(1, pdfReader.numPages):
          pageObj = pdfReader.getPage(pageNum)
          pdfWriter.addPage(pageObj)

# TODO: Сохранить результирующий PDF-документ в файле.
```

Код в цикле копирует каждый объект `Page` по отдельности в объект `PdfFileWriter`. Вспомните, что вы хотите опускать первую страницу. Поскольку в принятой в модуле PyPDF2 системе отсчета первой странице соответствует индекс `0`, цикл должен выполняться в диапазоне индексов, определяемом значениями `1` и `pdfReader.numPages` (последнее значение не включается в диапазон) ❶.

Шаг 4. Сохранение результатов

После выполнения вложенных циклов `for` переменная `pdfWriter` будет содержать объект `PdfFileWriter`, включающий страницы всех объединенных PDF-документов. Последний шаг состоит в том, чтобы записать это содержимое в файл на жестком диске. Добавьте в программу код, выделенный ниже полужирным шрифтом.

```
#! python3
# combinePdfs.py - Объединяет все PDF-документы, находящиеся
  # в текущем рабочем каталоге, в единый PDF-документ.

import PyPDF2, os

--пропущенный код--

# Организация цикла по всем PDF-файлам.
for filename in pdfFiles:
--пропущенный код--
    # Организация цикла по всем страницам (за исключением
    # первой) с их добавлением в результирующий документ.
    for pageNum in range(1, pdfReader.numPages):
    --пропущенный код--

# Сохранение результирующего PDF-документа в файле.
pdfOutput = open('allminutes.pdf', 'wb')
pdfWriter.write(pdfOutput)
pdfOutput.close()
```

Чтобы открыть выходной файл *allminutes.pdf* в режиме записи двоичных данных, функции `open()` передается строка `'wb'`. В результате последующей передачи результирующего объекта `File` методу `write()` создается фактический PDF-файл. Программу завершает вызов метода `close()`.

Идеи относительно создания аналогичных программ

Возможность создавать PDF-документы на основе страниц, извлекаемых из других PDF-документов, позволяет создавать программы для решения следующих задач:

- удаление указанных страниц из PDF-документов;
- реорганизация страниц в PDF-документах;
- создание PDF-документа на основе только тех страниц, которые содержат заданный текст, определяемый вызовом метода `extractText()`.

Документы Word

С помощью модуля Python `python-docx` можно создавать и изменять документы Word с расширением имени файла `.docx`. Чтобы установить этот модуль, следует выполнить команду `pip install python-docx`. (За более подробным описанием процедуры установки модулей, разработанных третьими сторонами, обратитесь к приложению А.)

Примечание

Используя команду `pip` *для установки модуля Python-Docx, обязательно вводите* `install python-docx`*, а не* `docx`*. Имя* `docx` *используется при установке другого модуля, который в данной книге не рассматривается. В то же время при импортировании модуля* `python-docx` *в программу следует использовать команду* `import docx`*, а не* `import python-docx`*.*

В случае отсутствия приложения Microsoft Word можете воспользоваться его бесплатными альтернативами LibreOffice Writer и OpenOffice Writer для операционных систем Windows, OS X и Linux, которые позволяют открывать *.docx*-файлы. Они доступны для загрузки на сайтах `https://www.libreoffice.org` и `http://openoffice.org` соответственно. Полная документация по модулю Python-Docx доступна по адресу `https://python-docx.readthedocs.org/`. Хотя и существует версия Word для OS X, в этой главе мы сосредоточимся на работе с версией Word для Windows.

В отличие от простых текстовых файлов, файлы с расширением *.docx* обладают развитой внутренней структурой. В модуле Python-Docx эта структура представлена тремя различными типами данных. На самом верхнем уровне объект `Document` представляет весь документ. Объект `Document` содержит список объектов `Paragraph`, которые представляют абзацы документа. (Новый абзац начинается всякий раз, когда пользователь нажимает клавишу <Enter> или <Return> при вводе текста в документ Word.) Каждый из абзацев содержит список, состоящий из одного или нескольких объектов `Run`, представляющих фрагменты текста с различными стилями форматирования. Представленный на рис. 13.4 одиночный абзац состоит из четырех таких фрагментов.

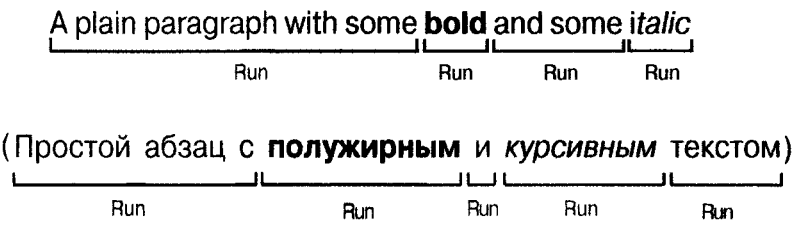


Рис. 13.4. Объекты Run, определенные в объекте `Paragraph`

Текст в документах Word — это нечто большее, чем просто текстовая строка. Он включает информацию, описывающую шрифт, цвет и размер шрифта, а также другую относящуюся к тексту информацию. Коллекция этих атрибутов образует *стиль* документа Word. Объект `Run` представляет непрерывный фрагмент текста, оформленный с использованием одного и того же стиля. Каждой смене стиля соответствует новый объект `Run`.

Чтение документов Word

Давайте поэкспериментируем с модулем python-docx. Загрузите файл *demo.docx* на сайте http://nostarch.com/automatestuff/ и сохраните документ в рабочем каталоге. После этого введите в интерактивной оболочке следующие команды.

```
  >>> import docx
❶ >>> doc = docx.Document('demo.docx')
❷ >>> len(doc.paragraphs)
  7
❸ >>> doc.paragraphs[0].text
  'Название документа'
❹ >>> doc.paragraphs[1].text
  'A plain paragraph with some bold and some italic'
❺ >>> len(doc.paragraphs[1].runs)
  4
❻ >>> doc.paragraphs[1].runs[0].text
  'A plain paragraph with some '
❼ >>> doc.paragraphs[1].runs[1].text
  'bold'
❽ >>> doc.paragraphs[1].runs[2].text
  ' and some '
❾ >>> doc.paragraphs[1].runs[3].text
  'italic'
```

Этими командами мы открываем *.docx*-файл в Python, вызываем функцию docx.Document() и передаем ей имя файла *demo.docx* ❶. Эта функция возвращает объект Document, атрибут paragraphs которого представляет собой список объектов Paragraph. Возвращаемое в результате вызова метода len() для списка doc.paragraphs ❷ значение 7 указывает на то, что в этом документе содержится семь объектов Paragraph. Каждый из этих объектов Paragraph имеет атрибут text, содержащий строку текста данного абзаца (без информации о стиле). В данном случае первый атрибут text содержит строку 'Название документа' ❸, а второй — строку 'A plain paragraph with some bold and some italic' ❹.

Кроме того, каждый объект Paragraph имеет атрибут runs, который представляет собой список объектов Run. Объекты Run также имеют атрибут text, который содержит лишь текст данного участка стиля форматирования. Обратимся к атрибутам text второго объекта Paragraph с текстом 'A plain paragraph with some bold and some italic'. Результат вызова метода len() для этого объекта говорит о том, что этот объект включает четыре объекта Run ❺. Тексту 'A plain paragraph with some ' соответствует первый объект Run ❻. Затем к тексту применяется полужирное начертание, и поэтому тексту 'bold' соответствует новый объект Run ❼. Далее следует текст с обычным форматированием ' and some ', которому соответствует третий

объект Run ❽. Четвертый объект Run содержит текст 'italic', к которому применено курсивное начертание ❾.

Имея в своем распоряжении модуль Python-Docx, ваши программы на Python смогут читать текст из файлов с расширением *.docx* и использовать его как любое другое текстовое значение.

Получение полного текста из файла .docx

Если в документе Word вас интересует только текст без информации о стиле форматирования, можете воспользоваться функцией getText(). Она принимает имя *.docx*-файла в качестве аргумента и возвращает строку, содержащую полный текст файла. Откройте новое окно в файловом редакторе, введите в него следующий код и сохраните программу в файле *readDocx.py*.

```
#! python3

import docx

def getText(filename):
    doc = docx.Document(filename)
    fullText = []
    for para in doc.paragraphs:
        fullText.append(para.text)
    return '\n'.join(fullText)
```

Функция getText() открывает документ Word, просматривает в цикле все объекты Paragraph в списке paragraphs, присоединяя содержащийся в них текст к списку fullText. По завершении выполнения цикла все строки, содержащиеся в списке fullText, объединяются с использованием символа новой строки в качестве разделителя.

Программу *readDocx.py* можно импортировать подобно любому другому модулю. Теперь, если все, что вам нужно, — это извлечь текст из документа Word, введите следующие команды.

```
>>> import readDocx
>>> print(readDocx.getText('demo.docx'))
Название документа
A plain paragraph with some bold and some italic
Заголовок, уровень 1
Выделенная цитата
Первый элемент маркированного списка
Первый элемент нумерованного списка
```

Вы также можете изменить строку, прежде чем возвратить ее. Например, чтобы выделить каждый абзац отступом, замените вызов append() в программе *readDocx.py* следующим его вариантом:

```
fullText.append(' ' + para.text)
```

Чтобы добавить удвоенный междустрочный интервал между абзацами, замените вызов `join()` следующим:

```
return '\n\n'.join(fullText)
```

Как видите, для написания функций, обеспечивающих чтение файла *.docx* и возвращение строки с его содержимым, видоизмененным в соответствии с вашими потребностями, достаточно всего нескольких строк кода.

Стилевое оформление абзаца и объекты Run

В Word для Windows стили можно увидеть, нажав комбинацию клавиш <Ctrl+Alt+Shift+S> для отображения окна Стили (рис. 13.5). В OS X для этого следует выбрать пункты меню View⇨Styles (Вид⇨Стили).

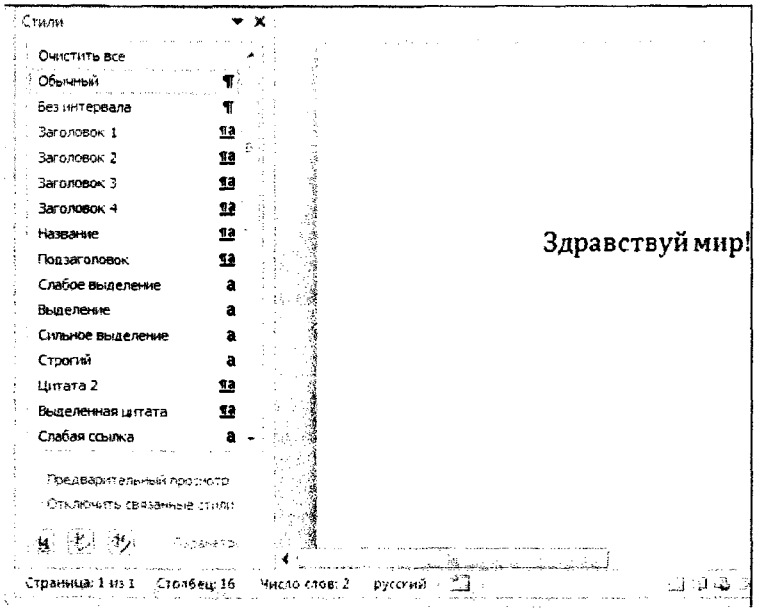


Рис. 13.5. Вид окна Стили *в Windows*

Стили используются в Word и других текстовых процессорах для придания документу или ряду однотипных документов единообразного внешнего вида, который к тому же можно легко изменить. Например, вы хотите, чтобы для абзацев использовался шрифт Times New Roman размером 11 пунктов, а текст выравнивался по левому краю без выключки справа. Можно создать стиль с этими параметрами и назначить его всем абзацам тела

документа. Впоследствии, если вы захотите изменить внешний вид абзацев по всему документу, достаточно будет изменить лишь стиль, и все абзацы автоматически изменят свой внешний вид.

В документах Word применяются три типа стилей: *стили абзацев*, которые могут применяться к объектам `Paragraph`, *стили символов*, которые могут применяться к объектам `Run`, и *связанные стили*, которые могут применяться к обоим типам объектов. Как объектам `Paragraph`, так и объектам `Run` можно назначать стили, присваивая их атрибутам `style` значение в виде строки. Этой строкой должно быть имя стиля. Если для стиля задано значение `None`, то у объекта `Paragraph` или `Run` не будет связанного с ним стиля

Ниже приведены имена стилей Word, используемых по умолчанию.

```
'Normal'        'Heading5'        'ListBullet'       'ListParagraph'
'BodyText'      'Heading6'        'ListBullet2'      'MacroText'
'BodyText2'     'Heading7'        'ListBullet3'      'NoSpacing'
'BodyText3'     'Heading8'        'ListContinue'     'Quote'
'Caption'       'Heading9'        'ListContinue2'    'Subtitle'
'Heading1'      'IntenseQuote'    'ListContinue3'    'TOCHeading'
'Heading2'      'List'            'ListNumber'       'Title'
'Heading3'      'List2'           'ListNumber2'
'Heading4'      'List3'           'ListNumber3'
```

Определяя значения атрибута `style`, не оставляйте пробелы в имени стиля. Например, если стиль называется "Subtle Emphasis", то в качестве значения атрибута `style` укажите строку `'SubtleEmphasis'`, а не `'Subtle Emphasis'`. Включение пробелов приведет к тому, что имя стиля не будет распознано Word, и он не будет применен.

Если связанный стиль применяется к объекту `Run`, то к его имени следует присоединять строку `'Char'`. Например, устанавливая для объекта `Paragraph` связанный стиль `Quote`, используйте инструкцию `paragraphObj.style = 'Quote'`, однако в случае объекта `Run` следует использовать инструкцию `runObj.style = 'QuoteChar'`.

В текущей версии модуля Python-Docx (0.7.4) единственные стили, которые можно использовать, — это заданные по умолчанию стили Word и стили, используемые в открытом *.docx*-документе. Возможность создания новых стилей в настоящее время отсутствует, хотя в будущих версиях модуля Python-Docx ситуация может измениться.

Создание документов Word с нестандартными стилями

Если вы хотите создавать документы Word, в которых используются стили, не являющиеся стандартными, прежде всего откройте в Word пустой документ и создайте стили самостоятельно, щелкнув на кнопке **Создать стиль** в нижней части окна `Стили` (на рис. 13.6 представлен фрагмент окна Word для Windows).

В результате откроется диалоговое окно Создание стиля, в котором можно настроить новый стиль. После этого вернитесь в интерактивную оболочку, откройте этот документ с помощью функции docx.Document() и используйте его в качестве основы для своего документа Word. Теперь имя, которое вы присвоили этому стилю, можно будет использовать вместе с модулем Python-Docx.

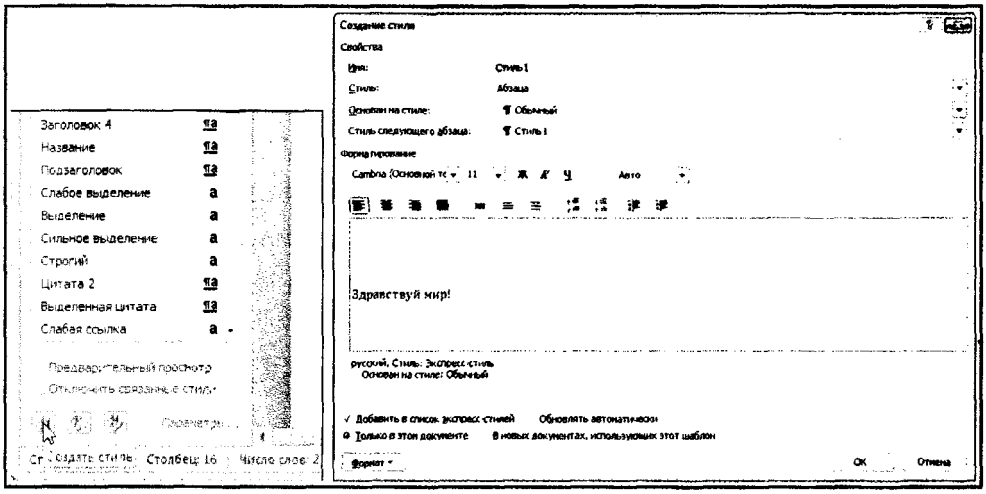


Рис. 13.6. Кнопка Создать стиль (слева) и диалоговое окно Создание стиля (справа)

Атрибуты объекта Run

Отдельные фрагменты текста, представляемые объектами Run, могут подвергаться дополнительному форматированию с помощью атрибутов text. Для каждого из этих атрибутов может быть задано одно из трех значений: True (атрибут постоянно активизирован, независимо от применения к данному фрагменту текста других стилей), False (атрибут постоянно отключен) и None (по умолчанию применяется стиль, установленный для данного объекта Run).

Атрибуты text, которые могут назначаться объектам Run, приведены в табл. 13.1.

Таблица 13.1. Атрибуты text объекта Run

Атрибут	Описание
bold	Полужирное начертание
italic	Курсивное начертание
underline	Подчеркнутый текст
strike	Зачеркнутый текст

Окончание табл. 13.1

Атрибут	Описание
double_strike	Текст, зачеркнутый двойной линией
all_caps	Отображение текста прописными буквами
small_caps	Отображение текста малыми прописными буквами (капитель), размер которых на два пункта больше размера строчных букв
shadow	Текст, отбрасывающий тени
outline	Контурный текст
rtl	Направление текста справа налево
imprint	Вдавленный текст
emboss	Рельефный текст

Поэкспериментируйте с изменением стилей форматирования текста из файла *demo.docx*, введя в интерактивной оболочке следующие команды.

```
>>> doc = docx.Document('demo.docx')
>>> doc.paragraphs[0].text
'Название документа'
>>> doc.paragraphs[0].style
'Title'
>>> doc.paragraphs[0].style = 'Normal'
>>> doc.paragraphs[1].text
'A plain paragraph with some bold and some italic'
>>> (doc.paragraphs[1].runs[0].text, doc.paragraphs[1].runs[1].text, doc.
paragraphs[1].runs[2].text, doc.paragraphs[1].runs[3].text)
 'A plain paragraph with some ', 'bold', ' and some ', 'italic')
>>> doc.paragraphs[1].runs[0].style = 'QuoteChar'
>>> doc.paragraphs[1].runs[1].underline = True
>>> doc.paragraphs[1].runs[3].underline = True
>>> doc.save('restyled.docx')
```

В этом примере показано, как просто исследовать содержимое абзацев документа с помощью атрибутов `text` и `style`. Вы видите, как легко можно разделить абзац на фрагменты текста и получать доступ к ним по отдельности. В данном случае мы извлекаем первый, второй и четвертый фрагменты второго абзаца, применяем к ним разные стили и сохраняем результат в новом документе.

К словам *Document Title* в начале документа *restyled.docx* вместо стиля Title применен стиль Normal, к объекту `Run` для текста *A plain paragraph with some* применен стиль QuoteChar, а атрибутам `underline` двух объектов `Run` для слов *bold* и *italic* присвоено значение `True`. Результат форматирования абзацев и отдельных фрагментов текста документа *restyled.docx* с помощью стилей показан на рис. 13.7.

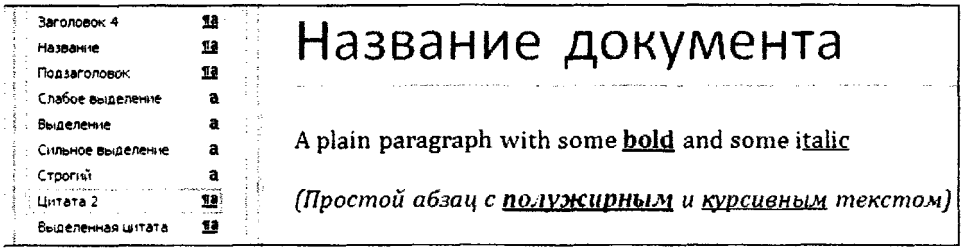


Рис. 13.7. Документ restyled.docx

Более подробную информацию относительно применения стилей форматирования в документах Word с помощью модуля Python-Docx можно найти по адресу `https://python-docx.readthedocs.org/en/latest/user/styles.html`.

Запись документов Word

Введите в интерактивной оболочке следующий код.

```
>>> import docx
>>> doc = docx.Document()
>>> doc.add_paragraph('Hello world!')
<docx.text.Paragraph object at 0x0000000003B56F60>
>>> doc.save('helloworld.docx')
```

Создайте собственный *.docx*-файл, вызвав функцию `docx.Document()`, которая возвратит новый, пустой объект `Document`. Метод `add_paragraph()` этого объекта добавляет новый абзац текста в документ и возвращает ссылку на добавленный объект `Paragraph`. Завершив добавление текста, сохраните документ в файле, передав строку с именем файла методу `save()` объекта `Document`.

В данном примере в текущем рабочем каталоге создается файл `helloworld.docx`, результат открытия которого в приложении Word показан на рис. 13.8.

Добавление абзацев осуществляется путем повторных вызовов метода `add_paragraph()` с текстом новых абзацев. Кроме того, можно добавить текст в конце существующего абзаца, вызвав для него метод `add_run()` с передачей ему строки текста. Введите в интерактивной оболочке следующие команды.

```
>>> import docx
>>> doc = docx.Document()
>>> doc.add_paragraph('Hello world!')
<docx.text.Paragraph object at 0x000000000366AD30>
>>> paraObj1 = doc.add_paragraph('Это второй абзац.')
>>> paraObj2 = doc.add_paragraph('Это еще один абзац.')
```

```
>>> paraObj1.add_run(' Этот текст был добавлен во второй абзац.')
<docx.text.Run object at 0x0000000003A2C860>
>>> doc.save('multipleParagraphs.docx')
```

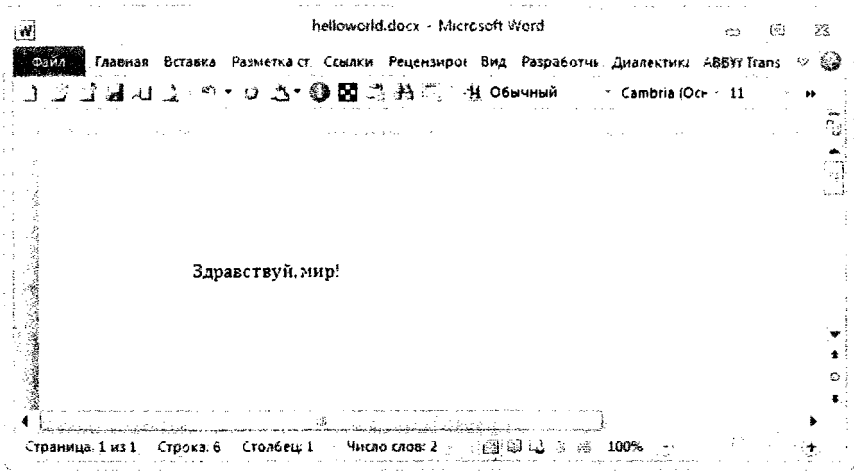


Рис. 13.8. Документ Word, созданный с использованием вызова `add_paragraph('Здравствуй, мир!')`

Результат показан на рис. 13.9. Обратите внимание на то, что текст `Этот текст был добавлен во второй абзац.` был добавлен в объект `Paragraph` (переменная `paraObj1`), представляющий второй из добавленных в документ абзацев. Функции `add_paragraph()` и `add_run()` возвращают объекты `Paragraph` и `Run` соответственно, что избавляет вас от лишних забот по извлечению необходимых фрагментов текста в различных ситуациях.

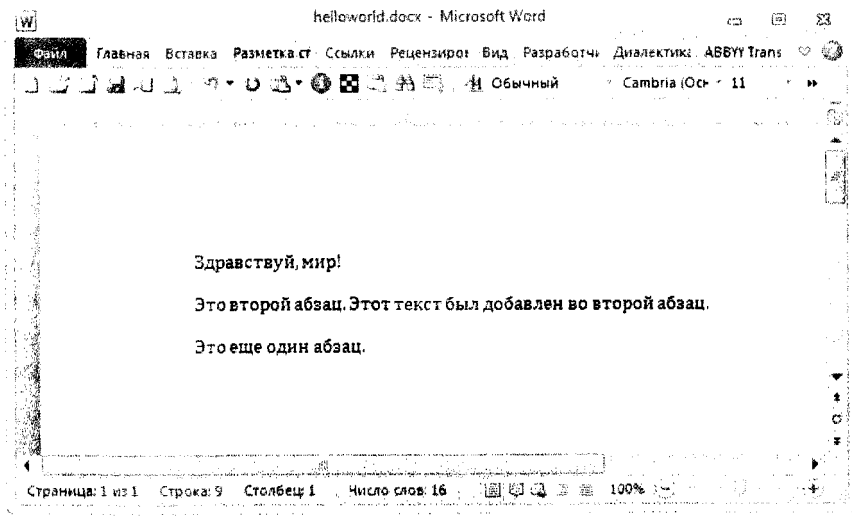


Рис. 13.9. Документ с несколькими добавленными объектами `Paragraph` *и* `Run`

Имейте в виду, что начиная с версии Python-Docx 0.5.3 новые объекты `Paragraph` можно добавлять только в конце документа, а новые объекты `Run` — только в конце абзаца.

Метод `save()` можно вызывать повторно для сохранения дополнительных изменений.

Оба метода, `add_paragraph()` и `add_run()`, принимают необязательный второй аргумент, содержащий строку стиля объекта `Paragraph` или `Run` соответственно, например:

```
>>> doc.add_paragraph('Здравствуй, мир!', 'Title')
```

Эта строка кода добавляет абзац с текстом `Здравствуй мир!`, отформатированным с использованием стиля `Title` (Название).

Добавление заголовков

Вызов метода `add_heading()` приводит к добавлению абзаца, отформатированного в соответствии с одним из возможных стилей заголовков. Введите в интерактивной оболочке следующие команды.

```
>>> doc = docx.Document()
>>> doc.add_heading('Header 0', 0)
<docx.text.Paragraph object at 0x00000000036CB3C8>
>>> doc.add_heading('Header 1', 1)
<docx.text.Paragraph object at 0x00000000036CB630>
>>> doc.add_heading('Header 2', 2)
<docx.text.Paragraph object at 0x00000000036CB828>
>>> doc.add_heading('Header 3', 3)
<docx.text.Paragraph object at 0x00000000036CB2E8>
>>> doc.add_heading('Header 4', 4)
<docx.text.Paragraph object at 0x00000000036CB3C8>
>>> doc.save('headings.docx')
```

Аргументами метода `add_heading()` являются строка текста и целое число, которое может иметь значение от `0` до `4`. Значению `0` соответствует стиль заголовка `Title` (Название), используемый в начале документа в качестве заголовка верхнего уровня. Целым числам от `1` до `4` соответствуют различные уровни заголовков, наивысшему из которых соответствует значение `1`, а наинизшему — `4`. Функция `add_heading()` возвращает объект `Paragraph`, избавляя вас от необходимости извлечения абзаца из документа отдельной операцией.

Вид результирующего документа *headings.docx* показан на рис. 13.10.

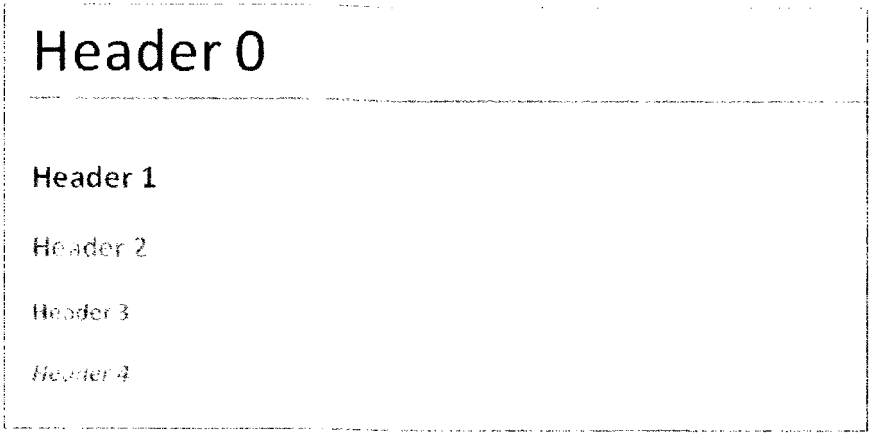


Рис. 13.10. Документ headings.docx с заголовками, соответствующими уровням от 0 до 4

Добавление разрывов строк и страниц

Чтобы добавить разрыв строки (а не начинать новый абзац), можно вызвать метод `add_break()` для того объекта `Run`, после которого вы хотите вставить разрыв строки. Если же требуется добавить разрыв страницы, то функции `add_break()` следует передать значение `docx.text.WD_BREAK.PAGE` в качестве единственного аргумента, как это сделано в следующем примере.

```
>>> doc = docx.Document()
>>> doc.add_paragraph('This is on the first page!')
<docx.text.Paragraph object at 0x0000000003785518>
❶ >>> doc.paragraphs[0].runs[0].add_break(docx.text.WD_BREAK.PAGE)
>>> doc.add_paragraph('This is on the second page!')
<docx.text.Paragraph object at 0x00000000037855F8>
>>> doc.save('twoPage.docx')
```

В результате создается двухстраничный документ Word с текстом `This is on the first page!` на первой странице и текстом `This is on the second page!` — на второй. Несмотря на то что на первой странице за текстом `This is on the first page!` остается еще много свободного места, мы принудительно начинаем следующий абзац на новой странице, вставляя разрыв страницы после первого объекта `Run` первого абзаца ❶.

Добавление изображений

Метод `add_picture()` объекта `Document` позволяет добавлять изображения в конце документа. Предположим, в вашем текущем рабочем каталоге находится файл `zophie.png`. Чтобы добавить в конце документа изображение *zophie.png* шириной 1 дюйм и высотой 4 сантиметра (Word распознает как

метрические, так и неметрические единицы измерения), введите следующие команды.

```
>>> doc.add_picture('zophie.png', width=docx.shared.Inches(1),
height=docx.shared.Cm(4))
<docx.shape.InlineShape object at 0x00000000036C7D30>
```

Первый аргумент — это строка, содержащая имя файла изображения. Необязательные именованные аргументы `width` и `height` задают ширину и высоту изображения в документе. Если их опустить, то значения этих аргументов по умолчанию будут определяться размерами самого изображения.

Вероятно, вы захотите указывать высоту и ширину изображения в привычных вам единицах — дюймах или сантиметрах, которые можно конкретизировать, используя функции `docx.shared.Inches()` и `docx.shared.Cm()` при указании значений именованных аргументов `width` и `height`.

Резюме

Текстовая информация — это не только простые текстовые файлы. В действительности вы, скорее всего, в большинстве случаев работаете с документами в форматах PDF и Word. Для чтения PDF-документов можно использовать модуль PyPDF2. К сожалению, при чтении текста из PDF-документов его преобразование в строку не всегда выполняется идеально в силу сложности файлового формата PDF, а некоторые PDF-документы прочесть вообще невозможно. Считайте, что в подобных случаях вам просто не повезло, и остается лишь надеяться, что в будущих версиях модуля PyPDF2 будет обеспечена дополнительная поддержка этих возможностей.

В этом смысле документы Word более надежны, и их можно читать с помощью модуля `python-docx`. Для манипулирования текстом в документах Word используют объекты `Paragraph` и `Run`. Им можно назначать стили форматирования, хотя возможности выбора стилей ограничиваются лишь стандартным набором стилей, а также стилями, уже существующими в документе. Разрешается добавлять новые абзацы, заголовки, разрывы строк и страниц, а также изображения, но только в конце документа.

Многие ограничения при работе с документами в форматах PDF и Word обусловлены тем, что эти форматы предназначены в первую очередь для того, чтобы документы было удобно читать людям, и они не ориентированы на анализ текста программными средствами. В следующей главе обсуждаются два других распространенных формата, используемых для хранения информации: JSON и CSV. Они предназначены для компьютерной обработки, и вы увидите, что работать с этими форматами в Python гораздо легче.

Контрольные вопросы

1. Функции `PdfFileReader()` модуля PyPDF2 не передается строковое значение имени PDF-файла. Что же в таком случае ей передается?
2. В каких режимах должны открываться объекты `File` для функций `PdfFileReader()` и `PdfFileWriter()`?
3. Как получить объект `Page` для страницы 5 из объекта `PdfFileReader`?
4. В какой переменной объекта `PdfFileReader` хранится количество страниц документа PDF?
5. Если PDF-документ объекта `PdfFileReader` зашифрован с использованием пароля *swordfish*, то что вы должны сделать, прежде чем сможете получить из него объекты `Page`?
6. Какие методы используются для поворота страницы?
7. Какой метод возвратит объект `Document` для файла *demo.docx*?
8. В чем суть различия между объектами `Paragraph` и `Run`?
9. Как получить список объектов `Paragraph` для объекта `Document`, который хранится в переменной `doc`?
10. Какой тип объектов имеет переменные `bold`, `underline`, `italic`, `strike` и `outline`?
11. Что соответствует значениям `True`, `False` и `None`, присваиваемым переменной bold?
12. Как создать объект `Document` для нового документа Word?
13. Как добавить абзац с текстом `'Hello there!'` в объект `Document`, хранящийся в переменной `doc`?
14. Какие целочисленные значения представляют уровни заголовков, доступные в документах Word?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

PDF-паранойя

Используя функцию `os.walk()` из главы 9, напишите сценарий, который выбирает все PDF-файлы в папке (и всех ее подпапках) и зашифровывает их с использованием пароля, предоставленного в командной строке. Сохраните каждый зашифрованный PDF-файл, добавляя к исходному имени файла суффикс `_encrypted.pdf`. Прежде чем удалять исходный файл, попытайтесь прочитать и дешифровать результирующий файл, чтобы убедиться в том, что файл был корректно зашифрован.

Затем напишите программу, которая находит все зашифрованные PDF-файлы в папке (и всех ее подпапках) и создает дешифрованную копию каждого из них, используя предоставленный пароль. В случае ввода неправильного пароля программа должна выводить предупреждающее сообщение для пользователя и переходить к обработке следующего файла.

Персонализированные приглашения в виде документов Word

Предположим, у вас есть текстовый файл *guests.txt* со списком гостей. Каждое имя в этом файле записано в отдельной строке.

```
Prof. Plum
Miss Scarlet
Col. Mustard
Al Sweigart
RoboCop
```

Напишите программу, которая генерирует документ Word, содержащий персонализированные приглашения (рис. 13.11).

Поскольку модуль Python-Docx может использовать лишь те стили, которые уже существуют в документе Word, вам придется сначала добавить эти стили в пустой файл Word, а затем открыть этот файл с помощью модуля Python-Docx. Результирующий документ Word должен содержать по одному приглашению на одной странице, поэтому вызывайте метод `add_break()` для добавления разрывов страниц за последним абзацем каждого приглашения. Благодаря этому, чтобы напечатать сразу все приглашения, вам нужно будет открыть только один документ Word.

Готовый образец файла *guests.txt* можно загрузить на сайте `http://nostarch.com/automatestuff/`.

Взлом паролей PDF методом грубой силы

Предположим, у вас имеется зашифрованный PDF-файл, пароль доступа к которому вы забыли, но помните, что им является какое-то слово на английском языке. Угадывание забытого пароля — довольно утомительная задача. Вместо этого вы можете написать программу, которая дешифрует PDF-файл, перебирая все возможные слова до тех пор, пока не будет найдено слово, совпадающее с паролем. Такой подход к взлому паролей носит название *атака методом грубой силы*. Загрузите текстовый файл *dictionary.txt* на сайте `http://nostarch.com/automatestuff/`. В этом файле словаря содержится свыше 44 тысяч английских слов, по одному слову в каждой строке.

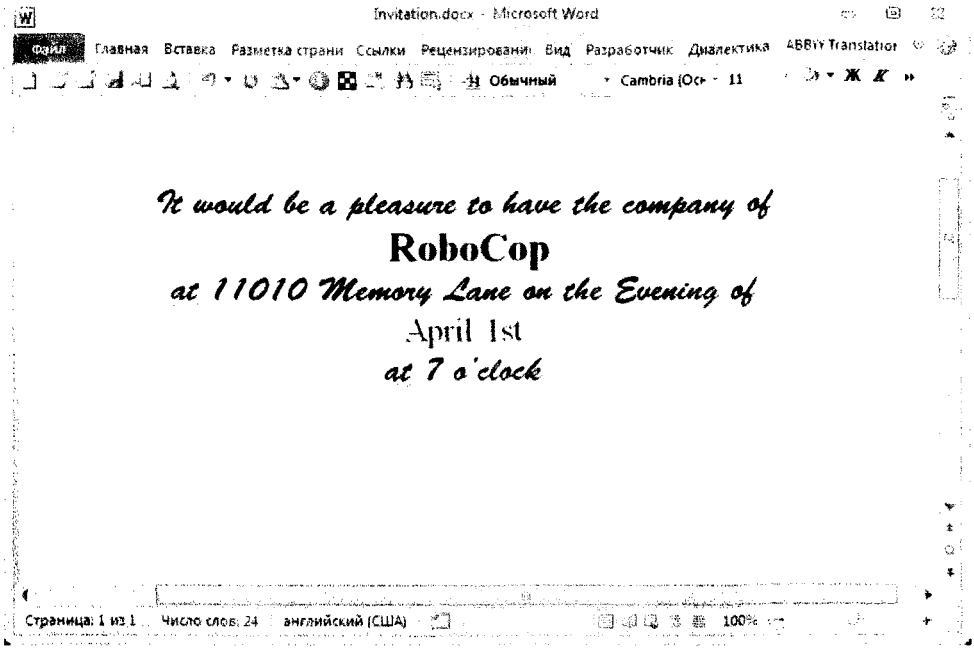
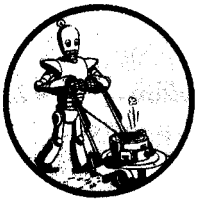


Рис. 13.11. Документ Word, полученный с помощью сценария, генерирующего персонализированные приглашения

Используя свои навыки в чтении файлов, приобретенные в главе 8, создайте список слов, прочитав содержимое этого файла. Затем организуйте цикл, в котором из словаря поочередно берутся слова, которые далее передаются методу `decrypt()`. Если метод возвращает `0`, значит, данное слово не является паролем, и программа должна переходить к следующему слову. Если же метод `decrypt()` возвращает значение `1`, то ваша программа должна выйти из цикла и вывести взломанное значение пароля. Каждое слово из словаря следует испытывать в верхнем и нижнем регистрах. (На моем ноутбуке перебор всех 88 тысяч вариантов пароля в верхнем и нижнем регистрах отнимает около двух минут. Вот почему нельзя использовать в качестве паролей простые слова.)

14

РАБОТА С CSV-ФАЙЛАМИ И ДАННЫМИ В ФОРМАТЕ JSON



В главе 13 вы научились извлекать текст из документов Word и PDF. Документы этого типа хранятся в файлах, записанных с использованием двоичного формата, и для доступа к содержащимся в них данным приходится привлекать специальные модули Python . С другой стороны, для записи CSV- и JSON-файлов используется простой текстовый формат. Их содержимое можно просматривать в любом текстовом редакторе, в том числе в файловом редакторе IDLE. Тем не менее для этих файлов также предусмотрены специальные модули `csv` и `json`, каждый из которых предоставляет функции, упрощающие работу с этими файловыми форматами.

CSV (Comma-Separated Values – значения, разделенные запятыми) – текстовый формат, ориентированный на работу с данными несложных электронных таблиц, хранящихся в обычных текстовых файлах. Модуль Python `csv` упрощает синтаксический анализ (парсинг) CSV-файлов.

JSON (JavaScript Object Notation; произносится как *джей-сон*, причем не важно, на каком именно слоге вы сделаете ударение, поскольку всегда найдутся люди, которые скажут, что ваш вариант произношения неправильный) – это формат, используемый в языке программирования JavaScript для хранения информации в обычных текстовых файлах. Для применения формата JSON не нужно знать JavaScript, и знакомство с этим форматом будет для вас полезным, так как он используется во многих веб-приложениях.

Модуль `csv`

Каждая строка CSV-файла представляет одну строку электронной таблицы, значения отдельных столбцов которой разделены запятыми. Например, электронная таблица, хранящаяся в файле *example.xlsx* на сайте `http://nostarch.com/automatestuff/`, будет иметь в формате CSV следующий вид.

```
05.04.2015 13:34,Яблоки,73
05.04.2015 3:41,Вишни,85
06.04.2015 12:46,Груши,14
08.04.2015 8:59,Апельсины,52
10.04.2015 2:07,Яблоки,152
10.04.2015 18:10,Бананы,23
10.04.2015 2:40,Земляника,98
```

Эта таблица будет использоваться в данной главе для выполнения примеров в интерактивной оболочке. Вы можете либо загрузить готовый файл *example.csv* на сайте `http://nostarch.com/automatestuff/`, либо самостоятельно ввести текст в каком-либо текстовом редакторе и сохранить его в файле *example.csv*.

С CSV-файлами легко работать, однако они не обеспечивают некоторые возможности, доступные при работе с электронными таблицами Excel:

- отсутствуют типы значений — все значения являются строками;
- отсутствует возможность настройки размера и цвета шрифта;
- отсутствует возможность использования нескольких рабочих листов;
- отсутствует возможность задания ширины и высоты ячеек таблицы;
- отсутствует возможность объединения ячеек;
- отсутствует возможность внедрения изображений и диаграмм.

Достоинством CSV-файлов является их простота. CSV-файлы широко поддерживаются многими типами программ, их можно просматривать в текстовых редакторах (включая файловый редактор IDLE) и они представляют непосредственно табличные данные. Формат CSV является в точности тем, о чем говорит его название: это обычный текстовый файл, в котором значения разделены запятыми.

Поскольку CSV-файлы — это всего лишь текстовые файлы, у вас может возникнуть соблазн читать их содержимое в виде строки, а затем обрабатывать полученную строку, используя методики, которые вы изучили в главе 8. Например, поскольку столбцы данных разделены в CSV-файле запятыми, вы можете пытаться извлекать значения, содержащиеся в каждой строке, с помощью метода `split()`. Однако не всякая запятая в CSV-файле представляет границу между двумя столбцами. Кроме того, CSV-файлы могут иметь собственные экранированные символы, позволяющие включать сами запятые в значения. Такие экранированные символы методом `split()` не обрабатываются. С учетом этих потенциальных ловушек лучше всего всегда читать и записывать CSV-файлы с помощью модуля `csv`.

Объекты Reader

Чтобы прочитать данные из CSV-файла, необходимо создать объект `Reader`. Объект `Reader` обеспечивает возможность итерирования по строкам CSV-файла. Убедившись в том, что в текущем рабочем каталоге присутствует файл *example.csv*, введите в интерактивной оболочке следующие команды.

```
❶ >>> import csv
❷ >>> exampleFile = open('example.csv')
❸ >>> exampleReader = csv.reader(exampleFile)
❹ >>> exampleData = list(exampleReader)
❺ >>> exampleData
[['05.04.2015 13:34', 'Яблоки', '73'], ['05.04.2015 3:41',
'Вишни', '85'],
['06.04.2015 12:46', 'Груши', '14'], ['08.04.2015 8:59',
'Апельсины', '52'], ['10.04.2015 2:07', 'Яблоки', '152'],
['10.04.2015 18:10', 'Бананы', '23'], ['10.04.2015 2:40',
'Земляника', '98']]
```

Модуль `csv` поставляется вместе с Python, поэтому мы можем просто импортировать его ❶ без предварительной установки.

Прежде чем читать CSV-файл с помощью модуля `csv`, откройте его с помощью функции `open()` ❷, как вы это делаете при открытии любого текстового файла. Но вместо того чтобы вызывать метод `read()` или `readlines()` для объекта `File`, возвращаемого функцией `open()`, передайте этот объект функции `csv.reader()` ❸. Эта функция возвращает объект `Reader`, который вы будете использовать в дальнейшем. Обратите внимание на то, что функции `csv.reader()` передается объект, а не просто строка с именем файла.

Самый прямой способ получения доступа к значениям в объекте `Reader` — передача этого объекта функции `list()` ❹ для преобразования в обычный список Python. В результате мы получаем список списков, который можно сохранить в переменной `exampleData`. Введя имя `exampleData` в интерактивной оболочке, можно увидеть этот список ❺.

Теперь, когда у вас есть CSV-файл в виде списка списков, можете обращаться к значениям, относящимся к отдельным строкам и столбцам таблицы, с помощью выражения `exampleData[row][col]`, где `row` — индекс одного из списков в объекте `exampleData`, а `col` — индекс нужного элемента из этого списка. Введите в интерактивной оболочке следующие команды.

```
>>> exampleData[0][0]
'05.04.2015 13:34'
>>> exampleData[0][1]
'Яблоки'
>>> exampleData[0][2]
```

```
'73'
>>> exampleData[1][1]
'Вишни'
>>> exampleData[6][1]
'Земляника'
```

Выражение `exampleData[0][0]` дает нам первую строку в первом списке, выражение `exampleData[0][2]` — третью строку в этом же списке и т.д.

Чтение данных из объектов Reader в цикле for

В случае крупных CSV-файлов целесообразно использовать объект `Reader` в цикле `for`. Тем самым удается избежать загрузки сразу всего файла в память компьютера. Например, введите в интерактивной оболочке следующие команды.

```
>>> import csv
>>> exampleFile = open('example.csv')
>>> exampleReader = csv.reader(exampleFile)
>>> for row in exampleReader:
        print('Строка #' + str(exampleReader.line_num) + ' ' +
↪ str(row))

Строка #1 ['05.04.2015 13:34', 'Яблоки', '73']
Строка #2 ['05.04.2015 3:41', 'Вишни', '85']
Строка #3 ['06.04.2015 12:46', 'Груши', '14']
Строка #4 ['08.04.2015 8:59', 'Апельсины', '52']
Строка #5 ['10.04.2015 2:07', 'Яблоки', '152']
Строка #6 ['10.04.2015 18:10', 'Бананы', '23']
Строка #7 ['10.04.2015 2:40', 'Земляника', '98']
```

Импортировав модуль `csv` и создав объект `Reader` из CSV-файла, можно организовать цикл по строкам в объекте `Reader`. Каждая строка — это список значений, каждое из которых представляет отдельную ячейку таблицы.

С помощью функции `print()` мы выводим номер строки и ее содержимое. Для получения номера строки используется переменная `line_num` объекта `Reader`, в которой хранится номер текущей строки.

Цикл по объекту `Reader` может выполняться только один раз. Для повторного чтения CSV-файла необходимо заново создать объект `Reader`, вызвав функцию `csv.reader()`.

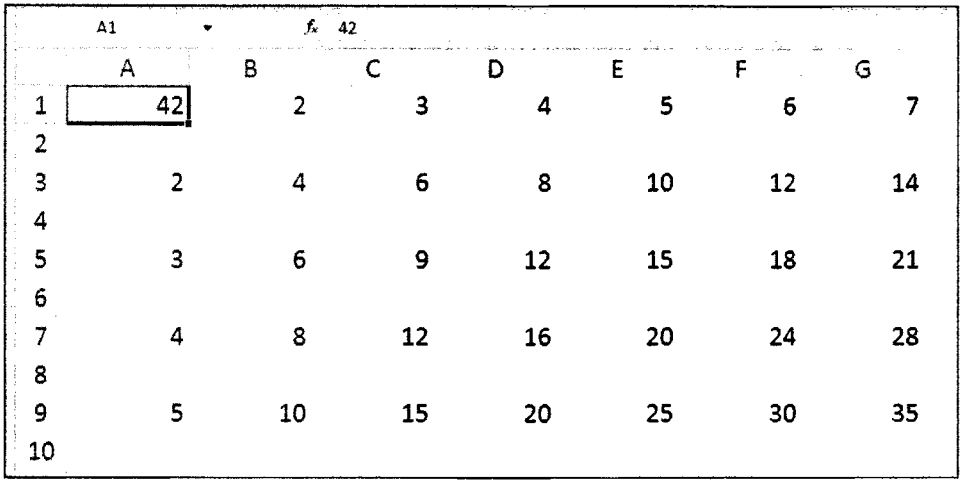
Объекты Writer

Объект `Writer` позволяет записывать данные в CSV-файл. Для создания объекта `Writer` используется функция `csv.writer()`. Введите в интерактивной оболочке следующие команды.

```
>>> import csv
❶ >>> outputFile = open('output.csv', 'w', newline='')
❷ >>> outputWriter = csv.writer(outputFile)
>>> outputWriter.writerow(['spam', 'eggs', 'bacon', 'ham'])
21
>>> outputWriter.writerow(['Здравствуй, мир!', 'eggs', 'bacon',
↳ 'ham'])
32
>>> outputWriter.writerow([1, 2, 3.141592, 4])
16
>>> outputFile.close()
```

Прежде всего необходимо вызвать функцию `open()` и передать ей аргумент `'w'` для открытия файла в режиме записи ❶. В результате этого создается объект, который далее передается функции `csv.writer()` ❷ для создания объекта `Writer`.

Если вы работаете в Windows, то в качестве значения именованного аргумента `newline` функции `open()` следует передавать пустую строку. В силу технических причин, обсуждение которых выходит за рамки данной книги, если этого не сделать, в выводе появятся лишние пустые строки (рис. 14.1).



A1	fx 42						
	A	B	C	D	E	F	G
1	42	2	3	4	5	6	7
2							
3	2	4	6	8	10	12	14
4							
5	3	6	9	12	15	18	21
6							
7	4	8	12	16	20	24	28
8							
9	5	10	15	20	25	30	35
10							

Рис. 14.1. Если вы забудете задать пустую строку в качестве значения именованного аргумента `newline` функции `open()`, то при выводе содержимого CSV-файла появятся лишние строки

Метод `writerow()` объекта `Writer` принимает аргумент в виде списка. Каждое значение этого списка помещается в отдельную ячейку выходного CSV-файла. Возвращаемым значением метода `writerow()` является число символов, записанных в файл для данной строки таблицы (включая символы новой строки).

Вот как выглядит содержимое файла *output.csv*, полученного в результате выполнения предыдущего кода.

```
spam,eggs,bacon,ham
"Здравствуй, мир!",eggs,bacon,ham
1,2,3.141592,4
```

Обратите внимание на то, как объект `Writer` автоматически экранирует кавычками запятую в значении `'Здравствуй, мир!'` в CSV-файле. Модуль `csv` избавляет вас от самостоятельной обработки подобных специальных случаев.

Именованные аргументы `delimiter` и `lineterminator`

Предположим, вы захотели использовать в качестве разделителя ячеек не запятую, а символ табуляции и при этом удвоить междустрочный интервал. Это можно обеспечить, введя в интерактивной оболочке следующий код.

```
>>> import csv
>>> csvFile = open('example.tsv', 'w', newline='')
❶ >>> csvWriter = csv.writer(csvFile, delimiter='\t',
                            lineterminator='\n\n')
>>> csvWriter.writerow(['яблоки', 'апельсины', 'грейпфруты'])
24
>>> csvWriter.writerow(['яйца', 'бекон', 'окорок'])
17
>>> csvWriter.writerow(['spam', 'spam', 'spam', 'spam', 'spam',
                        'spam'])
32
>>> csvFile.close()
```

В результате этого разделитель данных и разделитель строк в вашем файле изменятся. *Разделитель данных* (или просто — *разделитель*) — это символ, используемый для разделения значений в строке. По умолчанию в CSV-файлах в качестве разделителя используется запятая. *Разделитель строк* — это символ, добавляемый в конце строки таблицы. По умолчанию в качестве разделителя строк используется символ новой строки. Вместо этих символов можно использовать другие, передав соответствующие значения функции `csv.writer()` в качестве именованных аргументов `delimiter` и `lineterminator`.

В результате передачи аргументов `delimeter='\t'` и `lineterminator='\n\n'` ❶ разделителем становится символ табуляции, а разделителем строк — удвоенный символ новой строки. После этого троекратный вызов функции `writerow()` выводит три строки таблицы.

В результате выполнения этой программы мы получаем файл *example.tsv*, имеющий следующее содержимое.

```
яблоки апельсины грейпфруты
яйца бекон ветчина
spam spam spam spam spam spam
```

В данном случае в связи с тем, что ячейки таблицы со значениями столбцов разделены теперь символами табуляции, мы используем для файла расширение *.tsv*.

Проект: удаление заголовков из CSV-файла

Предположим, вам предстоит выполнить рутинную работу по удалению первых строк из нескольких сотен CSV-файлов. Возможно, вы собираетесь передавать эти файлы какому-то автоматизированному процессу, которому требуются только данные без заголовков столбцов. Вы *могли бы* открывать каждый файл в приложении Excel, удалять первую строку таблицы и заново сохранять файл, но это отняло бы у вас много часов времени. Гораздо лучше написать программу, которая выполнит эту работу вместо вас.

Программа должна будет открывать каждый из файлов с расширением *.csv*, находящихся в текущем рабочем каталоге, читать содержимое CSV-файла и перезаписывать содержимое без первой строки в файл с тем же именем. В результате старое содержимое CSV-файла будет заменено новым, в котором заголовки столбцов таблицы отсутствуют.

Предупреждение

Как обычно, всякий раз, когда вы пишете программу, изменяющую файлы, не забывайте создавать их резервные копии хотя бы для того, чтобы застраховать себя на тот случай, если что-то пойдет не так, как ожидается. Будет очень обидно, если в подобной ситуации у вас не окажется под рукой исходных файлов.

В целом программа должна делать следующее:

- выполнить поиск всех CSV-файлов в текущем рабочем каталоге;
- прочитать полное содержимое каждого файла;
- записать содержимое, опуская первую строку, в новый CSV-файл.

На уровне кода программа должна выполнять следующие операции:

- выполнить цикл по списку файлов, возвращенному вызовом функции `os.listdir()`, пропуская файлы с расширениями, отличными от *.csv*;
- создать объект `Reader` и прочитать содержимое файла, используя атрибут `line_num` для определения того, какую строку следует пропустить;
- создать объект `Writer` и записать прочитанные данные в новый файл.

Чтобы создать проект, откройте новое окно файлового редактора и сохраните его в файле *removeCsvHeader.py*.

Шаг 1. Цикл по всем CSV-файлам

Первое, что должна сделать ваша программа, — это организовать цикл по всем CSV-файлам, находящимся в текущем рабочем каталоге. Введите в файл `removeCsvHeader.py` следующий код.

```
#! python3
# removeCsvHeader.py - Удаляет заголовки из всех CSV-файлов
# в текущем рабочем каталоге.

import csv, os

os.makedirs('headerRemoved', exist_ok=True)

# Цикл по всем файлам в текущем рабочем каталоге.
for csvFilename in os.listdir('.'):
    if not csvFilename.endswith('.csv'):
        continue # пропустить файлы с расширением,
                 # отличным от .csv

    print('Удаление заголовка из файла ' + csvFilename + '...')

    # TODO: прочитать CSV-файл (с пропуском первой строки).

    # TODO: записать CSV-файл.
```

Вызов `os.makedirs()` создает папку *headerRemoved*, в которую будут записаны все CSV-файлы, не содержащие заголовков. Цикл `for` по элементам списка `os.listdir('.')` частично решает эту задачу, однако он итерирует по всем файлам, находящимся в рабочем каталоге, поэтому в начале цикла необходимо добавить код, обеспечивающий пропуск файлов, имена которых не заканчиваются расширением *.csv*. Если в цикле встречается такой файл, то инструкция `continue` ❶ обеспечивает его пропуск и переход к следующему имени файла.

Далее следует вызов функции `print()`, которая выводит на экран имя текущего обрабатываемого CSV-файла, что позволяет контролировать ход выполнения программы. Заканчивается программа комментариями TODO, напоминающими о том, что должна делать оставшаяся часть программы.

Шаг 2. Чтение CSV-файла

В действительности программа не удаляет первую строку каждого CSV-файла. Вместо этого она создает новую копию файла, но уже без первой

строки. Поскольку имя файла-копии совпадает с именем исходного файла, он перезаписывает оригинал.

В программе необходимо предусмотреть проверку, позволяющую определить, является ли текущая строка в цикле первой. Добавьте в файл *removeCsvHeader.py* выделенный ниже код.

```
#! python3
# removeCsvHeader.py - Удаляет заголовки из всех CSV-файлов
# в текущем рабочем каталоге.
--пропущенный код--
    # Прочитать CSV-файл (с пропуском первой строки).
    csvRows = []
    csvFileObj = open(csvFilename)
    readerObj = csv.reader(csvFileObj)
    for row in readerObj:
        if readerObj.line_num == 1:
            continue # пропустить первую строку
        csvRows.append(row)
    csvFileObj.close()

    # TODO: записать CSV-файл.
```

Для отслеживания номера строки CSV-файла, которая читается в данный момент, можно использовать атрибут `line_num` объекта `Reader`. Другой цикл `for` итерирует по строкам, возвращенным объектом `Reader`, и все строки, кроме первой, присоединяются к списку `csvRows`.

В процессе выполнения итераций цикла `for` по строкам код проверяет, является ли значение атрибута `readerObj.line_num` равным `1`. Если это так, то инструкция `continue` осуществляет переход к следующей строке, не присоединяя текущую строку к списку `csvRows`. Для всех последующих строк условие будет иметь ложное значение, и они будут присоединяться к указанному списку.

Шаг 3. Запись CSV-файла без первой строки

Теперь, когда в списке `csvRows` содержатся все строки, кроме первой, его нужно записать в CSV-файл, который будет находиться в папке *headerRemoved*. Добавьте в файл *removeCsvHeader. py* выделенный ниже код.

```
#! python3
# removeCsvHeader.py - Удаляет заголовки из всех CSV-файлов
# в текущем рабочем каталоге.
--пропущенный код--

# Цикл по всем файлам в текущем рабочем каталоге.
❶ for csvFilename in os.listdir('.'):
    if not csvFilename.endswith('.csv'):
```

```
            continue # пропуск файлов с расширением,
                     # отличным от .csv

        --пропущенный код--

        # Запись CSV-файла.
        csvFileObj = open(os.path.join('headerRemoved',
    ↳ csvFilename), 'w', newline='')
        csvWriter = csv.writer(csvFileObj)
        for row in csvRows:
            csvWriter.writerow(row)
        csvFileObj.close()
```

Объект `Writer` записывает список `csvRows` в CSV-файл в папке *headerRemoved*, используя переменную `csvFilename` (которую мы также использовали в объекте `Reader`). В результате этого исходный файл будет перезаписан.

Создав объект `Writer`, мы организуем цикл по всем подчиненным спискам, хранящимся в `csvRows`, и записываем каждый из них в файл.

После выполнения блока кода внешний цикл `for` ❶ перейдет к следующему имени файла из списка `os.listdir('.')`. По выполнении этого цикла программа завершается.

Чтобы протестировать программу, загрузите файл `removeCsvHeader.zip` с сайта `http://nostarch.com/automatestuff/` и распакуйте его содержимое в папку. Запустите программу *removeCsvHeader.py* в этой папке. Вы должны получить следующий вывод.

```
Удаление заголовка из файла NAICS_data_1048.csv...
Удаление заголовка из файла NAICS_data_1218.csv...
--пропущенный вывод--
Удаление заголовка из файла NAICS_data_9834.csv...
Удаление заголовка из файла NAICS_data_9986.csv...
```

Данная программа должна выводить имя файла всякий раз, когда из CSV-файла исключается первая строка.

Идеи относительно создания аналогичных программ

Вы можете писать аналогичные программы не только для CSV-файлов, но и для файлов Excel, поскольку в обоих случаях вы имеете дело с файлами электронных таблиц. Поэтому для вас не составит большого труда написать программы для решения следующих задач:

- сравнение данных, принадлежащих различным строкам в CSV-файле или различным CSV-файлам;
- копирование специфических данных из CSV-файла в файл Excel и обратно;

- контроль допустимости данных или ошибок форматирования в CSV-файлах и вывод предупреждений для пользователя об обнаруженных ошибках;
- чтение данных из CSV-файла с целью их использования в качестве входных данных для программ на Python.

JSON и интерфейсы прикладного программирования

JSON (JavaScript Object Notation) – популярный способ форматирования данных в виде одной строки, которую удобно читать людям. Формат JSON тесно связан с языком JavaScript, используется JavaScript-программами для записи структур данных и обеспечивает форму представления данных, которая напоминает вывод, получаемый с помощью функции `pprint()` языка Python. Для работы с данными в формате JSON знать JavaScript не нужно.

Вот пример представления данных в формате JSON.

```
{"name": "Zophie", "isCat": true,
"miceCaught": 0, "napsTaken": 37.5,
"felineIQ": null}
```

Знакомство с JSON никогда вам не помешает, поскольку именно этот формат предлагается многими веб-сайтами для обмена данными с программами. Набор средств, предоставляемых веб-сайтом для использования во внешних программных продуктах, называется *прикладным программным интерфейсом* (Application Programming Interface – API). Получение доступа к API – это то же самое, что получение доступа к любой веб-странице посредством URL-адреса. Различие состоит в том, что данные, возвращаемые API, форматируются (например, с использованием JSON) для чтения компьютерами; человеку читать такие данные трудно.

Доступ к данным в формате JSON обеспечивают многие веб-сайты. Facebook, Twitter, Yahoo, Google, Tumblr, Wikipedia, Flickr, Data.gov, Reddit, IMDb, Rotten Tomatoes, LinkedIn и другие популярные сайты предлагают интерфейсы прикладного программирования, которые вы сможете использовать в своих программах. На некоторых из этих сайтов необходимо предварительно зарегистрироваться, что в подавляющем большинстве случаев осуществляется бесплатно. Вам нужно найти документацию с описанием того, по каким URL-адресам ваша программа должна направлять запросы для получения требуемых данных, и выяснить, каков общий формат возвращаемых структур данных. Такая документация должна предоставляться тем сайтом, который предлагает данный API; если на сайте имеется страница "Developers" ("Разработчикам"), то начните поиск документации оттуда.

С помощью API можно писать программы, способные, в частности, решать следующие задачи.

- Автоматический сбор "сырых" (необработанных) данных с веб-сайтов (этот процесс называют *веб-скрапингом*). (Обращение к API зачастую оказывается гораздо более удобным, чем загрузка веб-страниц и синтаксический анализ HTML-разметки с помощью парсера Beautiful Soup.)
- Автоматическая загрузка новых постов с одной из ваших учетных записей в социальных сетях и их публикация для другой учетной записи. Например, вы можете взять свои посты с сервиса микроблогов Tumblr и опубликовать их в Facebook.
- Создание "энциклопедии кино" для своей личной коллекции кинофильмов, используя для сбора данных сайты IMDb, Rotten Tomatoes и Википедия и помещая их в один текстовый файл, хранящийся на вашем компьютере.

С некоторыми примерами JSON API вы сможете познакомиться на сайте `http://nostarch.com/automatestuff/`.

Модуль `json`

Модуль Python `json` выполняет всю работу по преобразованию данных из формата JSON в значения Python и обратно для функций `json.loads()` и `json.dumps()`. Формат JSON не обеспечивает хранение *всех* типов значений Python. Он позволяет хранить лишь следующие типы данных: строки, целые и вещественные числа, булевы значения, списки, словари и значение `NoneType`. Формат JSON не может представлять специфические объекты Python, такие как объекты `File`, `Reader` и `Writer`, предназначенные для работы с CSV-файлами, объекты `Regex` или объекты `WebElement` модуля Selenium.

Чтение данных JSON с помощью функции `loads()`

Чтобы транслировать строку, содержащую JSON-данные, в значение Python, передайте ее функции `json.loads()`. Введите в интерактивной оболочке следующие команды.

```
>>> stringOfJsonData = '{"name": "Zophie", "isCat": true,
                        "miceCaught": 0, "felineIQ": null}'
>>> import json
>>> jsonDataAsPythonValue = json.loads(stringOfJsonData)
>>> jsonDataAsPythonValue
{'isCat': True, 'miceCaught': 0, 'name': 'Zophie',
'felineIQ': None}
```

Импортировав модуль `json`, можно вызвать функцию `loads()` и передать ей строку JSON-данных. Обратите внимание на то, что в строках JSON всегда используются кавычки. Эта функция возвращает данные в виде словаря Python. Данные, хранящиеся в словарях Python, не упорядочены, поэтому при выводе значения переменной `jsonDataAsPythonValue` пары "ключ-значение" могут появляться в произвольном порядке.

Запись JSON-данных с помощью функции dumps()

Функция `json.dumps()` транслирует значение Python в строку данных в формате JSON. Введите в интерактивной оболочке следующий код.

```
>>> pythonValue = {'isCat': True, 'miceCaught': 0,
↳ 'name': 'Zophie', 'felineIQ': None}
>>> import json
>>> stringOfJsonData = json.dumps(pythonValue)
>>> stringOfJsonData
'{"isCat": true, "felineIQ": null, "miceCaught": 0,
"name": "Zophie" }'
```

Значением может быть один из следующих элементарных типов данных Python: словарь, список, целое или вещественное число, строка, булево значение и `None`.

Проект: получение текущего прогноза погоды

Вообще говоря, в получении сведений о погоде нет ничего сложного. Для этого достаточно открыть браузер, щелкнуть в адресной строке, ввести URL-адрес погодного сайта (или выполнить поиск таких сайтов и щелкнуть на одной из ссылок), выждать, пока загрузится страница, пропустить рекламу и т.д.

На самом деле при этом вам приходится выполнять множество лишних действий, от которых можно было бы избавиться, имея программу, загружающую прогноз погоды на несколько дней и выводящую его в виде простого текста. Для загрузки данных из Интернета программа будет использовать модуль `requests`, рассмотренный в главе 11.

В целом программа выполняет следующие действия:

- читает название населенного пункта, заданное в командной строке;
- загружает погодные данные в формате JSON с сайта OpenWeatherMap.org;
- преобразует строку JSON-данных в структуру данных Python;
- выводит прогноз погоды на сегодня и следующие два дня.

Следовательно, код должен выполнять следующие операции:

- объединять строки, хранящиеся в списке `sys.argv`, для получения названия запрошенного населенного пункта;
- вызывать функцию `requests.get()` для загрузки погодных данных;
- вызывать функцию `json.loads()` для преобразования JSON-данных в структуру данных Python;
- выводить прогноз погоды.

Откройте в файловом редакторе новое окно для этого проекта и сохраните его в файле *quickWeather.py*.

Шаг 1. Получение расположения из аргумента командной строки

Входные данные для этой программы поступают из командной строки. Введите в файл *quickWeather.py* следующее содержимое.

```
#! python3
# quickWeather.py - Выводит прогноз погоды для заданного
# населенного пункта

import json, requests, sys

# Определение названия населенного пункта из аргументов
# командной строки.
if len(sys.argv) < 2:
    print('Использование: quickWeather.py location')
    sys.exit()
location = ' '.join(sys.argv[1:])

# TODO: загрузить JSON-данные из API сайта OpenWeatherMap.org.

# TODO: загрузить JSON-данные в переменную Python.
```

В Python аргументы командной строки хранятся в списке `sys.argv`. За "магической строкой", которая начинается символами `#!`, и инструкцией `import` следует код, выполняющий проверку того, что командная строка содержит более одного аргумента. (Вспомните о том, что в списке `sys.argv` всегда будет присутствовать по крайней мере один элемент, `sys.argv[0]`, содержащий имя файла сценария Python.) Если в списке имеется только один элемент, значит, пользователь не предоставил в командной строке название населенного пункта, и в этом случае программа, прежде чем завершить работу, выводит сообщение, объясняющее способ ее вызова.

Аргументы командной строки разделяются пробелами. Следовательно, если пользователь предоставит название населенного пункта в виде `San Francisco, CA`, то в `sys.argv` будет содержаться следующий список:

['quickWeather.py', 'San', 'Francisco,', 'CA']. Поэтому мы должны объединить все хранящиеся в sys.argv строки, за исключением первой, вызвав метод join(). Объединенная строка сохраняется в переменной location.

Шаг 2. Загрузка JSON-данных

Сайт OpenWeatherMap.org предоставляет оперативную информацию о погоде в формате JSON. Вашей программе остается лишь загрузить страницу с URL-адресом http://api.openweathermap.org/data/2.5/forecast/daily?q=*<Город>*&cnt=3, где *<Город>* — название города, погодные условия в котором вас интересуют. Добавьте в файл *quickWeather.py* следующий код, выделенный ниже полужирным шрифтом.

```
#! python3
# quickWeather.py - Выводит прогноз погоды для заданного
# населенного пункта.

--пропущенный код--

# Загрузить JSON-данные из API сайта OpenWeatherMap.org.
url ='http://api.openweathermap.org/data/2.5/forecast/
↳ daily?q=%s&cnt=3' % (location)
response = requests.get(url)
response.raise_for_status()
# TODO: загрузить JSON-данные в переменную Python.
```

Мы определили название населенного пункта из аргументов командной строки. Для создания URL-адреса, к которому необходимо получить доступ, мы используем символ-заместитель %s и вставляем в эту позицию в строке URL строку, сохраненную в переменной location. Результат сохраняется в переменной url, которая передается функции requests.get(). Вызов requests.get() возвращает объект Response, корректность которого проверяется с помощью вызова raise_for_status(). В случае отсутствия исключений загруженный текст будет находиться в переменной-члене response.text.

Шаг 3. Загрузка JSON-данных и вывод прогноза погоды

В переменной-члене response.text хранится длинная строка, содержащая данные в формате JSON. Для преобразования этой строки в значение Python вызывается функция json.loads(). Полученные JSON-данные будут выглядеть примерно так.

```
{'city': {'coord': {'lat': 37.7771, 'lon': -122.42},
          'country': 'United States of America',
          'id': '5391959',
          'name': 'San Francisco',
          'population': 0},
```

```
 'cnt': 3,
 'cod': '200',
 'list': [{'clouds': 0,
           'deg': 233,
           'dt': 1402344000,
           'humidity': 58,
           'pressure': 1012.23,
           'speed': 1.96,
           'temp': {'day': 302.29,
                    'eve': 296.46,
                    'max': 302.29,
                    'min': 289.77,
                    'morn': 294.59,
                    'night': 289.77},
           'weather': [{'description': 'sky is clear',
                        'icon': '01d',
--опущено--
```

Вы можете увидеть эти данные, передав переменную `weatherData` (см. ниже) функции `pprint.pprint()`. Более подробное описание этих полей вы найдете на сайте `http://openweathermap.org/`. Например, из онлайновой документации можно узнать, что число `302.29`, указанное за строкой `'day'`, — это дневная температура, выраженная в градусах Кельвина, а не в градусах Цельсия или Фаренгейта.

Интересующие вас погодные данные указываются за строками `'main'` и `'description'`. Можно организовать их аккуратный вывод, добавив в файл *quickWeather.py* выделенный ниже код.

```
   ! python3
   # quickWeather.py - Выводит прогноз погоды для заданного
   # населенного пункта

   --опущено--

   # Загрузка JSON-данных в переменную Python.
   weatherData = json.loads(response.text)
   # Вывод прогноза погоды.
❶  w = weatherData['list']
   print('Погода сегодня в %s:' % (location))
   print(w[0]['weather'][0]['main'], '-',
         w[0]['weather'][0]['description'])
   print()
   print('Завтра:')
   print(w[1]['weather'][0]['main'], '-',
         w[1]['weather'][0]['description'])
   print()
   print('Послезавтра:')
   print(w[2]['weather'][0]['main'], '-',
         w[2]['weather'][0]['description'])
```

Обратите внимание на то, как код сохраняет данные о погоде `weatherData['list']` в переменной `w`, избавляя вас от лишнего ввода вручную ❶. Словари с погодными данными на сегодня, завтра и послезавтра извлекаются соответственно в элементы `w[0]`, `w[1]` и `w[2]`. В каждом из этих словарей имеется ключ `'weather'`, содержащий списковое значение. Вы заинтересованы в первом элементе списка с индексом `0`, представляющем собой вложенный словарь, который содержит несколько дополнительных ключей. Здесь мы выводим значения, которые сохранены в ключах `'main'` и `'description'`, разделенных дефисами.

Запустив эту программу с аргументом командной строки `quickWeather.py San Francisco, CA`, вы получите примерно такой вывод.

```
Погода сегодня в San Francisco, CA:
Clear - sky is clear
Завтра:
Clouds - few clouds
Послезавтра:
Clear - sky is clear
```

(Основная причина, по которой мне нравится жить в Сан-Франциско, — это хорошая погода!)

Идеи относительно создания аналогичных программ

Разработанный нами код для доступа к погодным данным может быть положен в основу многих типов программ. Вы можете создать аналогичные программы, позволяющие решать следующие задачи.

- Сбор прогнозов погоды для ряда географических областей с целью выбора кемпинга или пешеходного маршрута, наиболее благоприятного с точки зрения погодных условий в определенный период времени.
- Внести в планировщик заданий программу, выполняющуюся по расписанию, которая регулярно проверяет текущие погодные условия и заблаговременно предупреждает о заморозках, чтобы вы в случае необходимости успели занести комнатные растения с улицы в помещение. (Выполнение задач по расписанию обсуждается в главе 15, а отправке электронных сообщений посвящена глава 16.)
- Сбор прогнозов погоды нескольких веб-сайтов для их одновременного отображения или расчета и отображения усредненных показателей по совокупности прогнозов.

Резюме

CSV и JSON – распространенные текстовые форматы хранения данных. Их синтаксический анализ программами не вызывает никаких сложностей, и вместе с тем они легко читаются людьми, чем и обусловлено их широкое использование для представления данных простых электронных таблиц или веб-приложений. Модули `csv` и `json` значительно упрощают процесс чтения и записи CSV- и JSON-файлов.

Из нескольких предыдущих глав вы узнали о том, как использовать Python для синтаксического анализа информации, сохраненной в файлах различных форматов. Одна из часто встречающихся задач – получение данных, подготовленных в различных форматах, и их синтаксический анализ для извлечения конкретной информации, которая представляет для вас интерес. Часто эти задачи настолько специфичны, что использование коммерческих программ не является оптимальным выходом. Написав собственные сценарии, вы сможете заставить компьютер обрабатывать большие массивы данных, представленных в этих форматах.

В главе 15 мы обсудим организацию обмена данными путем отправки сообщений электронной почты и текстовых сообщений.

Контрольные вопросы

1. Какие возможности электронных таблиц Excel не могут быть реализованы в CSV-таблицах?
2. Какие аргументы необходимо передавать функциям `csv.reader()` и `csv.writer()` для создания объектов `Reader` и `Writer`?
3. С использованием каких режимов должны открываться объекты `File` для объектов `Reader` и `Writer`?
4. Какой метод принимает список аргументов и записывает его в файл?
5. Каково назначение именованных аргументов `delimiter` и `lineterminator`?
6. Какая функция принимает строку JSON-данных, а возвращает структуру данных Python?
7. Какая функция принимает структуру данных Python, а возвращает строку JSON-данных?

Учебный проект

Чтобы закрепить полученные знания на практике, напишите программу для предложенной ниже задачи.

Программа для преобразования данных из формата Excel в формат CSV

Excel позволяет сохранить электронную таблицу в CSV-файле несколькими щелчками мышью, но если нужно преобразовать из формата Excel в формат CSV сотни файлов, то придется щелкать мышью на протяжении многих часов. Используя модуль `openpyxl` из главы 12, напишите программу, которая читает все файлы Excel, расположенные в текущем каталоге, и выводит их в виде CSV-файлов.

В одном файле Excel может содержаться множество листов, поэтому необходимо создавать по одному CSV-файлу на один лист. Файлы в формате CSV должны носить имена следующего вида: *<имя_файла_excel>_<заголовок_листа>.csv*, где *<имя_файла_excel>* — имя файла в формате Excel без расширения (например, `'spam_data'`, а не `'spam_data.xlsx'`), а *<заголовок_листа>* — строка из переменной `title` объекта Worksheet.

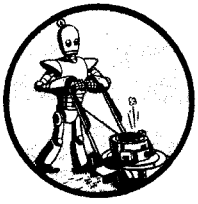
Данная программа включает ряд вложенных циклов `for`. Каркас программы может иметь примерно следующий вид.

```
for excelFile in os.listdir('.'):
    # Пропустить файлы, расширения имен которых отличны от .xlsx,
    # загрузить объект workbook.
    for sheetName in wb.get_sheet_names():
        # Цикл по листам рабочей книги.
        sheet = wb.get_sheet_by_name(sheetName)

        # Создание имени CSV-файла из имени Excel-файла
        # и титула листа рабочей книги.
        # Создание объекта csv.writer для этого CSV-файла.

        # Цикл по строкам электронной таблицы.
        for rowNum in range(1, sheet.get_highest_row() + 1):
            rowData = [] # присоединить каждую ячейку к списку
            # Цикл по всем ячейкам строки.
            for colNum in range(1, sheet.
                                get_highest_column() + 1):
                # Присоединение данных каждой ячейки к rowData.

            # Запись списка rowData в CSV-файл.

    csvFile.close()
```

Загрузите ZIP-файл *excelSpreadsheets.zip* с веб-сайта `http://nostarch.com/automatestuff/` и разархивируйте электронные таблицы в тот же каталог, в котором находится ваша программа. Вы сможете использовать эти файлы для тестирования программ.

15

ОБРАБОТКА ЗНАЧЕНИЙ ДАТЫ И ВРЕМЕНИ, ПЛАНИРОВЩИК ЗАДАНИЙ И ЗАПУСК ПРОГРАММ



Выполнять программы, сидя перед компьютером, — это, конечно, хорошо, но лучше бы программы выполнялись без непосредственного контроля над ними. И такая возможность действительно есть. Вы можете организовать запуск программ в определенное время определенного дня или через регулярные промежутки времени с привязкой к часам своего компьютера. Например, ваша программа может выполнять автоматический сбор данных на каком-либо сайте, отслеживая их обновление, или откладывать выполнение задач, требующих интенсивного использования процессорного времени, на 4 часа утра, пока вы будете спать. Эта функция обеспечивается модулями Python `time` и `datetime`.

Вы также можете писать программы, которые будут запускать другие программы по расписанию, используя модули `subprocess` и `threading`. Часто самый быстрый способ программирования заключается в использовании возможностей приложений, написанных другими людьми.

Модуль `time`

Системные часы вашего компьютера настраиваются на определенные дату, время и часовой пояс. Ваши программы на Python могут использовать системные часы для определения текущего времени с помощью встроенного модуля `time`. Чаще всего используются входящие в этот модуль функции `time.time()` и `time.sleep()`.

Функция time.time()

Общепринятая в программировании система описания времени основана на понятии так называемой "эры Unix", началом которой считается полночь 1 января 1970 года по шкале Всемирного координированного времени (Coordinated Universal Time – UTC). Функция `time.time()` возвращает количество секунд, истекших с этого момента времени, представленное вещественным числом. (Вспомните, что вещественными называют числа, содержащие десятичную точку.) Это количество секунд называется *временной меткой Unix*. Например, введите в интерактивной оболочке следующие команды.

```
>>> import time
>>> time.time()
1425063955.068649
```

Этот код был выполнен мною 27 февраля 2015 года в 11:05 по тихоокеанскому времени, или в 19:05 по времени UTC. Возвращаемое значение показывает, сколько секунд прошло с момента наступления эры Unix до момента вызова функции `time.time()`.

Примечание

Дата и время, отображаемые в примерах, которые я предлагаю для выполнения в интерактивной оболочке, соответствуют февралю 2015 года, когда я писал эту главу. Если только вы не являетесь путешественником во времени, ваша программа выведет другие значения даты и времени.

Временные метки Unix можно использовать для профилирования программы, т.е. для измерения периодов времени, в течение которых выполняются определенные фрагменты кода. Вызвав функцию `time.time()` в начале блока кода, время выполнения которого вы хотите измерить, а затем в конце этого блока, и вычтя значение первой временной метки из значения второй, вы получите длительность промежутка времени, прошедшего между двумя вызовами. Например, откройте новое окно в файловом редакторе и введите следующий код.

```
   import time
❶  def calcProd():
       # Вычисление произведения первых 100000 чисел.
       product = 1
       for i in range(1, 100000):
           product = product * i
       return product
```

```
❷ startTime = time.time()
  prod = calcProd()
❸ endTime = time.time()
❹ print('Длина результата: %s цифр.' % (len(str(prod))))
❺ print('Расчет занял %s секунд.' % (endTime - startTime))
```

В строке ❶ мы определяем функцию `calcProd()`, которая перебирает в цикле все целые числа от 1 до 100000 и возвращает их произведение. В строке ❷ мы вызываем функцию `time.time()` и сохраняем ее в переменной `startTime`. Сразу за вызовом функции `calcProd()` мы вновь вызываем функцию `time.time()` и сохраняем возвращенное ею значение в переменной `endTime` ❸. Программа заканчивается выводом количества цифр в произведении, возвращенном функцией `calcProd()` ❹, и длительности промежутка времени, в течение которого выполнялась функция `calcProd()` ❺.

Сохраните программу в файле *calcProd.py* и выполните ее. Вы получите примерно следующий вывод.

```
Длина результата: 456569 цифр.
Расчет занял 2.844162940979004 секунд.
```

Примечание

Существует и другая возможность профилирования кода – с помощью функции `cProfile.run()`, обеспечивающей намного более информативный уровень детализации, чем простая методика, основанная на использовании функции `time.time()`. Более подробно о функции `cProfile.run()` можно прочитать на сайте `https://docs.python.org/3/library/profile.html`.

Функция `time.sleep()`

Если требуется приостановить работу программы на некоторое время, вызовите функцию `time.sleep()`, передав ей аргумент, определяющий длительность паузы в секундах. Введите в интерактивной оболочке следующие команды.

```
>>> import time
>>> for i in range(3):
❶ print('Тик')
❷ time.sleep(1)
❸ print('Так')
❹ time.sleep(1)
Тик
Так
Тик
Так
```

```
   Тик
   Так
❺ >>> time.sleep(5)
```

В цикле `for` выводится на экран слово `Тик` ❶, выжидается пауза длительностью в одну секунду ❷, выводится слово `Так` ❸, выжидается пауза длительностью в одну секунду ❹, и так до тех пор, пока пара слов `Тик` и `Так` не будет выведена на экран три раза.

Функция `time.sleep()` блокируется, т.е. не возвращает управление до тех пор, пока не истечет количество секунд, переданное ей в качестве аргумента. Так, если вы введете инструкцию `time.sleep(5)` ❺, следующее приглашение ко вводу (>>>) вы увидите лишь через пять секунд.

Имейте в виду, что нажатие комбинации клавиш <Ctrl+C> не прервет выполнение вызова `time.sleep()` в IDLE. Среда IDLE будет находиться в состоянии ожидания до тех пор, пока не истечет заданная вами пауза, и лишь тогда возбудит исключение `KeyboardInterrupt`. Чтобы обойти эту проблему, можно создать паузу длительностью 30 секунд не одним вызовом `time.sleep(30)`, а тридцатью вызовами `time.sleep(1)`, выполняемыми в цикле.

```
>>> for i in range(30):
        time.sleep(1)
```

В этом случае нажатие комбинации клавиш <Ctrl+C> в любой момент на протяжении 30-секундного промежутка приведет к немедленному возбуждению исключения `KeyboardInterrupt`.

Округление чисел

В процессе работы со временем вы будете часто сталкиваться с вещественными значениями, содержащими очень много цифр после десятичной точки. Для упрощения работы с такими значениями их можно укоротить с помощью встроенной функции Python `round()`, которая округляет вещественные числа до заданной точности. Для этого достаточно передать ей число, подлежащее округлению, и необязательный второй аргумент, определяющий количество цифр после десятичной точки, до которых вы хотите округлить число. При опущенном втором аргументе функция `round()` округляет первый аргумент до ближайшего целого числа.

```
>>> import time
>>> now = time.time()
>>> now
1425064108.017826
>>> round(now, 2)
1425064108.02
```

```
>>> round(now, 4)
1425064108.0178
>>> round(now)
1425064108
```

После импортирования модуля `time` и сохранения возвращаемого значения функции `time.time()` в переменной `now` мы выполняем вызов `round(now, 2)` для округления значения `now` до двух цифр после десятичной точки, вызов `round(now, 4)` — для округления до четырех цифр и вызов `round(now)` — для округления до ближайшего целого числа.

Проект: суперсекундомер с остановом

Предположим, вы хотите наладить учет времени, непродуктивно расходуемого на выполнение трудоемких рутинных задач, которые вы еще не успели автоматизировать. Физического секундомера у вас нет, а найти аналогичное по функциональным возможностям бесплатное приложение для смартфона или ноутбука, работа которого не сопровождалась бы появлением на экране назойливой рекламы или "кражей" истории посещений вашего браузера для маркетинговых целей, на удивление трудно. (Права владельцев программ на совершение подобных действий оговариваются в лицензионных соглашениях, с которыми пользователи, как правило, соглашаются, не читая. Но вы-то читаете эти соглашения, не так ли?) Не лучше ли вместо этого самостоятельно написать на языке Python программу-хронометр?

Вот что должна делать ваша программа при высокоуровневом рассмотрении:

- отслеживать величину промежутков времени между нажатиями клавиши <Enter>, причем каждое очередное нажатие этой клавиши означает начало нового периода измерений;
- выводить номер измерения, суммарное время и длительность измерения.

Отсюда следует, что ваш код должен выполнять следующие операции:

- определять текущее время с помощью вызова `time.time()` и сохранять его в виде временной метки в начале работы программы, а также в начале каждого измерения;
- поддерживать счетчик измерений и инкрементировать его всякий раз, когда пользователь нажимает клавишу <Enter>;
- рассчитывать истекшее время путем вычитания временных меток;
- обрабатывать исключения `KeyboardInterrupt`, чтобы пользователь имел возможность прекратить работу программы, нажав комбинацию клавиш <Ctrl+C>.

Откройте новое окно в файловом редакторе и сохраните его в файле *stopwatch.py*.

Шаг 1. Создание каркаса программы для отслеживания времени

Программе-хронометру потребуется использовать текущее время, поэтому необходимо импортировать модуль `time`. Кроме того, ваша программа должна вывести краткие инструкции для пользователя еще до вызова функции `input()`, чтобы таймер мог начать отсчет сразу же после нажатия пользователем клавиши <Enter>. После этого код начнет отслеживать длительность временного промежутка.

Введите следующий код в файловом редакторе, написав комментарий TODO в качестве замены оставшейся части кода.

```
#! python3
# stopwatch.py - Простая программа-хронометр.

import time

# Отображение инструкции по использованию программы.
print('Чтобы начать отсчет, нажмите клавишу ENTER. Впоследствии
↪ для имитации щелчков кнопки секундомера нажимайте клавишу
↪ <Enter>. Для выхода из программы нажмите клавиши <Ctrl+C>.')
input()                   # нажатие клавиши <Enter> начинает
                          # отсчет
print('Отсчет начат.')
startTime = time.time()   # получение времени начала первого
                          # замера
lastTime = startTime
lapNum = 1

# TODO: Начать отслеживание замеров.
```

Написав код для отображения инструкций, мы начинаем первый замер, фиксируем время и устанавливаем значение счетчика замеров равным 1.

Шаг 2. Отслеживание и вывод длительности замеров

А теперь напишем код, который начинает каждый новый замер, рассчитывает длительность предыдущего замера и вычисляет суммарное время, истекшее с момента запуска хронометра. Далее мы отображаем длительность замера, а также суммарное время всех предыдущих замеров и увеличиваем значение счетчика для очередного замера. Добавьте в программу выделенный ниже код.

```
#! python3
# stopwatch.py - Простая программа-хронометр.

import time

--пропущенный код--
```

```
    # Начало отслеживание замеров.
❶ try:
❷       while True:
            input()
❸           lapTime = round(time.time() - lastTime, 2)
❹           totalTime = round(time.time() - startTime, 2)
❺           print('Замер #%s: %s (%s)' %
                (lapNum, totalTime, lapTime), end='')
            lapNum += 1
            lastTime = time.time() # переустановить время
                                   # последнего замера
❻ except KeyboardInterrupt:
      # Обработать исключение <Ctrl+C>, чтобы предотвратить
      # отображение его сообщений.
print('\nГотово.')
```

В случае, если пользователь останавливает хронометр нажатием комбинации клавиш <Ctrl+C>, возбуждается исключение `KeyboardInterrupt`, и программа завершается аварийно, если она выполняется не в блоке `try`. Чтобы избежать краха программы, мы помещаем код в инструкцию `try` ❶. Мы обрабатываем исключение в инструкции `except` ❻, поэтому в случае нажатия комбинации клавиш <Ctrl+C> и возбуждения исключения выполнение программы передается инструкции `except` для вывода сообщения `Готово` и предотвращения вывода сообщения об ошибке `KeyboardInterrupt`. Пока этого не произойдет, программа будет выполнять бесконечный цикл ❷, вызывающий функцию `input()`, и ожидать нажатия пользователем клавиши <Enter> для завершения замера. Далее мы рассчитываем длительность замера путем вычитания времени его начала, `lastTime`, из текущего времени, `time.time()` ❸. Суммарное истекшее время вычисляется путем вычитания общего начального времени запуска хронометра, `startTime`, из текущего времени ❹.

Поскольку все результаты временны*х* расчетов содержат чрезмерно большое количество цифр после десятичной точки (например, 4.766272783279419), мы округляем их до двух цифр с помощью функции `round()` (❸ и ❹).

В строке ❺ выводятся номер замера, суммарное истекшее время и длительность замера. Поскольку нажатие пользователем клавиши <Enter> в ответ на вызов функции `input()` инициирует вывод символа новой строки на экран, функции `print()` необходимо передать именованный аргумент `end=''`, устраняющий появление двойного междустрочного интервала в выводе. Выведя информацию о замере, мы выполняем подготовительные операции для следующего замера, прибавляя `1` к значению счетчика `lapNum` и устанавливая для переменной `lastTime` значение текущего времени, которое будет служить началом отсчета для следующего замера.

Идеи относительно создания аналогичных программ

Отслеживание времени открывает широкие возможности для ваших программ. И хотя для решения некоторых задач подобного рода можно найти готовые приложения, написание собственных программ обладает тем преимуществом, что за них не надо платить и они не будут содержать ненужные вам средства или надоедать назойливой рекламой. В частности, вы могли бы самостоятельно сделать следующее.

- Создайте приложение, реализующее простой табель, которое при вводе имени сотрудника записывает соответствующее время прихода и ухода с работы, используя текущие показания часов.
- Используя модуль `requests` (см. главу 11), добавьте в свою программу средство для отображения времени, истекшего с момента начала какого-либо процесса, например загрузки файла.
- Периодически проверяйте, как долго выполняется программа, и предоставьте пользователю возможность принудительного завершения долго выполняющихся задач.

Модуль datetime

Модуль `time` полезен для непосредственной работы с временными метками Unix. Но если вы хотите отобразить дату в более удобном формате или выполнить над датами арифметические действия (например, определить, какая дата была 205 дней назад или какая дата наступит через 123 дня), используйте модуль `datetime`.

Модуль `datetime` имеет собственный тип данных `datetime`. Значения `datetime` представляют определенные моменты времени. Введите в интерактивной оболочке следующие команды.

```
   >>> import datetime
❶  >>> datetime.datetime.now()
❷  datetime.datetime(2015, 2, 27, 11, 10, 49, 55, 53)
❸  >>> dt = datetime.datetime(2015, 10, 21, 16, 29, 0)
❹  >>> dt.year, dt.month, dt.day
   (2015, 10, 21)
❺  >>> dt.hour, dt.minute, dt.second
   (16, 29, 0)
```

Вызов функции `datetime.datetime.now()` ❶ возвращает объект `datetime` ❷ для текущих даты и времени в соответствии с показаниями часов вашего компьютера. Этот объект содержит информацию о годе, месяце, дне, часе, минуте, секунде и микросекунде текущего момента времени. Также можно получить объект `datetime` для любого заданного момента времени

с помощью функции datetime.datetime() ❸, передав ей целые числа, представляющие год, месяц, день, час, минуту и секунду желаемого момента времени. Эти целые числа будут храниться в атрибутах year, month, day ❹, hour, minute и second ❺ объекта datetime.

Временную метку Unix можно преобразовать в объект datetime с помощью функции datetime.datetime.fromtimestamp(). При этом дата и время объекта datetime будут соответствовать местному часовому поясу. Введите в интерактивной оболочке следующие команды.

```
>>> datetime.datetime.fromtimestamp(1000000)
datetime.datetime(1970, 1, 12, 5, 46, 40)
>>> datetime.datetime.fromtimestamp(time.time())
datetime.datetime(2015, 2, 27, 11, 13, 0, 604980)
```

Вызов функции datetime.datetime.fromtimestamp() с передачей ей аргумента 1000000 возвращает объект datetime для момента времени, соответствующего 1000000 секундам после наступления эры Unix. Функция time.time() при передаче ей временной метки Unix текущего момента времени возвращает объект datetime, соответствующий этому моменту. Поэтому выражения datetime.datetime.now() и datetime.datetime.fromtimestamp(time.time()) делают одно и то же: они оба возвращают объект datetime для настоящего момента времени.

Примечание

Эти примеры выполнялись на компьютере, настроенном на тихоокеанское время. Если вы находитесь в другом часовом поясе, ваши результаты будут выглядеть иначе.

Чтобы выяснить, какой из объектов datetime предшествует другому, используйте операторы сравнения. Более поздним объектам соответствуют "большие" значения. Введите в интерактивной оболочке следующие команды.

```
❶ >>> halloween2015 = datetime.datetime(2015, 10, 31, 0, 0, 0)
❷ >>> newyears2016 = datetime.datetime(2016, 1, 1, 0, 0, 0)
  >>> oct31_2015 = datetime.datetime(2015, 10, 31, 0, 0, 0)
❸ >>> halloween2015 == oct31_2015
  True
❹ >>> halloween2015 > newyears2016
  False
❺ >>> newyears2016 > halloween2015
  True
  >>> newyears2016 != oct31_2015
  True
```

Сначала здесь создается объект datetime для первого момента времени (полночь) 31 октября 2015 года. Этот объект сохраняется в переменной halloween2015 ❶. Затем создается объект datetime для первого момента времени 1 января 2016 года; для сохранения этого объекта используется переменная newyears2016 ❷. После этого создается еще один объект для полуночи 31 октября 1915 года. Сравнение halloween2015 и oct31_2015 показывает, что они равны ❸. Сравнение newyears2016 и halloween 2015 показывает, что значение newyears2016 больше (соответствует более позднему моменту времени), чем значение halloween2015 ❹ ❺.

Тип данных timedelta

Модуль datetime также предоставляет тип данных timedelta, который представляет длительности временны́х промежутков, а не моменты времени. Введите в интерактивной оболочке следующие команды.

```
❶ >>> delta = datetime.timedelta(days=11, hours=10,
                minutes=9, seconds=8)
❷ >>> delta.days, delta.seconds, delta.microseconds
  (11, 36548, 0)
  >>> delta.total_seconds()
  986948.0
  >>> str(delta)
  '11 days, 10:09:08'
```

Для создания объектов timedelta используется функция datetime.timedelta(). Функция datetime.timedelta() принимает именованные аргументы weeks, days, hours, minutes, seconds, milliseconds и microseconds. Аргументы month и year не предусмотрены, поскольку "month" (месяц) и "year" (год) представляют переменные отрезки времени. Объект timedelta имеет полную длительность, выражаемую в днях, секундах и микросекундах. Соответствующие числовые значения хранятся в атрибутах days, seconds и microseconds соответственно. Метод total_seconds() возвращает длительность, выраженную в секундах. При передаче методу str() объекта timedelta он возвращает аккуратно отформатированное, удобное для чтения людьми строковое представление объекта.

В этом примере мы передаем методу datetime.delta() именованные аргументы, определяющие длительность, равную 11 дням, 10 часам, 9 минутам и 8 секундам, и сохраняем возвращенный объект timedelta в переменной delta ❶. В атрибуте days объекта timedelta содержится значение 11, а в атрибуте seconds — значение 36548 (длительность 10 часов 9 минут и 8 секунд, пересчитанная в секунды) ❷. Вызов total_seconds() сообщает нам, что длительность в 11 дней 10 часов 9 минут и 8 секунд составляет 986948

секунд. Наконец, метод `str()` после передачи ему объекта `timedelta` возвращает строку, которая в ясной форме отображает длительность данного промежутка времени.

Для выполнения арифметических действий над значениями `datetime` используются арифметические операторы. Например, чтобы определить дату, которая наступит через тысячу дней от текущей даты, введите в интерактивной оболочке следующий код.

```
>>> dt = datetime.datetime.now()
>>> dt
datetime.datetime(2015, 2, 27, 18, 38, 50, 636181)
>>> thousandDays = datetime.timedelta(days=1000)
>>> dt + thousandDays
datetime.datetime(2017, 11, 23, 18, 38, 50, 636181)
```

Прежде всего мы создаем объект `datetime` для текущего момента времени и сохраняем его в переменной `dt`. Затем мы создаем объект `timedelta` для длительности 1000 дней и сохраняем его в переменной `thousandDays`. Далее значения переменных `dt` и `thousandDays` складываются для получения объекта `datetime` для будущей даты, которая наступит через 1000 дней после текущей даты. Python выполняет все необходимые арифметические операции и определяет, что датой, которая наступит через 1000 дней после 27 февраля 2015 года, является 23 ноября 2017 года. Это очень удобно, поскольку при проведении самостоятельных расчетов вам пришлось бы учитывать, сколько дней содержится в каждом месяце, не забывая при этом о високосных годах, т.е. помнить о множестве мелких деталей. Модуль `datetime` выполняет всю эту работу вместо вас.

Объекты `timedelta` могут участвовать в операциях сложения и вычитания с объектами `datetime` или другими объектами `timedelta` с использованием операторов `+` и `-`. Объект `timedelta` можно умножать или делить на целые или вещественные значения с помощью операторов `*` и `/`. Введите в интерактивной оболочке следующий код.

```
❶ >>> oct21st = datetime.datetime(2015, 10, 21, 16, 29, 0)
❷ >>> aboutThirtyYears = datetime.timedelta(days=365 * 30)
>>> oct21st
datetime.datetime(2015, 10, 21, 16, 29)
>>> oct21st - aboutThirtyYears
datetime.datetime(1985, 10, 28, 16, 29)
>>> oct21st - (2 * aboutThirtyYears)
datetime.datetime(1955, 11, 5, 16, 29)
```

В этом коде мы создаем объект `datetime` для 21 октября 2015 года ❶ и объект `timedelta` для длительности, равной примерно 30 лет (мы не

учитывали високосные годы) ❷. Вычитание `aboutThirtyYears` из `oct21st` дает нам объект `datetime`, которые соответствует дате, отстоящей на 30 лет назад от 21 октября 2015 года. В результате вычитания `2 * aboutThirtyYears` из `oct21st` мы получаем объект `datetime`, который соответствует дате, отстоящей от 21 октября 2015 года на 60 лет назад.

Организация паузы до наступления определенной даты

Метод `time.sleep()` позволяет приостанавливать работу программы на определенное количество секунд. Используя цикл `while`, вы можете приостановить работу программы до наступления определенной даты. Например, следующий код будет непрерывно выполнять цикл до наступления Хэллоуина в 2016 году.

```
import datetime
import time
halloween2016 = datetime.datetime(2016, 10, 31, 0, 0, 0)
while datetime.datetime.now() < halloween2016:
    time.sleep(1)
```

Вызов `time.sleep(1)` приостанавливает программу, чтобы компьютер не тратил драгоценные такты процессора на непрерывную проверку времени. Вместо этого цикл `while` всего лишь проверяет выполнение условия каждую секунду и передает управление остальной части программы, как только наступит Хэллоуин 2016 года (или любая наперед заданная дата).

Преобразование объектов `datetime` в строки

Временны*е* метки Unix и объекты `datetime` плохо приспособлены для чтения людьми. Чтобы отобразить объект `datetime` в виде строки, используйте метод `strftime()`. (Буква *f* в имени метода `strftime()` — от слова *format*.)

В методе `strftime()` используются директивы, аналогичные тем, которые используются при выводе форматированных строк Python. Полный перечень директив метода `strftime()` приведен в табл. 15.1.

Таблица 15.1. Директивы функции `strftime()`

Директива	Описание
`%Y`	Год с указанием столетия: '2014'
`%y`	Год без указания столетия: от '00' до '99' (1970–2069)
`%m`	Месяц в виде десятичного числа: от '01' до '12'

Окончание табл. 15.1

Директива	Описание
%B	Полное название месяца: 'November'
%b	Сокращенное название месяца: 'Nov'
%d	День месяца: от '01' до '31'
%j	День года: от '001' до '366'
%w	День недели: от '0' (воскресенье) до '6' (суббота)
%A	Полное название дня недели: 'Monday'
%a	Сокращенное название дня недели: 'Mon'
%H	Часы (24-часовая шкала): от '00' до '23'
%I	Часы (12-часовая шкала): от '01' до '12
%M	Минуты: от '00' до '59'
%S	Секунды: от '00' до '59'
%p	'AM' (до полудня) или 'PM' (после полудня)
%%	Литеральный символ '%'

Передайте функции `strrftime()` строку описания формата, содержащую директивы форматирования (вместе с любыми желаемыми символами обратной косой черты, двоеточиями и т.д.), и она возвратит информацию об объекте `datetime` в виде отформатированной строки. Введите в интерактивной оболочке следующий код.

```
>>> oct21st = datetime.datetime(2015, 10, 21, 16, 29, 0)
>>> oct21st.strftime('%Y/%m/%d %H:%M:%S')
'2015/10/21 16:29:00'
>>> oct21st.strftime('%I:%M %p')
'04:29 PM'
>>> oct21st.strftime("%B of '%y")
"October of '15"
```

Здесь мы имеем объект `datetime`, соответствующий дате 21 октября 2015 года и времени 16:29, который сохраняется в переменной `oct21st`. При передаче функции `strftime()` строки форматирования `'%Y/%m/%d %H:%M:%S'` эта функция возвращает числа `2015`, `10` и `21`, разделенные символами обратной косой черты, а также числа `16`, `29` и `00`, разделенные двоеточиями. В случае передачи этой функции строки `'%I:%M% p'` она возвращает значение `'04:29 PM'`, а в случае передачи строки `"%B of '%y"` — значение `"October of '15"`. Обратите внимание на то, что имя функции `strftime()` указывается без префикса `datetime.datetime`.

Преобразование строк в объекты datetime

Если у вас имеется строка, содержащая информацию о дате, например '2015/10/21 16:29:00' или 'October 21, 2015', которую необходимо преобразовать в объект datetime, то используйте функцию datetime.datetime.strptime(). Функция strptime() выполняет операцию, противоположную той, которую выполняет метод strftime. Чтобы функция strptime() смогла распознать и правильно выполнить синтаксический анализ строки описания формата, в этой строке должны использоваться те же директивы форматирования, что и в случае метода strftime(). (Буква *p* в имени функции strptime() — от слова *parse*.)

Введите в интерактивной оболочке следующий код.

```
❶ >>> datetime.datetime.strptime('October 21, 2015', '%B %d, %Y')
  datetime.datetime(2015, 10, 21, 0, 0)
  >>> datetime.datetime.strptime('2015/10/21 16:29:00',
↪  '%Y/%m/%d %H:%M:%S')
  datetime.datetime(2015, 10, 21, 16, 29)
  >>> datetime.datetime.strptime("October of '15", "%B of '%y")
  datetime.datetime(2015, 10, 1, 0, 0)
  >>> datetime.datetime.strptime("November of '63", "%B of '%y")
  datetime.datetime(2063, 11, 1, 0, 0)
```

Чтобы получить объект datetime из строки 'October 21, 2015', передайте функции strptime() эту строку в качестве первого аргумента, а строку описания формата, соответствующую строке 'October 21, 2015', — в качестве второго аргумента ❶. Строка с информацией о дате должна в точности соответствовать строке описания формата, иначе Python возбудит исключение ValueError.

Обзор функций Python для работы с датами и временем

Для работы с датами и временем в Python может потребоваться довольно большое количество различных типов данных и функций. Ниже кратко описаны три типа значений, используемых для представления времени.

- Временная метка Unix (используется модулем time) — это целое или вещественное число, представляющее количество секунд, истекших с полуночи 1 января 1970 года по шкале UTC.
- Объект datetime (из модуля datetime) содержит целочисленные значения, хранящиеся в атрибутах year, month, day, hour, minute и second.
- Объект timedelta (из модуля datetime) представляет длительность временного отрезка, а не конкретный момент времени.

Ниже приведено краткое описание функций для работы со временем, их параметров и возвращаемых значений.

- Функция `time.time()` возвращает временную метку Unix в виде вещественного значения, которая соответствует текущему моменту.
- Функция `time.sleep(`*`секунды`*`)` приостанавливает выполнение программы на время, определяемое количеством секунд, которое задается аргументом *секунды*.
- Функция `datetime.datetime(`*`год, месяц, день, час, минута, секунда`*`)` возвращает объект `datetime`, который соответствует моменту времени, специфицированному аргументами. Если аргументы *час*, *минута* или *секунда* не предоставлены, они принимают заданное по умолчанию значение `0`.
- Функция `datetime.datetime.now()` возвращает объект `datetime`, соответствующий текущему моменту времени.
- Функция `datetime.datetime.fromtimestamp(`*`временная_метка_Unix`*`)` возвращает объект `datetime`, который соответствует моменту времени, представленному аргументом *временная_метка_Unix*.
- Функция `datetime.timedelta(`*`недели, дни, часы, минуты, секунды, миллисекунды, микросекунды`*`)` возвращает объект `timedelta`, представляющий временную длительность. Все именованные аргументы этой функции необязательны и не включают аргументы *месяц* и *год*.
- Метод `total_seconds()` объектов `timedelta` возвращает количество секунд, представляемое объектом `timedelta`.
- Метод `strftime(`*`формат`*`)` возвращает строку времени, представленную объектом `datetime`, в пользовательском формате, основанном на строке описания формата. Более подробно о деталях формата читайте в табл. 15.1.
- Функция `datetime.datetime.strptime(`*`строка_времени, формат`*`)` возвращает объект `datetime`, который соответствует моменту времени, определенному аргументом *строка_времени*, синтаксически проанализированным с использованием строкового аргумента *формат*. Более подробно о деталях формата читайте в табл. 15.1.

Многопоточность

Чтобы ввести понятие многопоточности, лучше обратиться к конкретному примеру. Предположим, вы хотите назначить задержку или конкретное время начала выполнения некоторой программы. Вы могли бы добавить в начало программы примерно следующий код.

```
import time, datetime

startTime = datetime.datetime(2029, 10, 31, 0, 0, 0)
while datetime.datetime.now() < startTime:
    time.sleep(1)

print('Теперь программа запустится на Хэллоуин 2029 года')
--пропущенный код--
```

Этот код назначает запуск программы на полночь 31 октября 2029 года и вызывает функцию `time.sleep(1)` в цикле до наступления этого момента. В процессе ожидания завершения цикла вызовов функции `time.sleep()` ваша программа ничего не сможет делать и будет "спать" до тех пор, пока не наступит Хэллоуин 2029 года. События развиваются таким образом по той причине, что по умолчанию Python предоставляет программам только один *поток выполнения*.

Чтобы понять, что такое поток выполнения, вспомните о приведенной в главе 2 образной аналогии, когда процесс выполнения программы уподоблялся перемещению вашего пальца по строкам кода — либо последовательному, от одной строки кода к следующей, либо в соответствии с управляющими инструкциями, совершая скачки через строки кода. Выполнением однопоточных программ может управлять только один "палец". Однако у многопоточных программ таких управляющих "пальцев" может быть несколько. Каждый из них переходит от одной строки кода к другой в соответствии с управляющими инструкциями, но при этом "пальцы" могут перемещаться по разным участкам программы, независимо выполняя разные строки кода в одно и то же время. (Все программы, которые мы рассматривали до сих пор в этой книге, были однопоточными.)

Вместо того чтобы заставлять весь свой код находиться в состоянии ожидания до тех пор, пока не завершатся циклические вызовы функции `time.sleep()`, можно выделить для выполнения отложенной или запланированной по расписанию задачи отдельный поток выполнения, используя модуль Python `threading`. Этот отдельный поток будет находиться в состоянии ожидания все то время, пока вызывается функция `time.sleep()`. Тем временем ваша программа может выполнять другую полезную работу в исходном потоке.

Прежде чем можно будет создать поток, необходимо предварительно создать объект `Thread`, вызвав функцию `threading.Thread()`. Введите в новом окне файлового редактора следующий код и сохраните его в файле *threadDemo.py*.

```
import threading, time
print('Начало программы.')
```

```
❶ def takeANap():
    time.sleep(5)
    print('Проснись!')

❷ threadObj = threading.Thread(target=takeANap)
❸ threadObj.start()

    print('Конец программы.')
```

В строке ❶ мы определяем функцию, которая будет выполняться в новом потоке. Чтобы создать объект `Thread`, мы вызываем функцию `threading.Thread()` и передаем ей именованный аргумент `target=takeANap` ❷. Это означает, что функцией, которую мы хотим вызвать в новом потоке, является функция `takeANap()`. Обратите внимание на то, что именованный аргумент записывается как `target=takeANap`, а не как `target=takeANap()`. Это обусловлено тем, что в качестве аргумента мы хотим передать функцию `takeANap` как таковую, а не значение, возвращенное в результате ее вызова.

Сохранив объект `Thread`, созданный функцией `threading.Thread()`, в переменной `threadObj`, мы можем вызвать метод `threadObj.start()` ❸ для создания нового потока и запуска в нем целевой функции. Выполнив эту программу, вы получите следующий результат.

```
Начало программы.
Конец программы.
Подъем!
```

Этот результат может немного смущать. Если инструкция `print('Конец программы.')` — последняя в программе, то почему ее текст не был выведен последним? Причиной того, что последним выводится текст `Подъем!`, является то обстоятельство, что после вызова `threadObj.start()` целевая функция объекта `threadObj` выполняется в новом потоке. Возвращаясь к вышеупомянутой образной аналогии, можете считать, что при запуске функции `takeANap()` появляется второй управляющий "палец". Основной поток переходит к инструкции `print('Конец программы.')`. Тем временем новый поток, выполняющий вызов `time.sleep(5)`, выжидает в течение 5 секунд. После этого он выходит из своей 5-секундной спячки, выводит на экран строку `'Подъем!'` и осуществляет выход из функции `takeANap()`. В соответствии с описанной хронологией последнее, что выводит программа, — это строка `'Подъем!'`.

Обычно выполнение программы завершается с выполнением последней строки ее кода (или в результате вызова функции `sys.exit()`). Но в программе `threadDemo.py` существуют два потока. Один из них, первоначальный поток, в котором начала выполняться программа, завершается после выполнения инструкции `print('Конец программы.')`. Второй поток

создается вызовом `threadObj.start()`, начинает выполнение с запуском функции `takeANap()` и завершает выполнение, как только осуществляется возврат из этой функции.

В Python программа прекращает выполнение тогда, когда прекращают выполнение все ее потоки. Когда вы запустили программу *threadDemo.py*, то даже после того, как первоначальный поток завершился, второй поток все еще продолжал выполнение вызова `time.sleep(5)`.

Передача аргументов целевой функции

Если целевая функция, которую вы хотите выполнять в отдельном потоке, принимает аргументы, то можно передать их методу `threading.Thread()`. Предположим, например, что вы хотите выполнить следующий вызов `print()` в собственном потоке.

```
>>> print('Cats', 'Dogs', 'Frogs', sep=' & ')
Cats & Dogs & Frogs
```

В этом вызове используются три обычных аргумента, `'Cats'`, `'Dogs'` и `'Frogs'`, и один именованный, `sep=' & '`. Обычные аргументы могут быть переданы в виде списка именованному аргументу `args` функции `threading.Thread()`, а именованный аргумент в виде словаря — именованному аргументу `kwargs` этой функции.

Введите в интерактивной оболочке следующий код.

```
>>> import threading
>>> threadObj = threading.Thread(target=print, args=['Cats',
↪ 'Dogs', 'Frogs'],
kwargs={'sep': ' & '})
>>> threadObj.start()
Cats & Dogs & Frogs
```

Чтобы обеспечить передачу аргументов `'Cats'`, `'Dogs'` и `'Frogs'` функции `print()` в новом потоке, мы передаем именованный аргумент `args=['Cats', 'Dogs', 'Frogs']` функции `threading.Thread()`. Аналогичным образом мы поступаем в отношении именованного аргумента `sep=' & '`, передавая функции `threading.Thread()` именованный аргумент `kwargs={'sep': '& '}`.

Метод `threadObj.start()` создает новый поток выполнения для вызова функции `print()`, и он же передает ей строки `'Cats'`, `'Dogs'` и `'Frogs'` в качестве аргументов и `' & '` — в качестве значения именованного аргумента `sep`.

Ниже продемонстрирован неправильный способ создания нового потока выполнения для вызова функции `print()`.

```
threadObj = threading.Thread(target=print('Cats', 'Dogs',
↳ 'Frogs', sep=' & '))
```

В этом коде будет вызвана функция `print()`, и в качестве именованного аргумента `target` функции `threading.Thread()` будет передана *не* сама функция `print()`, а возвращаемое ею значение (каковым всегда является `None`). Для передачи аргументов функции, выполняющейся в новом потоке, следует использовать именованные аргументы `args` и `kwargs` функции `threading.Thread()`.

Проблемы параллелизма

Вы можете легко создать несколько новых потоков, и все они будут выполняться одновременно. Однако выполнение нескольких потоков может порождать так называемые *проблемы параллелизма*. Эти проблемы возникают в тех случаях, когда разные потоки одновременно пытаются читать и записывать одни и те же переменные, конкурируя между собой. Проблемы параллелизма трудно воспроизводить, поскольку они возникают, казалось бы, без какой-либо закономерности, что затрудняет отладку программы.

Многопоточное программирование — это отдельная обширная тема, рассмотрение которой выходит за рамки данной книги. Вам нужно лишь хорошо запомнить следующее: чтобы избежать проблем параллелизма, никогда не позволяйте нескольким потокам читать и записывать одни и те же переменные. Создавая новый объект `Thread`, убеждайтесь в том, что его целевая функция использует лишь локальные переменные данной функции. Тем самым вы избежите возникновения в своих программах проблем параллелизма, трудно поддающихся отладке.

Примечание

Руководство по многопоточному программированию для начинающих доступно на сайте `http://nostarch.com/automatestuff/`.

Проект: многопоточный загрузчик файлов с сайта XKCD

В главе 11 вы написали программу, загружающую все комиксы с сайта XKCD. Это была однопоточная программа: она загружала комиксы по одному за раз. Значительная часть времени программа расходовала на установление сетевого соединения, чтобы начать загрузку изображений и их запись на жесткий диск. При наличии широкополосного канала подключения к Интернету однопоточная программа будет использовать не всю доступную полосу пропускания.

Многопоточная программа, в которой загрузка комиксов, установление соединения и запись файлов изображений на жесткий диск осуществляются разными потоками, будет использовать подключение к Интернету гораздо более эффективно, что приведет к ускорению загрузки коллекции комиксов. Откройте новое окно в файловом редакторе и сохраните его в файле *multidownloadXkcd.py*. Мы модифицируем предыдущую программу, сделав ее многопоточной. Полный модифицированный код программы можно скачать на сайте `http://nostarch.com/automatestuff/`.

Шаг 1. Видоизменение программы путем вынесения ее кода в функцию

В этой программе для загрузки файлов будет использоваться в основном тот же код, что и в главе 11, поэтому я опущу описание кода, взаимодействующего с модулями Requests и BeautifulSoup. Основные изменения связаны с импортированием модуля `threading` и созданием функции `downloadXkcd()`, принимающей номера начального и конечного комиксов в качестве аргументов.

Например, вызов `downloadXkcd(140, 280)` выполнит в цикле код для загрузки комиксов, хранящихся на страницах `http://xkcd.com/140`, `http://xkcd.com/141`, `http://xkcd.com/142` и т.д. вплоть до комикса `http://xkcd.com/279`. Каждый создаваемый поток выполнения будет вызывать функцию `downloadXkcd()` с передачей ей разных диапазонов номеров комиксов, подлежащих загрузке.

Добавьте в программу *multidownloadXkcd.py* следующий код.

```
#! python3
# multidownloadXkcd.py - Загружает комиксы XKCD с
# использованием нескольких потоков выполнения.

import requests, os, bs4, threading
❶ os.makedirs('xkcd', exist_ok=True) # сохранить комиксы в
                                     #папке ./xkcd

❷ def downloadXkcd(startComic, endComic):
❸     for urlNumber in range(startComic, endComic):
          # Загрузка страницы.
          print('Загрузка страницы http://xkcd.com/%s...' %
      (urlNumber))
❹         res = requests.get('http://xkcd.com/%s' %
      (urlNumber))
          res.raise_for_status()

❺         soup = bs4.BeautifulSoup(res.text)

          # Поиск URL-адреса изображения комикса.
```

```
❻         comicElem = soup.select('#comic img')
          if comicElem == []:
              print('Не удается найти изображение комикса.')
          else:
❼             comicUrl = comicElem[0].get('src')
              # Загрузка изображения.
              print('Загрузка изображения %s...' % (comicUrl))
❽             res = requests.get(comicUrl)
              res.raise_for_status()

              # Сохранение изображения в папке ./xkcd.
              imageFile = open(os.path.join('xkcd',
                 os.path.basename(comicUrl)), 'wb')
              for chunk in res.iter_content(100000):
                  imageFile.write(chunk)
              imageFile.close()

  # TODO: Создать и запустить объекты Thread.
  # TODO: Дождаться завершения всех потоков.
```

Импортировав необходимые модули, мы создаем каталог для хранения комиксов ❶ и определяем функцию `downloadxkcd()` ❷. В этой функции мы организуем цикл для обработки комиксов в заданном диапазоне номеров ❸ и загружаем каждую страницу ❹. Далее мы используем модуль Beautiful Soup для просмотра HTML-кода каждой страницы ❺ и поиска изображения комикса ❻. В случае отсутствия изображения на странице выводится сообщение. В противном случае мы получаем URL-адрес изображения ❼ и загружаем само изображение ❽. Наконец, мы сохраняем изображение в созданном каталоге.

Шаг 2. Создание и запуск потоков выполнения

Теперь, когда у нас есть функция `downloadXkcd()`, мы создадим несколько потоков, каждый из которых вызывает функцию `downloadXkcd()` для загрузки комиксов с номерами, лежащими в различных диапазонах, с веб-сайта XKCD. Добавьте в файл *multidownloadXkcd.py* следующий код после определения функции `downloadXkcd()`.

```
#! python3
# multidownloadXkcd.py - Загружает комиксы XKCD с
# использованием нескольких потоков выполнения.

--пропущенный код--

# Создание и запуск объектов Thread.
downloadThreads = []          # список всех объектов Thread
for i in range(0, 1400, 100): # 14 итераций, создающих
                              # 14 потоков
```

```
    downloadThread = threading.Thread(target=downloadXkcd,
↳ args=(i, i + 99))
    downloadThreads.append(downloadThread)
    downloadThread.start()
```

Прежде всего мы создаем пустой список downloadThreads; этот список поможет нам отслеживать множество объектов Thread, которые мы будем создавать. Затем мы запускаем цикл for. На каждом проходе цикла мы создаем объект Thread с помощью вызова threading.Thread(), присоединяем его к списку и вызываем метод start() для запуска функции downloadXkcd() в новом потоке. Так как переменная i цикла for принимает значения от 0 до 1400 с шагом 100, в начале первой итерации она будет иметь значение 0, в начале второй — значение 100, в начале третьей — значение 200 и т.д. Поскольку мы передаем функции threading.Thread() список аргументов args=(i, i + 99), на первой итерации цикла два аргумента, образующие этот список, будут иметь значения 0 и 99, на второй — 100 и 199, на третьей — 200 и 299 и т.д.

В процессе вызова метода start() объекта Thread и запуска нового потока для выполнения кода функции downloadXkcd() основной поток будет продолжать выполнение, запуская очередной поток на следующей итерации цикла.

Шаг 3. Ожидание завершения всех потоков

В то время как другие потоки, которые мы создали, загружают комиксы, основной поток продолжает выполняться как обычно. Предположим, однако, что у вас есть некоторый код, начало выполнения которого до того, как завершится выполнение всех остальных потоков, для вас нежелательно. Вызов метода join() будет блокироваться до завершения данного потока. Используя цикл for для итерирования по всем объектам Thread, входящим в список downloadThreads, основной поток может вызвать метод join() для всех остальных потоков. Добавьте в конце программы следующий код.

```
#! python3
# multidownloadXkcd.py - Загружает комиксы XKCD с
# использованием нескольких потоков выполнения.

--пропущенный код--

# Ожидание завершения всех потоков выполнения.
for downloadThread in downloadThreads:
    downloadThread.join()
print('Готово.')
```

Строка 'Готово.' не будет выведена до тех пор, пока не будет выполнен возврат из всех вызовов метода join(). Если окажется так, что к моменту

вызова метода `join()` объект `Thread` уже завершил выполнение, то будет просто выполнен немедленный возврат из метода `join()`. Если вы хотите дополнить программу кодом, выполнение которого должно начаться лишь после того, как завершатся все потоки, то вам следует просто заменить этим кодом инструкцию `print('Готово.')`.

Запуск других программ из Python

Ваша программа на Python может запускать на компьютере другие программы с помощью функции `Popen()` встроенного модуля `subprocess`. (Буква *P* в имени функции `Popen()` происходит от слова *process*.) Если открыто несколько экземпляров какого-либо приложения, то каждый из этих экземпляров является отдельным процессом одной и той же программы. Например, если вы откроете одновременно несколько окон в своем браузере, то все они будут представлять собой разные процессы программы браузера. Пример нескольких одновременно выполняющихся процессов калькулятора приведен на рис. 15.1.

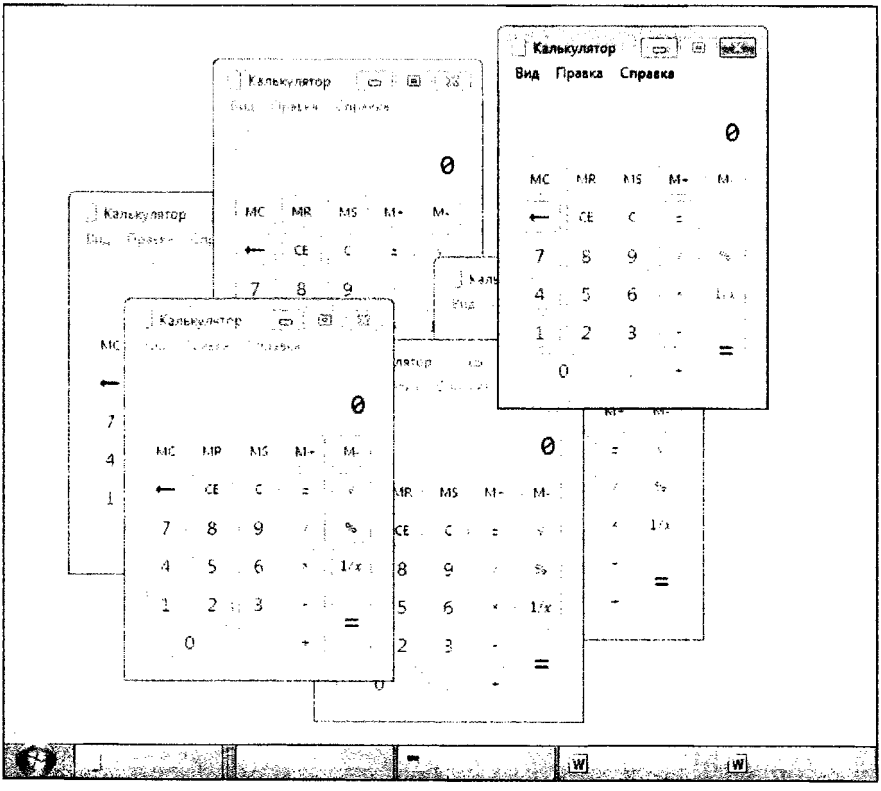


Рис. 15.1. Шесть одновременно выполняющихся процессов одного и того же приложения калькулятора

Каждый процесс может иметь несколько потоков выполнения. В отличие от потоков, процесс не может непосредственно читать и записывать переменные другого процесса. Если выполнение многопоточной программы можно образно представлять себе так, как будто оно управляется несколькими пальцами, скользящими по строкам исходного кода, то выполнение нескольких процессов одной и той же программы можно уподобить одновременному выполнению отдельных экземпляров этой программы, которые вы раздали своим друзьям. Каждый из вас может независимо выполнять одну и ту же программу.

Если вы хотите запустить внешнюю программу из сценария на Python, то передайте имя этой программы функции `subprocess.Popen()`. (Чтобы увидеть имя приложения в Windows, щелкните на значке приложения в меню Пуск правой кнопкой мыши и выберите в открывшемся контекстном меню пункт **Свойства**. В OS X, чтобы увидеть путь к исполняемому файлу, щелкните на значке приложения при нажатой клавише <Ctrl> и выберите пункт **Show Package Contents** (Показать содержимое пакета.) Функция `Popen()` немедленно выполнит возврат. Имейте в виду, что запущенная таким способом программа выполняется в потоке, отличном от того потока, в котором выполняется ваша программа на Python.

На компьютере, работающем под управлением Windows, введите в интерактивной оболочке следующие команды.

```
>>> import subprocess
>>> subprocess.Popen('C:\\Windows\\System32\\calc.exe')
<subprocess.Popen object at 0x0000000003055A58>
```

На компьютере, работающем под управлением Ubuntu Linux, выполните следующие команды.

```
>>> import subprocess
>>> subprocess.Popen('/usr/bin/gnome-calculator')
<subprocess.Popen object at 0x7f2bcf93b20>
```

На компьютере, работающем под управлением OS X, это делается немного иначе (см. раздел "Открытие файлов программами по умолчанию").

Возвращаемое значение представляет собой объект `Popen`, имеющий два полезных метода: `poll()` и `wait()`.

Образно говоря, метод `poll()` словно спрашивает вашего друга, завершил ли он выполнение кода, который вы ему предоставили. Метод `poll()` возвращает значение `None`, если в момент его вызова процесс все еще выполняется. Если же работа программы к этому моменту завершена, то он возвращает целочисленный *код выхода* процесса. Код выхода используется в качестве индикатора того, завершился ли процесс без ошибок (код выхода

0) или же его завершение было обусловлено ошибкой (ненулевой код выхода, обычно — 1, но могут быть и другие значения, в зависимости от программы).

Метод `wait()` словно выжидает, пока ваш друг не закончит работу со своим кодом, прежде чем предоставить вам возможность работать со своим. Метод `wait()` блокируется до завершения запущенного процесса. Это может быть полезным, если вы хотите, чтобы ваша программа выждала, пока пользователь не закончит работать с другой программой. Возвращаемым значением метода `wait()` является целочисленный код выхода.

Если вы используете Windows, то введите в интерактивной оболочке приведенный ниже код. Обратите внимание на то, что метод `wait()` блокируется до тех пор, пока вы не закончите работу с запущенной программой калькулятора.

```
❶ >>> calcProc =
  ↳ subprocess.Popen('c:\\Windows\\System32\\calc.exe')
❷ >>> calcProc.poll() == None
  True
❸ >>> calcProc.wait()
  0
  >>> calcProc.poll()
  0
```

Сначала мы открываем процесс калькулятора ❶. Затем мы проверяем, возвращает ли метод `poll()` значение `None` ❷. Соблюдение этого условия означает, что процесс все еще выполняется. После этого мы закрываем программу калькулятора и вызываем метод `wait()` для уже завершенного процесса ❸. Теперь оба метода, `wait()` и `poll()`, возвращают значение 0, указывающее на то, что выполнение процесса завершилось без ошибок.

Передача аргументов командной строки функции `Popen()`

Процессам, создаваемым с помощью функции `Popen()`, можно передавать аргументы командной строки. Для этого функции `Popen()` следует передать список в качестве единственного аргумента. Первой строкой в этом списке должно быть имя исполняемого файла программы, которую вы хотите запустить; все последующие строки — это аргументы командной строки, которые будут переданы программе при ее запуске. В конечном счете этот список будет значением `sys.argv` для запущенной программы.

Приложения с графическим интерфейсом пользователя (GUI) в основном используют аргументы командной строки менее интенсивно, чем приложения с текстовым пользовательским интерфейсом. Однако большинство GUI-приложений будут принимать одиночный аргумент в виде имении файла, который должен быть открыт приложением сразу после его запуска.

Например, если вы используете Windows, то создайте обычный текстовый файл *C:\hello.txt* и введите в интерактивной оболочке следующую команду.

```
>>> subprocess.Popen(['C:\\Windows\\notepad.exe',
↳ 'C:\\hello.txt'])
<subprocess.Popen object at 0x00000000032DCEB8>
```

Эта команда не только запустит приложение Notepad, но и немедленно откроет в нем файл *C:\hello.txt*.

Планировщик заданий Windows, система инициализации launchd и демон-планировщик cron

Если вы хорошо знакомы с компьютерами, то, возможно, вам известно, какие функции выполняют планировщик заданий в Windows, система инициализации `launchd` в OS X и демон-планировщик задач `cron` в Linux. Эти надежные и хорошо документированные инструментальные средства обеспечивают возможность периодического выполнения заданий в определенное время. Более подробную информацию о них вы сможете получить, воспользовавшись ссылками на соответствующие руководства на сайте `http://nostarch.com/automatestuff/`.

Встроенный планировщик заданий операционной системы избавляет от необходимости написания собственных программ, способных запускать задачи по расписанию. Тем не менее, если нужно всего лишь приостановить выполнение программы на короткое время, используйте функцию `time.sleep()`. Кроме того, вместо использования планировщика заданий операционной системы ваш код может организовать выполнение цикла до наступления определенных даты и времени, вызывая функцию `time.sleep(1)` на каждой итерации цикла.

Открытие веб-сайтов с помощью Python

Вместо того чтобы открывать приложение браузера путем вызова функции `subprocess.Popen()`, можно открывать браузер с переходом на сайт по заданному адресу, используя функцию `webbrowser.open()`. Для получения более подробной информации по этому вопросу обратитесь к разделу "Проект: программа *mapIt.py* с модулем `webbrowser`" главы 11.

Запуск других сценариев Python

Можно запускать выполнение сценариев под управлением Python в отдельном процессе точно так же, как любое другое приложение. Для этого нужно лишь передать функции `Popen()` в качестве аргументов имя

исполняемого файла Python (*python.exe*) и имя файла сценария с расширением *.py*, который вы хотите выполнить. Например, следующая команда выполнит сценарий *hello.py*, рассмотренный в главе 1.

```
>>> subprocess.Popen(['C:\\python34\\python.exe', 'hello.py'])
<subprocess.Popen object at 0x000000000331CF28>
```

Передавайте функции `Popen()` список, содержащий строку с путем доступа к исполняемому файлу Python и строку с именем файла сценария. Если запускаемый вами сценарий сам нуждается в аргументах командной строки, то добавьте их в список после имени файла сценария. Расположение исполняемого файла интерпретатора Python зависит от платформы: Windows – *C:\python34\python.exe*, OS X – */Library/Frameworks/Python.framework/Versions/3.3/bin/python3*, Linux – */usr/bin/python3*.

В отличие от программ на Python, импортированных в качестве модулей, программа на Python, запущенная вашей программой, выполняется в отдельном процессе, и обе программы не смогут обмениваться между собой своими переменными.

Открытие файлов программами по умолчанию

Двойной щелчок на значке файла с расширением *.txt* на вашем компьютере позволяет автоматически запустить приложение, ассоциированное с этим расширением. На вашем компьютере такие ассоциированные программы уже должны быть заданы и для ряда других расширений имен файлов. Python также может открывать файлы подобным образом с помощью функции `Popen()`.

В каждой операционной системе имеется программа, выполняющая те же функции, что и двойной щелчок на значке документа для его открытия. В Windows – это программа `start`, в OS – программа `open`, в Ubuntu Linux – программа `see`. Введите в интерактивной оболочке следующий код, передавая функции `Popen()` одну из строк `'start'`, `'open'` и `'see'`, в зависимости от того, какая операционная система установлена на вашем компьютере.

```
>>> fileObj = open('hello.txt', 'w')
>>> fileObj.write('Здравствуй, мир!')
12
>>> fileObj.close()
>>> import subprocess
>>> subprocess.Popen(['start', 'hello.txt'], shell=True)
```

Сначала мы записываем строку `Здравствуй, мир!` в новый файл *hello.txt*. Затем вызываем функцию `Popen()`, передавая ей список, содержащий имя

программы (в данном случае — `'start'` для Windows) и имя файла. Кроме того, мы передаем именованный аргумент `shell=True`, который требуется лишь в случае Windows. Операционной системе известны все ассоциативные соответствия программ расширениям имен файлов, и она может самостоятельно определить, какую программу следует выполнить, например Notepad.exe для обработки файла *hello.txt*.

Философия Unix

Программы, хорошо приспособленные к запуску других программ, предлагают гораздо более широкие возможности, чем просто их код сам по себе. Философия Unix зиждется на ряде принципов проектирования программного обеспечения, установленных разработчиками операционной системы Unix (на основе которой созданы современные операционные системы Linux и OS X). В соответствии с этой философией лучше писать небольшие программы узкоспециального назначения, которые могут взаимодействовать с другими программами, чем крупные приложения с богатыми возможностями. Меньшие программы более понятны, а простота взаимодействия с ними позволяет использовать их в качестве строительных кирпичиков для создания более мощных приложений.

Такому же подходу следуют и приложения для смартфонов. Если приложению-справочнику необходимо отображать для пользователя направление движения к указанному кафе или ресторану, то разработчики не должны заново изобретать велосипед, пытаясь написать собственный код для работы с картами местности. Такое приложение-справочник просто должно запускать отдельное приложение-карту, передавая ему адрес кафе, как это делает ваш код на Python, вызывая какую-либо функцию и передавая ей аргументы.

Программы на Python, которые вы пишете, работая с этой книгой, в основном укладываются в данную философию, особенно в одном важном отношении: вместо вызовов функции `input()` они используют аргументы командной строки. Если вся необходимая для работы программы информация может быть заранее предоставлена, то ее предпочтительно передавать в виде аргументов командной строки, а не ждать, пока пользователь ее введет. Тем самым аргументы командной строки могут не только вводиться пользователем, но и предоставляться другой программой. Благодаря такому подходу, обеспечивающему широкие возможности взаимодействия, ваша программа может выступать в роли многократно используемого компонента других программ.

Единственным исключением являются пароли, передача которых в качестве аргументов командной строки нежелательна, поскольку в этом случае пароль попадает в историю команд и сохраняется в ней. Поэтому в тех случаях, когда требуется вводить пароли, для этой цели лучше вызывать функцию `input()`.

Более подробно о философии Unix можно прочитать в статье Википедии: `https://ru.wikipedia.org/wiki/Философия_Unix/`.

В случае OS X программа open используется для открытия как документов, так и программ. Введите в интерактивной оболочке следующий код, если вы работаете на компьютере Mac.

```
>>> subprocess.Popen(['open', '/Applications/Calculator.app/'])
<subprocess.Popen object at 0x10202ff98>
```

В результате выполнения этой команды должно открыться приложение Calculator.

Проект: простая программа обратного отсчета времени

Найти простую программу для обратного отсчета времени не менее трудно, чем приложение, выполняющее функции хронометра. Давайте напишем программу, которая ведет обратный отсчет времени и сообщает о его завершении звуковым сигналом.

Вот что должна делать эта программа при высокоуровневом рассмотрении:

- вести обратный отсчет, начиная от 60;
- воспроизводить звуковой файл (*alarm.wav*), когда счетчик достигает нулевого значения.

Отсюда следует, что код программы должен выполнять следующие операции:

- создавать паузу длительностью 1 секунда перед выводом очередного значения счетчика, вызывая функцию `time.sleep()`;
- вызывать функцию `subprocess.Popen()` для открытия звукового файла с помощью программы по умолчанию.

Откройте новое окно в файловом редакторе и сохраните его в файле *countdown.py*.

Шаг 1. Обратный отсчет

Этой программе понадобится модуль `time` для вызова функции `time.sleep()` и модуль `subprocess` для вызова функции `subprocess.Popen()`. Введите следующий код в файл *countdown.py*.

```
#! python3
# countdown.py - Простой сценарий обратного отсчета.

import time, subprocess
```

```
❶ timeLeft = 60
  while timeLeft > 0:
❷     print(timeLeft, end='')
❸     time.sleep(1)
❹     timeLeft = timeLeft - 1

  # TODO: Воспроизвести звуковой файл по завершении
  # обратного отсчета.
```

Импортировав модули `time` и `subprocess`, мы создаем переменную `timeLeft`, в которой будет храниться количество секунд, оставшихся до окончания отсчета ❶. В данной программе обратный отсчет начинается от значения 60, однако вы можете изменить его в соответствии со своими потребностями или же сделать так, чтобы программа получала его в виде аргумента командной строки.

На каждой итерации цикла `while` отображается текущее значение обратного счетчика ❷, выдерживается пауза длительностью 1 секунда ❸ и декрементируется значение переменной `timeLeft` ❹. Цикл продолжается до тех пор, пока значение переменной `timeLeft` больше `0`. Как только это условие перестает соблюдаться, обратный отсчет прекращается.

Шаг 2. Воспроизведение звукового файла

Несмотря на то что существуют модули сторонних разработчиков, позволяющие воспроизводить звуковые файлы различных форматов, самый простой и быстрый способ заключается в запуске приложения, уже используемого пользователем для этих целей. Операционная система сама определит по расширению имени файла *.wav*, какое приложение следует запустить для воспроизведения файла. Разумеется, это может быть не только файл формата *.wav*, но и файлы других аналогичных форматов, таких как *.mp3* или *.ogg*.

В качестве файла, который воспроизводится в конце работы программы обратного отсчета, вы можете использовать любой звуковой файл, имеющийся на вашем компьютере, или загрузить файл *alarm.wav*, посетив сайт `http://nostarch.com/automatestuff/`.

Добавьте в программу следующий код.

```
#! python3
# countdown.py - Простой сценарий обратного отсчета.

import time, subprocess

--пропущенный код--
```

```
# Воспроизведение звукового файла по завершении
# обратного отсчета.
subprocess.Popen(['start', 'alarm.wav'], shell=True)
```

По завершении цикла `while` пользователь оповещается об окончании работы программы воспроизведением файла *alarm.wav* (или другого выбранного вами файла). В случае Windows не забудьте включить строку `'start'` в список, передаваемый функции `Popen()`, и одновременно передать именованный аргумент `shell=True`. В случае OS X передайте строку `'open'` вместо строки `'start'` и удалите аргумент `shell=True`.

Вместо того чтобы воспроизводить звуковой файл, можете подготовить текстовый файл с сообщением наподобие *Перерыв закончился!* и использовать функцию `Popen()` для его открытия по завершении обратного отсчета. В результате этого откроется окно предупреждения с текстом сообщения. Аналогичным образом можно использовать вызов функции `webbrowser.open()`, которая по завершении обратного отсчета будет открывать конкретный веб-сайт. В отличие от некоторых бесплатных программ подобного рода, которые можно найти в Интернете, в своей программе обратного отсчета вы сможете использовать любой звуковой файл, который только пожелаете!

Идеи относительно создания аналогичных программ

Обратный отсчет — это простейший способ организовать паузу, по окончании которой программа сможет продолжить выполнение. Эта возможность может быть использована в ряде других приложений, подобных описанным ниже.

- Используйте функцию `time.sleep()` для предоставления пользователю возможности отменить какое-либо действие, например удаление файлов, нажав комбинацию клавиш <Crl+C>. Ваша программа может выводить сообщение "Для отмены нажмите комбинацию клавиш <Ctrl+C>", а затем обрабатывать исключения `KeyboardInterrupt` с помощью инструкций `try` и `except`.
- Для организации обратного отсчета в течение длительных промежутков времени можно использовать объекты `timedelta`, чтобы отмерять количество дней, часов, минут и секунд, оставшихся до наступления определенного события (например, дня рождения или юбилея) в будущем.

Резюме

Эпоха Unix (1 января 1970 года, полночь, UTC) — это стандартная точка отсчета времени во многих языках программирования, включая Python. В то время как функция `time.time()` возвращает временную метку (т.е. вещественное число, представляющее количество секунд, истекших с момента наступления эпохи Unix), модуль `datetime` больше приспособлен для выполнения арифметических действий с датами, а также форматирования или анализа связанных с информацией о датах строк.

Функция `time.sleep()` блокируется (т.е. не выполняет возврат) на определенное количество секунд, что можно использовать для добавления пауз в программу. Но если вы хотите запланировать запуск своих программ на определенный момент времени, то инструкции, предоставленные на сайте `http://nostarch.com/automatestuff/`, подскажут вам, как как обеспечить это с помощью готового планировщика заданий, предоставляемого вашей операционной системой.

Модуль `threading` используется для создания нескольких потоков выполнения, что может пригодиться вам для загрузки многочисленных файлов или одновременного выполнения других задач. Однако при этом вы должны убедиться в том, что потоки читают и записывают только локальные переменные, иначе вы можете столкнуться с проблемами параллелизма.

Наконец, ваши программы на Python могут запускать другие приложения с помощью функции `subprocess.Popen()`. Функции `Popen()` могут передаваться аргументы командной строки, указывающие на документы, которые должны быть открыты в запускаемом приложении. Суть другого возможного варианта состоит в том, чтобы использовать совместно с функцией `Popen()` одну из программ `start`, `open` и `see` и позволить операционной системе автоматически определить, в каком приложении должен быть открыт документ, основываясь на заданных для компьютера файловых ассоциациях. Взаимодействуя с другими приложениями, установленными на компьютере, ваши программы на Python смогут воспользоваться их возможностями для автоматизации возникающих перед вами задач.

Контрольные вопросы

1. Что такое эпоха Unix?
2. Какая функция возвращает количество секунд, истекших с момента наступления эпохи Unix?
3. Как организовать в программе паузу длительностью ровно 5 секунд?
4. Что возвращает функция `round()`?
5. В чем состоит разница между объектами `datetime` и `timedelta`?

6. Предположим, у вас есть функция `spam()`. Как вызвать эту функцию и выполнить ее код в отдельном потоке?
7. Что нужно сделать для того, чтобы избежать проблем параллелизма при работе с несколькими потоками?
8. Каким образом ваша программа на Python может выполнить программу *calc.exe*, находящуюся в папке *C:\Windows\System32*?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Приукрашенный хронометр

Расширьте проект программы-хронометра, рассмотренный в начале главы, "украсив" вывод за счет использования в этой программе строковых методов `rjust()` и `ljust()` (эти методы обсуждались в главе 6). Вместо прежнего вывода

```
Замер #1: 3.56 (3.56)
Замер #2: 8.63 (5.07)
Замер #3: 17.68 (9.05)
Замер #4: 19.11 (1.43)
```

вы должны получить вывод следующего вида:

```
Замер # 1:   3.56 (  3.56)
Замер # 2:   8.63 (  5.07)
Замер # 3:  17.68 (  9.05)
Замер # 4:  19.11 (  1.43)
```

Обратите внимание на то, что для вызова вышеупомянутых строковых методов вам понадобятся строковые версии целочисленных и вещественных переменных `lapNum`, `lapTime` и `totalTime`.

На следующем этапе доработки программы используйте модуль `pyperclip`, рассмотренный в главе 6, для копирования текстового вывода в буфер обмена, благодаря чему пользователь сможет быстро вставить вывод в текстовый файл или в сообщение электронной почты.

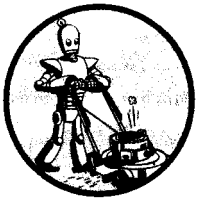
Загрузка веб-комиксов по расписанию

Напишите программу, которая проверяет несколько сайтов веб-комиксов и автоматически загружает изображения в случае обновления комикса со времени последнего посещения сайта программой. Планировщик

вашей операционной системы (планировщик заданий — в Windows, система инициализации *launchd* — в OS X и демон-планировщик *cron* — в Linux) может выполнять вашу программу ежедневно по одному разу. Программа на Python может загружать комикс, а затем копировать его на ваш рабочий стол, где его можно будет легко найти. Это избавит вас от необходимости самостоятельно посещать сайты для того, чтобы проверить, обновлялись ли комиксы. (Список сайтов с веб-комиксами доступен по адресу `http://nostarch.com/automatestuff/`.)

16

ОТПРАВКА СООБЩЕНИЙ ЭЛЕКТРОННОЙ ПОЧТЫ И ТЕКСТОВЫХ СООБЩЕНИЙ



Просмотр сообщений электронной почты и ответы на них отнимают огромное количество времени. Конечно, невозможно написать программу, которая обрабатывала бы электронную почту вместо вас, поскольку на каждое сообщение приходится отвечать по-разному. И все же имеется множество задач, связанных с обработкой электронной почты, которые поддаются автоматизации, коль скоро вам известно, как написать программу, способную отправлять и получать электронные письма.

Предположим, у вас есть электронная таблица с данными о клиентах, и вы хотите отправить каждому из них письмо, форма которого зависит как от возраста клиента, так и от другой конкретной информации. С поиском подходящей коммерческой программы у вас могут возникнуть трудности, но, к счастью, вы можете написать для этих целей собственную программу, которая позволит сэкономить массу времени, избавив вас от многократного копирования и вставки формы письма.

Кроме того, можно написать программы для отправки сообщений почты и SMS, доставляющих вам важные уведомления, даже если вы находитесь вдали от компьютера. Если вы автоматизировали задачу, которая может выполняться несколько часов, то вряд ли захотите контролировать компьютер каждые пять минут, чтобы проверить, не завершилась ли она. Вместо этого программа может отправить на ваш телефон текстовое сообщение сразу же после того, как будет выполнена, что даст вам возможность заняться другими делами.

SMTP

Во многом подобно тому, как HTTP используется в качестве протокола для пересылки веб-страниц через Интернет, сетевой протокол SMTP (Simple Mail Transfer Protocol – простой протокол передачи почты) используется

для передачи сообщений электронной почты. SMTP определяет порядок форматирования и шифрования сообщений электронной почты, стандартизирует процесс их ретрансляции между почтовыми серверами, а также описывает другие детали обработки сообщений компьютером, когда вы щелкаете на кнопке Отправить. Однако знать технические подробности вам вовсе не обязательно, поскольку для вас все сводится к работе с несколькими функциями модуля `smtplib`, предоставляемого Python.

Протокол SMTP отвечает лишь за отправку почты. Для получения электронной почты используется другой протокол – IMAP, описанный далее.

Отправка электронной почты

Возможно, вы привыкли отправлять электронную почту с помощью таких приложений, как Outlook или Thunderbird, или посредством таких сайтов, как Gmail или Yahoo! Mail. К сожалению, Python не предлагает такой же привлекательный графический интерфейс пользователя, как эти службы. Вместо этого для реализации всех основных действий протокола SMTP необходимо вызывать функции, как показано в следующем примере (интерактивная оболочка).

Примечание

Не пытайтесь выполнить этот пример в IDLE; он не будет работать, поскольку `smtp.example.com`, `bob@example.com` `MY_SECRET_PASSWORD` *и* `alice@example.com` *– это только заместители. Данный код дает вам лишь общее представление о том, как происходит отправка электронной почты с помощью Python.*

```
>>> import smtplib
>>> smtpObj = smtplib.SMTP('smtp.example.com', 587)
>>> smtpObj.ehlo()
(250, b'mx.example.com at your service, [216.172.148.131]\nSIZE
35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')
>>> smtpObj.starttls()
(220, b'2.0.0 Ready to start TLS')
>>> smtpObj.login('bob@example.com', 'МОЙ_СЕКРЕТНЫЙ_ПАРОЛЬ')
(235, b'2.7.0 Accepted')
>>> smtpObj.sendmail('bob@example.com', 'alice@example.com',
↳ 'Subject: So long.\nDear Alice, so long and thanks for all the
↳ fish. Sincerely, Bob')
{}
>>> smtpObj.quit()
(221, b'2.0.0 closing connection ko10sm23097611pbd.52 - gsmtp')
```

В следующем разделе мы тщательно проанализируем каждый шаг, подставляя вместо заместителей ваши реальные данные для установления

соединения и входа на SMTP-сервер, отправки сообщения и разрыва соединения с сервером.

Установление соединения с SMTP-сервером

Если вам когда-либо приходилось настраивать Thunderbird, Outlook или любую другую программу для подключения к своей учетной записи электронной почты, то, вероятно, вы знакомы с процедурой конфигурирования SMTP-сервера и порта. Для каждого провайдера почтовых услуг эти настройки будут разными, однако, выполнив в Интернете поиск по ключевым словам *<ваш_провайдер> настройки smtp*, вы сможете настроить соединение с сервером через нужный порт.

Как правило, доменное имя SMTP-сервера будет совпадать с именем домена вашего почтового провайдера, дополненным префиксом `smtp`. Например, в случае Gmail имя SMTP-сервера — `smtp.gmail.com`. В табл. 16.1 приведены имена провайдеров электронной почты и их SMTP-серверов. (Порт — это целочисленное значение, почти всегда равное 587, которое используется протоколом безопасности транспортного уровня TLS.)

Таблица 16.1. Провайдеры электронной почты и их SMTP-серверы

Провайдер	Доменное имя SMTP-сервера
Gmail	`smtp.gmail.com`
Outlook.com/Hotmail.com	`smtp-mail.outlook.com`
Yahoo Mail	`smtp.mail.yahoo.com`
AT&T	`smpt.mail.att.net` (порт 465)
Comcast	`smtp.comcast.net`
Verizon	`smtp.verizon.net` (порт 465)

Как только вы определите имя домена и порт, которые будут использоваться для установления соединения с вашим почтовым провайдером, создайте объект SMTP, вызвав функцию `smptlib.SMTP()` с передачей ей двух аргументов: строки, содержащей имя домена, и целого числа, идентифицирующего порт. Объект SMTP представляет соединение с почтовым SMTP-сервером и располагает методами для отправки сообщений электронной почты. Например, следующий вызов создает объект SMTP для подключения к Gmail.

```
>>> smtpObj = smtplib.SMTP('smtp.gmail.com', 587)
>>> type(smtpObj)
<class 'smtplib.SMTP'>
```

Результат выполнения команды `type(smtpObj)` демонстрирует, что в переменной `smtpObj` хранится объект SMTP. Этот объект потребуется вам для вызова методов, которые позволят выполнять процедуру входа и отправлять сообщения. Неудачный вызов `smptlib.SMTP()` может означать, что ваш SMTP-сервер не поддерживает протокол TLS через порт 587. В таком случае вам потребуется создать объект SMTP, используя вызов `smtplib.SMTP_SSL()` и порт 465.

```
>>> smtpObj = smtplib.SMTP_SSL('smtp.gmail.com', 465)
```

Примечание

В случае отсутствия подключения к Интернету Python сгенерирует исключение `socket.gaierror: [Errno 11004] getaddrinfo failed` *или аналогичное ему.*

Для ваших программ различие между протоколами TLS и SSL не является существенным. Чтобы подключиться к своему SMTP-серверу, вам достаточно лишь знать, какой именно стандарт шифрования он использует. Во всех последующих примерах, выполняемых в интерактивной оболочке, переменная `smtpObj` содержит объект SMTP, возвращаемый вызовом функции `smtplib.SMTP()` или `smtplib.SMTP_SSL()`.

Отправка строки приветствия SMTP-серверу

Создав объект SMTP, начните процедуру рукопожатия с почтовым сервером SDMTP, "поздоровавшись" с ним с помощью метода `ehlo()`. Отправка приветствия является в SMTP первым шагом процедуры установления соединения с сервером. Знать все детали соответствующих протоколов вам вовсе необязательно. Вам лишь надо твердо запомнить, что первое, что вы должны сделать после того, как создадите объект SMTP, – это вызвать метод `ehlo()`, иначе все последующие вызовы методов будут приводить к ошибке. Вот пример вызова метода `ehlo()` и возвращаемого им значения:

```
>>> smtpObj.ehlo()
(250, b'mx.google.com at your service, [216.172.148.131]\nSIZE 35882577\
n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nCHUNKING')
```

Если первым элементом возвращенного кортежа является целое число `250` (код успешного завершения операции в SMTP), то это означает, что процедура приветствия оказалась успешной.

Начало TLS-шифрования

Если вы подключаетесь к серверу SMTP через порт 587 (т.е. используете TLS-шифрование), то следующим должен быть вызван метод `starttls()`. Этот шаг активизирует шифрование для вашего соединения. Если вы подключаетесь к порту 465 (используя SSL), то шифрование уже настроено, и этот шаг нужно опустить.

Вот пример вызова метода `starttls()`:

```
>>> smtpObj.starttls()
(220, b'2.0.0 Ready to start TLS')
```

Метод `starttls()` переводит ваше SMTP-соединение в режим TLS. Возвращаемое этим методом значение `220` означает, что сервер готов к работе.

Выполнение процедуры входа на SMTP-сервер

Настроив зашифрованное соединение с SMTP-сервером, вы можете выполнить процедуру входа, указав свое имя пользователя (обычно это ваш адрес электронной почты) и пароль при вызове метода `login()`.

```
>>> smtpObj.login('my_email_address@gmail.com',
                  'MY_SECRET_PASSWORD')
(235, b'2.7.0 Accepted')
```

Использование паролей приложений в Gmail

В Gmail предусмотрено дополнительное средство обеспечения безопасности для учетных записей Google — так называемые *пароли приложений*. В случае получения сообщения об ошибке `Application-specific password required` при попытке выполнения вашей программой процедуры входа необходимо установить один из этих паролей для своего сценария на языке Python. Более подробные указания относительно того, как установить пароль приложения для своей учетной записи Google, вы найдете по адресу `http://nostarch.com/automatestuff/`.

В качестве первого аргумента методу передается адрес электронной почты, в качестве второго — пароль. Возвращаемое значение `235` говорит о том, что процедура аутентификации была успешно пройдена. В случае ввода неверного пароля Python возбудит исключение `smtplib.SMTPAuthenticationError`.

Предупреждение

Будьте осторожны, помещая пароли в исходный код. Если кто-то посторонний скопирует вашу программу, то он получит доступ к вашей учетной записи! В этом отношении имеет смысл вызывать метод `input()` *и предоставлять пользователю возможность ввода пароля. Возможно, необходимость вводить пароль каждый раз при запуске программы будет доставлять некоторое неудобство, но это позволит не допустить хранения вашего пароля в компьютере в незашифрованном файле, до которого легко может добраться хакер или вор, похитивший ваш ноутбук.*

Отправка почты

Выполнив вход на SMTP-сервер своего почтового провайдера, можете вызвать метод `sendmail()` для фактической отправки сообщения электронной почты. Вызов метода `sendmail()` выглядит так.

```
>>> smtpObj.sendmail('my_email_address@gmail.com',
↳ 'recipient@example.com', 'Subject: So long.\nDear Alice, so long
↳ and thanks for all the fish. Sincerely, Bob')
{}
```

Метод `sendmail()` принимает три аргумента:

- ваш адрес электронной почты в виде строки (для поля "От" адреса);
- адрес электронной почты получателя в виде строки или списка строк в случае нескольких получателей (для поля "Кому" адреса);
- тело сообщения в виде строки.

Строка с телом сообщения должна начинаться с текста `'Тема: \n'`, представляющего строку с темой сообщения. Символ новой строки `'\n'` отделяет строку темы от основного текста сообщения.

Возвращаемым значением метода `sendmail()` является словарь. В нем будет содержаться по одной паре "ключ–значение" для каждого из получателей, кому *не удалось* доставить сообщение. Пустой словарь означает, что сообщение было *успешно* доставлено всем получателям.

Разрыв соединения с SMTP-сервером

Отправив электронную почту, не забудьте вызвать метод `quit()`. Это приведет к отключению вашей программы от SMTP-сервера.

```
>>> smtpObj.quit()
(221, b'2.0.0 closing connection ko10sm23097611pbd.52 - gsmtp')
```

Возвращаемое значение 221 означает завершение сеанса связи.

Обзор всех шагов процедуры, которая должна быть выполнена для установления соединения и входа на сервер, отправки электронной почты и разрыва соединения, содержится в разделе "Отправка электронной почты".

IMAP

Подобно тому как SMTP — это протокол, управляющий отправкой электронной почты, протокол IMAP (Internet Message Access Protocol — *протокол прикладного уровня для доступа к электронной почте*) определяет порядок обмена данными с сервером почтового провайдера для извлечения электронных писем, отправленных в ваш адрес. Python поставляется вместе с модулем `imaplib`, но в действительности проще использовать модуль `imapclient` сторонних разработчиков. В данной главе вы познакомитесь с основами использования модуля IMAPClient; полная документация находится по адресу `http://imapclient.readthedocs.org/`.

Модуль `imapclient` загружает электронную корреспонденцию из хранилища на IMAP-сервере в довольно сложном формате. Вероятнее всего, вы захотите конвертировать письма из этого формата в простые строковые значения. Всю трудоемкую работу по синтаксическому анализу этих электронных сообщений вместо вас выполнит модуль `pyzmail`. Полную документацию по модулю PyzMail можно найти по адресу `http://www.magiksys.net/pyzmail/`.

Установите модули `imapclient` и `pyzmail` из командной строки. Описание процедуры установки модулей, разработанных сторонними компаниями, приведено в приложении А.

Извлечение и удаление сообщений электронной почты с помощью IMAP

Поиск и извлечение сообщений электронной почты в Python представляет собой многоэтапный процесс, требующий использования сторонних модулей `imapclient` и `pyzmail`. Чтобы вы могли получить общее представление об этом процессе, ниже приведен пример, демонстрирующий прохождение каждого его этапа: входа на IMAP-сервер, поиска сообщений, извлечения сообщений и последующего извлечения из них текста.

```
>>> import imapclient
>>> imapObj = imapclient.IMAPClient('imap.gmail.com', ssl=True)
>>> imapObj.login('my_email_address@gmail.com',
'МОЙ_СЕКРЕТНЫЙ_ПАРОЛЬ')
'my_email_address@gmail.com Jane Doe authenticated (Success)'
>>> imapObj.select_folder('INBOX', readonly=True)
```

```
>>> UIDs = imapObj.search(['SINCE 05-Jul-2014'])
>>> UIDs
[40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040, 40041]
>>> rawMessages = imapObj.fetch([40041], ['BODY[]', 'FLAGS'])
>>> import pyzmail
>>> message =
↳pyzmail.PyzMessage.factory(rawMessages[40041]['BODY[]'])
>>> message.get_subject()
'Hello!'
>>> message.get_addresses('from')
[('Edward Snowden', 'esnowden@nsa.gov')]
>>> message.get_addresses('to')
[(Jane Doe', 'jdoe@example.com')]
>>> message.get_addresses('cc')
[]
>>> message.get_addresses('bcc')
[]
>>> message.text_part != None
True
>>> message.text_part.get_payload().decode(message.text_part.charset)
'Follow the money.\r\n\r\n-Ed\r\n'
>>> message.html_part != None
True
>>> message.html_part.get_payload().decode(message.html_part.charset)
'<div dir="ltr"><div>So long, and thanks for all the fish!
<br><br></div>-Al<br></div>\r\n'
>>> imapObj.logout()
```

Вам не нужно запоминать эти шаги. После того как мы подробно разберем каждый шаг этой процедуры, вы сможете вернуться к этому обзорному примеру, чтобы освежить этот материал в памяти.

Соединение с IMAP-сервером

Точно так же, как для соединения с SMTP-сервером и отправки сообщений электронной почты требуется объект SMTP, для установления соединения с IMAP-сервером и получения сообщений требуется объект IMAPClient. Первое, что вам для этого потребуется, — это доменное имя IMAP-сервера вашего почтового провайдера. Это имя будет отличаться от доменного имени SMTP-сервера. В табл. 16.2 приведены имена некоторых популярных провайдеров электронной почты и их IMAP-серверов.

Таблица 16.2. Провайдеры электронной почты и их IMAP-серверы

Провайдер	Доменное имя IMAP-сервера
Gmail	imap.gmail.com
Outlook.com/Hotmail.com	imap-mail.outlook.com

Окончание табл. 16.2

Провайдер	Доменное имя IMAP-сервера
Yahoo Mail	`imap.mail.yahoo.com`
AT&T	`imap.mail.att.net`
Comcast	`imap.comcast.net`
Verizon	`incoming.verizon.net`

Определив доменное имя IMAP-сервера, вызовите функцию `imapclient.IMAPClient()` для создания объекта IMAPClient. Для большинства почтовых провайдеров необходимо SSL-шифрование, поэтому передайте функции именованный аргумент `ssl=True`. Введите в интерактивной оболочке следующие команды (используя доменное имя своего провайдера).

```
>>> import imapclient
>>> imapObj = imapclient.IMAPClient('imap.gmail.com', ssl=True)
```

Во всех примерах для интерактивной оболочки, приводимых в последующих разделах, переменная `imapObj` содержит объект IMAPClient, возвращаемый функцией `imapclient.IMAPClient()`. В данном контексте *клиент* — это объект, который соединяется с сервером.

Вход в учетную запись на IMAP-сервере

Создав объект IMAPClient, вызовите его метод `login()`, передав ему имя пользователя (обычно это ваш адрес электронной почты) и пароль в виде строк.

```
>>> imapObj.login('my_email_address@gmail.com',
                  'МОЙ_СЕКРЕТНЫЙ_ПАРОЛЬ')
'my_email_address@gmail.com Jane Doe authenticated (Success)'
```

Предупреждение

Запомните: нельзя записывать пароль непосредственно в код! Вместо этого следует проектировать программу так, чтобы она принимала пароль, возвращаемый функцией `input()`.

Если IMAP-сервер отвергнет предложенную ему комбинацию *имя_пользователя/пароль*, то Python возбудит исключение `imaplib.error`. В случае учетных записей Gmail вам, возможно, придется использовать пароль приложения (более подробно об этом читайте в разделе "Использование паролей приложений в Gmail).

Поиск сообщений

Фактическое извлечение сообщений, которое становится возможным после выполнения процедуры входа, выполняется в два этапа: сначала нужно выбрать папку, в которой будет выполняться поиск, а затем вызвать метод `search()` объекта IMAPClient, передав ему строку ключей поиска IMAP.

Выбор папки

Почти в каждой учетной записи по умолчанию имеется папка INBOX, но вы можете получить список папок, вызвав метод `list_folders()` объекта IMAPClient. Данный вызов возвращает список кортежей. Каждый кортеж содержит информацию об одной папке. Продолжите пример, введя в интерактивной оболочке следующие команды.

```
>>> import pprint
>>> pprint.pprint(imapObj.list_folders())
[(('\\HasNoChildren',), '/', 'Drafts'),
(('\\HasNoChildren',), '/', 'Filler'),
(('\\HasNoChildren',), '/', 'INBOX'),
(('\\HasNoChildren',), '/', 'Sent'),
--пропущенный код--
(('\\HasNoChildren', '\\Flagged'), '/', '[Gmail]/Starred'),
(('\\HasNoChildren', '\\Trash'), '/', '[Gmail]/Trash')]
```

Примерно так будет выглядеть ваш вывод, если у вас есть учетная запись Gmail. (В Gmail папки называются *ярлыками*, но это не меняет сути дела — они работают точно так же, как папки.) Три значения, входящие в каждый кортеж, например `(('\\HasNoChildren',), '/', 'INBOX')`, имеют следующий смысл:

- кортеж флагов папки (подробное обсуждение этих флагов выходит за рамки данной книги, так что можете смело игнорировать это поле);
- разделитель, используемый в строке имени для разделения родительских и подчиненных папок;
- полное имя папки.

Чтобы выбрать папку, в которой должен осуществляться поиск, передайте ее имя в виде строки методу `select_folder()` объекта IMAPClient.

```
>>> imapObj.select_folder('INBOX', readonly=True)
```

Значение, возвращаемое методом `select_folder()`, можете игнорировать. Если выбранной папки не существует, то Python возбудит исключение `imaplib.error`.

Именованный аргумент `readonly=True` позволяет избежать случайного изменения или удаления любого сообщения в данной папке при последующих

вызовах методов. Если вы не планируете удалять сообщения, то имеет смысл всегда устанавливать значение аргумента `readonly` равным `True`.

Выполнение поиска

Выбрав папку, можно приступить к поиску сообщений, используя метод `search()` объекта IMAPClient. Аргументом, передаваемым методу `search()`, является список строк, каждая из которых отформатирована поисковыми ключами IMAP (табл. 16.3).

Таблица 16.3. Поисковые ключи IMAP

Ключ поиска	Описание
`'ALL'`	Поиск всех сообщений, хранящихся в данной папке. Запрашивая все сообщения в большой папке, вы можете столкнуться с ограничениями, которые модуль `imaplib` налагает на допустимый суммарный размер загружаемых сообщений (см. раздел "Предельный размер сообщений")

`'BEFORE` *дата*`',`
`'ON` *дата*`',`
`'SINCE` *дата*`'`
	Эти три ключа задают поиск сообщений, помеченных соответственно более ранней датой, чем текущая, текущей датой и более поздней датой, чем текущая. Требуемый формат даты — 07-Апр-2016.
	Кроме того, в то время как строке `'SINCE 07-Апр-2016'` соответствуют сообщения, датированные 7 апреля, и более поздние, строке `'BEFORE 05-Jul-2015'` соответствуют лишь сообщения, предшествующие 7 апреля, тогда как сообщения, помеченные самой датой 7 апреля, этой строке не соответствуют

`'SUBJECT` *строка*`',`
`'BODY` *строка*`',`
`'TEXT` *строка*`'`
	Поиск сообщений, в поле темы, тела или в любом из этих полей которых соответственно содержится заданная строка. Строки, содержащие пробелы, следует заключать в кавычки: `'TEXT "поиск с учетом пробелов"'`

`'FROM` *строка*`',`
`'TO` *строка*`',`
`'CC` *строка*`',`
`'BCC` *строка*`'`
	Поиск сообщений, содержащих заданную строку в поле адреса "От", поле адресов "Кому", поле адресов "Сс" ("Копия") или поле адресов "Bcc" ("Скрытая копия") соответственно. При наличии нескольких адресов в строке их следует разделять пробелами или заключать в кавычки: `'CC "firstcc@example.com secondcc@example.com"'`

`'SEEN',` `'UNSEEN'`	Поиск сообщений, соответственно помеченных или не помеченных флагом `\Seen` (просмотрено). Этим флагом помечаются сообщения, к которым осуществлялся доступ с помощью метода `fetch()` (описан далее) или на которых вы выполнили щелчок во время просмотра почты с помощью клиентской почтовой программы или браузера. Чаще говорят о том, что сообщение "прочитано", а не "просмотрено", но оба варианта означают одно и то же
`'ANSWERED',` `'UNANSWERED'`	Поиск всех сообщений, соответственно помеченных или не помеченных флагом `\Answered`. Сообщение помечается этим флагом, если на него был дан ответ
`'DELETED',` `'UNDELETED'`	Поиск сообщений, соответственно помеченных или не помеченных флагом `\Deleted`. Сообщения, удаленные с помощью метода `delete_messages()`, снабжаются флагом `\Deleted`, но не удаляются безвозвратно до тех пор, пока не будет вызван метод `expunge()` (см. раздел "Удаление сообщений"). Имейте в виду, что некоторые почтовые провайдеры автоматически выполняют безвозвратное удаление сообщений

Окончание табл. 16.3

Ключ поиска	Описание
`'DRAFT'`, `'UNDRAFT'`	Поиск сообщений, соответственно помеченных или не помеченных флагом `\Draft` (черновик). Как правило, черновики сообщений хранятся в отдельной папке `Draft`, а не в папке `INBOX`
`'FLAGGED'`, `'UNFLAGGED'`	Поиск сообщений, соответственно помеченных или не помеченных флагом `\Flagged`. Обычно этот флаг используется для пометки сообщений как важных или срочных
`'LARGER N'`, `'SMALLER N'`	Поиск сообщений, размер которых соответственно больше или меньше N байт
`'NOT` *ключ_поиска*`'`	Поиск сообщений, которые указанный ключ поиска не возвратил бы
`'OR` *ключ_поиска1 ключ_поиска2*`'`	Поиск сообщений, которые соответствуют либо первому, либо второму ключу поиска

Следует отметить, что разные IMAP-серверы могут по-разному обрабатывать свои флаги и ключи поиска. Возможно, для того чтобы выяснить, как ведет себя конкретный сервер, вам придется немного поэкспериментировать в интерактивной оболочке.

Методу `search()` можно передать в списке аргументов несколько строк с поисковыми ключами IMAP. При этом метод возвратит те сообщения, которые соответствуют всем ключам поиска. Если вам достаточно соответствия любому из ключей, то используйте поисковый ключ OR. За ключами NOT и OR должны следовать один или два полных поисковых ключа соответственно.

Ниже приведены некоторые примеры вызовов метода `search()` с краткими комментариями.

- **`imapObj.search(['ALL'])`**. Возвращает все сообщения, хранящиеся в выбранной папке.
- **`imapObj.search(['ON 05-Jul-2015'])`**. Возвращает сообщения, отправленные 5 июля 2015 года.
- **`imapObj.search(['SINCE 01-Jan-2015', 'BEFORE 01-Feb-2015', 'UNSEEN'])`**. Возвращает все сообщения, отправленные в январе 2016 года, которые остались непрочитанными. (Заметьте, что указанный период включает *1 января и все последующие дни января* вплоть до, *но не включая*, 1 февраля.)
- **`imapObj.search(['SINCE 01-Jan-2015', 'FROM alice@example.com'])`**. Возвращает все сообщения от респондента `alice@example.com`, отправленные с начала 2016 года.
- **`imapObj.search(['SINCE 01-Jan-2015', 'NOT FROM alice@example.com'])`**. Возвращает все сообщения от всех респондентов, кроме `alice@example.com`, отправленные с начала 2016 года.

- `imapObj.search(['OR FROM alice@example.com FROM bob@example.com'])`. Возвращает все сообщения, отправленные когда-либо респондентами `alice@example.com` и `bob@example.com`.
- `imapObj.search(['FROM alice@example.com', 'FROM bob@example.com'])`. Пример с подвохом! В результате этого поиска не будет возвращено ни одно сообщение, поскольку сообщения должны соответствовать *всем* поисковым словам. Но в поле "От" сообщения может находиться только один адрес, так что не может быть сообщений, в которых в качестве отправителя фигурировали бы одновременно `alice@example.com` и `bob@example.com`.

Метод `search()` возвращает не сами сообщения, а их уникальные идентификаторы (UID) в виде целых чисел. Чтобы получить содержимое сообщений, вы можете передать эти уникальные идентификаторы методу `fetch()`.

Продолжите выполнение примера в интерактивной оболочке, введя следующие команды.

```
>>> UIDs = imapObj.search(['SINCE 05-Jul-2015'])
>>> UIDs
[40032, 40033, 40034, 40035, 40036, 40037, 40038, 40039, 40040, 40041]
```

Здесь список идентификаторов сообщений (полученных начиная с 5 июля), возвращенный методом `search()`, сохраняется в переменной `UIDs`. Список `UIDs`, возвращенный на вашем компьютере, будет отличаться от того, который представлен здесь; идентификаторы уникальны лишь для конкретной учетной записи электронной почты. Если впоследствии вы будете использовать уникальные идентификаторы в других вызовах функций, то используйте их значения, полученные вами, а не те, которые отображены в примерах.

Предельный размер сообщений

Если в результате поиска обнаруживается слишком большое количество сообщений, удовлетворяющих заданным критериям, то Python может возбудить исключение примерно со следующим текстом: `imaplib.error: got more than 10000 bytes` (imaplib.error: получено более чем 10000 байт). Если это произойдет, то вы должны разорвать, а затем восстановить соединение с IMAP-сервером и попытаться вновь выполнить поиск.

Этот предел введен с целью не позволить программам на Python потреблять слишком много памяти. К сожалению, заданный по умолчанию предельный размер сообщений слишком мал. Вы можете изменить этот предел с 10000 байт на 10000000 байт, выполнив следующий код.

```
>>> import imaplib
>>> imaplib._MAXLINE = 10000000
```

Это позволит вам избавиться от повторного получения таких сообщений. Возможно, вы захотите включить эти две строки кода во все программы для работы с IMAP, которые вы напишете.

Использование метода `gmail_search( )` объекта `IMAPClient`

Если вы войдете на сервер `imap.gmail.com` для доступа к учетной записи Gmail, то объект `IMAPClient` предоставит дополнительную функцию поиска, имитирующую поле поиска в верхней части веб-страницы (рис. 16.1).

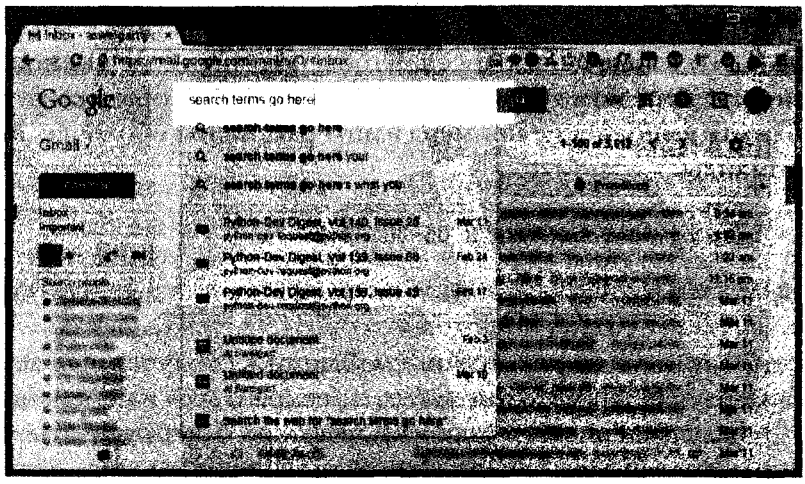


Рис. 16.1. Поле поиска в верхней части веб-страницы Gmail

Вместо того чтобы выполнять поиск с помощью поисковых ключей IMAP, можете использовать более сложный поисковый движок Gmail, который великолепно справляется с поиском соответствий для родственных слов и сортирует результаты поиска по наиболее значимым совпадениям. Кроме того, можно использовать расширенные операторы поиска Gmail (более подробную информацию об этом вы найдете по адресу `http://nostarch.com/automatestuff/`). Если вы вошли в учетную запись Gmail, то передайте поисковые слова методу `gmail_search()`, а не методу `search()`, как в следующем примере.

```
>>> UIDs = imapObj.gmail_search('meaning of life')
>>> UIDs
[42]
```

Ах да: в моей почте нашлось письмо, содержащее слова *meaning of life!* (смысл жизни), по которым я осуществлял поиск.

Извлечение сообщений электронной почты и снабжение прочитанных писем специальной меткой

Получив список UIDs, можно вызвать метод fetch() объекта IMAPClient для получения фактического содержимого ящика электронной почты.

Список UIDs будет первым аргументом метода fetch(). Вторым аргументом является список ['BODY[]'], в который метод fetch() должен загрузить все содержимое тела сообщений, указанных в вашем списке UIDs.

Продолжим выполнение нашего примера в интерактивной оболочке.

```
>>> rawMessages = imapObj.fetch(UIDs, ['BODY[]'])
>>> import pprint
>>> pprint.pprint(rawMessages)
{40040: {'BODY[]': 'Delivered-To: my_email_address@gmail.com\r\n'
                   'Received: by 10.76.71.167 with SMTP id '
--пропущенный код--
                   '\r\n'
                   '------=_Part_6000970_707736290.1404819487066--\r\n',
        'SEQ': 5430}}
```

Импортируйте модуль pprint и передайте значение, возвращенное методом fetch() и сохраненное в переменной rawMessages, функции pprint.pprint(), чтобы вывести его на "красивую печать", и вы увидите, что это значение представляет собой вложенный словарь сообщений, в котором идентификаторы UIDs служат ключами. Каждое сообщение сохраняется в виде словаря с двумя ключами: 'BODY[]' и 'SEQ'. Ключ 'BODY[]' отображается на фактическое тело электронного сообщения. Ключ 'SEQ' играет роль порядкового номера и выполняет функции, аналогичные UID. Можете смело его игнорировать.

Нетрудно заметить, что содержимое ключа 'BODY[]' выглядит довольно непонятно. Это обусловлено тем, что оно хранится в формате RFC 822, предназначенном для чтения серверами IMAP и не приспособленном для чтения человеком. Однако знать этот формат вам не надо; далее мы используем модуль pyzmail, позволяющий представить тело сообщения в удобочитаемом виде.

Выбрав папку, в которой следовало выполнить поиск, вы вызывали метод select_folder() с именованным аргументом readonly=True. Это делалось для того, чтобы предотвратить случайное удаление почты, но это также означает, что письма не будут помечены как прочитанные, если для их извлечения используется метод fetch(). Если необходимо, чтобы письма помечались как прочитанные при их извлечении, то методу select_folder() следует передать именованный аргумент readonly=False. Если же текущая папка уже была выбрана для работы в режиме "только чтение", то ее можно

выбрать заново с помощью другого вызова метода `select_ folder()`, на этот раз – с именованным аргументом `readonly=False`:

```
>>> imapObj.select_folder('INBOX', readonly=False)
```

Получение адресов электронной почты из "сырых" сообщений

"Сырые", т.е. необработанные, сообщения, возвращаемые методом `fetch()`, мало полезны тем, кому надо всего лишь прочитать свою почту. Модуль `pyzmail` выполняет синтаксический анализ "сырых" сообщений и возвращает их в виде объектов `PyzMessage`, обеспечивающих возможность простого доступа вашего кода на Python к теме, телу, полям "Кому" и "От", а также к другим разделам сообщений электронной почты.

Продолжите интерактивный пример, выполнив следующие команды (используя уникальные идентификаторы сообщений из своей учетной записи электронной почты, а не те, которые здесь представлены).

```
>>> import pyzmail
>>> message = pyzmail.PyzMessage.
factory(rawMessages[40041]['BODY[]'])
```

Прежде всего импортируйте модуль `pyzmail`. Затем создайте объект `PyzMessage` сообщения, вызвав функцию `pyzmail.PyzMessage.factory()` и передав ей раздел `'BODY[]'` необработанного сообщения. Сохраните результат в переменной `message`. Теперь в переменной `message` содержится объект `PyzMessage`, используя методы которого можно очень легко получить строку темы сообщения, а также адреса отправителя и получателя. Метод `get_subject()` возвращает тему сообщения в виде простого строкового значения. Метод `get_addresses()` возвращает список адресов для переданного ему поля. Например, вызов данного метода может выглядеть примерно так.

```
>>> message.get_subject()
'Hello!'
>>> message.get_addresses('from')
[('Edward Snowden', 'esnowden@nsa.gov')]
>>> message.get_addresses('to')
[(Jane Doe', 'my_email_address@gmail.com')]
>>> message.get_addresses('cc')
[]
>>> message.get_addresses('bcc')
[]
```

Обратите внимание на аргументы, передаваемые в этом примере методу `get_addresses()`: `'from'`, `'to'`, `'cc'` и `'bcc'`. Значение, возвращаемое методом `get_addresses()`, представляет собой список кортежей. Каждый кортеж

состоит из двух строк: первая из них — это имя, связанное с адресом электронной почты, вторая — сам адрес. Если в запрошенном поле адреса отсутствуют, то вызов `get_addresses()` возвращает пустой список. В данном случае такими являются поля `'cc'` (копия) и `'bcc'` (скрытая копия), и поэтому для них возвращаются пустые списки.

Получение тела письма из "сырого" сообщения

Сообщения электронной почты могут быть отправлены в формате простого текста, HTML или в комбинированном формате. В первом случае сообщения могут содержать только текст, тогда как в HTML-сообщениях могут использоваться цвета, различные шрифты, изображения и другие возможности, благодаря которым электронные письма будут выглядеть как небольшие веб-страницы. Если вся электронная почта — это простой текст, то значением атрибута `html_part` его объекта `PyzMessage` будет `None`. Подобным образом, если в сообщении используется только формат HTML, значением атрибута `text_part` его объекта `PyzMessage` будет `None`.

Во всех других случаях значение `text_part` или `html_part` будет иметь метод `get_payload()`, который возвращает тело сообщения в виде значения с типом данных *bytes*. (Рассмотрение типа данных *bytes* выходит за рамки данной книги.) Однако это все еще не то строковое значение, которое мы можем использовать. Ух! Последний шаг заключается в том, чтобы вызвать метод `decode()` для значения типа *bytes*, возвращенного методом `get_payload()`. Метод `decode()` принимает один аргумент: кодировку символов сообщения, которая хранится в атрибуте `text_part.charset` или `html_part.charset`. Окончательный результат, возвращаемый методом `decode()`, представляет собой строку с телом сообщения.

Продолжите выполнение интерактивного примера, введя следующие команды.

```
❶ >>> message.text_part != None
  True
  >>> message.text_part.get_payload().
  ↳ decode(message.text_part.charset)
❷ 'So long, and thanks for all the fish!\r\n\r\n-Al\r\n'
❸ >>> message.html_part != None
  True
❹ >>> message.html_part.get_payload().
  ↳ decode(message.html_part.charset)
  '<div dir="ltr"><div>So long, and thanks for all the
  fish!<br><br></div>-Al<br></div>\r\n'
```

Электронная почта, с которой мы работаем, включает как простой текст, так и HTML-содержимое, поэтому в объекте `PyzMessage`, сохраненном в переменной `message`, имеются атрибуты `text_part` и `html_part`, значения

которых не равны None ❶ ❸. В результате вызова метода get_payload() для атрибута text_part объекта message и последующего вызова метода decode() для значения типа *bytes* возвращается строка текстовой версии сообщения ❷. Использование вызовов методов get_payload() и decode() с атрибутом html_part объекта message возвращает HTML-версию сообщения ❹.

Удаление сообщений

Чтобы удалить сообщения, передайте список уникальных идентификаторов сообщений методу delete_messages() объекта IMAPClient. В результате этого сообщения помечаются флагом \Deleted (удалено). Вызов метода expunge() приведет к безвозвратному удалению всех сообщений в текущей выбранной папке, помеченных флагом \Deleted. Рассмотрим следующий интерактивный пример.

```
❶ >>> imapObj.select_folder('INBOX', readonly=False)
❷ >>> UIDs = imapObj.search(['ON 09-Июл-2015'])
  >>> UIDs
  [40066]
  >>> imapObj.delete_messages(UIDs)
❸ {40066: ('\\Seen', '\\Deleted')}
  >>> imapObj.expunge()
  ('Success', [(5452, 'EXISTS')])
```

В этом примере мы выбираем папку INBOX (Входящие), вызывая метод select_folder() объекта IMAPClient с передачей ему строки 'INBOX' в качестве первого аргумента. Кроме того, методу select_folder() передается именованный аргумент readonly=False, чтобы сделать возможным удаление сообщений ❶. В папке INBOX выполняется поиск сообщений с указанной датой получения, и идентификаторы возвращенных сообщений сохраняются в списке UIDs ❷. Вызов метода delete_message(), которому передается список UIDs, возвращает словарь. В этом словаре каждая пара "ключ–значение" представляет идентификатор сообщения и кортеж флагов сообщения, который теперь должен включать флаг \Deleted ❸. Последующий вызов метода expunge() безвозвратно удаляет сообщения, помеченные флагом \Deleted, и возвращает сообщение Success в случае успешного удаления сообщений. Имейте в виду, что некоторые провайдеры, например Gmail, автоматически безвозвратно удаляют сообщения, удаляемые с помощью метода delete_messages(), не дожидаясь поступления соответствующей команды от клиента IMAP.

Разрыв соединения с сервером IMAP

Чтобы разорвать соединение с сервером IMAP, когда ваша программа завершит извлечение или удаление сообщений электронной почты, достаточно вызвать метод `logout()` объекта `IMAPClient`:

```
>>> imapObj.logout()
```

Если ваша программа будет выполняться несколько минут или дольше, то сервер IMAP может автоматически отсоединиться по истечении заданного промежутка времени (тайм-аута). В подобных случаях следующая попытка вызова какого-либо метода объекта `IMAPClient` приведет к возникновению исключения наподобие следующего:

```
imaplib.abort: socket error: [WinError 10054] An existing connection
was forcibly closed by the remote host
```
(Удаленный хост-компьютер разорвал установленное соединение).

Если это произойдет, то для повторного установления соединения ваша программа должна будет вновь вызвать функцию `imapclient.IMAPClient()`.

Ух, наконец-то! На пришлось проделать очень длинный путь, пока мы дошли до конца, но зато теперь вы знаете, как заставить свои программы на Python входить в учетные записи электронной почты и извлекать сообщения. Всякий раз, когда вам нужно будет освежить в памяти детали того, как это делается, вы можете заглянуть в раздел "Извлечение и удаление сообщений электронной почты с помощью IMAP".

Проект: рассылка по электронной почте напоминаний о необходимости уплаты членских взносов

Предположим, вы на "добровольных" началах вызвались вести учет уплаты взносов членами *Клуба обязательного волонтерства*. Это утомительное занятие, требующее поддержки электронной таблицы, в которой отмечается, кто из членов клуба уплатил ежемесячный членский взнос, а кто не уплатил, причем последним необходимо рассылать электронной почтой уведомления, напоминающие о необходимости уплаты взноса. Вместо того чтобы вручную просматривать список членов клуба и готовить письма-напоминания тем, кто просрочил очередной платеж, копируя и вставляя в письма каждому из них один и тот же текст, лучше — и вы, наверное, сами уже об этом догадались — написать сценарий, который выполнит эту работу вместо вас.

Вот что должна делать такая программа при высокоуровневом рассмотрении:

- читать данные из электронной таблицы Excel;
- находить тех членов клуба, которые не уплатили взнос за последний месяц;
- находить их адреса электронной почты и отправлять им персональные напоминания.

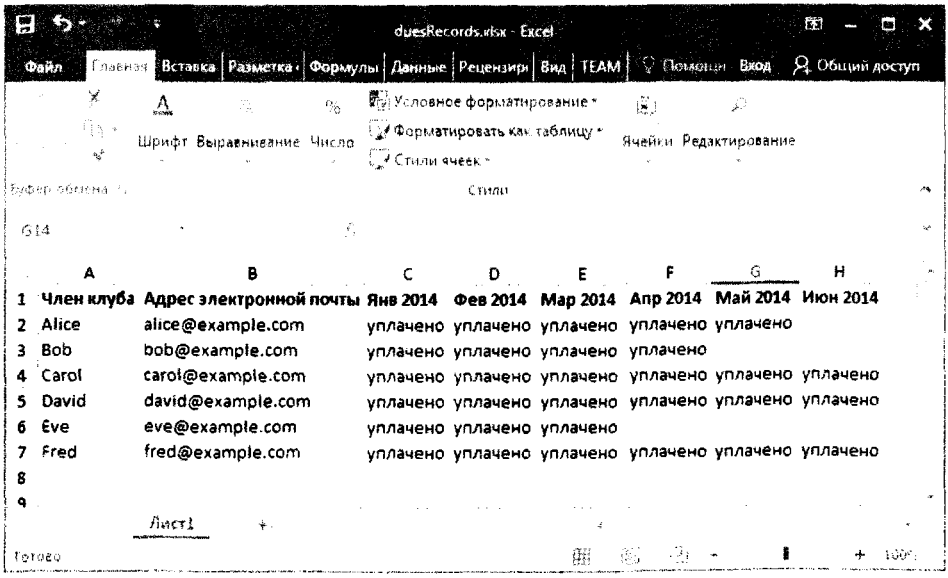
Это означает, что ваш код должен выполнять следующие действия:

- открывать документ Excel и читать содержимое его ячеек с помощью модуля `openpyxl` (о работе с файлами Excel читайте в главе 12);
- создавать словарь членов клуба, не уплативших членские взносы;
- входить в учетную запись на SMTP-сервере путем вызова функций и методов `smtplib.SMTP()`, `ehlo()`, `starttls()` и `login()`;
- отправлять электронной почтой персональные напоминания членам клуба, просрочившим уплату взносов, с помощью метода `sendmail()`.

Откройте новое окно в файловом редакторе и сохраните его в файле *sendDuesReminders.py*.

Шаг 1. Открытие файла Excel

Предположим, что электронная таблица Excel, которую вы используете для контроля уплаты членских взносов, выглядит примерно так, как показано на рис. 16.2, и хранится в файле *duesRecords.xlsx*. Этот файл можно загрузить на сайте `http://nostarch.com/automatestuff/`.



	A	B	C	D	E	F	G	H
1	**Член клуба**	**Адрес электронной почты**	**Янв 2014**	**Фев 2014**	**Мар 2014**	**Апр 2014**	**Май 2014**	**Июн 2014**
2	Alice	alice@example.com	уплачено	уплачено	уплачено	уплачено	уплачено	
3	Bob	bob@example.com	уплачено	уплачено	уплачено	уплачено		
4	Carol	carol@example.com	уплачено	уплачено	уплачено	уплачено	уплачено	уплачено
5	David	david@example.com	уплачено	уплачено	уплачено	уплачено	уплачено	уплачено
6	Eve	eve@example.com	уплачено	уплачено	уплачено			
7	Fred	fred@example.com	уплачено	уплачено	уплачено	уплачено	уплачено	уплачено
8								
9								

Рис. 16.2. Электронная таблица для учета уплаты членских взносов

В этой электронной таблице хранятся имена членов клуба и адреса их электронной почты. Каждому месяцу соответствует отдельный столбец, в котором делаются отметки об уплате взносов. Отметкой об уплате взноса служит текст `уплачено`.

Программа должна открывать файл *duesRecords.xlsx* и находить столбец, соответствующий последнему месяцу, вызывая метод `get_highest_column()`. (Для получения более подробной информации о доступе к ячейкам электронных таблиц Excel с помощью модуля `openpyxl` обратитесь к главе 12.) Введите в окне файлового редактора следующий код.

```
#! python3
# sendDuesReminders.py - Рассылает сообщения на основании
# сведений из электронной таблицы об уплате взносов.
import openpyxl, smtplib, sys
# Открытие электронной таблицы и получение последних данных об
# уплате взносов.
❶ wb = openpyxl.load_workbook('duesRecords.xlsx')
❷ sheet = wb.get_sheet_by_name('Лист1')
❸ lastCol = sheet.get_highest_column()
❹ latestMonth = sheet.cell(row=1, column=lastCol).value
# TODO: Проверить состояние уплаты взносов для каждого
#       члена клуба.
# TODO: Войти в учетную запись электронной почты.
# TODO: Отправить сообщения с напоминанием об уплате взносов.
```

Импортировав модули `openpyxl`, `smtplib` и `sys`, мы открываем файл *duesRecords.xlsx* и сохраняем результирующий объект `Workbook` в переменной `wb` ❶. Затем мы получаем `Лист 1` и сохраняем результирующий объект `Worksheet` в переменной `sheet` ❷. Теперь, когда в нашем распоряжении имеется объект `Worksheet`, мы можем обращаться к строкам, столбцам и ячейкам электронной таблицы. Мы сохраняем номер последнего столбца в переменной `lastCol` ❸, а затем используем ячейку с номером строки `1` и номером столбца `lastCol` для доступа к ячейке, в которой хранится номер последнего месяца. Значение этой ячейки мы сохраняем в переменной `latestMonth` ❹.

Шаг 2. Поиск всех членов клуба, не уплативших взнос

Определив номер последнего месяца (хранящийся в переменной `lastCol`), можно организовать цикл по всем строкам, кроме первой (в которой хранятся заголовки столбцов), чтобы выяснить, против фамилий каких членов клуба стоит отметка `уплачено` для проверяемого месяца. Если кто-либо из членов клуба не уплатил взнос, можно извлечь его имя и адрес электронной почты из столбцов 1 и 2 соответственно. Эта информация заносится в словарь `unpaidMembers`, с помощью которого будут отслеживаться те,

кто пропустил оплату за последний месяц. Добавьте в файл *sendDuesReminder.py* следующий код.

```
#! python3
# sendDuesReminders.py - Рассылает сообщения на основании
# сведений из электронной таблицы об уплате взносов.

--пропущенный код--

# Проверка состояния уплаты взносов для каждого члена клуба.
unpaidMembers = {}
❶ for r in range(2, sheet.get_highest_row() + 1):
❷     payment = sheet.cell(row=r, column=lastCol).value
      if payment != 'уплачено':
❸         name = sheet.cell(row=r, column=1).value
❹         email = sheet.cell(row=r, column=2).value
❺         unpaidMembers[name] = email
```

В этом коде создается пустой словарь `unpaidMembers` и выполняется цикл по всем строкам, кроме первой ❶. Для каждой строки значение, находящееся в последнем столбце, сохраняется в переменной `payment` ❷. Если значением `payment` не является строка `'уплачено'`, то значение, находящееся в первом столбце, сохраняется в переменной `name` ❸, а значение, находящееся во втором столбце — в переменной `email` ❹, и обе переменные, `name` и `email`, добавляются в словарь `unpaidMembers` ❺.

Шаг 3. Отправка персональных напоминаний по электронной почте

Получив список всех неплательщиков, можно отправить им напоминания по электронной почте. Добавьте в программу следующий код, использовав в нем свой реальный адрес электронной почты и информацию о почтовом провайдере.

```
#! python3
# sendDuesReminders.py - Рассылает сообщения на основании
# сведений из электронной таблицы об уплате взносов.

--пропущенный код--

# Вход в учетную запись электронной почты.
smtpObj = smtplib.SMTP('smtp.gmail.com', 587)
smtpObj.ehlo()
smtpObj.starttls()
smtpObj.login('my_email_address@gmail.com', sys.argv[1])
```

Создайте объект SMTP, вызвав функцию `smtplib.SMTP()` с передачей ей доменного имени и номера порта своего провайдера. Вызовите сначала

методы `ehlo()` и `starttls()`, а затем — метод `login()`, и передайте ему свой адрес электронной почты и значение переменной `sys.argv[1]`, в которой будет храниться строка с вашим паролем. Вы будете вводить пароль в качестве аргумента командной строки каждый раз при запуске программы, чтобы избежать его хранения в исходном коде.

После входа программы в вашу учетную запись электронной почты она должна проанализировать содержимое словаря `unpaidMembers` и отправить по электронной почте персональные напоминания должникам. Добавьте в файл `sendDuesReminders.py` следующий код.

```
    #! python3
    # sendDuesReminders.py - Рассылает сообщения на основании
    # сведений из электронной таблицы об уплате взносов.

    --пропущенный код--

    # Отправка сообщений с напоминанием об уплате взносов.
    for name, email in unpaidMembers.items():
❶       body = "Subject: %s dues unpaid.\nDear %s,\nRecords show
    ↪ that you have not paid dues for %s. Please make this payment
    ↪ as soon as possible. Thank you!'"
    ↪ % (latestMonth, name, latestMonth)
❷       print('Отправка письма по адресу %s...' % email)
❸       sendmailStatus =
    ↪ smtpObj.sendmail('my_email_address@gmail.com', email, body)
❹       if sendmailStatus != {}:
            print('There was a problem sending email to %s: %s'
    ↪ % (email, sendmailStatus))
    smtpObj.quit()
```

В этом коде выполняется цикл по именам и адресам электронной почты, хранящимся в словаре `unpaidMembers`. Для каждого члена клуба, просрочившего уплату взноса, создается персональное сообщение, в котором используется информация о последнем месяце и имени члена клуба, и это сообщение сохраняется в переменной `body` ❶. Мы также выводим информационное сообщение об отправке письма в адрес данного члена клуба ❷. Затем мы вызываем метод `sendmail()`, передав ему адрес отправителя и персонализированное сообщение ❸. Возвращаемое этим методом значение сохраняется в переменной `sendmailStatus`.

Помните о том, что в случае получения от SMTP-сервера сообщения об ошибке при попытке отправки какого-либо сообщения метод `sendmail()` возвращает значение в виде непустого словаря. В последней части цикла `for` ❹ проверяется, является ли словарь непустым, и, если это так, выводятся адрес электронной почты получателя и содержимое возвращенного словаря.

Когда программа завершит отправку всех сообщений, следует разорвать соединение с SMTP-сервером, вызвав метод `quit()`.

Выполнив программу, вы должны получить следующий вывод.

```
Отправка письма по адресу alice@example.com...
Отправка письма по адресу bob@example.com...
Отправка письма по адресу eve@example.com...
```

Полученные получателями сообщения электронной почты будут выглядеть так, как показано на рис. 16.3.

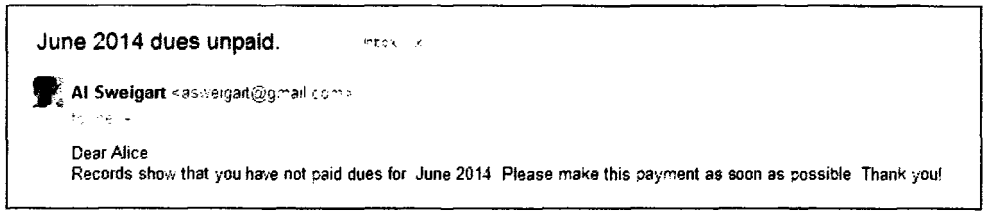


Рис. 16.3. Автоматическая отправка сообщений электронной почты с помощью программы sendDuesReminders.py

Отправка текстовых сообщений с помощью Twilio

Большинство людей предпочитают иметь под рукой телефон, а не компьютер, поэтому текстовые сообщения могут оказаться более непосредственным и надежным способом рассылки уведомлений, чем электронная почта. Кроме того, небольшой размер текстовых сообщений делает более вероятным их быстрое прочтение теми, кому они предназначены.

В этом разделе вы узнаете о том, как подписаться на бесплатную службу Twilio и использовать ее модуль Python для отправки текстовых сообщений. Twilio — это служба шлюзового интерфейса SMS, т.е. служба, с помощью которой вы сможете отправлять текстовые сообщения из своих программ. Несмотря на то что вы будете ограничены в количестве ежемесячных текстовых сообщений, которые вам будет разрешено отправлять, а их текст будет предваряться словами *Sent from a Twilio trial account* (отправлено с использованием пробной учетной записи Twilio), пробный вариант этой службы, вероятно, вполне удовлетворит потребности ваших персональных программ. Срок действия пробной версии неограничен, и вам не придется впоследствии в обязательном порядке обновлять ее до платной версии.

Twilio — не единственная из программ такого рода. Если вы не захотите использовать Twilio, то можете выбрать одну из альтернативных служб, выполнив в Интернете поиск по ключевым словам `free sms gateway`, `python sms api` или даже `twilio alternatives`.

Прежде чем создать учетную запись на Twilio, установите модуль `twilio`. Более подробно об установке модулей, разработанных сторонними компаниями, можно прочитать в приложении А.

Примечание

Этот раздел специфичен для США. Twilio предлагает услуги по обмену SMS-сообщениями и для других стран, однако их специфика в данной книге не рассматривается. Тем не менее модуль `twilio` и его функции будут работать одинаково и вне США. Более подробную информацию по этому вопросу можно найти на сайте `http://twilio.com/`.

Создание учетной записи Twilio

Перейдите на сайт `http://twilio.com/` и заполните регистрационную форму. После того как вы создадите новую учетную запись, вам нужно будет подтвердить номер мобильного телефона, на который вы хотите отправлять текстовые сообщения. (Верификация этого номера необходима для того, чтобы исключить возможность использования службы для рассылки спама на случайные номера.)

Получив текст с верификационным кодом, введите его на сайте Twilio в качестве доказательства того, что именно вы являетесь владельцем верифицируемого номера. Теперь вы сможете отправлять текстовые сообщения на этот номер с помощью модуля `twilio`.

Twilio предоставит вам пробную учетную запись с телефонным номером для использования в качестве отправителя текстовых сообщений. Вам потребуются еще два информационных элемента: идентификатор безопасности SID вашей учетной записи и аутентификационный маркер. Соответствующую информацию вы найдете на странице Dashboard, когда войдете в свою учетную запись Twilio. Эти значения будут играть роль вашего имени пользователя и пароля при входе в Twilio из программ на Python.

Отправка текстовых сообщений

Когда вы установите модуль `twilio`, заведете учетную запись Twilio, верифицируете свой телефонный номер, зарегистрируете телефонный номер Twilio и получите SID и аутентификационный маркер для своей учетной записи, это будет означать, что вы полностью готовы к отправке текстовых сообщений из своих сценариев на Python.

По сравнению с полной процедурой регистрации написание фактического кода на Python для использования службы Twilio не составляет труда. При условии, что ваш компьютер подключен к Интернету, введите в

интерактивной оболочке приведенный ниже код, используя в нем для переменных accountSID, authToken, myTwilioNumber и myCellPhone реальные значения своего идентификатора безопасности, аутентификационного маркера, телефонного номера Twilio и собственного телефонного номера.

```
❶ >>> from twilio.rest import TwilioRestClient
  >>> accountSID = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
  >>> authToken = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
❷ >>> twilioCli = TwilioRestClient(accountSID, authToken)
  >>> myTwilioNumber = '+14955551234'
  >>> myCellPhone = '+14955558888'
❸ >>> message = twilioCli.messages.create(body='Mr. Watson - Come
  ↪here - I want to see you.',
  ↪from_=myTwilioNumber, to=myCellPhone)
```

Спустя короткое время после ввода последней строки вы должны получить текстовое сообщение следующего содержания: *Sent from your Twilio trial account – Mr. Watson – Come here – I want to see you.*

В силу особенностей развертывания модуля twilio его нужно импортировать с помощью команды from twilio.rest import TwilioRestClient, а не просто команды import twilio ❶. Сохраните SID своей учетной записи в переменной accountSID, а свой аутентификационный маркер – в переменной authToken, после чего вызовите функцию TwilioRestClient(), передав accountSID и authToken в качестве аргументов. Вызов TwilioRestClient() возвращает объект TwilioRestClient ❷. Этот объект имеет атрибут messages, в свою очередь имеющий метод create(), который вы можете использовать для отправки текстовых сообщений. Именно этот метод инструктирует серверы Twilio о том, что ваше текстовое сообщение нуждается в отправке. Сохранив свой номер Twilio и номер сотового телефона в переменных myTwilioNumber и myCellPhone соответственно, вызовите метод create() и передайте ему именованные аргументы, задающие тело текстового сообщения, номер отправителя (myTwilioNumber) и номер получателя (myCellPhone) ❸.

Объект Message, возвращенный методом create(), будет хранить информацию об отправленном текстовом сообщении. Продолжите выполнение примера, введя в интерактивной оболочке следующие команды.

```
>>> message.to
'+14955558888'
>>> message.from_
'+14955551234'
>>> message.body
'Mr. Watson - Come here - I want to see you.'
```

В атрибутах to, from_ и body должны храниться соответственно номер вашего сотового телефона, телефонный номер Twilio и само сообщение.

Обратите внимание на то, что номер телефона отправителя хранится в атрибуте `from_` (имя которого содержит символ подчеркивания в конце), а не `from`. Это обусловлено тем, что `from` является ключевым словом в Python (например, оно уже встречалось вам в инструкциях импорта в форме `from` *имя_модуля* `import *`) и поэтому не может служить именем атрибута. Продолжите выполнение примера, введя в интерактивной оболочке следующие команды.

```
>>> message.status
'queued'
>>> message.date_created
datetime.datetime(2015, 7, 8, 1, 36, 18)
>>> message.date_sent == None
True
```

Атрибут `status` предоставляет строку. Атрибуты `date_created` и `date_sent` предоставляют объект `datetime`, если сообщение было создано и отправлено. Может показаться несколько странным, что атрибут `status` возвращает значение `'queued'` (помещено в очередь), а атрибут `date_sent` — значение `None`, когда текстовое сообщение уже получено. Объясняется это тем, что объект `Message` был сохранен в переменной `message` еще до фактической отправки текстового сообщения. Чтобы увидеть текущие значения атрибутов `status` и `date_sent`, следует заново извлечь объект `Message`. Каждому сообщению Twilio присваивается уникальный строковый идентификатор (SID), который можно использовать для извлечения последнего состояния объекта `Message`. Продолжите выполнение примера, введя в интерактивной оболочке следующие команды.

```
>>> message.sid
'SM09520de7639ba3af137c6fcb7c5f4b51'
❶ >>> updatedMessage = twilioCli.messages.get(message.sid)
>>> updatedMessage.status
'delivered'
>>> updatedMessage.date_sent
datetime.datetime(2015, 7, 8, 1, 36, 18)
```

Команда `message.sid` отображает длинное значение SID этого сообщения. Передав это значение SID методу `get()` клиента Twilio ❶, вы можете получить новый объект `Message`, содержащий обновленную информацию о сообщении. Теперь атрибуты `status` и `date_sent` этого нового объекта `Message` содержат корректные значения.

Атрибут `status` может принимать одно из следующих строковых значений: `'queued'` (помещено в очередь), `'sending'`(отправляется), `'sent'` (отправлено), `'delivered'` (доставлено), `'undelivered'` (не доставлено) или

'failed' (не отправлено). Названия этих состояний говорят сами за себя, но если вам нужны более подробные сведения, то ознакомьтесь с ресурсами на сайте http://nostarch.com/automatestuff/.

Получение текстовых сообщений с помощью Python

К сожалению, процедура получения текстовых сообщений с помощью Python немного сложнее процедуры отправки. Twilio требует, чтобы у вас был веб-сайт, выполняющий собственное веб-приложение. Рассмотрение этой темы выходит за рамки данной книги, но вы сможете получить более подробную информацию по этому вопросу, обратившись к ресурсам, представленным на сайте книги (http://nostarch.com/automatestuff/).

Проект: модуль "Черкни мне"

Вероятнее всего, персоной, которой вы будете чаще всего отправлять текстовые сообщения из своих программ, будете вы сами. Текстовые сообщения — замечательное средство отправлять себе напоминания, которые можно прочитать, находясь вдали от компьютера. Если вы автоматизировали рутинную задачу, выполняющуюся пару часов, то сможете получить текстовое сообщение, извещающее вас о ее завершении. Другой вариант — у вас есть программа, регулярно выполняющаяся по расписанию и при этом иногда нуждающаяся в установлении контакта с вами. В качестве примера можно привести программу, проверяющую прогнозы погоды и при необходимости сообщающую вам о том, что пора приготовить зонтик по случаю ожидаемого дождя.

Рассмотрим простой пример программы с функцией textmyself(), которая отправляет текстовое сообщение, переданное ей в качестве строкового аргумента. Откройте новое окно в файловом редакторе и введите приведенный ниже код, используя в нем для переменных accountSID, authToken, myTwilioNumber и myCellPhone собственные данные. Сохраните код в файле textMyself.py.

```
#! python3
# textMyself.py - Определяет функцию textmyself(), которая
# отправляет текстовое сообщение, переданное ей в виде строки
# Предустановленные значения:
accountSID = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
authToken = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
myNumber = '+15559998888'
twilioNumber = '+15552225678'
from twilio.rest import TwilioRestClient
❶ def textmyself(message):
❷ twilioCli = TwilioRestClient(accountSID, authToken)
```

```
❸ twilioCli.messages.create(body=message, from_=twilioNumber,
  ↳to=myNumber)
```

Прежде всего в программе сохраняются значения SID и аутентификационного маркера, а также телефонные номера отправителя и получателя. Затем определяется функция `textmyself()`, которая принимает аргумент ❶, создает объект `TwilioRestClient` ❷ и вызывает метод `create()`, используя переданное ей сообщение ❸.

Если вы захотите сделать функцию `textmyself()` доступной другим своим программам, то поместите файл *textMyself.py* в ту же папку, в которой находится исполняемый файл Python (`C:\Python34` в Windows, `/usr/local/lib/python3.4` в OS X и `/usr/bin/python3` в Linux). После этого вы сможете использовать эту функцию в других программах. Чтобы одна из ваших программ могла отправить вам текстовое сообщение, включите в нее следующий код:

```
import textmyself
textmyself.textmyself('Рутинная работа выполнена.')
```

Зарегистрироваться на сайте Twilio и написать код, обрабатывающий отправку текстовых сообщений, нужно только один раз. После этого для отправки текстового сообщения из любой из ваших программ вам потребуется добавить всего лишь пару строчек кода.

Резюме

Мы общаемся друг с другом через Интернет и посредством сотовых телефонных сетей, используя десятки всевозможных способов, но преимущественно это электронная почта и текстовые сообщения. Ваша программа может задействовать эти каналы связи, что открывает перед вами новые широкие возможности оповещения. Можно даже написать программу, которая будет выполняться на разных компьютерах, обеспечивая непосредственный обмен данными между ними посредством электронной почты, когда одна программа отправляет сообщения электронной почты по протоколу SMTP, а другая принимает их по протоколу IMAP.

Модуль Python `smtplib` предоставляет функции, с помощью которых вы можете отправлять сообщения через SMTP-сервер своего почтового провайдера. Подобным образом модули сторонних разработчиков `imapclient` и `pyzmail` позволяют получать доступ к серверам IMAP и получать отправленные вам сообщения. Протокол IMAP несколько сложнее протокола SMTP, однако предлагает довольно широкие возможности, включая поиск конкретных сообщений электронной почты, их загрузку и синтаксический

анализ для извлечения темы и тела сообщений в виде строковых значений. Обмен текстовыми сообщениями несколько отличается от электронной почты, поскольку, в отличие от последней, для отправки СМС требуется не только подключение к Интернету. К счастью, службы наподобие Twilio предоставляют модули, позволяющие отправлять текстовые сообщения из программ. Как только вы пройдете все этапы процесса начальной настройки, вы сможете отправлять СМС с помощью всего лишь пары строк кода.

Имея в своем распоряжении указанные модули, вы сможете программировать конкретные условия, при которых ваши программы должны отправлять уведомления или напоминания. В этом случае область действия ваших программ распространится на большие расстояния за пределами компьютера, на котором они выполняются!

Контрольные вопросы

1. Что такое протокол для отправки электронной почты? Что такое протокол для проверки и получения электронной почты?
2. Какие четыре функции/методы модуля `smtplib` необходимо вызвать для того, чтобы войти в учетную запись на SMTP-сервере?
3. Какие функции (методы) `imapclient` необходимо вызвать для того, чтобы войти в учетную запись на IMAP-сервере?
4. Аргумент какого типа передается методу `imapObj.search()`?
5. Какие меры вы предпримете в том случае, если ваш код получает сообщение об ошибке `got more than 10000 bytes` (получено более чем 10000 байт)?
6. Модуль `imapclient` отвечает за подключение к серверу IMAP и поиск сообщений электронной почты. Какой модуль отвечает за чтение сообщений электронной почты, извлеченных модулем `imapclient`?
7. Какие три элемента информации нужно получить от Twilio, прежде чем отправлять текстовые сообщения?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Распределение рутинных задач путем рассылки по электронной почте

Напишите программу, которая принимает список адресов электронной почты и список рутинных задач, подлежащих выполнению, которые

должны рассылаться исполнителям, выбираемым случайным образом. Если вы хотите поставить перед собой более амбициозные цели, то организуйте учет ранее распределенных заданий, чтобы программа не назначала исполнителю то же самое задание, которое он выполнял в прошлый раз. Как дополнительный вариант предусмотрите автоматическое выполнение программы по расписанию один раз в неделю.

Вот вам подсказка: если передавать функции `random.choice()` список, то она будет каждый раз возвращать один элемент списка, выбираемый случайным образом. Часть вашего кода может иметь примерно следующий вид.

```
chores = ['dishes', 'bathroom', 'vacuum', 'walk dog']
randomChore = random.choice(chores)
chores.remove(randomChore) # это задание уже распределено, и
                           # его можно удалить из списка
```

Напоминание о зонтике

В главе 11 было продемонстрировано, как организовать сбор данных с погодного сайта `http://weather.gov/` с помощью модуля `requests`. Напишите программу, которая запускается непосредственно перед вашим утренним пробуждением и проверяет, не ожидается ли в этот день дождь. В случае такого прогноза программа должна отправить вам текстовое сообщение с напоминанием о том, что, покидая дом, необходимо не забыть взять с собой зонт.

Автоматический отказ от подписки

Напишите программу, которая просматривает почтовый ящик вашей учетной записи электронной почты, находит все ссылки `Unsubscribe` (Отказаться от подписки) в сообщениях и автоматически открывает их в браузере. Программа должна входить в вашу учетную запись на IMAP-сервере почтового провайдера и загружать все сообщения. Для обнаружения HTML-дескрипторов ссылок, содержащих слово `Unsubscribe`, можно использовать модуль BeautifulSoup (см. главу 11).

Получив список соответствующих URL-адресов, можете использовать функцию `webbrowser.open()` для открытия в браузере ссылок, позволяющих отказаться от подписки.

Вам по-прежнему необходимо вручную проделать все остальные действия, связанные с отказом от подписки. В большинстве случаев это сводится к щелчку на ссылке, подтверждающему отказ.

Тем не менее этот сценарий избавляет вас от необходимости просматривать всю электронную почту в поиске ссылок `Unsubscribe`. Кроме того, можно передать этот сценарий своим друзьям, чтобы они могли выполнять

его применительно к собственным учетным записям электронной почты. (При этом обязательно проследите, чтобы в коде не был жестко запрограммирован ваш собственный пароль для доступа к электронной почте!)

Дистанционное управление компьютером посредством электронной почты

Напишите программу, которая проверяет вашу электронную почту каждые 15 минут, осуществляя поиск поступивших от вас инструкций, и в случае их наличия автоматически их выполняет. Рассмотрим, например, файлообменную систему BitTorrent. Используя бесплатное программное обеспечение BitTorrent, например qBittorrent, можно загружать большие мультимедийные файлы на свой домашний компьютер. Если отправить программе ссылку BitTorrent (это действие – совершенно законное и вовсе не пиратское), то программа в процессе очередной проверки электронной почты обнаружит это сообщение, извлечет ссылку, а затем запустит клиента qBittorrent для загрузки соответствующего файла. Таким образом, ваш компьютер сможет загружать файлы даже в ваше отсутствие, и к тому времени, когда вы вернетесь домой, файлы уже будут загружены.

В главе 15 речь шла о том, как запускать программы на компьютере с помощью функции `subprocess.Popen()`. Например, следующий код запустит программу qBittorrent для указанного торрент-файла.

```
qbProcess = subprocess.Popen(['C:\\Program Files (x86)\\
↳qBittorrent\\qbittorrent.exe',
↳shakespeare_complete_works.torrent'])
```

Разумеется, вы захотите, чтобы программа проверяла, поступило ли данное сообщение именно от вас. Для этого, в частности, можно потребовать, чтобы в сообщении содержался пароль, поскольку хакерам не составляет труда подделать адрес в поле "От" сообщения электронной почты. Программа должна удалять все подходящие сообщения, которые она обнаружит, чтобы не выполнять инструкции повторно каждый раз при проверке электронной почты. В качестве дополнительной возможности можете предусмотреть, чтобы программа отправляла вам электронное письмо или текстовое сообщение, подтверждающие начало выполнения порученной работы. Поскольку во время выполнения программы вы не будете сидеть за компьютером, будет неплохо, если вы организуете ведение журнала (см. главу 10), записываемого в текстовый файл, чтобы вы могли проверить, возникали ли ошибки в процессе загрузки файлов.

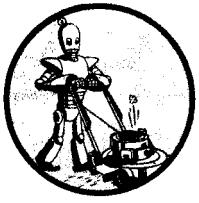
В программе qBittorrent (равно как и в других приложениях BitTorrent) предусмотрена возможность автоматического выхода по окончании

процесса загрузки. В главе 15 речь шла о том, как проконтролировать завершение работы запущенной программы с помощью метода `wait()` объектов `Popen`. Вызов метода `wait()` будет блокироваться до тех пор, пока выполнение программы qBittorrent не будет прекращено, после чего ваша программа сможет отправить электронное почтовое или текстовое сообщение, уведомляющее вас о завершении процесса загрузки.

Существует множество других возможностей, которые могут быть добавлены в подобный проект. Если вы столкнетесь с какими-либо трудностями, то можете загрузить примерную реализацию такой программы, посетив сайт `http://nostarch.com/automatestuff/`.

17

РАБОТА С ИЗОБРАЖЕНИЯМИ



Если у вас есть цифровая камера или же вы просто размещаете свои фотографии на Facebook, то вам, вероятно, постоянно приходится иметь дело с файлами цифровых изображений. Возможно, вы даже умеете пользоваться простыми программами для обработки графики, такими как Microsoft Paint или Paintbrush, или даже более сложными, как, например, Adobe Photoshop. Но если вам нужно отредактировать большой массив изображений, такая рутинная обработка вручную может отнять у вас много времени.

Воспользуйтесь возможностями Python. Pillow — это библиотека Python от сторонних разработчиков, предназначенная для взаимодействия с файлами изображений. В данной библиотеке предусмотрен ряд функций, упрощающих выполнение таких операций, как обрезка, масштабирование и изменение содержимого изображений. Располагая возможностями манипулирования изображениями примерно с той же эффективностью, какую обеспечивают такие программы, как Microsoft Paint, Python позволяет легко автоматизировать редактирование сотен и даже тысяч изображений.

Основы компьютерной обработки изображений

Чтобы манипулировать изображениями, нужно хотя бы на элементарном уровне понимать, как компьютеры обрабатывают графическую информацию и как выполнять соответствующие операции с помощью библиотеки Pillow. Но прежде чем мы продолжим наше рассмотрение, вам необходимо установить модуль `pillow`. Подробное описание процедуры установки модулей сторонних разработчиков приведено в приложении А.

Цвета и RGBA-значения

В компьютерных программах для представления цвета часто используют *RGBA-значения*. Значение RGBA — это группа чисел, задающих долю красной (red), зеленой (green), синей (blue) и *альфа*-составляющей (прозрачности)

в цвете. Каждое из этих значений представляет собой целое число в интервале от 0 (отсутствие данного цвета) до 255 (максимум). Эти RGBA-значения присваиваются отдельным пикселям. *Пиксель* — это наименьший элемент изображения, который может быть представлен на компьютерном экране (количество пикселей на экране исчисляется миллионами). RGB-значения пикселей в точности определяют оттенок цвета для отображения пикселя. Использование альфа-составляющей приводит к созданию RGBA-значений. Если изображение выводится на экране поверх фонового изображения или обоев рабочего стола, то значение параметра "альфа" (или, как говорят, альфа-канала) определяет, насколько интенсивно должен "просматриваться" фон через пиксель изображения.

В библиотеке Pillow значения RGBA представляются кортежем из четырех целочисленных значений. Например, красный цвет представляется кортежем `(255, 0, 0, 255)`. В этом цвете содержится максимальное количество красного цвета, зеленый и синий цвета отсутствуют, а альфа-составляющая имеет максимальное значение, которому соответствует полная непрозрачность. Зеленый цвет представляется кортежем `(0, 255, 0, 255)`, а синий — кортежем `(0, 0, 255, 255)`. Белый цвет, являющийся сочетанием всех цветов, представляется кортежем `(255, 255, 255, 255)`, тогда как черный, соответствующий полному отсутствию цветов, — кортежем `(0, 0, 0, 255)`.

Если значение альфа-канала в цвете равно 0, то этот цвет — невидимый, и в действительности от конкретных значений параметров RGB ничего не зависит. В конце концов, невидимый красный — это все равно что невидимый черный.

В библиотеке Pillow используются стандартные названия цветов, принятые в HTML. В табл. 17.1 приведены некоторые стандартные названия цветов и соответствующие им RGBA-значения.

Таблица 17.1. Стандартные названия цветов и их RGBA-значения

Название	Значение RGBA	Название	Значение RGBA
Белый	(255, 255, 255, 255)	Красный	(255, 0, 0, 255)
Зеленый	(0, 128, 0, 255)	Синий	(0, 0, 255, 255)
Серый	(128, 128, 128, 255)	Желтый	(255, 255, 0, 255)
Черный	(0, 0, 0, 255)	Пурпурный	(128, 0, 128, 255)

Библиотека Pillow предлагает функцию `ImageColor.getcolor()`, которая избавляет от необходимости запоминать RGBA-значения цветов, планируемых к использованию. Эта функция принимает название цвета в качестве первого аргумента и строку `'RGBA'` в качестве второго аргумента, а возвращает кортеж значений RGBA.

Цветовые схемы CMYK и RGB

Из курса физики средней школы вам должно быть известно, что смешиванием красной, желтой и синей красок можно получить все остальные цвета. Например, если вы смешаете синюю и желтую краски, то получите зеленую. Эта схема называется *субтрактивной цветовой моделью* и применяется при производстве красителей, чернил и пигментов. По этой причине цветные принтеры оборудуются чернильными картриджами CMYK: смешивание чернил Cyan (голубой), Magenta (пурпурный), Yellow (желтый) и blacK (черный) позволяет сформировать любой цвет.

Однако в физике света используется *аддитивная цветовая модель*. Комбинируя лучи света различной частоты (как в случае света, излучаемого компьютерным экраном), можно сочетать красный, зеленый и синий световые лучи для формирования любого другого цвета. Вот почему в компьютерных программах цвета представляются RGB-значениями.

Чтобы увидеть, как работает эта функция, введите в интерактивной оболочке следующий код.

```
❶ >>> from PIL import ImageColor
❷ >>> ImageColor.getcolor('red', 'RGBA')
  (255, 0, 0, 255)
❸ >>> ImageColor.getcolor('RED', 'RGBA')
  (255, 0, 0, 255)
  >>> ImageColor.getcolor('Black', 'RGBA')
  (0, 0, 0, 255)
  >>> ImageColor.getcolor('chocolate', 'RGBA')
  (210, 105, 30, 255)
  >>> ImageColor.getcolor('CornflowerBlue', 'RGBA')
  (100, 149, 237, 255)
```

Прежде всего, необходимо импортировать модуль `ImageColor` из модуля PIL ❶ (обратите внимание — не Pillow; вскоре вы увидите, почему так). Строка с названием цвета, которую вы передаете функции `ImageColor.getcolor()`, нечувствительна к регистру, поэтому при передаче как `'red'` ❷, так и `'RED'` ❸ мы получаем один и тот же кортеж RGBA. Возможна передача и таких необычных названий цветов, как `'chocolate'` (шоколадный) и `'Cornflower Blue'` (васильковый). Библиотека Pillow поддерживает огромное количество названий цветов, от `'aliceblue'` до `'whitesmoke'`. Полный список, включающий более 100 стандартных названий цветов, можно найти в ресурсах, предоставляемых на сайте `http://nostarch.com/automatestuff/`.

Кортежи координат и прямоугольников

Для адресации пикселей изображений используют координаты *x* и *y*, характеризующие расположение пикселя в изображении соответственно в горизонтальном и вертикальном направлениях. *Началом отсчета* служит пиксель, располагающийся в верхнем левом углу изображения, координаты которого записываются в виде `(0, 0)`. Первый нуль представляет *x*-координату, значения которой начинаются с нуля и увеличиваются в направлении слева направо. Второй нуль представляет *y*-координату, значения которой начинаются с нуля и увеличиваются в направлении сверху вниз. Последнее утверждение стоит повторить: *y*-координаты увеличиваются в направлении вниз – противоположном тому, которое предполагается в математике. На рис. 17.1 продемонстрировано, как работает эта система координат.

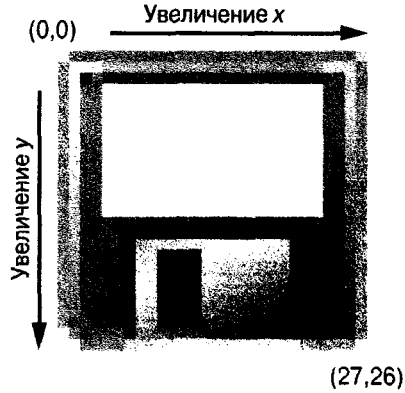


Рис. 17.1. Координаты x и y для области изображения размером 27×26 пикселей, представляющего некое древнее устройство для хранения данных

Многие из функций и методов Pillow принимают в качестве аргумента *кортеж прямоугольника*. Это означает, что они ожидают кортеж из четырех целочисленных координат, который представляет прямоугольную область изображения. Вот что означают эти целые числа в порядке их следования.

- **Левая:** *x*-координата левой стороны прямоугольника.
- **Верхняя:** *y*-координата верхней стороны прямоугольника.
- **Правая:** *x*-координата внешнего пикселя, примыкающего к правой стороне прямоугольника. Это число должно быть больше того, которое определяет положение левой стороны.
- **Нижняя:** *y*-координата внешнего пикселя, примыкающего к нижней стороне прямоугольника. Это число должно быть больше того, которое определяет положение верхней стороны.

Обратите внимание на то, что прямоугольник включает точки, соответствующие координатам *левая* и *верхняя*, и в то же время, простираясь до координат *правая* и *нижняя*, не включает соответствующие им точки. Например, кортеж `(3, 1, 9, 6)` представляет все пиксели, принадлежащие области черного прямоугольника, показанного на рис. 17.2.

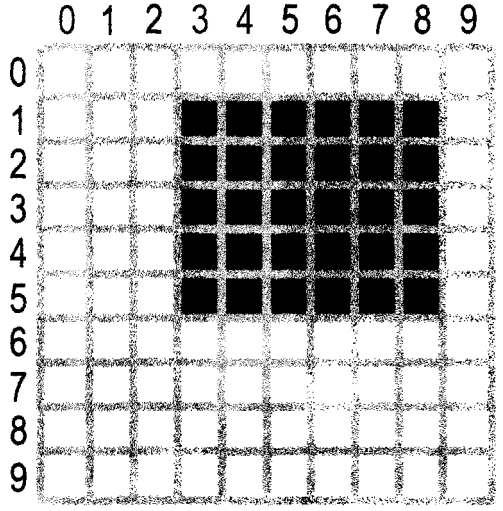


Рис. 17.2. Область, представляемая кортежем прямоугольника (3, 1, 9, 6)

Манипулирование изображениями с помощью библиотеки Pillow

Теперь, когда вам уже известно, как обрабатываются цвета и координаты в Pillow, мы используем эту библиотеку для манипулирования изображениями. На рис. 17.3 показано изображение, которое мы будем использовать в этой главе в примерах, выполняемых в интерактивной оболочке. Изображение доступно для загрузки на сайте `http://nostarch.com/automatestuff/`.

Поместив файл *Zophie.png* в текущий рабочий каталог, вы сможете загружать изображение Зофи в Python следующим образом.

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.tiff')
```

Чтобы загрузить изображение, вы импортируете модуль `Image` из библиотеки Pillow и вызываете функцию `Image.open()`, передавая ей имя файла изображения. Затем вы можете сохранить изображение в переменной `CatIm`. В качестве имени модуля Pillow используется `PIL`, чтобы обеспечить

обратную совместимость со старым модулем, который называется "Python Imaging Library", и именно по этой причине вы должны выполнять команду `from PIL import Image`, а не просто `from Pillow import Image`. В силу особенностей способа настройки модуля `pillow` создателями Pillow необходимо использовать команду импорта вида `from PIL import Image`, а не вида `import PIL`.

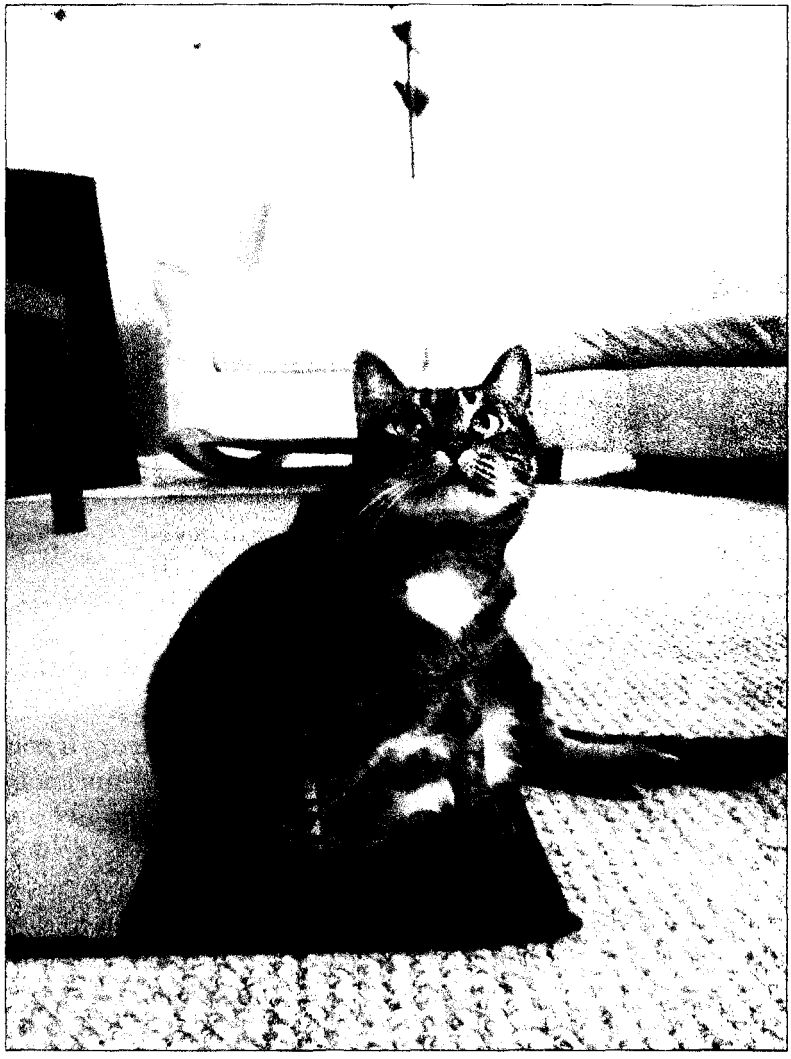


Рис. 17.3. Моя кошка Зофи. Камера добавила ей лишних 10 фунтов (весьма заметное прибавление веса для кошки)

Если файл изображения находится не в текущем каталоге, сделайте рабочим каталогом папку, в которой содержится файл изображения, вызвав функцию `os.chdir()`.

```
>>> import os
>>> os.chdir('C:\\папка_с_файлом_изображения')
```

Функция `Image.open()` возвращает значение в виде объекта типа `Image`: именно так Pillow представляет изображения значениями Python. Объект `Image` можно загрузить из файла изображения (любого формата), передав функции `Image.open()` строку с именем файла. Любые изменения, внесенные в объект `Image`, могут быть сохранены в файле изображения (также в любом формате) с помощью метода `save()`. Все операции поворота, изменения размеров, обрезки, рисования и другие манипуляции изображением осуществляются посредством вызовов методов для этого объекта `Image`.

Чтобы сократить размер примеров, приводимых в этой главе, предположим, что вы уже импортировали модуль Pillow `Image` и сохранили изображение Зофи в переменной `catIm`. Проследите за тем, чтобы файл *zophie.png* находился в текущем рабочем каталоге, где его сможет найти функция `Image.open()`. В противном случае нужно будет указать полный абсолютный путь к файлу в строковом аргументе, передаваемом методу `Image.open()`.

Работа с типом данных Image

Объект `Image` имеет несколько полезных атрибутов, предоставляющих основную информацию относительно файла изображения, из которого он был загружен: ширину и высоту изображения, имя файла и графический формат (например, JPEG, GIF или PNG).

Например, введите в интерактивной оболочке следующие команды.

```
>>> from PIL import Image
>>> catIm = Image.open('zophie.png')
>>> catIm.size
❶ (816, 1088)
❷ >>> width, height = catIm.size
❸ >>> width
816
❹ >>> height
1088
>>> catIm.filename
'zophie.png'
>>> catIm.format
'PNG'
>>> catIm.format_description
'Portable network graphics'
❺ >>> catIm.save('zophie.jpg')
```

Создав объект `Image` на основе файла *Zophie.png* и сохранив его в переменной `catIm`, мы видим, что атрибут `size` объекта содержит кортеж значений

ширины и высоты изображения, выраженных в пикселях ❶. Эти значения можно назначить переменным `width` и `height` ❷, что обеспечит возможность независимого доступа к значениям ширины ❸ и высоты ❹ изображения. Атрибут `filename` содержит имя исходного файла. Атрибуты `format` и `format_description` — это строки, содержащие описание формата изображения в исходном файле (причем в атрибуте `format_description` содержится несколько более развернутое описание формата).

Наконец, вызвав метод `save()` с передачей ему строки `'zophie.jpg'` в качестве параметра, мы сохраняем изображение в новом файле *zophie.jpg* на жестком диске ❺. Модуль Pillow замечает, что расширением имени файла является *.jpg*, и автоматически сохраняет изображение с использованием формата JPEG. Теперь на вашем жестком диске имеются два изображения — *zophie.png* и *zophie.jpg*. Несмотря на то что оба эти файла основаны на одном и том же изображении, они не идентичны, поскольку имеют различные форматы.

Кроме того, модуль Pillow предоставляет функцию `Image.new()`, которая возвращает объект `Image` аналогично тому, как это делает функция `Image.open()`, однако в данном случае изображение, представляемое объектом `Image.new()`, будет пустым. Аргументы функции `Image.new()` описаны ниже.

- Строка `'RGBA'`, устанавливающая цветовую модель RGBA. (Также возможна работа с другими цветовыми моделями, которые в данной книге не рассматриваются.)
- Размер в виде кортежа из двух значений, представляющих ширину и высоту нового изображения.
- Цвет фона, определяющий начальный вид изображения, который задается кортежем из четырех целочисленных значений, представляющих значение RGBA. В качестве этого аргумента можно использовать значение, возвращаемое функцией `ImageColor.getcolor()`. Другим возможным вариантом является передача функции `Image.new()` строки, содержащей стандартное название цвета.

Введите в интерактивной оболочке следующие команды в качестве примера.

```
>>> from PIL import Image
❶ >>> im = Image.new('RGBA', (100, 200), 'purple')
>>> im.save('purpleImage.png')
❷ >>> im2 = Image.new('RGBA', (20, 20))
>>> im2.save('transparentImage.png')
```

В этом примере мы создаем объект `Image` для изображения, ширина и высота которого составляют соответственно 100 и 200 пикселей и которое

имеет пурпурный цвет фона ❶. Затем это изображение сохраняется в файле *purpleImage.png*. Далее мы вновь вызываем функцию `Image.new()` для создания другого объекта `Image`, на этот раз с размерами `(20, 20)`, но без указания цвета фона ❷. В тех случаях, когда цвет не задается, по умолчанию используется невидимый черный цвет, `(0, 0, 0, 0)`, поэтому второе изображение имеет прозрачный фон; это прозрачное прямоугольное изображение с размерами `20×20` мы сохраняем в файле *transparentImage.png*.

Обрезка изображений

Под *обрезкой* изображения понимают выбор прямоугольной области внутри изображения и удаление всего, что находится за пределами этого прямоугольника. Метод `crop()` объекта `Image` принимает кортеж прямоугольника и возвращает объект `Image`, представляющий обрезанное изображение. Процесс обрезки выполняется не на месте, т.е. исходный объект `Image` остается нетронутым, и метод `crop()` возвращает новый объект `Image`. Вспомните о том, что кортеж прямоугольника, в данном случае — области обрезки, включает левый столбец и верхнюю строку пикселей и лишь достигает правого столбца и нижней строки пикселей, но *не* включает их.

Введите в интерактивной оболочке следующие команды.

```
>>> croppedIm = catIm.crop((335, 345, 565, 560))
>>> croppedIm.save('cropped.png')
```

Здесь мы создаем новый объект `Image` для обрезанного изображения, сохраняем его в переменной `croppedIm`, а затем вызываем метод `save()`, чтобы сохранить обрезанное изображение в файле *cropped.png*. В результате этого на основе исходного изображения создается новый файл *cropped.png* (рис. 17.4).

Копирование и вставка изображений в другие изображения

Метод `copy()` возвращает новый объект `Image` с тем же изображением, что и объект `Image`, для которого он был вызван. Это может пригодиться в тех случаях, когда необходимо получить измененную версию изображения, но при этом сохранить нетронутым оригинал. Введите в интерактивной оболочке следующие команды.

```
>>> catIm = Image.open('zophie.png')
>>> catCopyIm = catIm.copy()
```

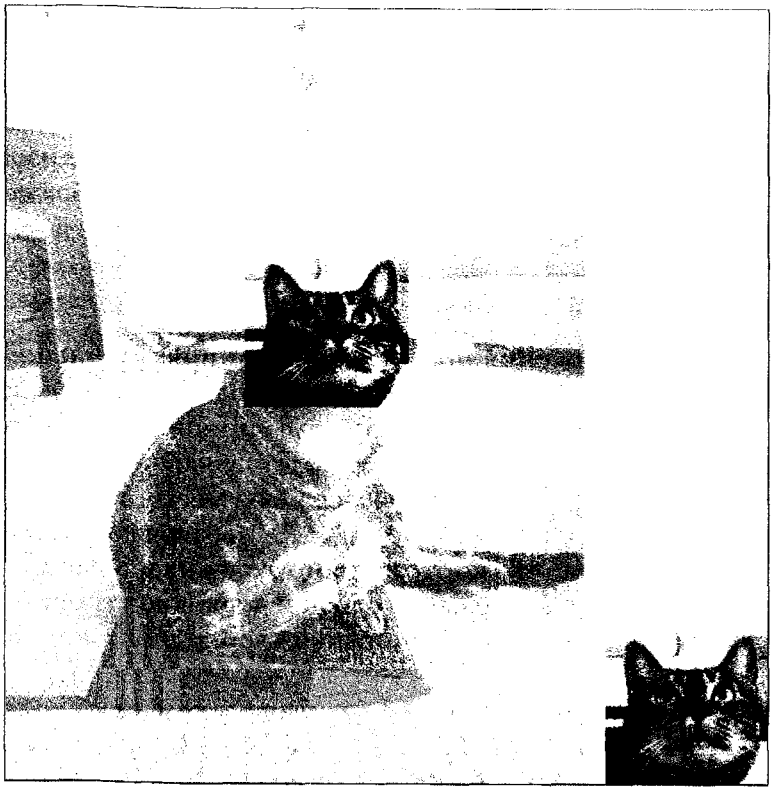


Рис. 17.4. Новое изображение будет лишь частью исходного

В переменных `catIm` и `catCopyIm` содержатся два независимых объекта `Image`, в каждом из которых хранится одно и то же изображение. Теперь, когда в переменной `catCopyIm` хранится копия исходного объекта `Image`, вы можете изменить ее по своему усмотрению и сохранить в другом файле, оставив файл *zophie.png* нетронутым. В качестве примера изменим изображение, хранящееся в переменной `catCopyIm`, с помощью метода `paste()`.

Будучи вызванным для объекта `Image`, метод `paste()` помещает поверх него другое изображение. Продолжим выполнение примера в интерактивной оболочке, вставив поверх изображения, хранящегося в переменной `catCopyIm`, изображение с меньшими размерами.

```
>>> faceIm = catIm.crop((335, 345, 565, 560))
>>> faceIm.size
(230, 215)
>>> catCopyIm.paste(faceIm, (0, 0))
>>> catCopyIm.paste(faceIm, (400, 500))
>>> catCopyIm.save('pasted.png')
```

Сначала мы передаем методу `crop()` кортеж прямоугольника, задающий область изображения `zophie.png`, соответствующую голове Зофи. В результате создается объект `Image`, представляющий обрезанное изображение с размерами 230×215, которое сохраняется в переменной `faceIm`. Теперь мы можем поместить это изображение поверх изображения `catCopyIm`. Метод `paste()` принимает два аргумента: объект `Image` вставляемого изображения и кортеж, определяющий координаты *x* и *y* точки на основном изображении, в которую должен быть помещен верхний левый угол вставляемого изображения. В данном примере метод `paste()` вызывается для переменной `catCopyIm` дважды: первый раз с передачей кортежа `(0, 0)` и второй раз — с передачей кортежа `(400, 500)`. В результате изображение `faceIm` помещается поверх изображения `catCopyIm` дважды. В первом случае верхний левый угол изображения `faceIm` помещается в точку `(0, 0)` изображения `catCopyIm`, а во втором — в точку `(400, 500)`. Наконец, мы сохраняем измененное изображение в файле *pasted.png*. Изображение, сохраненное в файле *pasted.png*, выглядит так, как показано на рис.17.5.

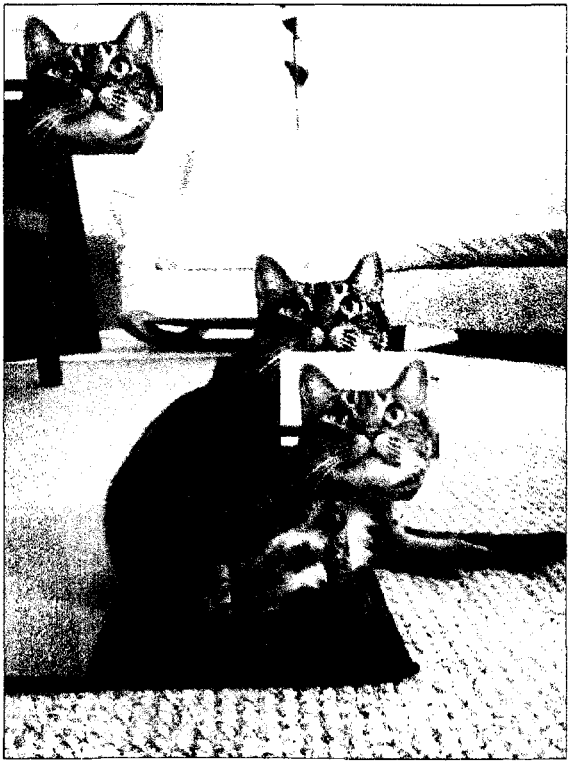


Рис. 17.5. Мордочка Зофи была скопирована дважды

Примечание

Пусть названия методов `copy()` *и* `paste()` *модуля Pillow не вводят вас в заблуждение: их работа никоим образом не связана с буфером обмена вашего компьютера.*

Обратите внимание на то, что метод `paste()` изменяет объект `Image` на месте, а не возвращает объект `Image` со вставленным изображением. Если вы хотите вызвать метод `paste()` и при этом сохранить нетронутой версию исходного изображения, то сначала создайте его копию, а затем вызовите метод `paste()` для этой копии.

Предположим, вы хотите покрыть изображениями головы Зофи всю область основного изображения (рис. 17.6). Для создания этого эффекта нам хватит пары циклов. Продолжите выполнение примера в интерактивной оболочке, введя следующие команды.

```
>>> catImWidth, catImHeight = catIm.size
>>> faceImWidth, faceImHeight = faceIm.size
❶ >>> catCopyTwo = catIm.copy()
❷ >>> for left in range(0, catImWidth, faceImWidth):
❸         for top in range(0, catImHeight, faceImHeight):
              print(left, top)
              catCopyTwo.paste(faceIm, (left, top))
0 0
0 215
0 430
0 645
0 860
0 1075
230 0
230 215

--пропущенный код--

690 860
690 1075
>>> catCopyTwo.save('tiled.png')
```

Здесь значения ширины и высоты изображения `catIm` сохраняются в переменных `catImWidth` и `catImHeight`. В строке ❶ мы создаем копию `catIm` и сохраняем ее в переменной `catCopyTwo`. Теперь, когда у нас есть копия, на которую можно поместить другие изображения, мы организуем цикл для вставки изображения `faceIm` поверх изображения `catCopyTwo`. Значение переменной `left` при запуске внешнего цикла `for` равно `0` и на каждой итерации этого цикла получает приращение, равное `faceImWidth` (`230`) ❷. Значение переменной `top` при запуске внутреннего цикла равно `0` и на каждой итерации этого цикла получает приращение, равное `faceImHeight` (`215`) ❸. Получаемые в этих вложенных циклах значения переменных `left` и `top` обеспечивают

покрытие всего изображения `catCopyTwo` изображениями `faceIm` (рис. 17.6). Чтобы можно было проследить за тем, как работают оба цикла, мы выводим значения переменных `left` и `top`. По завершении вставки всех изображений мы сохраняем измененное изображение `catCopyTwo` в файле `tiled.png`.

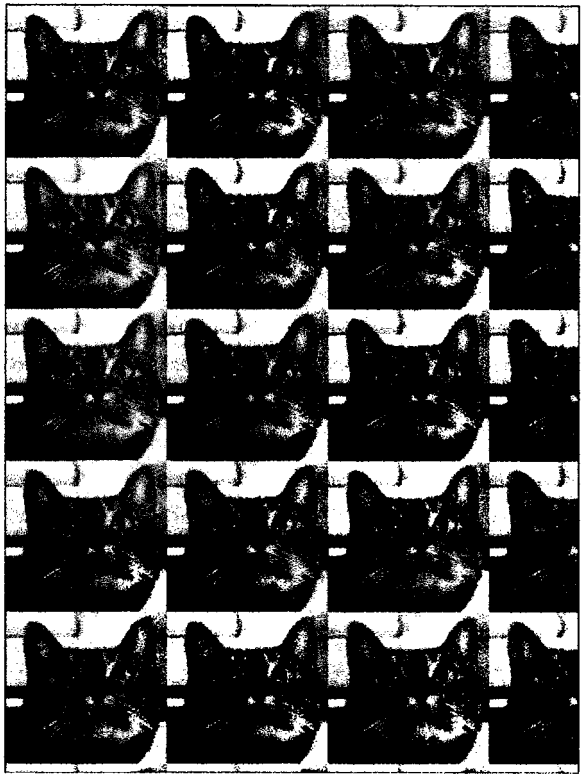


Рис. 17.6. Использование вложенных циклов и метода `paste()` для создания дубликатов изображения головы кошки

Вставка прозрачных пикселей

Обычно прозрачные пиксели вставляются в виде белых пикселей. Если в изображении, которое вы хотите вставить, имеются прозрачные пиксели, то передайте методу `paste()` объект `Image` в качестве третьего аргумента, предотвращающего вставку прямоугольника, закрашенного сплошным цветом. Этот третий аргумент является "маской" объекта `Image`. *Маска* — это объект `Image`, в котором учитывается лишь значение альфа-канала, тогда как красная, зеленая и синяя составляющие игнорируются. Маска сообщает методу `paste()`, какие пиксели следует копировать, а какие оставить прозрачными. Подробное рассмотрение масок выходит за рамки данной книги, но если вы хотите вставить изображение, содержащее прозрачные пиксели, то передайте объект `Image` методу `paste()` вновь в качестве третьего аргумента.

Изменение размеров изображения

Метод `resize()` вызывается для объекта `Image` и возвращает новый объект `Image` с заданными значениями ширины и высоты. Он принимает аргумент в виде кортежа из двух целочисленных значений, представляющих новые значения ширины и высоты возвращаемого объекта. Введите в интерактивной оболочке следующие команды.

```
❶ >>> width, height = catIm.size
❷ >>> quartersizedIm = catIm.resize((int(width / 2),
  ↳ int(height / 2)))
  >>> quartersizedIm.save('quartersized.png')
❸ >>> svelteIm = catIm.resize((width, height + 300))
  >>> svelteIm.save('svelte.png')
```

Здесь два значения, образующие кортеж `catIm.size`, присваиваются переменным `width` и `height` ❶. Использование отдельных переменных `width` и `height` вместо элементов `catIm.size[0]` и `catIm.size[1]` повышает удобочитаемость остальной части кода.

В первом вызове метода `resize()` ему передаются значения `int(width / 2)` и `int(height / 2)` в качестве новых значений ширины и высоты ❷, поэтому объект `Image`, возвращенный методом `resize()`, будет иметь половинные значения ширины и высоты исходного изображения, т.е. в целом будет в четыре раза меньше него. Метод `resize()` принимает в качестве аргумента лишь кортеж целочисленных значений, и именно поэтому обе операции деления на 2 должны быть обернуты в вызовы `int()`.

В данном случае ширина и высота изменяются в одинаковой пропорции. Однако новые значения ширины и высоты, передаваемые методу `resize()`, не обязаны сохранять исходные пропорции изображения. Переменная `svelteIm` содержит объект `Image`, ширина которого совпадает с первоначальной, но высота увеличена на 300 пикселей ❸, что делает Зофи более стройной.

Заметьте, что метод `resize()` не изменяет объект `Image` на месте, а возвращает новый объект `Image`.

Поворот и зеркальное отображение изображений

Изображения можно поворачивать с помощью метода `rotate()`, который возвращает новый объект `Image` повернутого изображения, оставляя исходный объект Image неизменным. Аргументом метода `rotate()` является целое или вещественное число, представляющее величину угла поворота в градусах против часовой стрелки. Введите в интерактивной оболочке следующие команды.

```
>>> catIm.rotate(90).save('rotated90.png')
>>> catIm.rotate(180).save('rotated180.png')
>>> catIm.rotate(270).save('rotated270.png')
```

Обратите внимание на то, как здесь используется возможность образования цепочек вызовов методов, когда метод `save()` вызывается непосредственно для объекта `Image`, возвращенного методом `rotate()`. Вызовы методов `rotate()` и `save()` создают новый объект `Image`, который представляет изображение, повернутое на 90° против часовой стрелки, и сохраняют это изображение в файле *rotated90.png*. Во второй и третьей строках кода делается то же самое, но с поворотом исходного изображения на 180° и 270° соответственно. Результат представлен на рис. 17.7.

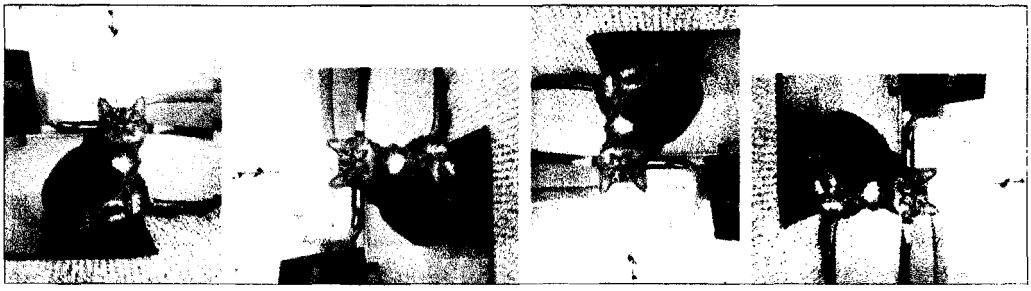


Рис. 17.7. Исходное изображение (слева) и изображения, повернутые на 90°, 180° и 270°

Заметьте, что при повороте на 90° или 270° ширина и высота изображения изменяются. При повороте на другие углы поддерживаются исходные размеры изображения. В Windows для заполнения образуемых зазоров используется черный фон. В OS X для этого используются прозрачные пиксели.

В методе `rotate()` предусмотрен необязательный именованный аргумент `expand`, установка значения которого в `True` приводит к увеличению размеров изображения таким образом, чтобы оно вписалось в ограничивающий прямоугольник нового, повернутого изображения. Например, введите в интерактивной оболочке следующие команды.

```
>>> catIm.rotate(6).save('rotated6.png')
>>> catIm.rotate(6, expand=True).save('rotated6_expanded.png')
```

В первой строке кода изображение поворачивается на 6° и сохраняется в файле *rotate6.png* (рис. 17.8, *слева*). Во второй строке изображение поворачивается на 6° при значении аргумента `expand`, равном `True`, и сохраняется в файле *rotate6_expanded.png* (рис. 17.8, *справа*).

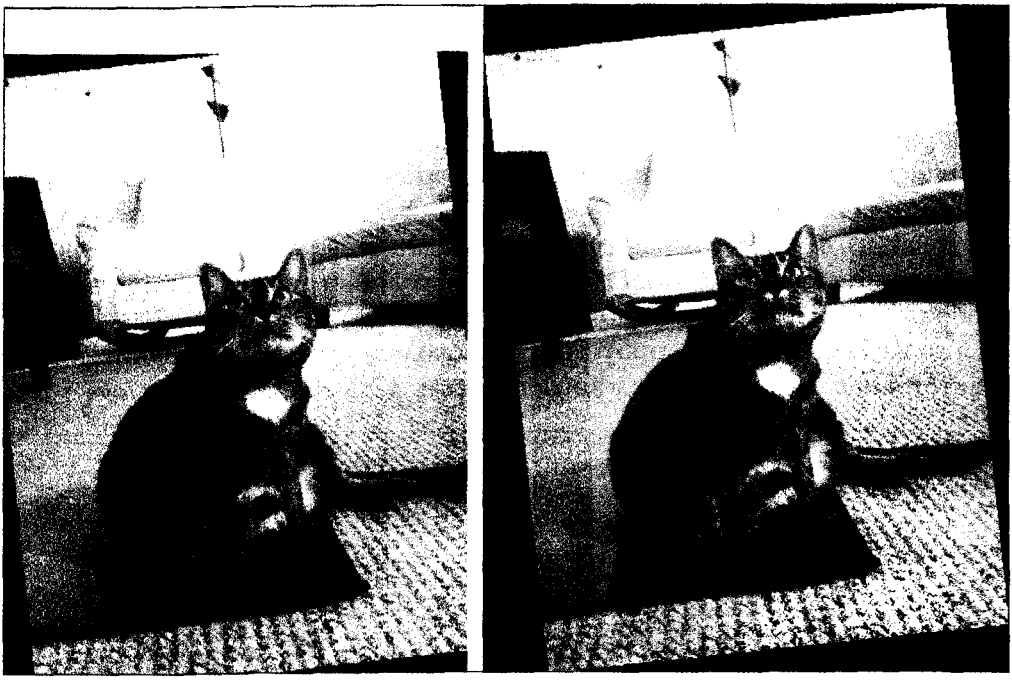


Рис. 17.8. Изображение, повернутое на 6° в обычном режиме (слева) и в режиме `expand=True` *(справа)*

У вас также есть возможность получить "зеркально отображенное" изображение с помощью метода `transpose()`. Методу `transpose()` должен передаваться аргумент `Image.FLIP_LEFT_RIGHT` или `Image.FLIP_TOP_BOTTOM`. Введите в интерактивной оболочке следующие команды.

```
>>> catIm.transpose(Image.FLIP_LEFT_RIGHT).
↳ save('horizontal_flip.png')
>>> catIm.transpose(Image.FLIP_TOP_BOTTOM).
↳ save('vertical_flip.png')
```

Как и метод `rotate()`, метод `transpose()` создает новый объект `Image`. В этом коде данному методу передается аргумент `Image.FLIP_LEFT_RIGHT`, в результате чего изображение отражается по горизонтали, а затем сохраняется с помощью метода `save()` в файле *horizontal_flip.png*. Для отображения по вертикали мы передаем методу `transpose()` аргумент `Image.FLIP_TOP_BOTTOM`, а затем сохраняем результирующее изображение в файле *vertical_flip.png*. Полученные в результате этих преобразований изображения представлены на рис. 17.9.

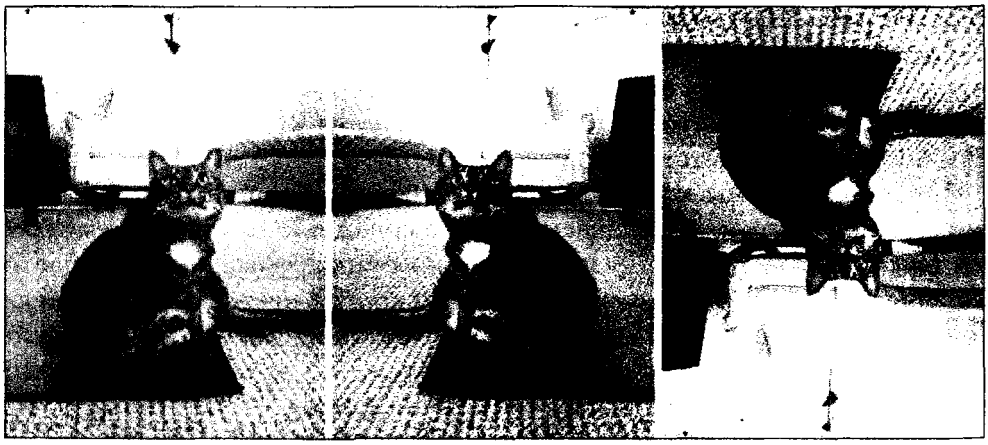


Рис. 17.9. Исходное изображение (слева), результат горизонтального отражения (в центре) и результат вертикального отражения (справа)

Изменение отдельных пикселей

Для получения или установки цвета отдельного пикселя используются методы `getpixel()` и `putpixel()`. Оба метода принимают в качестве аргумента кортеж, представляющий координаты *x* и *y* пикселя. Кроме того, метод `putpixel()` принимает дополнительный аргумент в виде кортежа, определяющего цвет пикселя. Этим аргументом может быть либо RGBA-кортеж из трех целых чисел, либо RGB-кортеж, включающий три целых числа. Введите в интерактивной оболочке следующие команды.

```
❶ >>> im = Image.new('RGBA', (100, 100))
❷ >>> im.getpixel((0, 0))
  (0, 0, 0, 0)
❸ >>> for x in range(100):
          for y in range(50):
❹             im.putpixel((x, y), (210, 210, 210))
  >>> from PIL import ImageColor
❺ >>> for x in range(100):
          for y in range(50, 100):
❻             im.putpixel((x, y), ImageColor.
                                  getcolor('darkgray', 'RGBA'))
  >>> im.getpixel((0, 0))
  (210, 210, 210, 255)
  >>> im.getpixel((0, 50))
  (169, 169, 169, 255)
  >>> im.save('putPixel.png')
```

В строке ❶ мы создаем новое изображение в виде прозрачного квадрата с размерами 100×100. Вызов метода `getpixel()` для одной из точек этого изображения возвращает кортеж `(0, 0, 0, 0)`, поскольку изображение

прозрачно ❷. Для назначения цвета пикселям этого изображения мы можем использовать вложенные циклы `for`, перебирая все пиксели в верхней половине изображения ❸ и вызывая для каждого из них метод `putpixel()`. ❹. В данном случае методу `putpixel()` передается RGB-кортеж `(210, 210, 210)`, которому соответствует светло-серый цвет.

Предположим, мы хотим закрасить нижнюю половину изображения темно-серым цветом, но не знаем, какие значения RGB-кортежа ему соответствуют. Метод `putpixel()` не принимает стандартные названия цветов наподобие `'darkgray'`, поэтому мы получаем кортеж цветовых значений для цвета `'darkgray'` с помощью функции `ImageColor.getcolor()`. Организуйте цикл по пикселям нижней половины изображения ❺ с передачей методу `putpixel()` значения, возвращаемого функцией `ImageColor.getcolor()` ❻, и вы получите изображение, верхняя половина которого закрашена светло-серым цветом, а нижняя — темно-серым (рис. 17.10). Для проверки того, что цвет любого заданного пикселя соответствует ожидаемому, можно вызвать метод `getpixel()`. Наконец сохраните изображение в файле *putPixel.png*.

Разумеется, рисовать изображение по одному пикселю за один раз не очень удобно. Если вам надо рисовать фигуры, используйте функции `ImageDraw`, о которых речь пойдет далее.

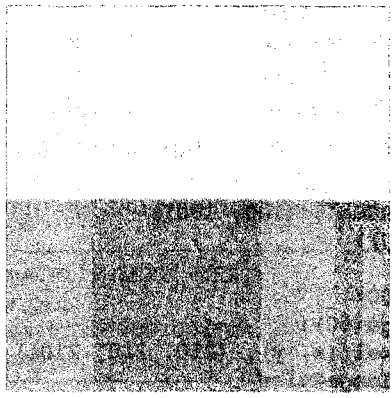


Рис. 17.10. Изображение putPixel.png

Проект: добавление логотипа

Предположим, вам предстоит рутинная работа по изменению размеров тысяч изображений и добавления в углу каждого из них небольшого логотипа в виде водяного знака. Выполнение такой задачи с помощью простых графических программ наподобие Paintbrush или Paint длилось бы целую вечность. В более сложных графических приложениях, таких как Photoshop, существует возможность пакетной обработки, но такое программное обеспечение стоит не одну сотню долларов.

На рис. 17.11 показан логотип, который мы будем добавлять в нижний правый угол каждого изображения. Этот логотип представляет собой стилизованный профиль кошки с белым контуром и прозрачной остальной частью изображения.

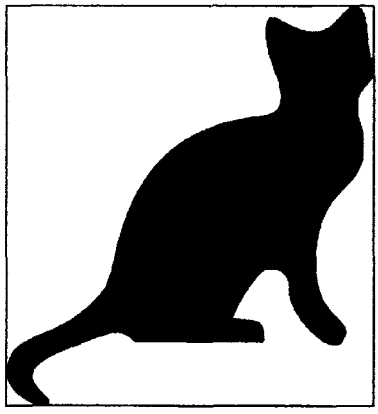


Рис. 17.11. Логотип, который будет добавляться в изображение

Вот что должна делать данная программа при высокоуровневом рассмотрении:

- загружать изображение логотипа;
- обходить в цикле все файлы с расширениями *.png* и *.jpg* в рабочем каталоге;
- проверять, не превышает ли ширина или высота изображения 300 пикселей;
- в случае указанного превышения уменьшать ширину или высоту (в зависимости от того, что больше) до 300 пикселей, уменьшая другой размер в той же пропорции;
- вставлять логотип в угол изображения;
- сохранять измененные изображения в другой папке.

Это означает, что код должен выполнять следующие операции:

- открывать файл *catlogo.png* в качестве объекта `Image`;
- проходить в цикле по всем строкам, возвращаемым функцией `os.listdir('.')`;
- получать ширину и высоту изображения из атрибута `size`;
- рассчитывать новые значения ширины и высоты изображения;
- вызывать метод `resize()` для изменения размеров изображения;
- вызывать метод `paste()` для вставки логотипа;
- вызывать метод `save()` для сохранения изменений, используя имя исходного файла.

Шаг 1. Открытие изображения логотипа

Для работы с этим проектом откройте новое окно в файловом редакторе, введите следующий код и сохраните его в файле *resizeAndAddLogo.py*.

```
#! python3
# resizeAndAddLogo.py - Изменяет размеры всех изображений
# в текущем рабочем каталоге таким образом, чтобы они
# вписывались в квадрат с размерами 300x300, и добавляет
# изображение catlogo.png в его нижний правый угол.

import os
from PIL import Image

❶ SQUARE_FIT_SIZE = 300
❷ LOGO_FILENAME = 'catlogo.png'

❸ logoIm = Image.open(LOGO_FILENAME)
❹ logoWidth, logoHeight = logoIm.size

# TODO: Организовать цикл по всем файлам в текущем
#       рабочем каталоге.

# TODO: Проверить, нуждается ли изображение в
#       изменении размеров.

# TODO: Рассчитать необходимые новые значения ширины и высоты.

# TODO: Изменить размеры изображения.

# TODO: Добавить логотип.

# TODO: Сохранить изменения.
```

Задав в начале программы значения констант `SQUARE_FIT_SIZE` ❶ и `LOGO_FILENAME` ❷, мы упростили внесение возможных изменений в программу в будущем. Предположим, вы захотите использовать в качестве логотипа другой рисунок или ограничить размеры логотипа не 300 пикселями, а другой величиной. Расположив определения этих констант в самом начале программы, вы должны будете внести изменения, если это потребуется, только в одном месте программы. (Возможен и другой вариант, когда значения этих констант предоставляются в виде аргументов командной строки при вызове программы.) Без использования этих констант вам пришлось бы просматривать весь код в поиске всех вхождений значений `300` и `'catlogo.png'` и заменять их вручную для каждого нового проекта. Короче говоря, константы делают программу более универсальной.

Объект `Image` логотипа возвращается вызовом функции `Image.open()` ❸. Для улучшения удобочитаемости кода программы значения, содержащиеся

в атрибуте `logoIm.size`, присваиваются отдельным переменным `logoWidth` и `logoHeight` ❹.

По состоянию на данный момент каркас остальной части программы представлен комментариями TODO.

Шаг 2. Цикл по всем файлам и открытым изображениям

Теперь мы должны организовать в текущем рабочем каталоге последовательный поиск всех *.png*- и *.jpg*-файлов. При этом следует учесть, что в добавлении изображения логотипа к самому логотипу нет никакой необходимости, и поэтому программа должна пропускать любое изображение с тем же именем файла, которое содержится в константе `LOGO_FILENAME`. Добавьте в программу следующий код.

```
#! python3
# resizeAndAddLogo.py - Изменяет размеры всех изображений
# в текущем рабочем каталоге таким образом, чтобы они
# вписывались в квадрат с размерами 300x300, и добавляет
# изображение catlogo.png в его нижний правый угол.

import os
from PIL import Image

--пропущенный код--

os.makedirs('withLogo', exist_ok=True)
# Цикл по всем файлам в текущем рабочем каталоге.
❶ for filename in os.listdir('.'):
❷     if not (filename.endswith('.png') or \
          filename.endswith('.jpg')) or filename == LOGO_FILENAME:
❸         continue # пропустить файлы, не являющиеся файлами
                   # изображений, и файл самого логотипа

❹     im = Image.open(filename)
      width, height = im.size

--пропущенный код--
```

Прежде всего, мы создаем с помощью вызова `os.makedirs()` отдельную папку `withLogo`, предназначенную для хранения версий изображений с логотипами, чтобы не затирать исходные файлы. Указав именованный аргумент `exist_ok=True`, можно избежать возбуждения исключений в методе `os.makedirs()` в том случае, если папка `withLogo` уже существует. В процессе выполнения цикла по всем файлам текущего рабочего каталога с использованием вызова `os.listdir('.')` ❶ длинная инструкция `if` ❷ проверяет расширение имени каждого файла. Если файл имеет расширение *.png* или *.jpg* или же если это файл самого изображения логотипа, то цикл должен

пропустить его и использовать инструкцию `continue` ❸ для перехода к следующему файлу. Если же имя файла заканчивается расширением *.png* или *.jpg* (и это не файл логотипа), то можно открыть его в виде объекта `Image` ❹ и задать ширину и высоту изображения.

Шаг 3. Изменение размеров изображений

Программа должна изменять размеры изображений лишь в том случае, если ширина или высота превышает значение, определяемое константой `SQUARE_FIT_SIZE` (в данном случае — 300 пикселей), поэтому необходимый для этого код следует поместить в инструкцию `if`, проверяющую значения переменных `width` и `height`. Добавьте в программу следующий код.

```
  #! python3
  # resizeAndAddLogo.py - Изменяет размеры всех изображений
  # в текущем рабочем каталоге таким образом, чтобы они
  # вписывались в квадрат с размерами 300x300, и добавляет
  # изображение catlogo.png в его нижний правый угол.

import os
from PIL import Image

--пропущенный код--

      # Проверка необходимости изменения размеров изображения.
      if width > SQUARE_FIT_SIZE and height > SQUARE_FIT_SIZE:
          # Расчет необходимых новых значений ширины и высоты.
          if width > height:
❶             height = int((SQUARE_FIT_SIZE / width) * height)
              width = SQUARE_FIT_SIZE
          else:
❷             width = int((SQUARE_FIT_SIZE / height) * width)
              height = SQUARE_FIT_SIZE

          # Изменение размеров изображения.
          print('Изменение размеров изображения %s...'
                                                % (filename))
❸             im = im.resize((width, height))

  --пропущенный код--
```

Если размеры изображения действительно надо изменить, то в этом случае следует определить, какой именно из размеров превышает допустимый предел — ширина или высота. Если ширина изображения больше его высоты, то последнюю следует уменьшить в той же пропорции, что и ширину ❶. Величина коэффициента пропорциональности находится делением значения `SQUARE_FIT_SIZE` на текущее значение ширины. Тогда новое значение высоты будет равно ее текущему значению, умноженному на найденное

значение коэффициента пропорциональности. Поскольку результатом операции деления является вещественное число, а метод `resize()` требует задания целочисленных размеров, не забудьте преобразовать полученный результат в целое число с помощью функции `int()`. Наконец, новое значение ширины просто устанавливается равным `SQUARE_FIT_SIZE`.

Случай, когда высота изображения больше ширины или равна ей (оба случая обрабатываются с помощью инструкции `else`), обрабатывается с использованием такой же расчетной схемы, за исключением того, что переменные `height` и `width` меняются ролями ❷.

Установив в переменных `width` и `height` новые значения размеров, передайте их методу `resize()` и сохраните возвращенный объект `Image` в переменной `im` ❸.

Шаг 4. Добавление логотипа и сохранение изменений

Независимо от того, изменялись ли размеры изображения, логотип должен помещаться в нижний правый угол изображения. В какую именно позицию он должен вставляться, определяется размерами как изображения, так и самого логотипа. На рис. 17.12 показано, как рассчитать позицию вставки. Левая координата позиции для вставки изображения должна быть равна разности между шириной изображения и шириной логотипа, тогда как верхняя координата должна быть равна разности между высотой изображения и высотой логотипа.

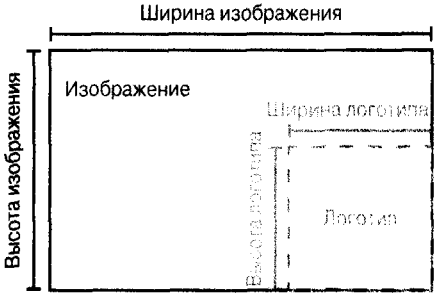


Рис. 17.12. Координаты левого и верхнего краев логотипа при его помещении в нижний правый угол изображения определяются соответственно разностью значений ширины изображения и ширины логотипа и высоты изображения и высоты логотипа

После того как ваш код вставит логотип в изображение, он должен сохранить измененный объект `Image`. Добавьте в программу следующий код.

```
#! python3
# resizeAndAddLogo.py - Изменяет размеры всех изображений
# в текущем рабочем каталоге таким образом, чтобы они
# вписывались в квадрат с размерами 300x300, и добавляет
```

```
  # изображение catlogo.png в его нижний правый угол.

  import os
  from PIL import Image

  --пропущенный код--

  # Проверка необходимости изменения размеров изображения.
  --пропущенный код--

  # Добавление логотипа.
❶ print('Добавление логотипа в изображение %s...' % (filename))
❷ im.paste(logoIm, (width - logoWidth, height - logoHeight),
  ↳ logoIm)
  # Сохранение изменений.
❸ im.save(os.path.join('withLogo', filename))
```

Новый код выводит сообщение, которое извещает пользователя о добавлении логотипа ❶, помещает изображение `logoIm` в позицию с рассчитанными координатами ❷ и сохраняет изменения в файле в каталоге `withLogo` ❸. Если вы запустите программу, когда единственным изображением в рабочем каталоге будет `zophie.png`, то получите следующий вывод.

```
Изменение размеров изображения zophie.png...
Добавление логотипа в изображение zophie.png...
```

Изображение `zophie.png` будет преобразовано в изображение с размерами 225×300 пикселей, представленное на рис. 17.13. Не забывайте о том, что метод `paste()` не вставит прозрачные пиксели в результирующее изображение, если вы не передадите ему дополнительно объект `logoIm` в качестве третьего аргумента. Данная программа способна "логотипизировать" сотни изображений и соответствующим образом изменить их размеры буквально за пару минут.

Идеи относительно создания аналогичных программ

Возможность композиции и изменения размеров изображений в режиме пакетной обработки может быть полезной во многих приложениях. В частности, это позволяет делать следующее:

- добавлять текст или URL-адреса веб-сайтов в изображения;
- добавлять временны*е* метки в изображения;
- копировать или перемещать изображения в различные папки, исходя из размеров файлов;
- добавлять водяные знаки, как правило, прозрачные, в изображения с целью предотвращения копирования последних.

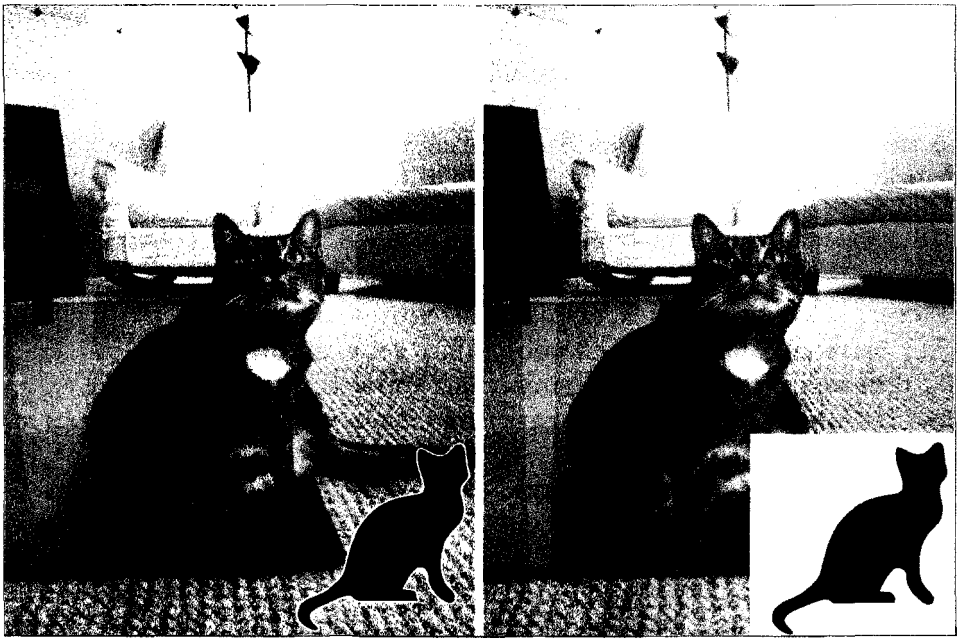


Рис. 17.13. Изображение zophie.png с измененными размерами и добавленным логотипом (слева). Если вы забудете о третьем аргументе, то прозрачные пиксели логотипа скопируются в виде сплошных белых пикселей (справа).

Рисование изображений

В тех случаях, когда возникает необходимость в рисовании отрезков, прямоугольников, окружностей и других простых фигур, используйте модуль `ImageDraw` библиотеки Pillow. Введите в интерактивной оболочке следующий код.

```
>>> from PIL import Image, ImageDraw
>>> im = Image.new('RGBA', (200, 200), 'white')
>>> draw = ImageDraw.Draw(im)
```

Прежде всего мы импортируем модули `Image` и `ImageDraw`. Затем мы создаем новое изображение (в данном случае — квадрат белого цвета с размерами `200×200` пикселей) и сохраняем объект `Image` в переменной `im`. Далее этот объект `Image` передается функции `ImageDraw.Draw()` для получения объекта `ImageDraw`. Этот объект имеет несколько методов, предназначенных для рисования фигур и текста. Сохраните объект `ImageDraw` в переменной `draw`, чтобы его можно было легко использовать в последующих примерах.

Рисование фигур

Описанные ниже методы объекта `ImageDraw` предназначены для рисования различного рода фигур. Параметры `fill` (заливка) и `outline` (обводка) этих методов являются необязательными, и по умолчанию для них устанавливается белый цвет.

Точки

Метод `point(xy, fill)` прорисовывает отдельные пиксели. Аргумент `xy` представляет список точек, которые вы хотите нарисовать. Этим списком может быть список кортежей координат *x* и *y*, такой как `[(x, y), (x, y), ...]`, или же список *x*- и *y*-координат без кортежей, например `[x1, y1, x2, y2, ...]`. Аргумент `fill` определяет цвет точек и может задаваться либо RGBA-кортежем, либо строкой с названием цвета, например `'red'`. Этот аргумент является необязательным.

Отрезки

Метод `line(xy, fill, width)` предназначен для рисования отрезков прямых линий или последовательностей таких отрезков. `xy` — это либо список кортежей, такой как `[(x, y), (x, y), ...]`, либо список целых чисел, например `[x1, y1, x2, y2, ...]`. Каждая точка является одной из соединительных точек рисуемых отрезков. Необязательный аргумент `fill` определяет цвет линий и задается в виде RGBA-кортежа или названия цвета. Необязательный аргумент `width` определяет толщину линий и по умолчанию имеет значение `1`, если не определено иначе.

Прямоугольники

Метод `rectangle(xy, fill, outline)` предназначен для рисования прямоугольников. Аргумент `xy` — это кортеж прямоугольника вида (`left`, `top`, `right`, `bottom`). Значения `left` и `top` определяют *x*- и *y*-координаты верхнего левого угла прямоугольника, тогда как значения `right` и `bottom` — координаты нижнего правого угла. Необязательный аргумент `fill` определяет цвет заливки, а необязательный аргумент `outline` — цвет обводки прямоугольника.

Эллипсы

Метод `ellipse(xy, fill, outline)` предназначен для рисования эллипсов. В случае совпадения ширины и высоты эллипса рисуется окружность. Аргумент `xy` — это кортеж прямоугольника `(left, top, right, bottom)`, который представляет прямоугольник, описанный вокруг эллипса. Необязательный аргумент `fill` определяет цвет заливки, а необязательный аргумент `outline` — цвет обводки эллипса.

Многоугольники

Метод `polygon(xy, fill, outline)` предназначен для рисования многоугольников с любым числом сторон. Аргумент *xy* — это список кортежей, такой как `[(x, y), (x, y), ...]`, или целых чисел, например `[x1, y1, x2, y2, ...]`, представляющий соединительные точки сторон многоугольника. Последняя пара координат будут автоматически соединяться с первой парой. Необязательный аргумент *fill* определяет цвет заливки, а необязательный параметр *outline* — цвет обводки многоугольника.

Пример рисования фигур

Введите в интерактивной оболочке следующий код.

```
>>> from PIL import Image, ImageDraw
>>> im = Image.new('RGBA', (200, 200), 'white')
>>> draw = ImageDraw.Draw(im)
❶ >>> draw.line([(0, 0), (199, 0), (199, 199), (0, 199),
↪ (0, 0)], fill='black')
❷ >>> draw.rectangle((20, 30, 60, 60), fill='blue')
❸ >>> draw.ellipse((120, 30, 160, 60), fill='red')
❹ >>> draw.polygon(((57, 87), (79, 62), (94, 85),
↪ (120, 90), (103, 113)), fill='brown')
❺ >>> for i in range(100, 200, 10):
        draw.line([(i, 0), (200, i - 100)], fill='green')
>>> im.save('drawing.png')
```

Создав объект `Image` для белого квадрата с размерами 200×200 пикселей, передав его методу `ImageDraw.Draw()` для получения объекта `ImageDraw` и сохранив этот объект в переменной `draw`, мы можем вызывать для нее методы, обеспечивающие рисование фигур. В данном случае мы создаем тонкую черную обводку вдоль краев изображения ❶, голубой прямоугольник, координатами верхнего левого и нижнего правого углов которого являются соответственно `(20, 30)` и `(60, 60)` ❷, красный эллипс, определяемый описанным прямоугольником с углами `(120, 30)` и `(160, 60)` ❸, коричневый пятиугольник ❹ и узор, нарисованный зелеными прямолинейными отрезками с помощью цикла `for` ❺. Результирующее изображение, сохраненное в файле *drawing.png*, показано на рис. 17.14.

Существуют и другие методы объекта `ImageDraw`, предназначенные для рисования фигур. Полная документация по ним доступна по адресу `http://pillow.readthedocs.org/en/latest/reference/ImageDraw.html`.

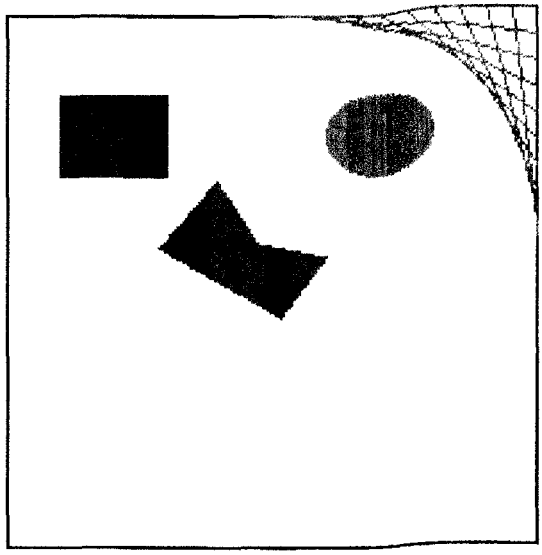


Рис. 17.14. Результирующее изображение drawing.png

Рисование текста

Кроме описанных методов, объект `ImageDraw` имеет также метод `text()`, предназначенный для рисования текста. Этот метод принимает четыре аргумента: *xy*, *text*, *fill* и *font*.

- Аргумент *xy* — это кортеж из двух целых чисел, определяющий координаты верхнего левого угла текстового окна.
- Аргумент *text* — это текстовая строка, которую вы хотите записать.
- Необязательный аргумент *fill* определяет цвет текста.
- Необязательный аргумент *font* — это объект `ImageFont`, используемый для задания шрифта и размера текста. Этот объект описан более подробно в следующем разделе.

Поскольку во многих случаях трудно заранее определить, каким будет размер текстового блока при конкретном значении шрифта, в модуле `ImageDraw` предусмотрен метод `textsize()`. Его первый аргумент — текстовая строка, размер которой вы хотите измерить, а второй — необязательный объект `ImageFont`. Метод `textsize()` возвращает кортеж из двух целых чисел, представляющих ширину и высоту, которые будет иметь блок текста при его прорисовке с заданным размером шрифта. Вы можете использовать эти значения для расчета точных размеров текста, который хотите поместить в изображение.

Первые три аргумента метода `text()` не нуждаются в отдельных пояснениях. Прежде чем использовать метод `text()` для рисования текста на

изображении, рассмотрим его необязательный четвертый аргумент — объект `ImageFont`.

И метод `text()`, и метод `textsize()` принимают необязательный объект `ImageFont` в качестве своего последнего аргумента. Чтобы создать один из этих объектов, необходимо предварительно выполнить следующий код:

```
>>> from PIL import ImageFont
```

Теперь, когда вы импортировали модуль `ImageFont` библиотеки Pillow, можно вызвать функцию `ImageFont.truetype()`, которая принимает два аргумента. Первый аргумент — это строка с именем *файла шрифта TrueType*, существующего на вашем жестком диске. Файлы шрифтов TrueType имеют расширение *.ttf* и обычно располагаются в следующих папках:

- Windows: *C:\Windows\Fonts*;
- OS X: */Library/Fonts and /System/Library/Fonts*;
- Linux: */usr/share/fonts/truetype*.

Вы не должны фактически вводить эти пути в качестве части строки, содержащей имя файла шрифта TrueType, поскольку Python известны эти каталоги, в которых он выполнит автоматический поиск шрифтов. Однако, если Python не удастся найти указанный шрифт, он выведет сообщение об ошибке.

Вторым аргументом функции `ImageFont.truetype()` является целое число, определяющее размер шрифта в *пунктах* (а не, скажем, в пикселях). Имейте в виду, что по умолчанию Pillow создает изображения в формате PNG с разрешением 72 пикселя на дюйм, а пункт — это 1/72 дюйма.

Введите в интерактивной оболочке следующий код, заменив константу `FONT_FOLDER` фактическим именем папки, которая используется вашей операционной системой.

```
   >>> from PIL import Image, ImageDraw, ImageFont
   >>> import os
❶  >>> im = Image.new('RGBA', (200, 200), 'white')
❷  >>> draw = ImageDraw.Draw(im)
❸  >>> draw.text((20, 150), 'Hello', fill='purple')
   >>> fontsFolder = 'FONT_FOLDER' # e.g. '/Library/Fonts'
❹  >>> arialFont = ImageFont.
   ↳ truetype(os.path.join(fontsFolder, 'arial.ttf'), 32)
❺  >>> draw.text((100, 150), 'Привет', fill='gray', font=arialFont)
   >>> im.save('text.png')
```

Импортировав модули `Image`, `ImageDraw`, `ImageFont` и `os`, мы создаем сначала объект `Image` для нового изображения в виде квадрата белого цвета с размерами 200×200 пикселей ❶, а затем объект `ImageDraw` на базе объекта

`Image` ❷. Далее мы используем метод `text()` для прорисовки текста *Hello* пурпурного цвета с началом в позиции с координатами `(20, 150)` ❸. В данном случае мы не передаем методу `text()` необязательный четвертый аргумент, поэтому гарнитура и размер шрифта выбираются по умолчанию.

Чтобы задать гарнитуру и размер шрифта, мы предварительно сохраняем имя папки (наподобие */Library/Fonts*) в переменной `fontsFolder`. Затем мы вызываем функцию `ImageFont.truetype()`, передав ей имя *.ttf*-файла желаемого шрифта и целочисленный аргумент, определяющий размер шрифта ❹. Объект `Font`, возвращенный вызовом `ImageFont.truetype()`, сохраняется в переменной `arialFont`, и эта переменная передается методу `text()` в его последнем именованном аргументе. Вызов метода `text()` ❺ прорисовывает текст `Привет` серого цвета с началом в позиции с координатами `(100, 150)` и с использованием шрифта Arial размером 32 пункта.

Вид результирующего изображения, сохраненного в файле *text.png*, показан на рис. 17.15.

Рис. 17.15. Результирующее изображение text.png

Резюме

Изображения представляют собой коллекции пикселей, каждый из которых имеет RGBA-значение, определяющее его цвет и прозрачность, и адресуется с помощью координат *x* и *y*. Двумя распространенными форматами изображений являются JPEG и PNG. Модуль `pillow` способен обрабатывать изображения как этих форматов, так и многих других.

После загрузки изображения в объект `Image` его ширина и высота сохраняются в виде кортежа из двух целочисленных значений в атрибуте `size`. Объекты типа `Image` имеют также методы, позволяющие различным образом манипулировать изображениями: `crop()`, `copy()`, `paste()`, `resize()`, `rotate()` и `transpose()`. Для сохранения объекта `Image` в файле изображения вызовите метод `save()`.

Если у вас возникает необходимость рисовать фигуры на изображениях, используйте функции модуля `ImageDraw` для рисования точек, прямолинейных отрезков, прямоугольников, эллипсов и многоугольников. Кроме того, этот модуль предоставляет методы для рисования текста с использованием выбранных вами гарнитур и размеров шрифтов.

Несмотря на то что гораздо более мощные (и дорогие) приложения, такие как Photoshop, предоставляют возможности автоматической пакетной обработки изображений, можно использовать сценарии Python для выполнения тех же операций, но бесплатно. В предыдущих главах вы писали программы на Python для обработки простых текстовых файлов, электронных таблиц, документов в формате PDF и других форматах. С модулем `pillow` вы расширяете сферу действия своих программ еще и на обработку изображений!

Контрольные вопросы

1. Что такое RGBA-значение?
2. Как получить из модуля Pillow RGBA-значение для стандартного цвета `'CornflowerBlue'`?
3. Что такое кортеж прямоугольника?
4. С помощью какой функции можно получить объект `Image`, скажем, для файла изображения *zophie.png*?
5. Как определить ширину и высоту изображения, представляемого объектом `Image`?
6. Какой метод вы вызовете, чтобы получить объект `Image` для изображения размером `100×100` пикселей, исключая его нижнюю левую четверть?
7. Как сохранить файл изображения после внесения изменений в представляющий его объект `Image`?
8. Какой модуль содержит код Pillow для рисования фигур?
9. Объекты `Image` не имеют методов, позволяющих выполнять операции рисования. Какие объекты имеют эти методы? Как получить объекты этого рода?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Расширение и доработка программ основного проекта этой главы

Рассмотренная в этой главе программа *resizeAndAddLogo.py* работает с файлами форматов PNG и JPEG, но библиотека Pillow поддерживает намного больше форматов. Расширьте программу *resizeAndAddLogo.py* таким образом, чтобы она могла обрабатывать также изображения в форматах GIF и BMP.

Есть еще одна небольшая проблема, суть которой заключается в том, что программа может работать с файлами PNG и JPEG лишь в том случае, если их расширения указаны в нижнем регистре. Например, она обработает файл *zophie.png*, но не файл *zophie.PNG*. Измените код таким образом, чтобы программа была нечувствительна к регистру расширения имен файлов.

Наконец, предполагается, что логотип, добавляемый в нижний правый угол изображения, должен выглядеть лишь как небольшая метка. Но если размеры изображения и логотипа примерно одинаковы, то композиционное изображение будет выглядеть примерно так, как показано на рис. 17.16. Измените программу *resizeAndAddLogo.py* таким образом, чтобы логотип добавлялся лишь в том случае, если размер изображения по крайней мере в два раза превышает размер логотипа. Если это условие не выполняется, то логотип не должен добавляться.

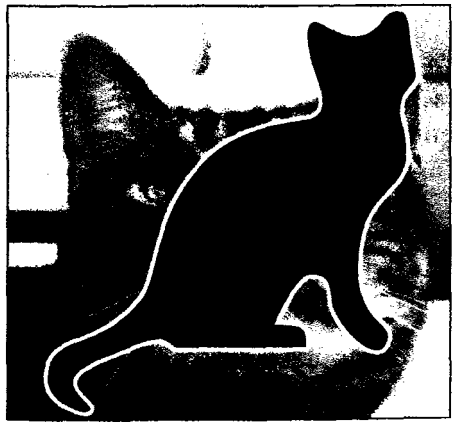


Рис. 17.16. Когда размер основного изображения лишь ненамного больше размера логотипа, вся композиция выглядит уродливо

Обнаружение папок с фотографиями на жестком диске

У меня есть плохая привычка — перемещать файлы из своей цифровой камеры во временные папки на жестком диске, имена которых, конечно же, я впоследствии забываю. Было бы неплохо иметь программу, которая просматривала бы весь жесткий диск и находила эти забытые "фотопапки".

Напишите программу, которая просматривает все папки на вашем жестком диске и находит потенциальные папки, в которых могут находиться фотографии. Разумеется, сначала вы должны определиться с тем, какие папки следует относить к этой категории. Например, это могут быть папки, более половины файлов в которых составляют фотографии. Но как определить, какие файлы являются фотографиями?

Прежде всего, файл с фотографией должен иметь расширение *.png* или *.jpg*. Кроме того, фотографии — это большие изображения; ширина и высота фотоизображения должны превышать 500 пикселей. Это безопасное предположение, поскольку ширина или высота большинства фотографий, получаемых с помощью цифровых камер, составляет несколько тысяч пикселей.

Вот вам небольшая подсказка в виде грубого каркаса, на основе которого может быть построена подобная программа.

```
#! python3
# Импортировать модули и описать данную программу в комментарии.

for foldername, subfolders, filenames in os.walk('C:\\'):
    numPhotoFiles = 0
    numNonPhotoFiles = 0
    for filename in filenames:
        # Обнаруживать файлы, не имеющие расширения .png или .jpg.
        if TODO:
            numNonPhotoFiles += 1
            continue # перейти к следующему файлу

        # Открыть файл изображения, используя модуль Pillow.

        # Обнаруживать файлы с изображениями, ширина или высота
        # которых превышает 500.
        if TODO:
            # Размеры файла изображения слишком велики, чтобы
            # его можно было считать фотографией.
            numPhotoFiles += 1
        else:
            # Изображение слишком мало, чтобы его можно было
            # считать фотографией.
            numNonPhotoFiles += 1

    # Если более половины фотографий оказались фотографиями,
    # вывести абсолютный путь к папке.
    if TODO:
        print(TODO)
```

После запуска программа должна вывести на экран абсолютные пути доступа ко всем папкам, содержащим фотографии.

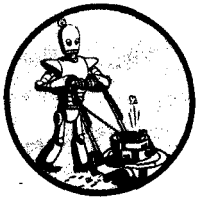
Персональные приглашения

Глава 13 включала учебный проект по созданию персональных приглашений на основе списка гостей, хранящегося в простом текстовом файле. Расширьте проект, добавляя в каждое приглашение фотографию гостя с помощью модуля `pillow`. Для каждого из гостей, включенных в список, который хранится в файле *guests.txt* на сайте `http://nostarch.com/automatestuff/`, сгенерируйте файл изображения, включающего элементы декоративного оформления и имя гостя. Для этой цели можете воспользоваться изображением цветка, которое предоставляется вместе с другими ресурсами на сайте `http://nostarch.com/automatestuff/`.

Чтобы все пригласительные билеты были одного размера, добавьте прямоугольную границу черного цвета, окружающую изображение, которая будет служить ориентиром при разрезании приглашений после их вывода на печать. Библиотека Pillow позволяет создавать PNG-файлы с разрешением 72 пикселя на дюйм, так что для приглашения с размерами 4×5 дюймов потребуется изображение с размерами 288×360 пикселей.

18

УПРАВЛЕНИЕ КЛАВИАТУРОЙ И МЫШЬЮ С ПОМОЩЬЮ СРЕДСТВ АВТОМАТИЗАЦИИ ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ



К этому времени вы успели изучить довольно много модулей Python, которые могут пригодиться для автоматизации самых разных задач, таких как редактирование электронных таблиц, загрузка файлов или запуск программ по расписанию. Но в некоторых ситуациях не удается подыскать модуль, соответствующий запросам приложения. В этом случае единственным выходом, позволяющим автоматизировать выполнение приложения, может оказаться написание программы, обеспечивающей непосредственное управление клавиатурой и мышью. Такие программы способны управлять приложениями, отправляя им виртуальные нажатия клавиш или щелчки мышью, как если бы вы сами взаимодействовали с приложением. Для этой техники существует специальное название — *автоматизация графического пользовательского интерфейса* (сокращенно — *GUI-автоматизация*). В этом случае программы выполняются так, будто их работой управляет человек, сидящий за компьютером, с тем дополнительным преимуществом, что попадание на клавиатуру случайно пролитого кофе полностью исключается.

Средства автоматизации пользовательского интерфейса можно считать своеобразным программным роботом. Вы можете запрограммировать руки робота так, чтобы он нажимал клавиши и перемещал мышь вместо вас. Эта техника особенно полезна для решения тех задач, в которых необходимо выполнять множество щелчков на элементах управления или заполнять формы.

Модуль `pyautogui` предлагает функции, позволяющие имитировать перемещения мыши, щелчки ее кнопками и вращение колесика. В этой главе

рассматривается лишь некоторое подмножество средств автоматизации PyAutoGUI; полную документацию по этому вопросу вы найдете на сайте `http://pyautogui.readthedocs.org/`.

Установка модуля pyautogui

Модуль `pyautogui` способен посылать сигналы виртуальных нажатий клавиш и щелчков мышью операционным системам Windows, OS X и Linux. В зависимости от типа используемой вами операционной системы, вам может понадобиться установка некоторых других модулей (называемых *зависимостями*) перед установкой библиотеки PyAutoGUI.

- В Windows установка дополнительных модулей не нужна.
- В OS X выполните последовательно команды `pip3 install pyobjc-framework-Quartz`, `sudo pip3 install pyobjc-core` и `sudo pip3 install pyobjc`.
- В Linux выполните последовательно команды `sudo pip3 install python3-xlib`, `sudo apt-get install scrot`, `sudo apt-get install python3-tk` и `sudo apt-get install python3-dev`. (Scrot — это программа для захвата снимков экрана, которую использует библиотека PyAutoGUI.)

Установив необходимые зависимости, выполните команду `pip install pyautogui` (или `pip3` в случае OS X и Linux), чтобы установить библиотеку PyAutoGUI.

Полная информация, касающаяся установки модулей сторонних разработчиков, содержится в приложении A. Вы можете протестировать корректность установки PyAutoGUI, проверив, не сопровождается ли выполнение команды `import pyautogui` в интерактивной оболочке выводом сообщений об ошибках.

Сохранение контроля над клавиатурой и мышью

Прежде чем приступить к непосредственному использованию средств GUI-автоматизации, необходимо узнать о том, как избежать проблем, которые при этом могут возникать. Python может перемещать указатель мыши и выполнять виртуальные нажатия клавиш с невероятной скоростью. В действительности эта скорость может оказаться несоизмеримо большой для других программ. Кроме того, если что-то пойдет не так, но ваша программа будет по-прежнему перемещать мышь, то у вас могут возникнуть затруднения с точным определением причин неполадок и поиском способов их устранения. Подобно волшебной метле из мультипликационного фильма Диснея "Ученик чародея", которая не прекращала доливать воду в бак Микки Мауса и после того, как она рекой полилась через края, ваша программа

может выйти из-под контроля, даже если она будет идеально выполнять ваши инструкции. Если указатель мыши начнет безостановочно бегать по всему экрану, не давая вам возможности щелкнуть на кнопке закрытия окна IDLE, то остановить работу программы будет чрезвычайно трудно. К счастью, существуют способы, позволяющие избегать или преодолевать последствия проблем, связанных с GUI-автоматизацией.

Прекращение выполнения всех задач путем выхода из учетной записи

Возможно, самый простой способ остановки вышедшей из-под контроля программы GUI-автоматизации заключается в выходе из учетной записи пользователя, что приведет к прекращению выполнения всех запущенных в ней программ. В системах Windows и Linux для этого следует нажать комбинацию из трех клавиш <Ctrl+Alt+Del>. В OS X это комбинация клавиш <⌘+Shift+Option+Q>. Выйдя из учетной записи, вы потеряете несохраненные результаты работы, но зато вам не придется перезагружать всю систему.

Паузы и безопасный резервный выход

У вас есть возможность предусмотреть в своем сценарии паузы после каждого вызова функции автоматизации, которые вы можете использовать для того, чтобы перехватить управление мышью и клавиатурой, если что-то пойдет не так. Для этого следует установить в переменной `pyautogui.PAUSE` длительность паузы в секундах. Например, после выполнения инструкции `pyautogui.PAUSE = 1.5` каждый вызов любой из функций PyAutoGUI будет завершаться полуторасекундной паузой. Все остальные вызовы такой паузой сопровождаться не будут.

Кроме того, для средств PyAutoGUI предусмотрена возможность безопасного резервного выхода из программы. Перемещение указателя мыши в верхний левый угол экрана приведет к возбуждению исключения `pyautogui.FailSafeException`. Ваша программа может либо обработать это исключение с помощью инструкций `try` и `except`, либо позволить исключению аварийно завершить выполнение программы. В любом случае максимально быстрое перемещение указателя мыши в верхний левый угол экрана завершит работу программы. Это средство можно отключить инструкцией `pyautogui.FAILSAFE = False`. Введите в интерактивной оболочке следующие команды.

```
>>> import pyautogui
>>> pyautogui.PAUSE = 1
>>> pyautogui.FAILSAFE = True
```

Здесь мы импортируем модуль `pyautogui` и с помощью инструкции `pyautogui.PAUSE = 1` устанавливаем паузу длительностью 1 секунда после каждого вызова функции. Мы устанавливаем для переменной `pyautogui.FAILSAFE` значение `True`, чтобы активизировать средство безопасного резервного выхода из программы.

Управление перемещениями указателя мыши

В этом разделе вы узнаете о том, как перемещать указатель мыши и отслеживать его позицию на экране с помощью модуля PyAutoGUI, но сначала вам надо понять, как PyAutoGUI работает с координатами.

Функции PyAutoGUI для работы с мышью используют координаты *x* и *y*. Система координат для компьютерного экрана показана на рис. 18.1; она аналогична системе координат, используемой при работе с изображениями, которая обсуждалась в главе 17. Начало координат, которому соответствуют нулевые значения обеих координат, находится в верхнем левом углу экрана. Координата *x* увеличивается в направлении слева направо, координата *y* — в направлении сверху вниз. Все координаты — положительные целые числа; координаты не могут быть отрицательными.

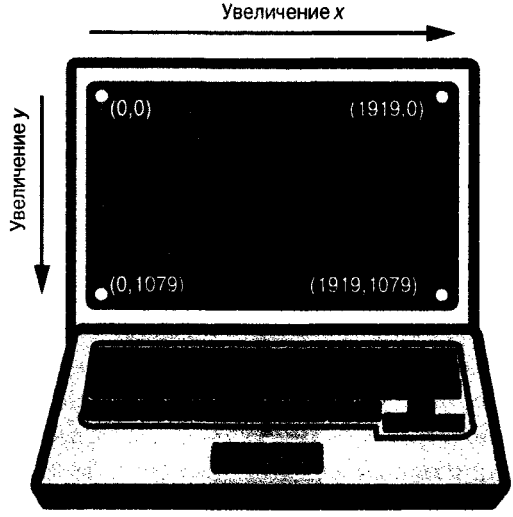


Рис. 18.1. Система координат на компьютерном экране разрешением 1920×1080

Разрешение экрана указывает на его ширину и высоту в пикселях. Если разрешение экрана установлено равным 1920×1080, то координаты его верхнего левого угла — `(0, 0)`, а нижнего правого — `(1919, 1079)`.

Функция `pyautogui.size()` возвращает кортеж из двух целых чисел, представляющих ширину и высоту экрана в пикселях. Введите в интерактивной оболочке следующие команды.

```
>>> import pyautogui
>>> pyautogui.size()
(1920, 1080)
>>> width, height = pyautogui.size()
```

Функция `pyautogui.size()` возвращает кортеж `(1920, 1080)` на компьютере с разрешением экрана 1920×1080; в зависимости от разрешения, установленного для вашего экрана, вы можете получить другие значения. Эти значения можно сохранить в переменных `width` и `height` для улучшения удобочитаемости программы.

Перемещение указателя мыши

Теперь, когда вы хорошо понимаете, что собой представляют экранные координаты, мы можем заняться перемещением указателя мыши. Функция `pyautogui.moveTo()` немедленно перемещает указатель в указанную позицию на экране. В качестве первого и второго аргументов этой функции задаются координаты *x* и *y* соответственно. Необязательный именованный аргумент `duration` в виде целого или вещественного числа задает (в секундах) длительность перемещения указателя в конечную точку. Если его опустить, то он принимает значение `0`, соответствующее мгновенному перемещению. (Все необязательные именованные аргументы `duration` функций PyAutoGUI являются необязательными.) Введите в интерактивной оболочке следующие команды.

```
>>> import pyautogui
>>> for i in range(10):
      pyautogui.moveTo(100, 100, duration=0.25)
      pyautogui.moveTo(200, 100, duration=0.25)
      pyautogui.moveTo(200, 200, duration=0.25)
      pyautogui.moveTo(100, 200, duration=0.25)
```

В этом примере указатель мыши обходит все стороны квадрата в направлении по часовой стрелке десять раз. Каждое перемещение по стороне квадрата осуществляется за четверть секунды, как это задано именованным аргументом `duration=0.25`. Если опустить третий параметр в любом из вызовов функции `pyautogui.moveTo()`, то указатель мыши будет мгновенно телепортироваться из одной точки в другую.

Функция `pyautogui.moveRel()` перемещает указатель мыши относительно его текущей позиции. В следующем примере указатель также перемещается по сторонам квадрата, за исключением того, что роль начальной точки квадрата играет та точка экрана, в которой указатель находится в момент запуска кода.

```
>>> import pyautogui
>>> for i in range(10):
      pyautogui.moveRel(100, 0, duration=0.25)
      pyautogui.moveRel(0, 100, duration=0.25)
      pyautogui.moveRel(-100, 0, duration=0.25)
      pyautogui.moveRel(0, -100, duration=0.25)
```

Функция `pyautogui.moveRel()`, как и предыдущая функция, принимает три аргумента: величина перемещения в пикселях вправо по горизонтали и вниз по вертикали, а также (необязательный аргумент) длительность перемещения (в секундах). Отрицательное значение первого или второго аргумента означает перемещение указателя влево или вверх соответственно.

Получение позиции указателя мыши

Чтобы определить текущую позицию указателя мыши, вызовите функцию `pyautogui.position()`, которая возвращает кортеж из двух координат *x* и *y* указателя мыши на момент вызова функции. Введите в интерактивной оболочке следующие команды, перемещая указатель мыши после каждого вызова.

```
>>> pyautogui.position()
(311, 622)
>>> pyautogui.position()
(377, 481)
>>> pyautogui.position()
(1536, 637)
```

Разумеется, возвращаемые функцией значения будут зависеть от того, где в данный момент находится указатель мыши.

Проект "Где сейчас находится указатель мыши?"

Возможность определять позицию указателя мыши является важной частью настройки ваших сценариев GUI-автоматизации. Однако, просто глядя на экран, определить точные координаты пикселя практически невозможно. Было бы удобно иметь программу, которая постоянно отображала бы координаты указателя мыши, когда вы перемещаете его по экрану.

Вот что должна делать такая программа при высокоуровневом рассмотрении:

- отображать текущие значения координат *x* и *y* указателя мыши;
- обновлять эти координаты при перемещении указателя мыши по экрану.

Это означает, что ваш код должен выполнять следующие действия:

- вызывать функцию `position()` для извлечения текущих значений координат;
- затирать ранее выведенные значения координат путем вывода символов забоя (`\b`) на экран;
- обрабатывать исключение `KeyboardInterrupt`, чтобы пользователь мог выходить из программы посредством нажатия клавиш <Ctrl+C>.

Откройте новое окно в файловом редакторе и сохраните его в файле *mouseNow.py*.

Шаг 1. Импортирование модуля

Начните программу вводом следующих инструкций.

```
#! python3
# mouseNow.py - Отображает текущую позицию указателя мыши.
import pyautogui
print('Для выхода нажмите клавиши <Ctrl+C>.')
#TODO: Получить и вывести координаты указателя мыши.
```

Этот код импортирует модуль `pyautogui` и выводит для пользователей напоминание о том, что для выхода из программы следует нажать комбинацию клавиш <Ctrl+C>.

Шаг 2. Код выхода из программы и бесконечный цикл

Вы можете обеспечить постоянный вывод текущих координат указателя мыши, возвращаемых функцией `mouse.position()`, организовав бесконечный цикл. Что касается кода для выхода из программы, вам надо будет перехватывать исключение `KeyboardInterrupt`, которое генерируется, когда пользователь нажимает комбинацию клавиш <Ctrl+C>. Если вы не обработаете это исключение, то на экран будет выведен пугающий стек обратной трассировки вызовов и сообщение об ошибке для пользователя. Добавьте в свою программу следующий код, выделенный ниже полужирным шрифтом.

```
#! python3
# mouseNow.py - Отображает текущую позицию указателя мыши.
import pyautogui
print('Для выхода нажмите клавиши <Ctrl+C>.')
try:
    while True:
        # TODO: Получить и вывести координаты указателя мыши.
❶ except KeyboardInterrupt:
❷     print('\nГотово.')
```

Чтобы обработать исключение, поместите бесконечный цикл в тело инструкции `try`. Когда пользователь нажмет комбинацию клавиш <Ctrl+C>, выполнение программы перейдет к инструкции `except` ❶, и в новой строке будет выведено сообщение `Done` ❷.

Шаг 3. Получение и вывод координат указателя мыши

Код в теле цикла `while` должен получать текущие координаты указателя мыши, форматировать их для улучшения удобочитаемости и выводить на экран. Добавьте следующий код в тело цикла `while`.

```
#! python3
# mouseNow.py - Отображает текущую позицию указателя мыши.
import pyautogui
print('Для выхода нажмите клавиши <Ctrl+C>.')
--пропущенный код--
        # Получить и вывести координаты указателя мыши.
        x, y = pyautogui.position()
        positionStr = 'X: ' + str(x).rjust(4) +
                      ' Y: ' + str(y).rjust(4)
--пропущенный код--
```

Используя трюк с групповым присваиванием, мы можем присвоить переменным `x` и `y` значения двух целых чисел, возвращаемых в составе кортежа функцией `pyautogui.position()`. Передав переменные `x` и `y` функции `str()`, можно получить целочисленные координаты в виде строк. Строковый метод `rjust()` выравнивает эти строки вправо, так что они будут занимать одинаковое пространство, независимо от того, из скольких цифр они состоят — из одной, двух, трех или четырех. Конкатенация выровненных вправо строк с подписями `'X: '` и `' Y: '` дает аккуратно отформатированную строку, которая будет сохраняться в переменной `positionStr`.

Добавьте в конце программы следующий код.

```
#! python3
    # mouseNow.py - Отображает текущую позицию указателя мыши.
    --пропущенный код--
        print(positionStr, end='')
❶       print('\b' * len(positionStr), end='', flush=True)
```

Этот код осуществляет фактический вывод значения переменной `positionStr` на экран. Именованный аргумент `end=''` функции `print()` предотвращает добавление заданного по умолчанию символа новой строки в конец строки, выводимой на экран. Существует возможность затереть текст, который вы уже вывели на экран, но это относится только к последней строке текста. Как только вы выведете символ новой строки, вы уже не сможете удалить ранее выведенный текст.

Чтобы удалить текст, выведите управляющий символ забоя (\b). Этот специальный символ удаляет символ, занимающий последнюю позицию в текущей строке на экране. В строке кода ❶ используется репликация строк для того, чтобы получить такое количество следующих один за другим символов \b, которое совпадало бы с длиной строки, сохраненной в переменной positionStr, что внешне проявляется как эффект затирания последней выведенной строки на экране.

В силу технических причин, рассмотрение которых выходит за рамки данной книги, функции print(), выводящей символы \b, следует всегда передавать именованный аргумент flush=True. В противном случае текст на экране может не обновляться так, как вам хотелось бы.

Поскольку цикл while выполняется чрезвычайно быстро, пользователь фактически даже не заметит, что вы удаляете и заново выводите на экран числа целиком. Например, если *x*-координата равна 563, а указатель мыши сместится на один пиксель вправо, то все будет выглядеть так, будто только цифра 3 в числе 563 была заменена цифрой 4.

Запустив программу, вы увидите на экране только две выведенные строки, которые будут выглядеть примерно так.

```
Для выхода нажмите клавиши <Ctrl+C>.
X: 290 Y: 424
```

Первая строка выводит пояснительную инструкцию относительно нажатия комбинации клавиш <Ctrl+C> для выхода из программы. Вторая строка с координатами указателя мыши будет изменяться по мере того, как вы будете перемещать мышь. Используя эту программу, вы сможете определять координаты указателя мыши, чтобы использовать их в своих сценариях GUI-автоматизации.

Управление взаимодействием с мышью

Теперь, когда вы уже знаете, как перемещать указатель мыши и определять его местоположение на экране, приступим к выполнению таких виртуальных операций, как щелчки, перетаскивание и прокрутка.

Щелчки мышью

Чтобы отправить компьютеру виртуальный щелчок мышью, вызовите метод pyautogui.click(). По умолчанию предполагается, что этот щелчок выполняется левой кнопкой в месте текущего расположения указателя мыши. Если требуется выполнить щелчок в другом месте, передайте координаты *x* и *y* соответствующей точки в качестве необязательных первого и второго аргументов.

Если вы хотите указать кнопку, которой должен быть выполнен щелчок, то включите в вызов именованный аргумент `button` с одним из следующих значений: `'left'` (левая), `'middle'` (средняя) или `'right'` (правая). Например, вызову `pyautogui.click(100, 150, button='left')` соответствует щелчок левой кнопкой в точке экрана с координатами `(100, 150)`, тогда как вызову `pyautogui.click(200, 250, button='right')` — щелчок правой кнопкой в точке экрана с координатами `(200, 250)`.

Введите в интерактивной оболочке следующий код.

```
>>> import pyautogui
>>> pyautogui.click(10, 5)
```

Вы должны увидеть, как указатель мыши переместится в область экрана вблизи его верхнего левого угла и выполнит однократный щелчок. Полный "щелчок" определяется как нажатие кнопки мыши и последующее ее отпускание без перемещения курсора. Также возможно выполнение щелчков с помощью вызова `pyautogui.mouseDown()`, которому соответствует лишь нажатие кнопки, и вызова `pyautogui.mouseUp()`, которому соответствует лишь отпускание кнопки. Эти функции принимают те ж аргументы, что и функция `click()`, и в действительности функция `click()` просто используется в качестве оболочки для выполнения вызовов этих двух функций.

Дополнительные возможности предоставляют функция `pyautogui.doubleClick()`, выполняющая двойной щелчок левой кнопкой мыши, а также функции `pyautogui.rightClick()` и `pyautogui.middleClick()`, которые выполняют щелчок соответственно правой и средней копками.

Перетаскивание указателя мыши

Термин *перетаскивание* означает перемещение указателя мыши при одновременном удерживании одной из ее кнопок. Например, можно перемещать файлы между папками, перетаскивая их значки, или перемещать назначенные встречи в приложении календаря.

Библиотека PyAutoGUI предоставляет функции `pyautogui.dragTo()` и `pyautogui.dragRel()`, позволяющие перетаскивать указатель мыши в новое местоположение или местоположение, заданное относительно текущего. Функции `dragTo()` и `dragRel()` принимают те же аргументы, что и функции `moveTo()` и `moveRel()`: *x*-координату/горизонтальное смещение, *y*-координату/вертикальное смещение и необязательный аргумент, определяющий длительность перемещения. (В OS X передача этого аргумента является желательной, поскольку при слишком быстром перемещении указателя мыши операция перетаскивания может выполняться некорректно.)

Чтобы испытать, как работают эти функции, откройте какое-либо приложение для работы с графикой, например Paint в Windows, Paintbrush в OS X или GNU Paint в Linux. (Если подходящее приложение не установлено на вашем компьютере, можете воспользоваться в онлайновом режиме тем, которое предлагается на сайте `http://sumopaint.com/`.) Для выполнения операций рисования в этих приложениях я буду использовать библиотеку PyAutoGUI.

При условии, что указатель мыши находится на холсте графического приложения и в качестве текущего инструмента выбран **Pencil** (Карандаш) или **Brush** (Кисть), введите в новом окне файлового редактора следующий код и сохраните его в файле *spiralDraw.py*.

```
import pyautogui, time
❶ time.sleep(5)
❷ pyautogui.click() # щелчок для перевода фокуса в
                    # программу рисования
distance = 200
while distance > 0:
❸     pyautogui.dragRel(distance, 0, duration=0.2) # сдвиг вправо
❹     distance = distance - 5
❺     pyautogui.dragRel(0, distance, duration=0.2) # сдвиг вниз
❻     pyautogui.dragRel(-distance, 0, duration=0.2) # сдвиг влево
      distance = distance - 5
      pyautogui.dragRel(0, -distance, duration=0.2) # сдвиг вверх
```

Когда вы запустите эту программу, вам будет предоставлена пятисекундная пауза ❶, чтобы вы успели переместить указатель мыши в окно графической программы, в которой к этому времени должен быть выбран в качестве инструмента карандаш или кисть. После этого сценарий *spiralDraw.py* перехватит управление мышью и выполнит щелчок для получения фокуса в программе рисования ❷. Фокус получен окном, если в нем виден активный мерцающий курсор и предпринимаемые вами действия, в данном случае перетаскивание указателя мыши, воздействуют на него. Как только графическая программа получит фокус, сценарий *spiralDraw.py* нарисует квадратную спираль наподобие той, которая представлена на рис. 18.2.

Начальным значением переменной `distance` является `200`, поэтому на первой итерации цикла `while` первый вызов `dragRel()` перемещает курсор на `200` пикселей вправо за `0,2` секунды ❸. Затем значение переменной `distance` уменьшается до `195` ❹, и второй вызов `dragRel()` перемещает курсор на `195` пикселей вниз ❺. Третий вызов `dragRel()` перемещает курсор на `-195` по горизонтали (`195` пикселей влево) ❻, значение переменной `distance` уменьшается до `190`, а последний вызов `dragRel()` перемещает курсор на `190` пикселей вверх. На каждой итерации мышь перемещается вправо, вниз, влево и вверх, а переменная `distance` принимает несколько меньшее

значение, чем на предыдущей итерации. Выполняя этот код в цикле, вы перемещаете курсор таким образом, что он рисует квадратную спираль.

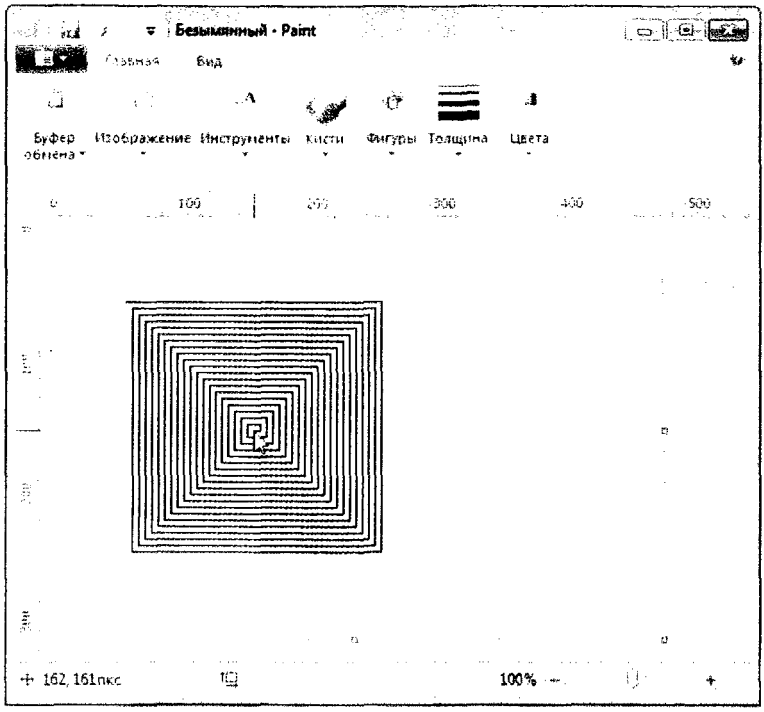


Рис. 18.2. Результат работы примера программы, в которой используется функция `pyautogui.dragRel()`

Вы могла бы нарисовать эту спираль и вручную (вернее, с помощью мыши), но для точного рисования спирали вам пришлось бы работать очень медленно. Модуль PyAutoGUI способен выполнить эту работу за считанные секунды!

Примечание

Вы могли бы нарисовать это изображение, используя функции модуля `pillow` *(см. главу 17). Однако средства GUI-автоматизации позволяют использовать более сложные инструменты рисования, предоставляемые графическими программами, такие как градиенты, различные кисти или инструмент заливки.*

Прокрутка

Последней из функций библиотеки PyAutoGUI для работы с мышью мы рассмотрим функцию `scroll()`, которая принимает целочисленный аргумент, определяющий количество единиц прокрутки в направлении вверх

или вниз. Величина единицы прокрутки изменяется в зависимости от операционной системы и приложения, поэтому в каждой конкретной ситуации придется провести самостоятельные эксперименты, чтобы выяснить ее конкретную величину. Прокрутка выполняется в текущей позиции курсора. Положительное значение аргумента означает прокрутку вверх, отрицательное — прокрутку вниз. Выполните следующую команду в интерактивной оболочке, предварительно расположив указатель мыши в окне IDLE:

```
>>> pyautogui.scroll(200)
```

Вы увидите, как окно IDLE быстро прокрутится вверх, а затем вернется в исходную позицию вниз. Прокрутка вниз происходит по той причине, что IDLE делает это автоматически после выполнения инструкции. Введите теперь другой код.

```
>>> import pyperclip
>>> numbers = ''
>>> for i in range(200):
      numbers = numbers + str(i) + '\n'

>>> pyperclip.copy(numbers)
```

В этом коде прежде всего импортируется модуль `pyperclip` и определяется пустая строка `numbers`. Далее код циклически перебирает целые числа от `0` до `199` и добавляет каждое из них в строку `numbers` вместе с символом новой строки. После выполнения инструкции `pyperclip.copy(numbers)` в буфер обмена будут загружены 200 строк чисел. Откройте новое окно в файловом редакторе и вставьте в него содержимое буфера. В результате вы получите окно с длинным текстом, удобным для наблюдения за прокруткой. Введите в интерактивной оболочке следующий код.

```
>>> import time, pyautogui
>>> time.sleep(5); pyautogui.scroll(100)
```

Во второй строке вы вводите две команды, разделенные точкой с запятой, которые воспринимаются Python так, как если бы они находились на разных строках. Единственное отличие состоит в том, что в этом случае Python не выводит приглашение для ввода команды между двумя инструкциями. В данном примере это важно, поскольку мы хотим, чтобы вызов функции `pyautogui.scroll()` был автоматически выполнен по истечении паузы. (Заметьте, что, несмотря на всю полезность помещения двух команд в интерактивной оболочке в одной строке, в программах по-прежнему необходимо вводить инструкции в разных строках.)

После нажатия клавиши <Enter> у вас будет пять секунд на то, чтобы щелкнуть мышью в окне файлового редактора для перевода в него фокуса. Сразу же по истечении пятисекундной задержки вызов `pyautogui.scroll()` выполнит прокрутку окна файлового редактора в направлении вверх.

Работа с экраном

Ваши программа GUI-автоматизации не должны вслепую выполнять щелчки и нажимать виртуальные клавиш. Библиотека PyAutoGUI предлагает средства, обеспечивающие получение снимков экрана и создание файлов изображений на основе его текущего содержимого. Эти функции также могут возвращать объекты `Image` Pillow, соответствующие текущему виду экрана. Если вы не читали эту книгу глава за главой, то вам следует обратиться к главе 17 и установить модуль `pillow`, прежде чем продолжить чтение этого раздела.

Чтобы функции PyAutoGUI, предназначенные для получения снимков экрана, можно было использовать на компьютерах с Linux, на них должна быть установлена программа `scrot`. Для установки этой программы следует выполнить в окне Terminal команду `sudo-apt-get install scrot`. Если вы используете компьютеры, работающие под управлением операционной системы Windows или OS X, то опустите этот шаг и продолжите чтение данного раздела.

Получение снимка экрана

Для получения снимка экрана в Python вызовите функцию `pyautogui.screenshot()`. Введите в интерактивной оболочке следующие команды.

```
>>> import pyautogui
>>> im = pyautogui.screenshot()
```

Переменная `im` будет содержать объект `Image` снимка экрана. Теперь вы сможете вызывать методы для объекта `Image`, хранящегося в переменной `im`, как для любого другого объекта `Image`. Введите в интерактивной оболочке следующие команды.

```
>>> im.getpixel((0, 0))
(176, 176, 175)
>>> im.getpixel((50, 200))
(130, 135, 144)
```

Передайте функции `getpixel()` кортеж координат, например `(0, 0)` или `(50, 200)`, и она вернет вам цвет пикселя изображения в точке с этими

координатами. Возвращаемое значение функции `getpixel()` — это RGB-кортеж из трех целых чисел, представляющих соответственно количество красной, зеленой и синей составляющих в цвете пикселя. (Четвертая составляющая — альфа-канал — отсутствует, поскольку снимки экрана полностью непрозрачны.) Именно так ваша программа может "видеть", что находится в любой точке экрана в данный момент.

Анализ снимка экрана

Предположим, что одним из действий, выполняемых вашей программой GUI-автоматизации, является щелчок на серой кнопке. Прежде чем вызвать метод `click()`, можно получить снимок экрана и проверить цвет пикселя, в котором программа собирается выполнить щелчок. Если цвет пикселя не совпадает с серым цветом кнопки, значит, что-то пошло не так. Возможно, окно было неожиданно перемещено или же всплывающее диалоговое окно перекрыло кнопку. В этом случае, вместо того чтобы продолжить выполнение, что имело бы непредсказуемые последствия из-за щелчка в неподходящем месте, программа может определить, что выполнять щелчок не следует, и предпринять другие соответствующие действия.

Функция `pixelMatchesColor()` библиотеки PyAutoGUI возвращает значение `True`, если цвет пикселя с заданными экранными координатами *x* и *y* совпадает с заданным цветом. Ее первый и второй аргументы — это целые числа, представляющие координаты *x* и *y*, а третий — это кортеж из трех целых чисел, представляющих RGB-цвет, которому должен соответствовать экранный пиксель. Введите в интерактивной оболочке следующие команды.

```
>>> import pyautogui
>>> im = pyautogui.screenshot()
❶ >>> im.getpixel((50, 200))
(130, 135, 144)
❷ >>> pyautogui.pixelMatchesColor(50, 200, (130, 135, 144))
True
❸ >>> pyautogui.pixelMatchesColor(50, 200, (255, 135, 144))
False
```

После того как вы получите снимок экрана и используете функцию `getpixel()` для получения RGB-кортежа, соответствующего цвету пикселя с заданными координатами ❶, передайте эти координаты и RGB-кортеж функции `pixelMatchesColor()` ❷, которая возвратит значение `True`. Затем измените значение RGB-кортежа и вновь вызовите функцию `pixelMatchesColor()` для тех же координат ❸. На этот раз функция должна возвратить значение `False`. Эту функцию полезно вызывать всякий раз перед тем, как ваша программа GUI-автоматизации должна вызвать функцию `click()`. Обратите внимание на то, что совпадение цвета пикселя с

заданным цветом в данном случае должно быть полным. Даже если отличие весьма незначительно, например пиксель имеет цвет (255, 255, 254) вместо (255, 255, 255), функция `pixelMatchesColor()` все равно вернет значение `False`.

Проект: расширение программы mouseNow.py

Рассмотренную ранее программу `mouseNow.py` можно расширить так, чтобы она выдавала не только координаты *x* и *y* текущей позиции указателя мыши, но и RGB-цвет находящегося под ним пикселя. Модифицируйте программу `mouseNow.py`, внеся в код, находящийся в теле цикла `while`, следующие изменения.

```
#! python3
# mouseNow.py - Отображает текущую позицию указателя мыши.
--пропущенный код--
        positionStr = 'X: ' + str(x).rjust(4) +
                      ' Y: ' + str(y).rjust(4)
        pixelColor = pyautogui.screenshot().getpixel((x, y))
        positionStr += ' RGB: (' + str(pixelColor[0]).rjust(3)
        positionStr += ', ' + str(pixelColor[1]).rjust(3)
        positionStr += ', ' + str(pixelColor[2]).rjust(3) + ')'
        print(positionStr, end='')
--пропущенный код--
```

Теперь, если вы запустите программу *mouseNow.py*, вывод будет дополнительно включать информацию об RGB-цвете пикселя, находящегося под указателем мыши.

```
Для выхода нажмите клавиши <Ctrl+C>.
X:  406 Y:   17 RGB: (161,  50,  50)
```

Использование этой информации совместно с функцией `pixelMatchesColor()` упростит добавление проверки цвета пикселей в ваши программы GUI-автоматизации.

Распознавание образов

Но что делать, если вам неизвестно заранее, в каком месте экрана программа должна выполнить виртуальный щелчок? В этом случае можно воспользоваться распознаванием образов. Передайте модулю PyAutoGUI изображение, на котором следует выполнить щелчок, и позвольте программе самостоятельно определить нужные координаты.

Например, если ранее вы получили снимок экрана для захвата изображения кнопки Отправить, которое сохранили в файле *submit.png*, то функция

`locateOnScreen()` возвратит координаты местонахождения этого изображения на экране. Чтобы увидеть, как работает функция `locateOnScreen()`, попробуйте получить снимок небольшой области экрана, сохраните это изображение, а затем введите в интерактивной оболочке следующие команды, заменив имя файла `'submit.png'` реальным именем файла полученного вами изображения.

```
>>> import pyautogui
>>> pyautogui.locateOnScreen('submit.png')
(643, 745, 70, 29)
```

Кортеж из четырех целых чисел, возвращаемый функцией `locateOnScreen()`, представляет *x*-координату левого края, *y*-координату верхнего края, а также ширину и высоту изображения в первой из позиций на экране, в которой оно обнаружено. Если вы попытаетесь выполнить этот код на своем компьютере с собственным снимком экрана, то ваши числовые данные будут отличаться от тех, которые показаны здесь.

Если функции `locateOnScreen()` не удается обнаружить указанное изображение на экране, то она возвращает значение `None`. Обратите внимание на то, что для того, чтобы изображение на экране могло быть распознано, оно должно в точности совпадать с предоставленным изображением. Если оно отличается хотя бы одним пикселем, то функция `locateOnScreen()` возвратит значение `None`.

Если функция `locateAllOnScreen()` находит изображение в нескольких местах экрана, она возвращает объект `Generator`, который можно передать функции `list()` для возврата списка кортежей, включающих по четыре целых числа. Всего будет по одному такому кортежу для каждого расположения на экране, в котором было найдено заданное изображение. Продолжите выполнение примера в интерактивной оболочке, введя следующие команды (с заменой строки `'submit.png'` именем файла с подготовленным вами изображением).

```
>>> list(pyautogui.locateAllOnScreen('submit.png'))
[(643, 745, 70, 29), (1007, 801, 70, 29)]
```

Каждый из кортежей, включающих четыре целых числа, представляет одну область экрана. Если ваше изображение обнаружено лишь в одной области экрана, то в результате использования функций `list()` и `locateAllOnScreen()` вы получите список, содержащий один кортеж.

Зная область экрана, в которой обнаружено искомое изображение, можно выполнить щелчок в центре этой области, передав соответствующий кортеж функции `center()`, которая возвратит координаты *x* и *y* центра данной области. Введите в интерактивной оболочке следующие команды,

заменяя аргументы именем собственного файла изображения, а также соответствующими кортежем и парой координат.

```
>>> pyautogui.locateOnScreen('submit.png')
(643, 745, 70, 29)
>>> pyautogui.center((643, 745, 70, 29))
(678, 759)
>>> pyautogui.click((678, 759))
```

Передав полученные из функции `center()` координаты функции `click()`, вы выполните виртуальный щелчок мышью по центру области экрана, совпадающей с изображением, которое вы передали функции `locateOnScreen()`.

Управление клавиатурой

Библиотека PyAutoGUI также содержит функции, позволяющие посылать компьютеру виртуальные клавиатурные нажатия, что дает вам возможность заполнять формы или вводить текст в приложениях.

Отправка строки, набранной на виртуальной клавиатуре

Функция `pyautogui.typewrite()` посылает компьютеру виртуальные клавиатурные нажатия. Какие последствия это будет иметь, зависит от того, в каком окне и каком поле находится фокус ввода. Чтобы гарантировать нахождение фокуса в нужном месте, можно предварительно выполнить виртуальный щелчок мышью в соответствующем поле.

В качестве простого примера используем Python для автоматического ввода текста `Hello world!` в окне файлового редактора. Прежде всего, откройте в файловом редакторе новое окно и расположите его в верхнем левом углу экрана, чтобы в него можно было переместить фокус ввода, выполнив с помощью средств автоматизации PyAutoGUI виртуальный щелчок мышью в подходящем месте. Затем введите в интерактивной оболочке следующие команды:

```
>>> pyautogui.click(100, 100); pyautogui.typewrite('Hello world!')
```

Обратите внимание на размещение двух команд, разделенных точкой с запятой, в одной строке, что предотвращает отображение приглашения ко вводу между выполнением двух инструкций. Это позволяет избежать случайного перемещения фокуса ввода в новое окно между вызовами функций `click()` и `typewrite()`, что внесло бы путаницу в выполнение примера.

Сначала Python выполняет виртуальный щелчок мышью в точке экрана с координатами `(100, 100)`, что приводит к выполнению щелчка в окне файлового редактора и перемещению в него фокуса ввода. Вызов `typewrite()` отправляет текст `Hello world!` в это окно (рис. 18.3). Теперь вы располагаете кодом, который может набирать текст вместо вас!

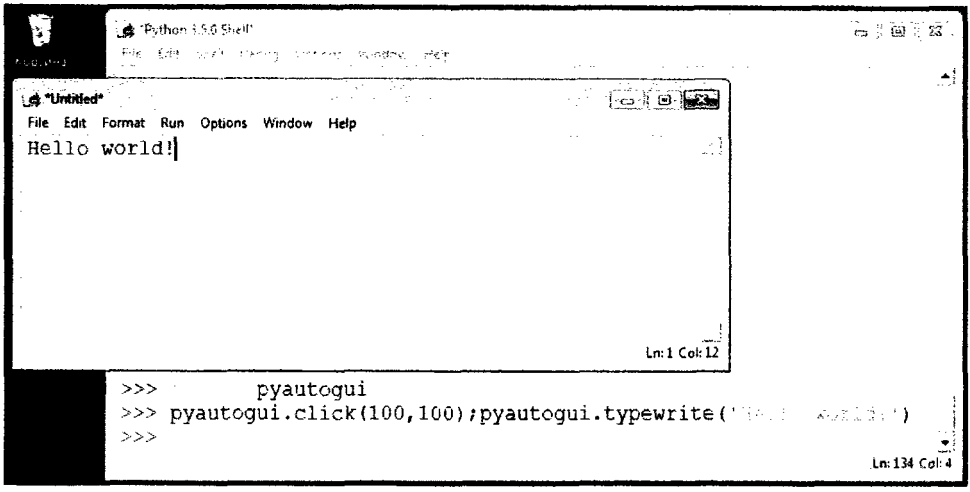


Рис. 18.3. Использование средств PyAutoGUI для выполнения щелчка в окне файлового редактора и ввода в нем текста `Hello world!`

По умолчанию функция `typewrite()` немедленно вводит всю строку. Но можно добавить небольшую задержку между символами, передав функции необязательный второй аргумент. Им является целое или вещественное число, выражающее длительность паузы в секундах. Например, вызов `pyautogui.typewrite('Hello world!', 0.25)` будет выжидать одну четвертую долю секунды после ввода буквы `H`, затем еще одну четвертую долю секунды после ввода буквы `e` и т.д. Такой постепенный ввод текста может быть полезным в случае медленных приложений, которые не в состоянии обрабатывать нажатия клавиш с той же скоростью, с какой их выполняет модуль PyAutoGUI.

Для таких символов, как `A` или `!`, PyAutoGUI автоматически имитирует одновременное нажатие клавиши <Shift>.

Обозначения клавиш

Не все клавиши могут быть представлены одиночными символами. Например, это невозможно сделать для клавиш <Shift> и <↑>. В библиотеке PyAutoGUI эти клавиши представляются короткими строковыми значениями, например `'esc'` — для клавиши <Esc> или `'enter'` — для клавиши <Enter>.

Вместо одиночного строкового значения функции `typewrite()` можно передавать целый список строк, соответствующих таким клавишам. Например, следующий вызов соответствует нажатию клавиши <A> и клавиши <B>, затем – двум нажатиям клавиши <←> и наконец – нажатиям клавиш <X> и <Y>:

```
>>> pyautogui.typewrite(['a', 'b', 'left', 'left', 'X', 'Y'])
```

Поскольку нажатию клавиши <←> соответствует перемещение курсора клавиатуры на один символ влево, результатом выполнения этой команды будет вывод текста `XYab`. В табл. 18.1 приведены строковые обозначения клавиш PyAutoGUI, которые можно передавать функции `typewrite()` для имитации нажатий любых комбинаций клавиш.

Вы сможете ознакомиться со всеми строковыми обозначениями клавиш, допустимыми в PyAutoGUI, выведя список `pyautogui.KEYBOARD_KEYS`. Строка `'shift'` ссылается на левую клавишу <Shift> и эквивалентна клавише `'shiftleft'`. То же самое относится к строкам `'ctrl'`, `'alt'` и `'win'`; все они ссылаются на одноименные левые клавиши.

Таблица 18.1. Атрибуты PyKeyboard

Строковые обозначения клавиш	Описание
`'a'`, `'b'`, `'c'`, `'A'`, `'B'`, `'C'`, `'1'`, `'2'`, `'3'`, `'!'`, `'@'`, `'#'` и т.д.	Клавиши одиночных символов
`'enter'` (а также `'return'` или `'\n'`)	Клавиша <Enter>
`'esc'`	Клавиша <Esc>
`'shiftleft'`, `'shiftright'`	Левая и правая клавиши <Shift>
`'altleft'`, `'altright'`	Левая и правая клавиши <Alt>
`'ctrlleft'`, `'ctrlright'`	Левая и правая клавиши <Ctrl>
`'tab'` (или `'\t'`)	Клавиша <Tab>
`'backspace'`, `'delete'`	Клавиши <Backspace> и <Delete>
`'pageup'`, `'pagedown'`	Клавиши <Page Up> и <Page Down>
`'home'`, `'end'`	Клавиши <Home> и <End>
`'up'`, `'down'`, `'left'`, `'right'`	Клавиши <↑>, <↓>, <←> и <→>
`'f1'`, `'f2'`, `'f3'` и т.д.	Клавиши от <F1> до <F12>
`'volumemute'`, `'volumedown'`, `'volumeup'`	Клавиши отключения, уменьшения и увеличения звука (на некоторых клавиатурах эти клавиши отсутствуют, но операционная система все равно будет распознавать эти имитированные нажатия клавиш)
`'pause'`	Клавиша <Pause>

Окончание табл. 18.1

Строковые обозначения клавиш	Описание
`'capslock'`, `'numlock'`, `'scrolllock'`	Клавиши <Caps Lock>, <Num Lock> и <Scroll Lock>
`'insert'`	Клавиша <Ins> или <Insert>
`'printscreen'`	Клавиша <Prtsc> или <Print Screen>
`'winleft'`, `'winright'`	Левая и правая клавиши <Win> (в Windows)
`'command'`	Клавиша <Command (⌘)> (в OS X)
`'option'`	Клавиша <Option> (в OS X)

Нажатие и отпускание клавиш

В тесной аналогии с функциями `mouseDown()` и `mouseUp()` функции `pyautogui.keyDown()` и `pyautogui.keyUp()` посылают компьютеру сигналы виртуальных нажатий и отпусканий клавиш. В качестве аргумента эти функции принимают строковое обозначение клавиши (см. табл. 18.1). Для большего удобства библиотека PyAutoGUI предоставляет функцию `pyautogui.press()`, которая вызывает обе эти функции для имитации полного цикла нажатия клавиши.

Выполните следующий код, который выведет знак доллара (получаемый в результате нажатия клавиши <4> при одновременно нажатой клавише <Shift>):

```
>>> pyautogui.keyDown('shift'); pyautogui.press('4');
pyautogui.keyUp('shift')
```

Эта строка имитирует нажатие клавиши <Shift>, нажатие (и отпускание) клавиши <4> с последующим отпусканием клавиши <Shift>. Для ввода строки в текстовом поле лучше подойдет функция `typewrite()`. Но в случае приложений, нуждающихся в одноклавишных командах, функция `press()` обеспечивает более простой подход.

Горячие клавиши

Горячие клавиши, или клавиши быстрого вызова, — это комбинации клавиш, предназначенные для активизации определенных функций приложения. Так, распространенными горячими клавишами являются комбинация клавиш <Ctrl+C> (Windows и Linux) и <⌘+C> (OS X). Пользователь нажимает и удерживает клавишу <Ctrl>, затем нажимает клавишу <C> и после этого отпускает клавиши <C> и <Ctrl>. Чтобы сделать это с помощью функций PyAutoGUI `keyDown()` и `keyUp()`, вам нужно ввести следующие команды.

```
pyautogui.keyDown('ctrl')
pyautogui.keyDown('c')
pyautogui.keyUp('c')
pyautogui.keyUp('ctrl')
```

Вводить все эти инструкции довольно утомительно. Вместо этого лучше использовать функцию `pyautogui.hotkey()`, которая принимает несколько строковых аргументов, обозначающих клавиши, выполняет виртуальные нажатия клавиш в указанном порядке, а затем отпускает их в обратном порядке. Для комбинации клавиш <Ctrl+C> вызов будет выглядеть так:

```
pyautogui.hotkey('ctrl', 'c')
```

Эта функция особенно полезна в случае длинных клавиатурных комбинаций. В приложении Word комбинация клавиш <Ctrl+Alt+Shift+S> отображает панель **Стили**. Вместо того чтобы выполнять восемь разных вызовов функций (четыре вызова `keyDown()` и четыре вызова `keyUp()`), достаточно вызвать функцию `hotkey('ctrl', 'alt', 'shift', 's')`.

Переместив новое окно файлового редактора IDLE в верхний левый угол экрана, введите в интерактивной оболочке следующие команды (в OS X вместо строки `'alt'` следует использовать строку `'ctrl'`).

```
>>> import pyautogui, time
>>> def commentAfterDelay():
❶     pyautogui.click(100, 100)
❷     pyautogui.typewrite('In IDLE, Alt-3 comments out a line.')
      time.sleep(2)
❸     pyautogui.hotkey('alt', '3')

>>> commentAfterDelay()
```

В этом коде определяется функция `commentAfterDelay()`, которая выполняет щелчок в окне файлового редактора, чтобы переместить в него фокус ввода ❶, вводит текст *In IDLE, Atl-3 comments out a line* (в IDLE нажатие клавиш <Alt+3> превращает строку в комментарий) ❷, выдерживает паузу в течение 2 секунд, а затем имитирует нажатие комбинации клавиш <Alt+3> (или <Ctrl+3> в OS X) ❸. Это клавиатурное сокращение добавляет два символа решетки (`#`) в текущей строке, тем самым превращая ее в комментарий. (Данный прием будет вам очень полезен при написании собственного кода в IDLE.)

Обзор функций PyAutoGUI

Поскольку в этой главе обсуждалось много различных функций, ниже приведен их краткий справочный обзор.

- `moveTo(x, y)`. Перемещает указатель мыши в точку экрана с заданными координатами *x* и *y*.
- `moveRel(xOffset, yOffset)`. Перемещает указатель мыши относительно его текущей позиции.
- `dragTo(x, y)`. Перемещает указатель мыши, удерживая нажатой ее левую кнопку.
- `dragRel(xOffset, yOffset)`. Перемещает указатель мыши относительно его текущей позиции, удерживая нажатой ее левую кнопку.
- `click(x, y, button)`. Имитирует щелчок (по умолчанию левой кнопкой мыши).
- `rightClick()`. Имитирует щелчок правой кнопкой мыши.
- `middleClick()`. Имитирует щелчок средней кнопкой мыши.
- `doubleClick()`. Имитирует двойной щелчок левой кнопкой мыши.
- `mouseDown(x, y, button)`. Имитирует нажатие указанной кнопки мыши в точке экрана с координатами *x* и *y*.
- `mouseUp(x, y, button)`. Имитирует отпускание указанной кнопки мыши в точке экрана с координатами *x* и *y*.
- `scroll(units)`. Имитирует прокручивание колесика мыши. Положительному значению аргумента соответствует прокрутка вверх, отрицательному — прокрутка вниз.
- `typewrite(message)`. Вводит символы в указанной строке сообщения.
- `typewrite([key1, key2, key3])`. Вводит указанные строковые обозначения клавиш.
- `press(key)`. Нажимает клавишу, заданную указанной строкой.
- `keyDown(key)`. Имитирует нажатие указанной клавиши.
- `keyUp(key)`. Имитирует отпускание указанной клавиши.
- `hotkey([key1, key2, key3])`. Имитирует нажатие клавиш, заданных их строковыми обозначениями, в указанном порядке с последующим их отпусканием в обратном порядке.
- `screenshot()`. Возвращает снимок экрана в виде объекта `Image`. (Более подробную информацию об объекте `Image` вы найдете в главе 17.)

Проект: автоматическое заполнение формы

Самая надоедливая из всех рутинных задач — это заполнение форм. Считайте, что вам повезло: сейчас мы приступаем к рассмотрению проекта, который поможет вам с легкостью решать задачи подобного рода. Предположим, что в электронной таблице хранится огромный массив данных и вам предстоит кропотливая работа по переносу этих данных в форму другого приложения, а свободного стажера, который сделал бы все это вместо вас, рядом не оказалось. Хотя в некоторых приложениях и предусмотрена возможность импорта данных, иногда обстоятельства складываются таким

образом, что единственным приходящим на ум решением является многочасовое щелканье мышью и ввод данных с клавиатуры. Но поскольку вы уже прочитали эту книгу почти до конца, то, конечно же, вам хорошо известно, что существует альтернативный вариант.

В этом проекте будет использоваться форма Google Docs, которую можно скачать на сайте `http://nostarch.com/automatestuff`. Вид этой формы показан на рис. 18.4.

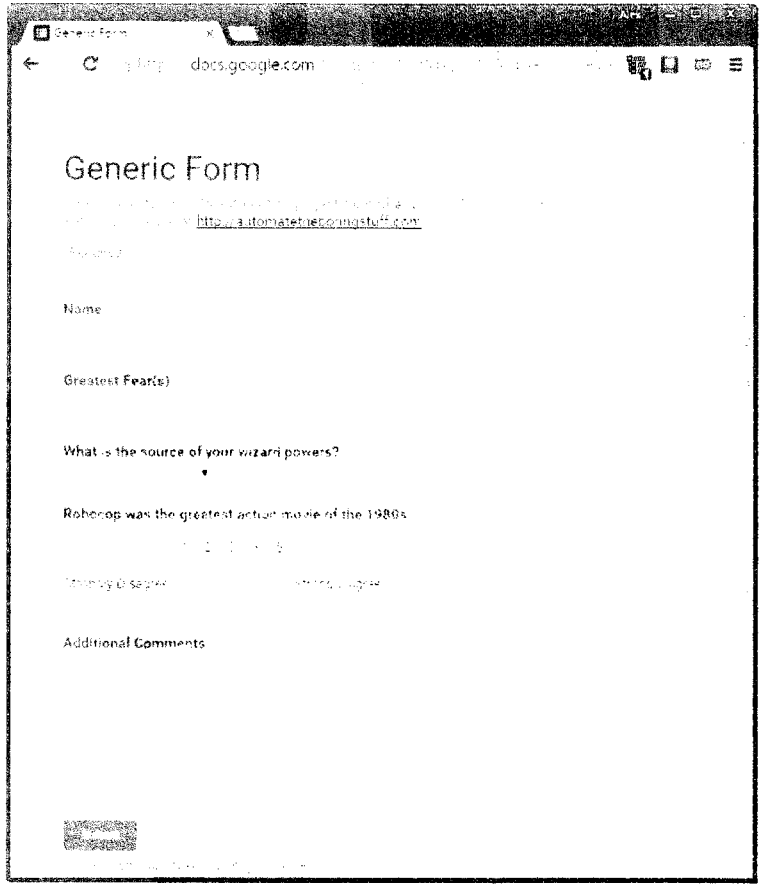


Рис. 18.4. Форма, используемая в этом проекте

Вот что должна делать данная программа при высокоуровневом рассмотрении:

- щелкнуть на первом текстовом поле формы;
- перемещаться по форме, вводя информацию в каждое поле;
- щелкнуть на кнопке Готово;
- повторить весь процесс для следующего набора данных.

Это означает, что ваш код должен выполнять следующие операции:

- вызывать функцию `pyautogui.click()` для выполнения щелчков на форме и кнопке Готово;
- вызывать функцию `pyautogui.typewrite()` для ввода текста в соответствующих полях;
- обрабатывать исключение `KeyboardInterrupt`, чтобы пользователь мог выйти из программы, нажав комбинацию клавиш <Ctrl+C>.

Откройте новое окно в файловом редакторе и сохраните его в файле *formFiller.py*.

Шаг 1. Составление плана действий

Прежде чем приступить к написанию кода, определим точную последовательность нажатий клавиш и щелчков мышью для однократного прохода по форме. Рассмотренный ранее сценарий *mouseNow.py* поможет вам определять конкретные значения координат указателя мыши. Единственное, что вам нужно знать, – это координаты первого текстового поля. После того как будет выполнен щелчок на первом текстовом поле, вам будет достаточно нажимать клавишу <Tab> для перехода к следующему полю. Это избавит вас от необходимости определять координаты *x* и *y* для каждого поля, чтобы выполнить на нем щелчок.

Ниже приведена пошаговая процедура для ввода данных в поля формы.

1. Щелкнуть на поле Name (Имя). (Используйте функцию *mouseNow.py*, чтобы определить координаты этого поля после развертывания окна браузера на весь экран. В случае OS X вам может потребоваться щелкнуть дважды: один раз – для перемещения фокуса в браузер, а второй – для выполнения щелчка в поле Name.)
2. Ввести имя и нажать клавишу <Tab>.
3. Указать предмет наибольшей угрозы (Greatest Fear) и нажать клавишу <Tab>.
4. Нажать клавишу <↓> необходимое количество раз для выбора источника магической силы (Wizard Power Source): 1 – волшебная палочка (Wand), 2 – амулет (Amulet), 3 – хрустальный шар (Crystal Ball) и 4 – деньги (Money). Затем нажать клавишу <Tab>. (Обратите внимание на то, что в случае OS X клавишу <↓> необходимо нажимать на один раз больше для каждой опции. Возможно, в некоторых браузерах потребуется нажимать также клавишу <Enter>.)
5. Нажать клавишу <→> необходимое количество раз для выбора варианта ответа на вопрос о том, был ли *Робокоп* лучшим фильмом 1980-х годов (раздел параметров Robocop was the greatest action movie of the 1980s).

Ее следует нажать один раз для выбора варианта 2, два раза — для выбора варианта 3, три раза — для выбора варианта 4 и четыре раза — для выбора варианта 5. Для выбора варианта 1 (который подсвечен по умолчанию) достаточно нажать клавишу пробела. После этого нажать клавишу <Tab>.

6. Ввести дополнительный комментарий и нажать клавишу <Tab>.
7. Нажать клавишу <Enter> для выполнения "щелчка" на кнопке Submit (Отправить).
8. После отправки формы браузер перейдет на страницу, на которой вам нужно будет щелкнуть на ссылке для возврата на страницу формы.

Обратите внимание на то, что в случае последующего повторного запуска программы может потребоваться обновление координат точки для щелчка, поскольку позиция окна браузера за это время могла измениться. Чтобы устранить эту проблему, всегда убеждайтесь в том, что окно браузера развернуто на весь экран, прежде чем определять координаты первого поля формы. Кроме того, учтите, что различные браузеры ведут себя по-разному в различных операционных системах. Поэтому всегда проверяйте, как работают используемые комбинации клавиш на вашем компьютере, прежде чем запускать программу.

Шаг 2. Настройка координат

Загрузите форму примера (см. рис. 18.4) в браузер и разверните окно на весь экран. Откройте новое окно терминала или командной строки, чтобы выполнить сценарий *mouseNow.py*, а затем поместите указатель мыши над полем Name для определения его координат. Найденные координаты будут присвоены переменной `nameField`. Кроме того, определите значения координат и RGB-значения для голубой кнопки Submit. Эти значения будут присвоены переменным `submitButton` и `submitButtonColor` соответственно.

После этого заполните форму любыми фиктивными данными и щелкните на кнопке Submit. Вам нужно увидеть, что собой представляет следующая страница, чтобы можно было определить координаты ссылки *Submit another response* на ней с помощью сценария *mouseNow.py*.

Откройте новое окно в файловом редакторе и введите следующий код, заменив в нем значения, выделенные курсивом, значениями координат, найденными вами в собственных тестах.

```
#! python3
# formFiller.py - Автоматически заполняет форму.

import pyautogui, time
```

```
# Задайте значения координат, соответствующие вашему компьютеру.
nameField = (648, 319)
submitButton = (651, 817)
submitButtonColor = (75, 141, 249)
submitAnotherLink = (760, 224)

# TODO: Предоставить пользователю возможность прекратить
        выполнение сценария.

# TODO: Дождаться окончания загрузки страницы формы.

# TODO: Заполнить поле Name.

# TODO: Заполнить поле Greatest Fear(s).

# TODO: Заполнить поле Source of Wizard Powers.

# TODO: Заполнить поле RoboCop.

# TODO: Заполнить поле Additional Comments.

# TODO: Щелкнуть на кнопке Submit.

# TODO: Дождаться окончания загрузки страницы формы.

# TODO: Щелкнуть на ссылке Submit another response.
```

Теперь вам нужны данные, которые вы хотите фактически вводить в эту форму. В реальном мире эти данные могут браться из электронных таблиц, простых текстовых файлов или веб-сайтов, и для их загрузки в программу может потребоваться дополнительный код. Однако для данного проекта вам будет достаточно закодировать эти данные в переменной. Добавьте в программу следующий код.

```
#! python3
# formFiller.py - Автоматически заполняет форму.
--пропущенный код--
formData = [{'name': 'Alice', 'fear': 'eavesdroppers', 'source':
'wand',
'robocop': 4, 'comments': 'Tell Bob I said hi.'},
{'name': 'Bob', 'fear': 'bees', 'source': 'amulet', 'robocop': 4,
'comments': 'n/a'},
{'name': 'Carol', 'fear': 'puppets', 'source': 'crystal ball',
'robocop': 1, 'comments': 'Please take the puppets out of the
break room.'},
{'name': 'Alex Murphy', 'fear': 'ED-209', 'source': 'money',
'robocop': 5, 'comments': 'Protect the innocent. Serve the public
trust. Uphold the law.'},
]
--пропущенный код--
```

Список formData содержит по одному словарю для четырех различных лиц. В каждом словаре ключами служат имена текстовых полей, а значениями — ответы на соответствующие вопросы. Последний элемент настройки — установка для переменной PAUSE значения, обеспечивающего создание паузы длительностью полсекунды после каждого вызова функции. Добавьте в программу вслед за инструкцией присваивания значения переменной formData следующую инструкцию:

```
pyautogui.PAUSE = 0.5
```

Шаг 3. Начало ввода данных

Виртуальный ввод данных в текстовые поля будет осуществляться в цикле for, выполняющем итерации по словарям, которые содержатся в списке formData, с передачей значений словаря соответствующим функциям библиотеки PyAutoGUI.

Добавьте в программу следующий код.

```
#! python3
# formFiller.py - Автоматически заполняет форму.

--пропущенный код--

for person in formData:
    # Предоставление пользователю возможности прекратить
    # выполнение сценария.
    print('>>> 5-СЕКУНДНАЯ ПАУЗА, ЧТОБЫ ПОЛЬЗОВАТЕЛЬ УСПЕЛ
↳ НАЖАТЬ КОМБИНАЦИЮ КЛАВИШ <CTRL+C> <<<')
❶   time.sleep(5)

    # Дождаться окончания загрузки страницы формы.
❷   while not pyautogui.pixelMatchesColor(submitButton[0],
↳ submitButton[1], submitButtonColor):
        time.sleep(0.5)

--пропущенный код--
```

В качестве дополнительной меры безопасности в сценарии предусмотрена пятисекундная пауза ❶, предоставляющая пользователю возможность нажать комбинацию клавиш <Ctrl+C> (или переместить указатель мыши в верхний левый угол экрана для возбуждения исключения FailSafeException) с целью прекращения работы программы, если что-то пойдет не так. Затем программа дожидается того момента, когда на экране отобразится кнопка Submit ❷, что будет свидетельствовать о завершении загрузки формы. Вспомните о том, что информация о координатах *x* и *y* и

цвете, необходимая для определения места выполнения щелчка, была сохранена в переменных `submitButton` и `submitButtonColor` на шаге 2. Эта информация передается функции `pixelMatchesColor()` в виде аргументов `submitButton[0]`, `submitButton[1]` и `submitButtonColor` соответственно.

Добавьте в программу следующий код, введя его после кода, организующего задержку до появления на экране кнопки Submit.

```
#! python3
# formFiller.py - Автоматически заполняет форму.

--пропущенный код--

❶ print('Вводится информация о %s...' % (person['name']))
❷ pyautogui.click(nameField[0], nameField[1])

    # Заполнение поля Name.
❸ pyautogui.typewrite(person['name'] + '\t')

    # Заполнение поля Greatest Fear(s).
❹ pyautogui.typewrite(person['fear'] + '\t')

--пропущенный код--
```

Чтобы информировать пользователя о ходе выполнения программы, мы добавили вызов функции `print()`, отображающий статус программы в окне её терминала ❶.

Поскольку к этому моменту времени программе уже известно, что форма загружена, ничто не мешает вызвать функцию `click()` для выполнения виртуального щелчка на поле Name ❷ и функцию `typewrite()` для ввода строки из элемента `person['name']` ❸. Символ `'\t'`, который добавляется в конце строки, передаваемой функции `typewrite()`, имитирует нажатие клавиши <Tab>, что перемещает фокус клавиатуры в следующее поле — Greatest Fear(s). Второй вызов функции `typewrite()` вводит в это поле строку из элемента `person['fear']` и выполняет переход к следующему полю формы с помощью виртуальной клавиши <Tab> ❹.

Шаг 4. Обработка списков выбора и переключателей

Обработка раскрывающегося списка, предлагающего варианты источника "магической силы" (Wizard Powers), и кнопок-переключателей, предлагающих варианты ответа на вопрос о фильме *Робокоп*, требует несколько бо́льших усилий, чем обработка текстовых полей. Выбор этих опций с помощью виртуальных щелчков мышью требует определения координат *x* и *y* каждого из соответствующих элементов управления. Для этой цели проще использовать нажатия виртуальных клавиш стрелок.

Добавьте в программу следующий код.

```
#! python3
# formFiller.py - Автоматически заполняет форму.

--пропущенный код--

    # Заполнение поля Source of Wizard Powers.
❶   if person['source'] == 'wand':
❷       pyautogui.typewrite(['down', '\t'])
    elif person['source'] == 'amulet':
        pyautogui.typewrite(['down', 'down', '\t'])
    elif person['source'] == 'crystal ball':
        pyautogui.typewrite(['down', 'down', 'down', '\t'])
    elif person['source'] == 'money':
        pyautogui.typewrite(['down', 'down', 'down', 'down',
↳ '\t'])

    # Заполнение поля RoboCop.
❸   if person['robocop'] == 1:
❹       pyautogui.typewrite([' ', '\t'])
    elif person['robocop'] == 2:
        pyautogui.typewrite(['right', '\t'])
    elif person['robocop'] == 3:
        pyautogui.typewrite(['right', 'right', '\t'])
    elif person['robocop'] == 4:
        pyautogui.typewrite(['right', 'right', 'right', '\t'])
    elif person['robocop'] == 5:
        pyautogui.typewrite(['right', 'right', 'right', 'right',
↳ '\t'])

--пропущенный код--
```

Как только раскрывающийся список получит фокус ввода (вспомните, что написали код, имитирующий нажатие клавиши <Tab> после заполнения поля Greatest Fear(s)), нажатие клавиши <↓> осуществит переход к следующему элементу списка выбора. Количество нажатий клавиши <↓>, которые должна имитировать программа, прежде чем перейти с помощью клавиши <Tab> к следующему полю, определяется значением, хранящимся в элементе `person['source']`. Если значением ключа `'source'` в словаре данного пользователя является `'wand'` ❶, то мы имитируем однократное нажатие клавиши <↓> (для выбора волшебной палочки) и нажатие клавиши <Tab> ❷. Если этим значением является `'amulet'`, то мы имитируем два нажатия клавиши <↓> (для выбора амулета) и нажатие клавиши <Tab> и так далее для всех остальных возможных вариантов выбора.

Для выбора переключателей, соответствующих различным вариантам ответа на вопрос о фильме *Робокоп*, можно использовать имитацию

нажатий клавиши <→> или, если вы хотите выбрать первый из вариантов ❸, ограничиться нажатием клавиши пробела ❹.

Шаг 5. Отправка формы и ожидание

Можно заполнить поле Additional Comments (Дополнительные комментарии) с помощью функции `typewrite()`, передав ей в качестве аргумента элемент `person['comments']`. Дополнительно можно ввести символ `'\t'`, чтобы переместить фокус ввода на кнопку Submit. Как только кнопка Submit получит фокус, вызов `pyautogui.press('enter')` имитирует нажатие клавиши <Enter> и отправит форму. После отправки формы ваша программа будет в течение пяти секунд ожидать загрузки следующей страницы.

Как только загрузится другая страница, она получит ссылку *Submit another response*, которая перенаправит браузер на новую, пустую страницу формы. Вы сохраняете координаты этой ссылки в виде кортежа в переменной `submitAnotherLink` на шаге 2, поэтому передайте эти координаты функции `pyautogui.click()` для имитации щелчка на этой ссылке.

Когда новая форма будет готова к работе, внешний цикл `for` сможет перейти к следующей итерации и ввести в форму информацию, относящуюся к следующему лицу.

Завершите программу, добавив в нее следующий код.

```
#! python3
# formFiller.py - Автоматически заполняет форму.

--пропущенный код--

    # Заполнение поля Additional Comments.
    pyautogui.typewrite(person['comments'] + '\t')

    # Выполнение щелчка на кнопке Submit.
    pyautogui.press('enter')

    # Дождаться окончания загрузки страницы формы.
    print('Выполнен щелчок на кнопке Submit.')
    time.sleep(5)

    # Щелчок на ссылке Submit another response link.
    pyautogui.click(submitAnotherLink[0], submitAnotherLink[1])
```

Как только основной цикл `for` завершит свою работу, программа будет располагать полной информацией по каждому лицу. В данном примере мы имеем дело всего лишь с четырьмя лицами. Но если речь идет о четырех тысячах человек, то написание подобной программы сэкономит вам массу времени и избавит от необходимости потеть за клавиатурой!

Резюме

Средства GUI-автоматизации, предлагаемые модулем `pyautogui`, позволяют вам взаимодействовать с приложениями, выполняющимися на вашем компьютере, посредством управления мышью и клавиатурой. Несмотря на то что этот подход достаточно гибок для того, чтобы выполнять все те действия, которые может выполнять человек, у него есть один недостаток, заключающийся в определенной слепоте программ, в которых данный подход используется для имитации щелчков мышью и клавиатурного ввода. При написании программ GUI-автоматизации всегда старайтесь обеспечить возможность их быстрого аварийного завершения, если им были предоставлены неподходящие инструкции. Аварийное завершение может раздражать, но все же это лучше, чем некорректное выполнение программы, дающей непредсказуемые результаты.

Используя средства PyAutoGUI, можно перемещать указатель мыши по экрану, а также имитировать щелчки мышью и нажатия обычных и горячих клавиш. Кроме того, модуль `pyautogui` обеспечивает возможность проверки цвета пикселей на экране, что может быть использовано для анализа содержимого экрана с целью контроля нормального процесса выполнения программ GUI-автоматизации. Вы даже можете предоставлять средствам PyAutoGUI снимки экрана и позволить им определять координаты области, в которой следует выполнить виртуальный щелчок мышью.

Все эти возможности библиотеки PyAutoGUI можно комбинировать для автоматизации выполнения любых повторяющихся задач. В действительности вид указателя мыши, автоматически перемещающегося по экрану, или текста, появляющегося в полях формы без вашего вмешательства, действует завораживающе. Почему бы не потратить сэкономленное время на то, чтобы откинуться на спинку кресла и понаблюдать, как программа выполняет всю работу за вас? Это ведь так приятно — осознавать, что твои умения и смекалка позволили тебе избавиться от бремени рутинной работы!

Контрольные вопросы

1. Как запустить средства резервного выхода PyAutoGUI для прекращения работы программы?
2. Какая функция возвращает текущее разрешение экрана?
3. Какая функция возвращает координаты текущей позиции указателя мыши?
4. В чем различие между функциями `pyautogui.moveTo()` и `pyautogui.moveRel()`?
5. Какую функцию можно использовать для перетаскивания указателя мыши?

6. Вызов какой функции введет текст «`Hello world!`»?
7. Как сымитировать нажатия специальных клавиш клавиатуры, таких как <←>?
8. Как сохранить текущее содержимое экрана в файле изображения *screenshot.png*?
9. С помощью какого кода можно задать паузу длительностью две секунды после каждого вызова функции библиотеки PyAutoGUI?

Учебные проекты

Чтобы закрепить полученные знания на практике, напишите программы для предложенных ниже задач.

Как притвориться занятым

Многие из программ мгновенного обмена сообщениями определяют, нет ли вас на месте, т.е. перед компьютером, обнаруживая отсутствие перемещений мыши в течение определенного периода времени, скажем, десяти минут. Возможно, вы отошли от компьютера, чтобы перекусить, но не хотите, чтобы другие люди могли подумать, основываясь на отображаемом статусе программы-мессенджера, что вы бездельничаете. Напишите сценарий, который будет слегка перемещать указатель мыши каждые десять минут. Эти перемещения должны быть достаточно небольшими, чтобы не мешать вашей работе, если вам потребуется использовать компьютер во время работы сценария.

Бот для отправки мгновенных сообщений

Google Talk, Skype, Yahoo Messenger, AIM и другие приложения для обмена мгновенными сообщениями нередко используют проприетарные протоколы, затрудняющие другим разработчикам написание сценариев на Python, способных взаимодействовать с этими программами. Но даже эти проприетарные протоколы не смогут помешать вам писать программы GUI-автоматизации.

В приложении Google Talk имеется строка поиска, позволяющая вам ввести имя пользователя из списка ваших друзей и открывающая окно сообщений при нажатии клавиши <Enter>. Фокус ввода автоматически перемещается в новое окно. Другие приложения-мессенджеры предлагают аналогичные способы открытия новых окон сообщений. Напишите программу для автоматической рассылки уведомлений группе людей из списка друзей. Вашей программе придется обрабатывать различные исключительные случаи, такие, например, как отсутствие адресатов в сети, появление окна чата

в разных местах экрана или открытие диалоговых окон для подтверждения действий, что может прерывать работу сценария. Ваша программа должна будет получать снимки экрана и использовать их для управления взаимодействием с графическим интерфейсом пользователя, а также предусматривать способы обнаружения ситуаций, в которых отправка виртуальных клавиатурных нажатий не может быть осуществлена.

Примечание

Возможно, вам стоит создать тестовые фиктивные учетные записи, чтобы вы случайно не забросали спамом своих друзей, пока пишете эту программу.

Руководство по созданию игрового бота

На сайте `http://nostarch.com/automatestuff/` вы найдете замечательное практическое руководство под названием "Создание бота, играющего в онлайн-игры, на языке Python". В этом руководстве объясняется, как создать программу, основанную на средствах GUI-автоматизации Python, которая способна играть во флеш-игру под названием *Суши-бар*. В этой игре нужно щелкать на кнопках, соответствующих различным ингредиентам, для оформления заказов суши. Чем быстрее вы заполните заказы, не допустив ошибок, тем больше очков заработаете. Эта игра идеально приспособлена для того, чтобы запрограммировать ее средствами GUI-автоматизации и при этом... немного схитрить, добавив себе лишних очков! В этом руководстве затронуты многие вопросы, рассмотренные в данной главе, а кроме того, описаны базовые возможности распознавания образов средствами PyAutoGUI.

A

УСТАНОВКА МОДУЛЕЙ СТОРОННИХ РАЗРАБОТЧИКОВ

Помимо стандартной библиотеки модулей, входящей в поставку Python, можно использовать модули сторонних разработчиков, расширяющие возможности исходного набора. Основным средством для установки модулей Python, разработанных другими лицами, является утилита `pip`. Она обеспечивает безопасную загрузку и установку модулей Python, доступных на сайте организации Python Software Foundation по адресу `https://pypi.python.org/`. Сайт PyPI (от англ. Python Package Index – каталог пакетов Python) играет роль своего рода "магазина бесплатных приложений" для Python.

Утилита `pip`

Исполняемый файл утилиты `pip` для Windows называется *pip*, а для OS X и Linux – *pip3*. Путь к нему в Windows – *C:\Python34\Scripts\pip.exe*, в OS X – */Library/Frameworks/Python.framework/Versions/3.4/bin/pip3*, в Linux – */usr/bin/pip3*.

В то время как в Windows и OS X утилита `pip` устанавливается автоматически вместе с Python 3.4, в Linux ее нужно устанавливать отдельно. Чтобы установить `pip3` на компьютерах, работающих под управлением Ubuntu или Debian Linux, откройте новое окно терминала и введите команду `sudo apt-get install python3-pip`. В случае Fedora Linux для этого следует ввести в окне терминала команду `sudo yum install python3-pip`. Возможно, для установки этого программного обеспечения вам придется ввести пароль системного администратора.

Установка сторонних модулей

Утилита `pip` предназначена для запуска из командной строки: ей передается команда `install`, за которой следует имя устанавливаемого модуля. Например, на компьютерах с Windows необходимо ввести команду `pip install` *`Имя_модуля`*. На компьютерах с OS X или Linux утилита `pip3` должна запускаться с префиксом `sudo`, предоставляющим административные привилегии для установки модуля. В этом случае команда принимает следующий вид: `sudo pip3 install` *`Имя_модуля`*.

Если модуль уже установлен, но вы хотите обновить его до последней версии, доступной на сайте PyPI, выполните следующую команду: `pip install -U` *`Имя_модуля`* (или `pip3 install -U` *`Имя_модуля`* в случае OS X или Linux).

После того как модуль установлен, можно протестировать корректность его установки, выполнив команду `import` *`Имя_модуля`* в интерактивной оболочке. В случае отсутствия сообщения об ошибке можно полагать, что установка модуля прошла успешно.

Чтобы установить все модули, о которых шла речь в книге, выполните перечисленные ниже команды. (Не забывайте о том, что в случае OS X или Linux вместо `pip` следует использовать `pip3`.)

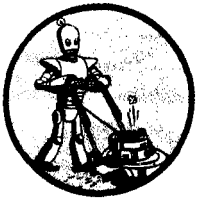
- `pip install send2trash`
- `pip install requests`
- `pip install beautifulsoup4`
- `pip install selenium`
- `pip install openpyxl`
- `pip install PyPDF2`
- `pip install python-docx` (именно `python-docx`, а не `docx`)
- `pip install imapclient`
- `pip install pyzmail`
- `pip install twilio`
- `pip install pillow`
- `pip install pyobjc-core` (только на компьютерах с OS X)
- `pip install pyobjc` (только на компьютерах с OS X)
- `pip install python3-xlib` (только на компьютерах с Linux)
- `pip install pyautogui`

Примечание

Пользователям OS X: для установки модуля `pyobjc` может потребоваться около 20 минут или даже больше, поэтому не беспокойтесь, если она выполняется не так быстро, как вы рассчитывали. Кроме того, первым следует установить модуль `pyobjc-core`, что позволит уменьшить общее время установки.

Б

ЗАПУСК ПРОГРАММ



Выполнить программу, открытую в окне файлового редактора IDLE, не составляет труда — для этого достаточно нажать клавишу <F5> или выбрать пункты меню Run⇨Run Module (Выполнить⇨Выполнить модуль). Это простейший способ запуска программ в процессе их написания, но открывать IDLE для запуска готовых программ — слишком обременительный метод. Для выполнения сценариев, написанных на языке Python, существуют более удобные способы.

"Магическая" строка

Каждая ваша программа на Python должна начинаться с "магической" строки (англ. "shebang line"), которая сообщает компьютеру о том, что выполнение данной программы вы поручаете Python. "Магическая" строка начинается символами `#!`, но в остальном ее вид зависит от используемой вами операционной системы, а именно:

- Windows — `#! python3`;
- OS X — `#! /usr/bin/env python3`;
- Linux — `#! /usr/bin/python3`.

При запуске сценариев на Python в окне IDLE "магическая" строка является излишней, однако она необходима при запуске сценария из командной строки.

Запуск программ на Python в Windows

В Windows интерпретатору Python 3.4 соответствует путь `C:\Python34\python.exe`. Удобный альтернативный вариант предлагает программа `py.exe`, которая читает "магическую" строку в начале *.py*-файла, содержащего исходный код, и запускает версию Python, подходящую для этого сце-

нария. Программа *py.exe* гарантированно запускает нужную версию Python, если на компьютере установлены сразу несколько версий.

Чтобы обеспечить удобный запуск своей программы на Python, создайте пакетный файл (файл с расширением *.bat*), который будет запускать программу с помощью исполняемого файла *py.exe*. Для этого создайте простой текстовый файл, содержащий всего одну строку следующего вида:

```
@py.exe C:\путь\к\вашему\сценарию\pythonScript.py %*
```

Подставьте вместо указанного здесь пути абсолютный путь к своей программе и сохраните этот файл с расширением `.bat` (например, как файл *pythonScript.bat*). Этот пакетный файл избавит вас от необходимости вводить полный абсолютный путь к файлу программы на Python при каждом ее запуске. Я рекомендую сохранять все свои *.bat-* и *.py*-файлы в одной папке, например *C:\MyPythonScripts* или *C:\Пользователи\Ваше_имя\PythonScripts*.

Имя папки *C:\MyPythonScripts* следует добавить в список каталогов Windows, в которых расположены исполняемые файлы, чтобы пакетные файлы можно было запускать из диалогового окна **Выполнить**. Для этого измените содержимое переменной среды `PATH`. Щелкните на кнопке **Пуск** и начните вводить текст *Изменение переменных среды текущего пользователя*. Эта опция должна автоматически отобразиться средством автозавершения по мере ввода вами ее названия. Щелчок на этой опции открывает диалоговое окно **Переменные среды** (рис. Б.1).

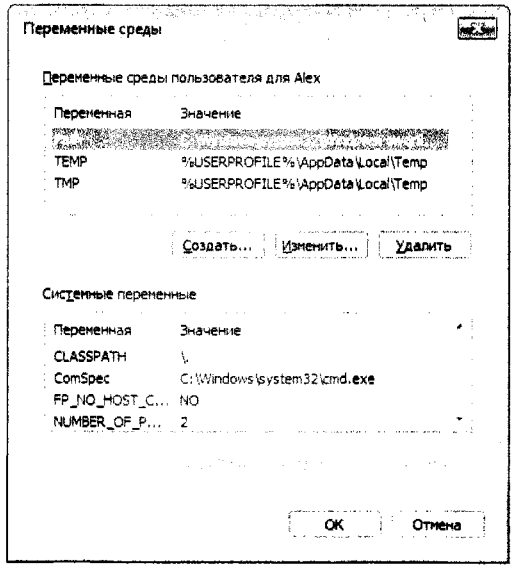


Рис. Б.1. Окно Переменные среды в Windows

В разделе Переменные среды пользователя для *<Ваше_имя>* выделите переменную `Path` и щелкните на кнопке **Изменить**. Добавьте в конце текстового поля **Значение** точку с запятой и введите *C:\MyPythonScripts*, после чего щелкните на кнопке **OK**. Теперь для запуска любого сценария, файл которого находится в папке *C:\MyPythonScripts*, достаточно будет нажать комбинацию клавиш <Win+R> и ввести имя сценария. Например, ввод имени *pythonScript* запустит пакетный файл *pythonScript.bat*, который, в свою очередь, избавит вас от необходимости вводить в диалоговом окне **Выполнить** длинную команду `py.exe C:\MyPythonScripts\pythonScript.py`.

Запуск программ на Python в OS X и Linux

В OS X последовательно выберите **Applications⇨Utilities⇨Terminal**, что приведет к открытию окна **Terminal**. Окно терминала обеспечивает возможность ввода команд исключительно в виде текста, а не с помощью мыши посредством графического интерфейса. Чтобы открыть окно терминала на компьютерах с Ubuntu Linux, нажмите клавишу <Win> (или <Super>) для открытия интерфейса Dash и введите `Terminal`.

Окно терминала открывается в базовой папке вашей учетной записи пользователя. Если моим именем пользователя является *asweigart*, то базовой папкой будет */Users/asweigart* в OS X и */home/asweigart* в Linux. Символ "тильда" (~) является сокращенным обозначением базовой папки, поэтому для перехода к ней можно ввести команду `cd ~`. Команду `cd` также можно использовать для смены текущего рабочего каталога. Как в OS X, так и в Linux для вывода имени текущего рабочего каталога используется команда `pwd`.

Чтобы запустить программу, написанную на языке Python, сохраните ее *.py*-файле в своей базовой папке. Затем измените права доступа к этому файлу, выполнив команду `chmod +x pythonScript.py`. Рассмотрение прав доступа к файлам и каталогам выходит за рамки данной книги, но вам обязательно нужно будет выполнить эту команду по отношению к файлу программы, если вы хотите запускать ее из окна терминала. Сделав это, вы сможете в любой момент запустить программу, открыв окно терминала и введя команду `./pythonScript.py`. "Магическая" строка в начале сценария сообщит операционной системе, где искать исполняемый файл интерпретатора Python.

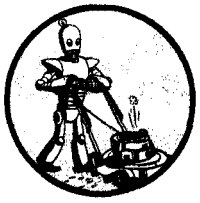
Запуск программ на Python с отключенными утверждениями

Вы сможете несколько увеличить скорость выполнения своих программ на Python, отключив инструкции утверждений. Для этого, запуская

программу из окна терминала, укажите переключатель `-O` после `python` или `python3` и перед именем *.py*-файла программы. Это приведет к запуску оптимизированной версии вашей программы, в которой инструкции утверждений будут игнорироваться.

B

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ



В этом приложении даны ответы на контрольные вопросы, приведенные в конце каждой главы. Я настоятельно рекомендую вам пытаться самостоятельно находить правильные ответы на них и обращаться к данному приложению лишь для сверки. Чтобы научиться программировать, одного лишь знания синтаксиса языка и названий функций недостаточно. Как и при изучении иностранного языка, чем больше вы будете применять свои знания на практике, тем лучше будут результаты. Существует множество веб-сайтов, на которых можно аналогичным образом проверить свои знания. Список их адресов приведен на сайте `http://nostarch.com/automatestuff/`.

Глава 1

1. Операторами являются `+`, `-`, `*` и `/`. Значениями являются `'hello'`, `-8.8` и `5`.
2. Строка — `'spam'`; переменная — `spam`. Строки всегда заключаются в кавычки или апострофы.
3. Три типа данных, введенных в этой главе, — это целые числа, вещественные числа и строки.
4. Выражение — это сочетание значений и операторов. Любое выражение сводится к одиночному значению.
5. Любое выражение вычисляется, давая одиночное значение. В случае инструкций это не так.
6. Для переменной `bacon` установлено значение `20`. Выражение `bacon + 1` не изменяет значение `bacon` (для этого потребовалась бы инструкция присваивания: `bacon = bacon + 1`).
7. Вычисление обоих выражений дает одну и ту же строку `'spamspamspam'`.
8. Имена переменных не могут начинаться с цифры.

9. Целочисленную, вещественную и строковую версии передаваемого им значения возвращают соответственно функции `int()`, `float()` и `str()`.
10. В данном выражении ошибка возникает из-за того, что `99` — целое число, тогда как с помощью оператора + могут конкатенироваться только строки. Для получения правильного результата следует использовать запись вида `'Я съел ' + str(99) + ' лепешек.'`.

Глава 2

1. Это значения `True` и `False`, причем первые буквы, `T` и `F`, — прописные, остальные — строчные.

   `and`, `or`, and `not`.
2. `True and True` — `True`.

   `True and False` — `False`.

   `False and True` — `False`.

   `False and False` — `False`.

   `True or True` — `True`.

   `True or False` — `True`.

   `False or True` — `True`.

   `False or False` — `False`.

   `not True` — `False`.

   `not False` — `True`.
3. `False`

   `False`

   `True`

   `False`

   `False`

   `True`
4. `==`, `!=`, `<`, `>`, `<=` и `>=`.
5. Оператор равенства `==` сравнивает два значения и возвращает результат сравнения в виде булева значения, тогда как оператор присваивания `=` сохраняет значения в переменной.
6. Условие — это используемое в управляющих инструкциях выражение, вычисление которого дает булево значение.
7. Тремя блоками являются: все тело инструкции `if`, строка `print('bacon')` и строка `print('ham')`.

```
print('eggs')
if spam > 5:
```

```
    print('bacon')
else:
    print('ham')
print('spam')
```

8. Код:

```
if spam == 1:
    print('Hello')
elif spam == 2:
    print('Howdy')
else:
    print('Greetings!')
```

9. Для прекращения выполнения программы, увязшей в бесконечном цикле, следует нажать комбинацию клавиш <Ctrl+C>.
10. Инструкция `break` осуществляет выход из цикла и продолжает выполнение с инструкции, следующей за циклом. Инструкция `continue` осуществляет переход в начало цикла и продолжает его выполнение со следующей итерации.
11. Все эти вызовы делают одно и то же. Вызов `range(10)` задает диапазон изменения переменной цикла от `0` до (но не включая) `10`; вызов `range(0, 10)` в явном виде задает начальное значение переменной цикла равным `0`; вызов `range(0, 10, 1)` в явном виде задает инкремент переменной цикла равным `1` на каждой итерации.
12. Код:

```
for i in range(1, 11):
    print(i)
```

и

```
i = 1
while i <= 10:
    print(i)
    i = i + 1
```

13. С помощью вызова `spam.bacon()`.

Глава 3

1. Функции уменьшают потребность в дублировании кода, что положительно сказывается на размере программ и их удобочитаемости и упрощает их обновление.
2. Код функции выполняется тогда, когда она вызывается, а не тогда, когда определяется.

3. Для определения (т.е. создания) функций предназначена инструкция `def`.
4. Функция включает инструкцию `def` и код, образующий ее тело. Вызов функции — это передача управления потоком выполнения программы в тело функции и выполнение ее кода для вычисления возвращаемого значения.
5. Существует только одна глобальная область видимости, тогда как каждый вызов функции создает свою локальную область видимости.
6. Когда происходит возврат из функции, ее локальная область видимости уничтожается и вся информация о ее переменных теряется.
7. Возвращаемое значение — это значение, вычисляемое в результате вызова функции. Как и любое другое значение, возвращаемое функцией значение может быть использовано в качестве части выражения.
8. Если в функции отсутствует инструкция `return`, ее возвращаемым значением является `None`.
9. Для этого следует предварить имя переменной ключевым словом `global`.
10. Типом данных значения `None` является `NoneType`.
11. Эта инструкция `import` импортирует модуль `areallyourpetsnamederic`. (Кстати, такого модуля Python не существует.)
12. Эту функцию можно вызвать с помощью вызова `spam.bacon()`.
13. Поместить строку кода, в которой может возникнуть ошибка, в тело инструкции `try`.
14. В инструкцию `try` помещается код, в котором возможно возникновение ошибки. В инструкцию `except` помещается код, который должен быть выполнен в случае возникновения ошибки.

Глава 4

1. Пустое списковое значение, т.е. списковое значение, не содержащее элементов. Здесь наблюдается полная аналогия с тем, как `''` является пустым строковым значением.
2. `spam[2] = 'hello'`. (Обратите внимание на то, что третьим значением в списке является элемент с индексом `2`, поскольку индексация начинается с `0`.)
3. `'d'`. (Заметьте, что результатом вычисления выражения `'3' * 2` является строка `'33'`, которая передается функции `int()` до выполнения операции деления на `11`. Конечным результатом является целочисленное значение `3`. Выражения могут использоваться везде, где используются значения.)

4. `'d'`. (Отрицательные значения индексов отсчитываются с конца.)
5. `['a', 'b']`
6. `1`
7. `[3.14, 'cat', 11, 'cat', True, 99]`
8. `[3.14, 11, 'cat', True]`
9. Оператором конкатенации списков является `+`, оператором репликации — `*`. (То же самое, что и для строк.)
10. В то время как функция `append()` добавляет значения только в конце списка, функция `insert()` может добавлять их в любом месте списка.
11. Для удаления значений из списка могут использоваться инструкция `del` и списковый метод `remove()`.
12. Как списки, так и строки могут передаваться функции `len()`, иметь индексы и срезы, использоваться в циклах `for`, конкатенироваться и реплицироваться, а также использоваться совместно с операторами `in` и `not in`.
13. Списки изменяемы; они допускают добавление, удаление и изменение хранящихся в них значений. Кортежи неизменяемы; они не допускают вообще никаких изменений. Кроме того, кортежи записывают с использованием круглых скобок, `( and )`, тогда как списки — с использованием квадратных скобок, `[ and ]`.
14. `(42,)`. (Завершающая запятая обязательна.)
15. С помощью функций `tuple()` и `list()` соответственно.
16. Они содержат ссылки на списковые значения.
17. Функция `copy.copy()` создает поверхностную копию списка, тогда как функция `copy.deepcopy()` — полную копию. Это означает, что только функция `copy .deepcopy()` будет дублировать списки, содержащиеся в составе других списков.

Глава 5

1. Две фигурные скобки: `{}`.
2. `{'foo': 42}`
3. Элементы словаря не упорядочены, тогда как элементы списка упорядочены.
4. Возникнет ошибка `KeyError`.
5. Между ними нет никакой разницы. Оператор `in` проверяет существование значения по существованию ключа в словаре.
6. Выражение `'cat' in spam` проверяет, существует ли ключ `'cat'` в словаре, тогда как выражение `'cat' in spam.values()` проверяет, существует ли значение `'cat'` для одного из ключей в словаре `spam`.

7. `'spam.setdefault('color', 'black')`
8. `pprint.pprint()`

Глава 6

1. Экранированные символы представляют те символы в строковых значениях, которые иначе было бы трудно или невозможно ввести в код.
2. `\n` — символ новой строки; `\t` — символ табуляции.
3. Символ обратной косой черты представляется в строке экранированным символом `\\`.
4. Апостроф в строке `"Howl's is fine"` вполне уместен, поскольку начало и конец строки обозначены кавычками.
5. Многострочные блоки допускают включение символов новой строки без использования экранированного символа `\n`.
6. Вычисление этих выражений даст следующие результаты:
   - `'e'`
   - `'Hello'`
   - `'Hello'`
   - `'lo world!`
7. Вычисление этих выражений даст следующие результаты:
   - `'HELLO'`
   - `True`
   - `'hello'`
8. Вычисление этих выражений даст следующие результаты:
   - `['Remember,', 'remember,', 'the', 'fifth', 'of', 'November.']`
   - `'There-can-be-only-one.'`
9. Строковые методы `rjust()`, `ljust()` и `center()` соответственно.
10. Пробельные символы в начале и конце строки удаляются с помощью методов `lstrip()` и `rstrip()` соответственно.

Глава 7

1. Функция `re.compile()` возвращает объект `Regex`.
2. "Сырые" строки используют для того, чтобы символы обратной косой черты не надо было экранировать.
3. Метод `search()` возвращает объект `Match`.
4. Метод `group()` возвращает строки, соответствующие шаблону регулярного выражения.

5. Группа `0` соответствует всему совпавшему тексту, группа `1` — тексту, совпавшему с выражением в первой паре круглых скобок, группа `2` — тексту, совпавшему с выражением во второй паре круглых скобок.
6. Точки и круглые скобки можно экранировать символами обратной косой черты: `\.`, `\(` и `\)`.
7. Если в регулярном выражении отсутствуют группы, возвращается список строк; в противном случае возвращается список кортежей строк.
8. Символ `|` означает поиск соответствия одной из двух альтернативных групп.
9. Символ `?` может означать либо поиск нулевого или единичного количества вхождений предыдущей группы, либо нежадный поиск.
10. Символ `+` означает одно и более вхождений искомого выражения. Символ `*` означает нуль или несколько вхождений искомого выражения.
11. Записи `{3}` соответствуют ровно три вхождения предыдущей группы. Записи `{3,5}` соответствуют от трех до пяти вхождений.
12. Сокращенные символьные классы `\d`, `\w` и `\s` означают поиск одиночной цифры, словарного или пробельного символа соответственно.
13. Сокращенные символьные классы `\D`, `\W` и `\S` означают поиск одиночного символа, не являющегося цифрой, словарным или пробельным символом соответственно.
14. Передача константы `re.I` или `re.IGNORECASE` в качестве второго аргумента функции `re.compile()` сделает регулярное выражение нечувствительным к регистру.
15. Обычно символ `.` совпадает с любым символом, за исключением символа новой строки. Если функции `re.compile()` передать константу `re.DOTALL` в качестве второго аргумента, то точке будет соответствовать также символ новой строки.
16. Символы `.*` задают режим жадного поиска, а символы `.*?` — нежадного.
17. Либо `[0-9a-z]`, либо `[a-z0-9]`
18. `'X drummers, X pipers, five rings, X hens'`
19. Аргумент `re.VERBOSE` делает возможным добавление пробельных символов и комментариев в строку, передаваемую функции `re.compile()`.
20. `re.compile(r'^\d{1,3}(,{3})*$')`, но возможны и другие варианты строк регулярных выражений, используемых в качестве аргумента.
21. `re.compile(r'[A-Z][a-z]*\sNakamoto')`
22. `re.compile(r'(Alice|Bob|Carol)\s(eats|pets|throws)\ s(apples|cats|baseballs)\.', re.IGNORECASE)`

Глава 8

1. Относительные пути определяются относительно текущего рабочего каталога.
2. Абсолютные пути начинаются с корневой папки, например / или C:\.
3. Функция os.getcwd() возвращает текущий рабочий каталог. Функция os.chdir() изменяет текущий рабочий каталог.
4. Папка . — это текущая папка, папка .. — ее родительская папка.
5. C:\bacon\eggs — это имя каталога, тогда как spam.txt — базовое имя.
6. Строка 'r' задает режим чтения, строка 'w' — режим записи, а строка 'a' — режим присоединения.
7. Содержимое существующего файла, открытого в режиме записи, уничтожается и полностью перезаписывается.
8. Метод read() возвращает все содержимое файла в виде одного строкового значения. Метод readlines() возвращает список строк, в котором каждая строка представляет строку содержимого файла.
9. Организация хранилища, создаваемого с помощью модуля shelve, напоминает организацию словаря; в хранилищах, как и в словарях, используются ключи и значения, а также методы keys() и values(), работающие аналогично одноименным методам словарей.

Глава 9

1. Функция shutil.copy() копирует одиночный файл, тогда как функция shutil.copytree() копирует всю папку вместе со всем ее содержимым.
2. Функция shutil.move() используется для переименования файлов, а также для их перемещения.
3. Функции модуля send2trash перемещает файл или папку в корзину, тогда как соответствующая функция модуля shutil безвозвратно удаляет файлы и папки.
4. функция zipfile.ZipFile() эквивалентна функции open(); ее первым аргументом является имя файла, а вторым — режим, в котором открывается ZIP-файл (чтение, запись, присоединение).

Глава 10

1. assert(spam >= 10, 'Значение переменной spam меньше 10.')
2. assert(eggs.lower() != bacon.lower(), 'Переменные eggs и bacon содержат одинаковые строки!') **или** assert(eggs.upper() != bacon.upper(), 'Переменные eggs и bacon содержат одинаковые строки!')

3. `assert(False, 'Это утверждение всегда возбуждает исключение AssertionError.')`
4. Чтобы можно было вызывать функцию `logging.debug()`, в начале программы должны находиться следующие две строки кода.

```
import logging
logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s -
↳ %(levelname)s - %(message)s')
```

5. Чтобы можно было записывать сообщения в файл журнала `programLog.txt` с помощью функции `logging.debug()`, в начале программы должны находиться следующие две строки кода.

```
import logging
logging.basicConfig(filename='programLog.txt',
↳ level=logging.DEBUG, format=' %(asctime)s - %(levelname)s -
↳ %(message)s')
```

6. `DEBUG`, `INFO`, `WARNING`, `ERROR` и `CRITICAL`
7. `logging.disable(logging.CRITICAL)`
8. Вывод сообщений протоколирования можно отключить, не удаляя вызовы функций протоколирования. Вы можете селективно отключать вывод сообщений протоколирования, соответствующих ошибкам более низких уровней критичности. Сообщения протоколирования можно создавать. Сообщения протоколирования обеспечивают создание временных меток.
9. После щелчка на кнопке Step отладчик вызывает функцию и останавливается на первой строке ее кода. После щелчка на кнопке Over выполняется не только вызов функции, но и весь ее код в обычном режиме. После щелчка на кнопке Out строки кода выполняются в обычном режиме до тех пор, пока не будет осуществлен возврат из текущей функции.
10. После щелчка на кнопке Go запускается процесс обычного выполнения программы до ее полного завершения или до достижения строки кода с точкой останова.
11. Установление точки останова для строки кода приводит к тому, что при достижении этой строки отладчик приостанавливает выполнение программы.
12. Чтобы установить точку останова в IDLE, следует щелкнуть правой кнопкой мыши на строке кода и выбрать в контекстном меню пункт Set Breakpoint.

Глава 11

1. В модуле `webbrowser` имеется метод `open()`, который запускает браузер и направляет его по определенному URL-адресу, и на этом все. Модуль `requests` может загружать файлы и страницы из Интернета. Модуль Beautiful Soup осуществляет синтаксический анализ HTML-документов. Наконец, модуль `selenium` может запускать браузер и управлять его выполнением.
2. Функция `requests.get()` возвращает объект `Response`, имеющий атрибут `text`, который содержит загруженное содержимое в виде строки.
3. Метод `raise_for_status()` возбуждает исключение в случае возникновения проблем в процессе загрузки и ничего не делает в случае успешной загрузки.
4. Атрибут `status_code` объекта `Response` содержит код состояния HTTP.
5. Для этого следует открыть на компьютере новый файл в режиме записи двоичных данных (`'wb'`) и использовать цикл `for` для итерирования по возвращаемому значению метода `iter_content()` объекта `Response` для записи порций содержимого в файл.

```
saveFile = open('filename.html', 'wb')
for chunk in res.iter_content(100000):
    saveFile.write(chunk)
```

6. Для открытия панели инструментов разработчика в браузере Chrome следует нажать клавишу <F12>. В браузере Firefox для этого следует нажать комбинацию клавиш <Ctrl+Shift+C> (Windows и Linux) или <⌘+Option+C> (OS X).
7. Для этого следует щелкнуть правой кнопкой мыши на элементе на странице и выбрать в открывшемся контекстном меню пункт Просмотреть код.
8. `'#main'`
9. `'.highlight'`
10. `'div div'`
11. `'button[value=»favorite»]'`
12. `spam.getText()`
13. `linkElem.attrs`
14. Для импортирования модуля `selenium` следует использовать инструкцию `from selenium import webdriver`.

15. Методы `find_element_*` возвращают первый совпавший элемент в виде объекта `WebElement`. Методы `find_elements_*` возвращают список всех совпавших элементов в виде объектов `WebElement`.
16. Методы `click()` и `send_keys()` имитируют щелчки мышью и нажатия клавиш соответственно.
17. Вызов метода `submit()` для любого элемента формы инициирует ее отправку.
18. Щелчки на этих кнопках браузера имитируются методами `forward()`, `back()` и `refresh()` объекта `WebDriver`.

Глава 12

1. Функция `openpyxl.load_workbook()` возвращает объект `Workbook`.
2. Функция `get_sheet_names()` возвращает объект `Worksheet`.
3. Для этого следует вызвать метод `wb.get_sheet_by_name('Sheet1')`.
4. Для этого следует вызвать метод `wb.get_active_sheet()`.
5. `sheet['C5'].value` или `sheet.cell(row=5, column=3).value`.
6. `sheet['C5'] = 'Hello'` или `sheet.cell(row=5, column=3).value = 'Hello'`.
7. `cell.row` и `cell.column`.
8. Они возвращают целочисленный номер последнего столбца и последней строки листа, содержащих значения.
9. `openpyxl.cell.column_index_from_string('M')`
10. `openpyxl.cell.get_column_letter(14)`
11. `sheet['A1':'F1']`
12. `wb.save('example.xlsx')`
13. Формула задается для ячейки подобно любому значению. Для этого следует установить в качестве значения атрибута `value` строку с текстом формулы. Не забывайте о том, что формула начинается со знака =.
14. Для этого следует передать значение `True` в качестве именованного аргумента `data_only` при вызове функции `load_workbook()`.
15. `sheet.row_dimensions[5].height = 100`
16. `sheet.column_dimensions['C'].hidden = True`
17. OpenPyXL 2.0.5 не загружает закрепленные области, заголовки, изображения и диаграммы.
18. Закрепленные области — это строки и столбцы, которые всегда видны на экране. Их полезно использовать в качестве заголовков.
19. `openpyxl.charts.Reference()`, `openpyxl.charts.Series()`, `openpyxl.charts.BarChart()`, `chartObj.append(seriesObj)` и `add_chart()`.

Глава 13

1. Объект `File`, возвращенный функцией `open()`
2. В режиме чтения двоичных данных (`'rb'`) для `PdfFileReader()` и в режиме записи двоичных данных (`'wb'`) для `PdfFileWriter()`.
3. Вызов `getPage(4)` возвратит объект `Page` для страницы 5, поскольку первой страницей является страница `0`.
4. Целочисленное значение количества страниц в PDF-документе хранится в переменной `numPages` объекта `PdfFileReader`.
5. Вызвать функцию `decrypt('swordfish')`.
6. Методы `rotateClockwise()` и `rotateCounterClockwise()`. Величина угла поворота в градусах передается в виде целочисленного аргумента.
7. `docx.Document('demo.docx')`
8. Документ содержит множество абзацев, представляемых объектами `Paragraph`. Абзац начинается с новой строки и состоит из нескольких участков с различными стилями форматирования, представляемых объектами `Run`. Эти участки текста представляют собой группы символов, прилегающие одна к другой в пределах абзаца.
9. Использовать выражение `doc.paragraphs`.
10. Эти переменные имеет объект `Run` (объект `Paragraph` их не имеет).
11. Установка для переменной `bold` значения `True` приводит к тому, что к объекту `Run` всегда будет применяться выделение полужирным шрифтом, а значения `False` — к тому, что оно никогда не будет к нему применяться, какой бы ни была настройка его стиля. При значении `None` к объекту `Run` выделение полужирным шрифтом применяется в соответствии с установленным для него стилем.
12. Для этого следует вызвать функцию `docx.Document()`.
13. `doc.add_paragraph('Hello there!')`
14. Целочисленные значения `0`, `1`, `2`, `3` и `4`.

Глава 14

1. В электронных таблицах могут храниться значения с типами данных, отличными от строкового; ячейки могут иметь различные настройки шрифтов, размера или цвета; ширина и высота ячеек могут изменяться; смежные ячейки могут объединяться; в электронные таблицы можно внедрять изображения и диаграммы.
2. Этим функциям передается объект `File`, возвращенный вызовом функции `open()`.

3. Объекты `File` необходимо открывать в режиме чтения двоичных данных (`'rb'`) для объектов `Reader` и в режиме записи двоичных данных (`'wb'`) для объектов `Writer`.
4. Метод `writerow()`.
5. Аргумент `delimiter` заменяет строку, используемую в качестве разделителя ячеек в строке таблицы. Аргумент `lineterminator` заменяет строку, используемую в качестве разделителя строк таблицы.
6. `json.loads()`
7. `json.dumps()`

Глава 15

1. Начало отсчета времени, используемое многими программами, предназначенными для работы со временем и датами. Им считается 0 часов 1 января 1970 года, UTC.
2. `time.time()`
3. `time.sleep(5)`
4. Она возвращает целое число, ближайшее к переданному аргументу; например `round(2.4)` вернет значение `2`.
5. Объект `datetime` представляет определенный момент времени. Объект `timedelta` представляет промежуток времени.
6. Код:

```
threadObj = threading.Thread(target=spam)
threadObj.start()
```

7. Следует убедиться в том, что код, выполняющийся в одном потоке, не читает и не записывает те же переменные в коде, выполняющимся в другом потоке.
8. `subprocess.Popen('c:\\Windows\\System32\\calc.exe')`

Глава 16

1. SMTP и IMAP соответственно.
2. `smtplib.SMTP()`, `smtpObj.ehlo()`, `smptObj.starttls()` и `smtpObj.login()`.
3. `imapclient.IMAPClient()` и `imapObj.login()`.
4. Список строк, содержащих ключевые слова IMAP, такие как `'BEFORE <date>'`, `'FROM<string>'` и `'SEEN'`.
5. Следует присвоить переменной `imaplib._MAXLINE` большое целочисленное значение, такое как `10000000`.

6. Чтение загруженных сообщений электронной почты обеспечивает модуль `pyzmail`.
7. Потребуется SID учетной записи Twilio, аутентификационный маркер и телефонный номер пользователя Twilio.

Глава 17

1. RGBA-значение — это кортеж из четырех целых чисел, каждое из которых может иметь значение в пределах от `0` до `255`. Эти четыре числа выражают количественные доли красной, зеленой и синей составляющих, а также альфа-канала (прозрачности) в цвете.
2. Вызов `ImageColor.getcolor('CornflowerBlue', 'RGBA')` вернет для этого цвета RGBA-значение `(100, 149, 237, 255)`.
3. Кортеж прямоугольника — это кортеж из четырех целых чисел: *x*-координата левой стороны прямоугольника, *y*-координата верхней стороны прямоугольника, ширина и высота прямоугольника соответственно.
4. `Image.open('zophie.png')`
5. `imageObj.size` — это кортеж из двух целых чисел: ширины и высоты изображения.
6. `imageObj.crop((0, 50, 50, 50))`. Заметьте, что методу `crop()` передается кортеж прямоугольника, а не четыре отдельных целочисленных аргумента.
7. Для этого следует вызвать метод `imageObj.save('new_filename.png')` объекта `Image`.
8. Код для рисования изображений содержится в модуле `ImageDraw`.
9. Методы, предназначенные для рисования объектов, такие как `point()`, `line()` или `rectangle()`, имеют объекты `ImageDraw`. Эти объекты возвращаются функцией `ImageDraw.Draw()`, которой передается объект `Image`.

Глава 18

1. Для этого следует переместить указатель мыши в верхний левый угол экрана, т.е. в позицию с координатами `(0, 0)`.
2. Разрешение экрана в виде кортежа из двух целых чисел, представляющих ширину и высоту экрана, возвращает функция `pyautogui.size()`.
3. Текущие координаты указателя мыши в виде кортежа из двух целых чисел, представляющих *x*- и *y*-координату, возвращает функция `pyautogui.position()`.

4. Функция `moveTo()` перемещает указатель мыши в позицию с заданными абсолютными координатами на экране, тогда как функция `moveRel()` перемещает указатель мыши относительно его текущей позиции.
5. `pyautogui.dragTo()` и `pyautogui.dragRel()`.
6. `pyautogui.typewrite('Hello world!')`
7. Для этого следует либо передать список строк с обозначениями клавиш (таких, как `'left'`) функции `pyautogui.typewrite()`, либо передать строку с обозначением клавиши функции `pyautogui.press()`.
8. `pyautogui.screenshot('screenshot.png')`
9. `pyautogui.PAUSE = 2`

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ