

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. Результатирующий 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 доступна 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.

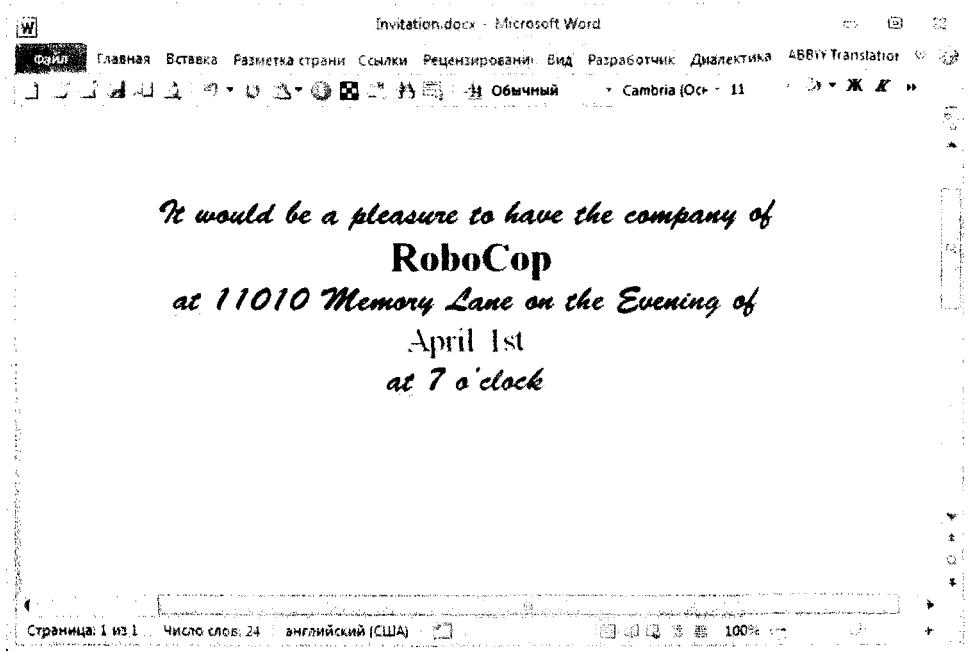
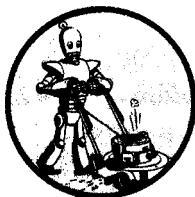


Рис. 13.11. Документ Word, полученный с помощью сценария, генерирующего персонализированные приглашения

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 представляет 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 чём 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 существует 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',
❸ '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	B	C	D	E	F	G
1	42	2	3	4	5	6	7
2		4	6	8	10	12	14
3	3	6	9	12	15	18	21
4		8	12	16	20	24	28
5	4	8	12	16	20	24	28
6		3	6	9	12	15	18
7	5	10	15	20	25	30	35
8		4	8	12	16	20	24
9	2	4	6	8	10	12	14
10		3	6	9	12	15	18

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 межстрочный 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 `delimiter='\\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
```

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

❶    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 BeautifulSoup.)
- 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, булевые 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,
    &lt; '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 цюлей 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.

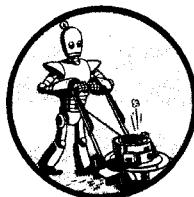
            # Запись списка rowData в CSV-файл.

    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 программы.

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
```

```
❶ startTime = time.time()
❷ 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

❸ >>> `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 высокочувствительном 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 нажмайте клавишу
    <Enter>. Для выхода из программы нажмите клавиши <Ctrl+C>.')
input()                                # нажатие клавиши <Enter> начинает
                                         # отсчет
print('Отсчет начал.')
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:
❸         input()
❹         lapTime = round(time.time() - lastTime, 2)
❺         totalTime = round(time.time() - startTime, 2)
❻         print('Замер #{}: {} ({})'
❾             .format(lapNum, totalTime, lapTime), end='')
❿         lapNum += 1
❬         lastTime = time.time() # переустановить время
                                # последнего замера
❭ 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` ❷, 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 ❸, 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()` ❹. Суммарное истекшее время вычисляется путем вычитания общего начального времени запуска хронометра, `startTime`, из текущего времени ❺.

Поскольку все результаты временных расчетов содержат чрезмерно большое количество цифр после десятичной точки (например, 4.766272783279419), мы округляем их до двух цифр с помощью функции `round()` (❻ и ❼).

В строке ❼ 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=''`, устраниющий появление двойного междусторочного 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 году, 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)
❸ >>> oct31_2015 = datetime.datetime(2015, 10, 31, 0, 0, 0)
❹ >>> halloween2015 == oct31_2015
True
❺ >>> halloween2015 > newyears2016
False
❻ >>> newyears2016 > halloween2015
True
❼ >>> 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 `halloween2015` 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 временных 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.timedelta()` 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) ②. 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

Временные 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
<code>%Y</code>	Год с указанием столетия: '2014'
<code>%Y</code>	Год без указания столетия: от '00' до '99' (1970–2069)
<code>%m</code>	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 `strrrftime()` 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 `strrrftime()` 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 `strrrftime()` 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*.)

Ведите 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',
❷   '%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. Стока 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 выполнятьсь 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 воспроизвести, 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`, убедитесь 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
① os.makedirs('xkcd', exist_ok=True) # сохранить комиксы в
                                    # папке ./xkcd

② def downloadXkcd(startComic, endComic):
③     for urlNumber in range(startComic, endComic):
        # Загрузка страницы.
        print('Загрузка страницы http://xkcd.com/%s...' %
        ④     (urlNumber))
        res = requests.get('http://xkcd.com/%s' %
        ⑤     (urlNumber))
        res.raise_for_status()

⑥     soup = bs4.BeautifulSoup(res.text)

        # Поиск URL-адреса изображения комикса.
```

```

❶ comicElem = soup.select('#comic img')
if comicElem == []:
    print('Не удается найти изображение комикса.')
else:
    comicUrl = comicElem[0].get('src')
    # Загрузка изображения.
    print('Загрузка изображения %s...' % (comicUrl))
    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 ❷ и загружаем само изображение ❸. 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

вызыва 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.

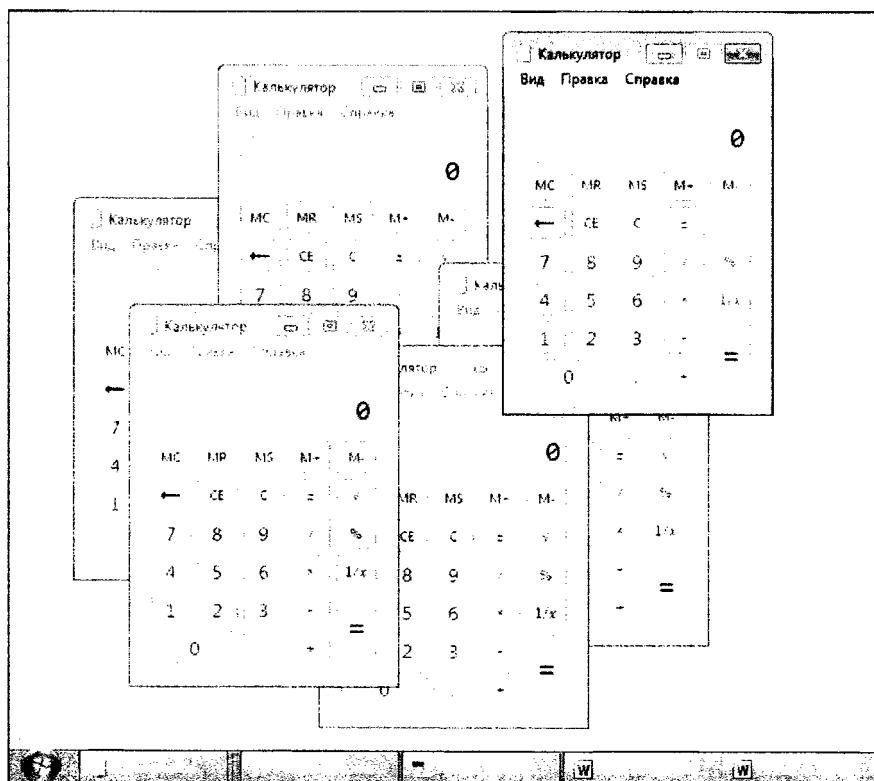


Рис. 15.1. Шесть одновременно выполняющихся процессов одного и того же приложения калькулятора

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 0x000000003055A58>
```

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 =
❷   subprocess.Popen('c:\\Windows\\System32\\calc.exe')
❸ >>> calcProc.poll() == None
True
❹ >>> 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 имени 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 *tarIt.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 представлена, 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 высокогорневом 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 `<Ctrl+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 задач, 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 `sram()`. 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 `ruperclip`, 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.

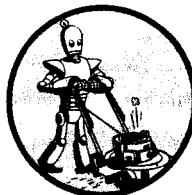
Загрузка веб-комиксов по расписанию

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 многоократного 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 ko10sm23097611pb.52 - gsmtp')
```

В следующем разделе мы тщательно проанализируем каждый шаг, представляя 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	<i>smtp.gmail.com</i>
Outlook.com/Hotmail.com	<i>smtp-mail.outlook.com</i>
Yahoo Mail	<i>smtp.mail.yahoo.com</i>
AT&T	<i>smpt.mail.att.net</i> (порт 465)
Comcast	<i>smtp.comcast.net</i>
Verizon	<i>smtp.verizon.net</i> (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 *smtplib.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 позволяют выполнять процедуру входа и отправлять сообщения. Неудачный вызов `smtplib.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 SMTP, “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 `rutzmail`. Полную документацию по модулю RutzMail можно найти по адресу <http://www.magiksys.net/rutzmail/>.

Установите модули `imapclient` и `rutzmail` 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 `rutzmail`. 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 =
<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 <code>imaplib</code> 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, теле 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 "Cc" ("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 <code>fetch()</code> (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 <code>delete_messages()</code> , снабжаются флагом \Deleted, но не удаляются безвозвратно до тех пор, пока не будет вызван метод <code>expunge()</code> (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).

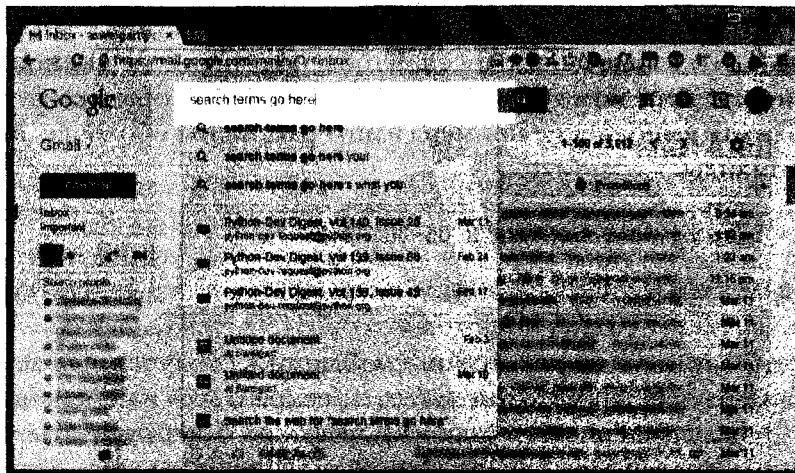


Рис. 16.1. Поле поиска в верхней части веб-страницы Gmail

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, предназначенному 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 `PyzMessagе`, 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 `PyzMessagе` сообщения, вызвав функцию `pyzmail.PyzMessage.factory()` и передав ей раздел `'BODY[]'` необработанного сообщения. Сохраните результат в переменной `message`. Теперь в переменной `message` содержится объект `PyzMessagе`, 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().
❷ decode(message.text_part.charset)
❸ 'So long, and thanks for all the fish!\r\n\r\n-Al\r\n'
❹ >>> message.html_part != None
True
❺ >>> message.html_part.get_payload().
❻ decode(message.html_part.charset)
'<div dir="ltr"><div>So long, and thanks for all the
fish!<br><br>-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 ❸.

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 выполнятьсь 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 рассыпать 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 высокуюровневом 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.

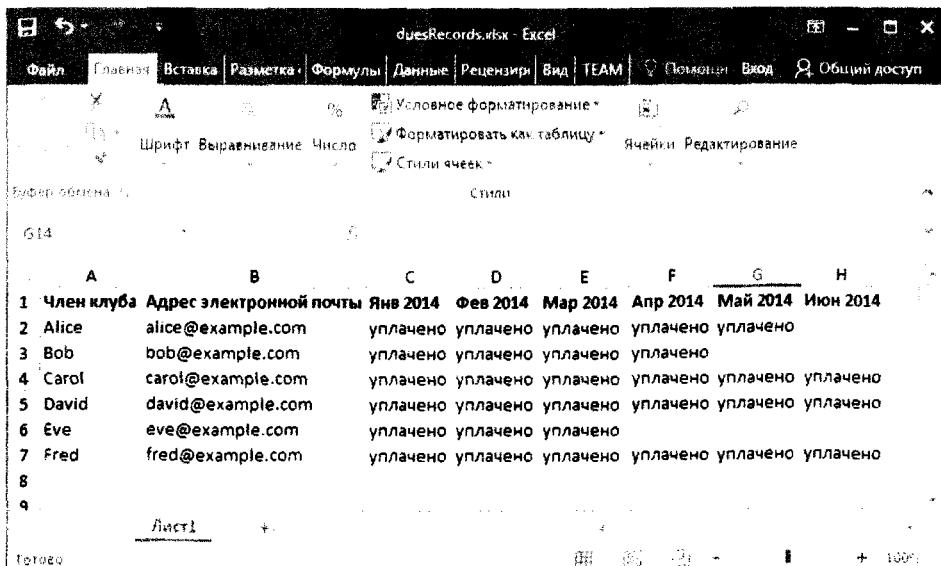
Это означает, что ваш код должен выполнять следующие действия:

- 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/>.



The screenshot shows a Microsoft Excel spreadsheet titled "duesRecords.xlsx - Excel". The table has columns labeled from A to H, corresponding to months from January to June. Row 1 contains the header "Член клуба" (Member) and "Адрес электронной почты" (Email Address). Rows 2 through 7 contain data for members Alice, Bob, Carol, David, Eve, and Fred, respectively. Each row has five entries under the month columns, all of which are filled with the word "уплачено" (paid).

	А	В	С	Д	Е	Ф	Г	Н
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	уплачено
3	Bob	bob@example.com	уплачено	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	уплачено	уплачено	
7	Fred	fred@example.com	уплачено	уплачено	уплачено	уплачено	уплачено	уплачено
8								
9								
Лист1								
100%								

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 - Рассыпает 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 **❶**. Затем мы получаем Лист 1 и сохраняем результирующий объект `Worksheet` в переменной **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 - Рассыпает 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, используя в нем свой реальный адрес электронной почты и информацию о постовом провайдере.

```
#! python3
# sendDuesReminders.py - Рассыпает 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 - Рассыпает 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.

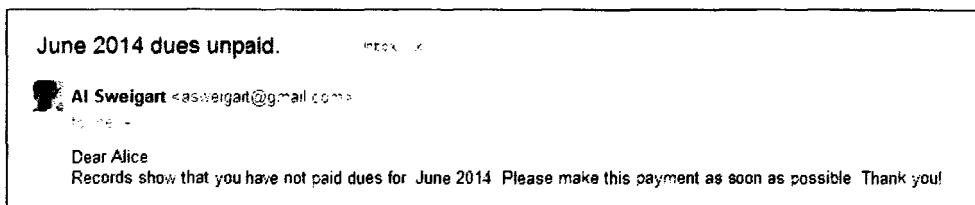


Рис. 16.3. Автоматическая отправка сообщений электронной почты с помощью программы `sendDuesReminders.py`

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 подписатья 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 = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
>>> authToken = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
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 = 'ACxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
authToken = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
myNumber = '+15559998888'
twilioNumber = '+15552225678'
from twilio.rest import TwilioRestClient
❶ def textmyself(message):
❷     twilioCli = TwilioRestClient(accountSID, authToken)
```

```
❶ 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 `ruzmil` 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, выбранный случайнym 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, подтверждающей 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.

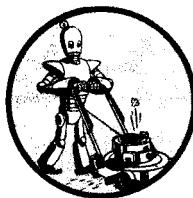
В программе qBittorrent (равно как и в других приложениях BitTorrent) предусмотрена возможность автоматического выхода по окончании

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). Стока 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.

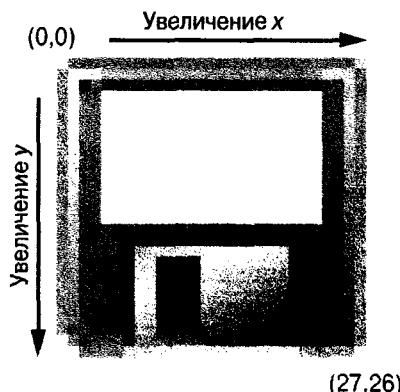


Рис. 17.1. Координаты x и y для области изображения размером 27×26 пикселей, представляющего некое древнее устройство для хранения данных

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.

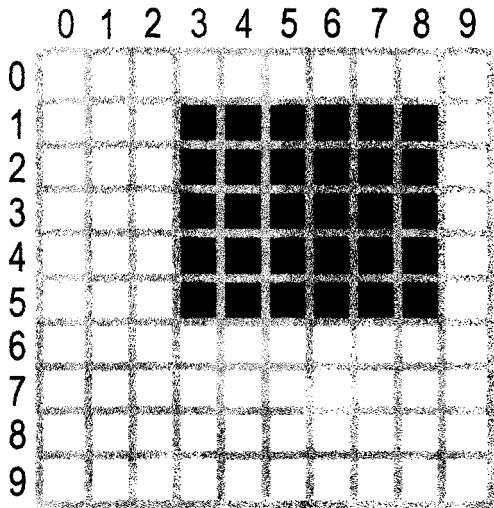


Рис. 17.2. Область, представляемая кортежем
прямоугольника $(3, 1, 9, 6)$

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 (весъ-
ма 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.

- Стока `'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.

Ведите 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. Ведите в интерактивной оболочке следующие команды.

```
>>> catIm = Image.open('zophie.png')
>>> catCopyIm = catIm.copy()
```

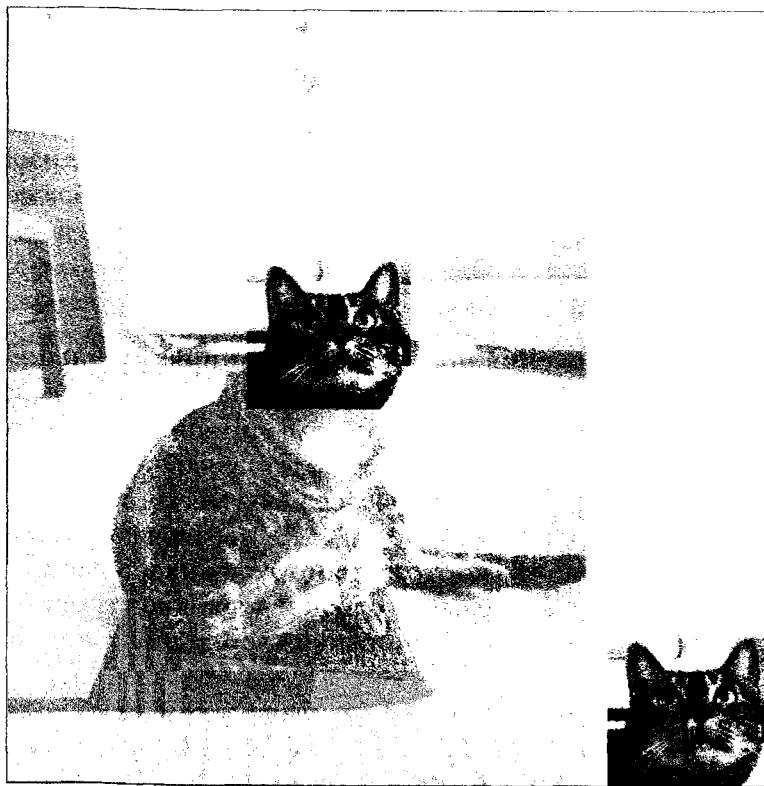


Рис. 17.4. Новое изображение будет лишь частью исходного

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 Софи. 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 Софи была скопирована дважды

Примечание

Пусть названия методов `copy()` и `paste()` модуля `Pillow` не вводят вас в заблуждение: их работа никаким 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 Софи 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`.

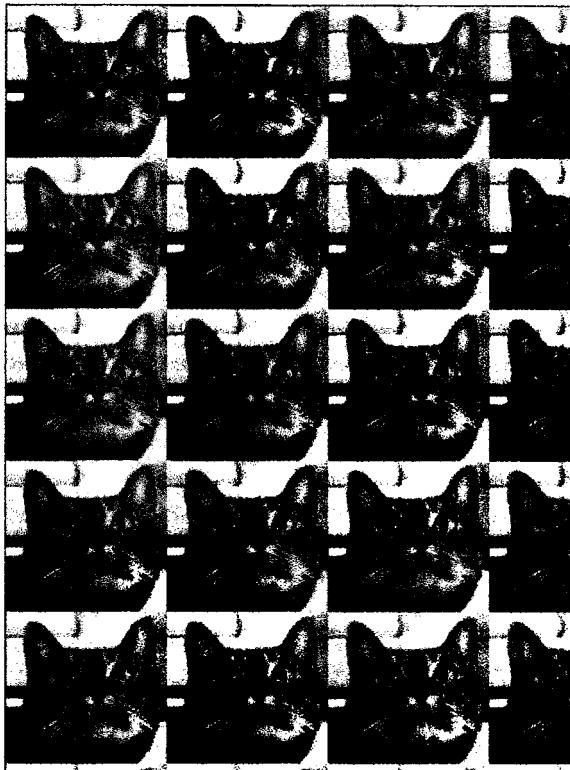


Рис. 17.6. Использование вложенных циклов и метода `paste()` для создания дубликатов изображения головы кошки

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 `rotated6.png` (рис. 17.8, слева). Во второй строке изображение поворачивается на 6° при значении аргумента `expand`, равном `True`, и сохраняется в файле `rotated6_expanded.png` (рис. 17.8, справа).

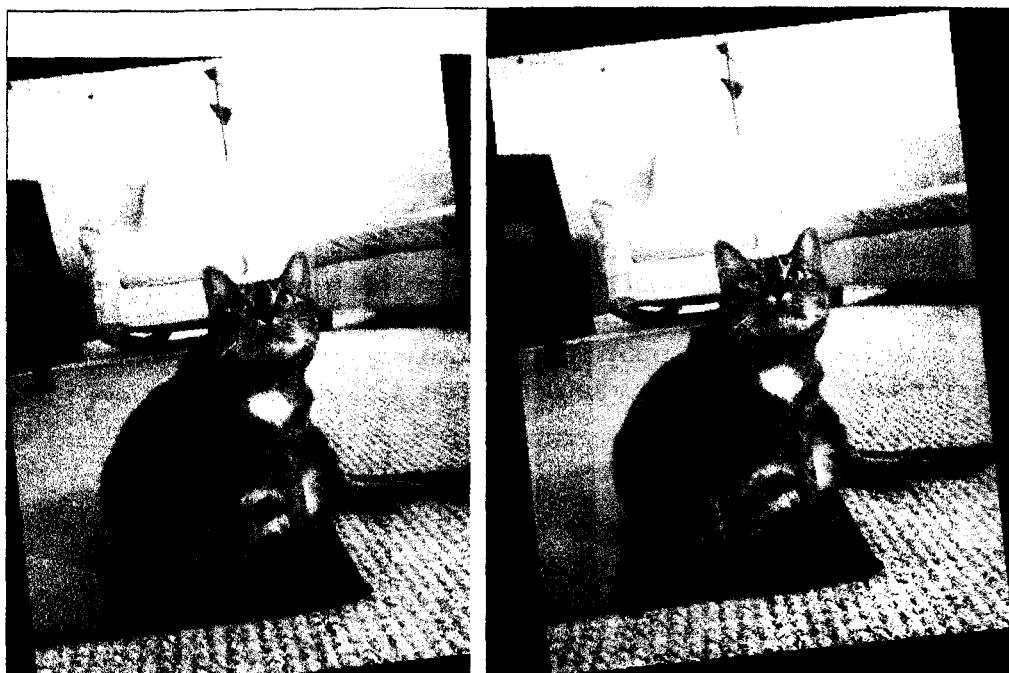


Рис. 17.8. Изображение, повернутое на 6° в обычном режиме (слева) и в режиме `expand=True` (справа)

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 сохраним результатирующее 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))
❺ >>> from PIL import ImageColor
❻ >>> for x in range(100):
           for y in range(50, 100):
❾             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 ②. Для назначения цвета пикселям этого изображения мы можем использовать вложенные циклы `for`, перебирая все пиксели в верхней половине изображения ③ и вызывая для каждого из них метод `putpixel()`. ④. 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 ⑤ с передачей методу `putpixel()` значения, возвращаемого функцией `ImageColor.getColor()` ⑥, 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 пикселью за один раз не очень удобно. Если вам надо рисовать фигуры, используйте функции `ImageDraw`, о которых речь пойдет далее.

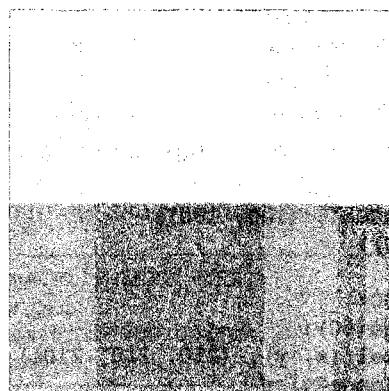


Рис. 17.10. Изображение `putPixel.png`

Проект: добавление логотипа

Предположим, вам предстоит рутинная работа по изменению размеров тысяч изображений и добавления в угол 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.

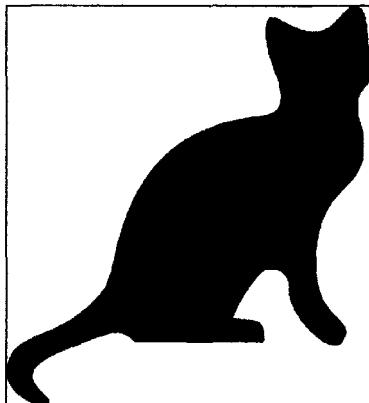


Рис. 17.11. Логотип, который будет добавляться в изображение

Вот что должна делать данная программа при высокогорневом 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` ❶.

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 \
❸         filename.endswith('.jpg')) or filename == LOGO_FILENAME:
❹         continue # пропустить файлы, не являющиеся файлами
# изображений, и файл самого логотипа

❺     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 ②.

Установив в переменных `width` и `height` новые значения размеров, перейдите их методу `resize()` и сохраните возвращенный объект `Image` в переменной `im` ③.

Шаг 4. Добавление логотипа и сохранение изменений

Независимо от того, изменились ли размеры изображения, логотип должен помещаться в нижний правый угол изображения. В какую именно позицию он должен вставляться, определяются 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.

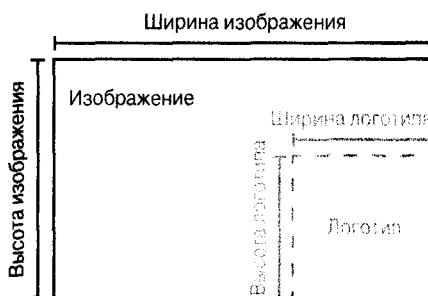


Рис. 17.12. Координаты левого и верхнего краев логотипа при его помещении в нижний правый угол изображения определяются соответственно разностью значений ширины изображения и ширины логотипа и высоты изображения и высоты логотипа

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 информирует 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 временные 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.

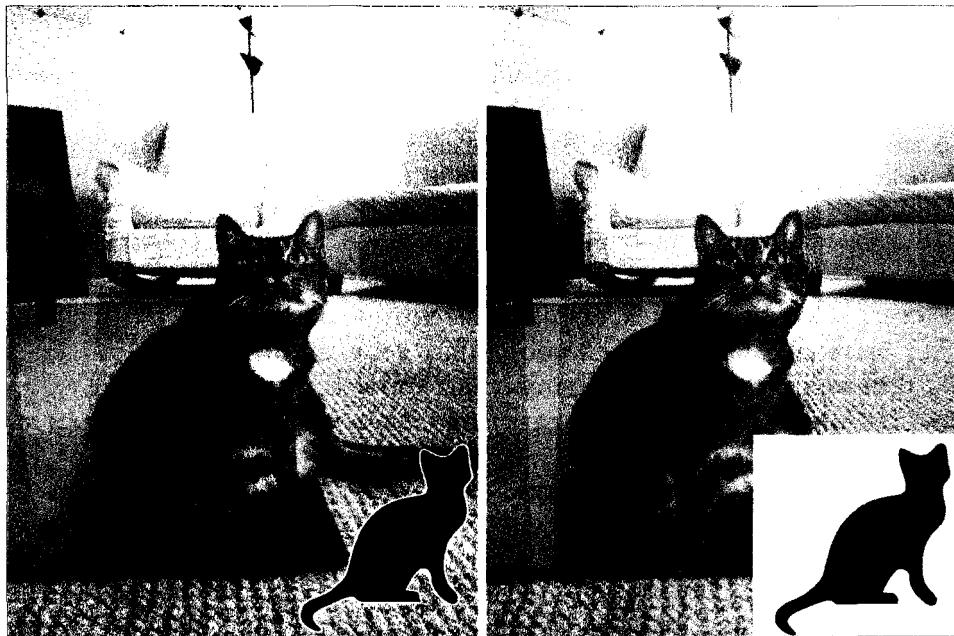


Рис. 17.13. Изображение zophie.png с измененными размерами и добавленным логотипом (слева). Если вы забудете о третьем аргументе, то прозрачные пиксели логотипа скопируются в виде сплошных белых пикселей (справа).

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

Ведите 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. Результатирующее 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>.

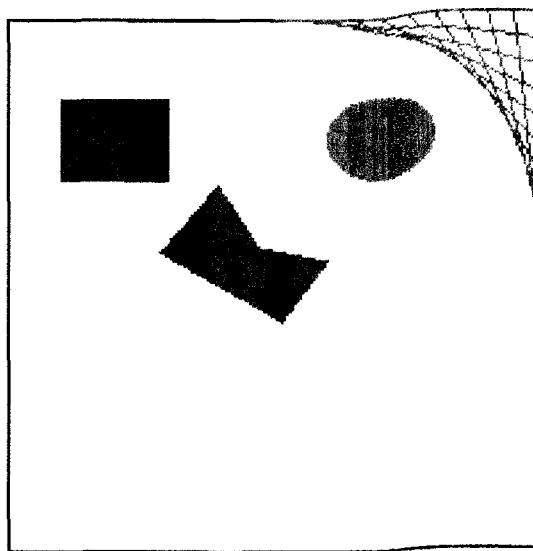


Рис. 17.14. Результатирующее изображение drawing.png

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 будут 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.

Ведите 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 ②. Далее мы используем метод `text()` для прорисовки текста *Hello* пурпурного цвета с началом в позиции с координатами (20, 150) ③. 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 ④. Объект `Font`, возвращенный вызовом `ImageFont.truetype()`, сохраняется в переменной `arialFont`, и эта переменная передается методу `text()` в его последнем именованном аргументе. Вызов метода `text()` ⑤ 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 расширите 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.

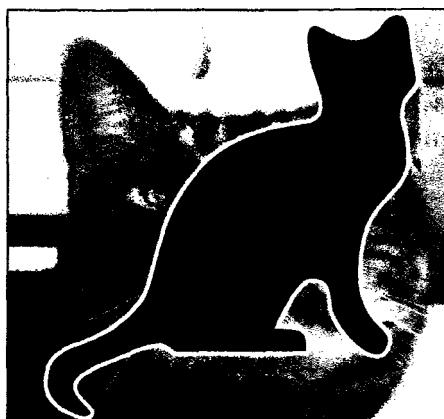


Рис. 17.16. Когда размер основного изображения лишь немножко больше размера логотипа, вся композиция выглядит уродливо

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.

```
#!/usr/bin/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.

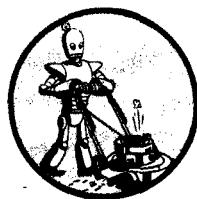
Персональные приглашения

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 посыпать 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 А. 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 Мики 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>**. Выходя 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.

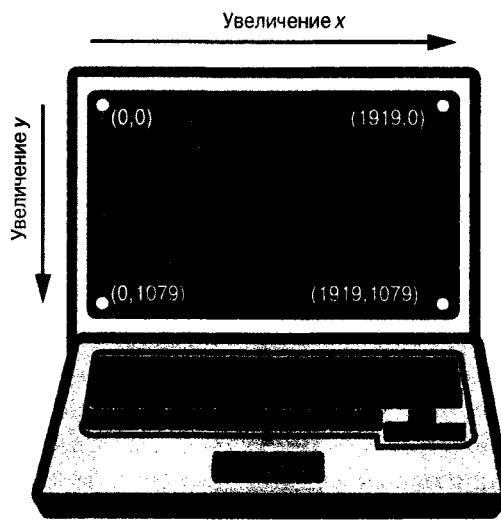


Рис. 18.1. Система координат на компьютерном экране разрешением 1920×1080

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 высокоДуровневом 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. Импортowanie 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 работаете 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.
① except KeyboardInterrupt:
②     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)`.

Ведите 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 же 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 кнопками.

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()`: `х-координату/горизонтальное смещение, у-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
    ❼ 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.

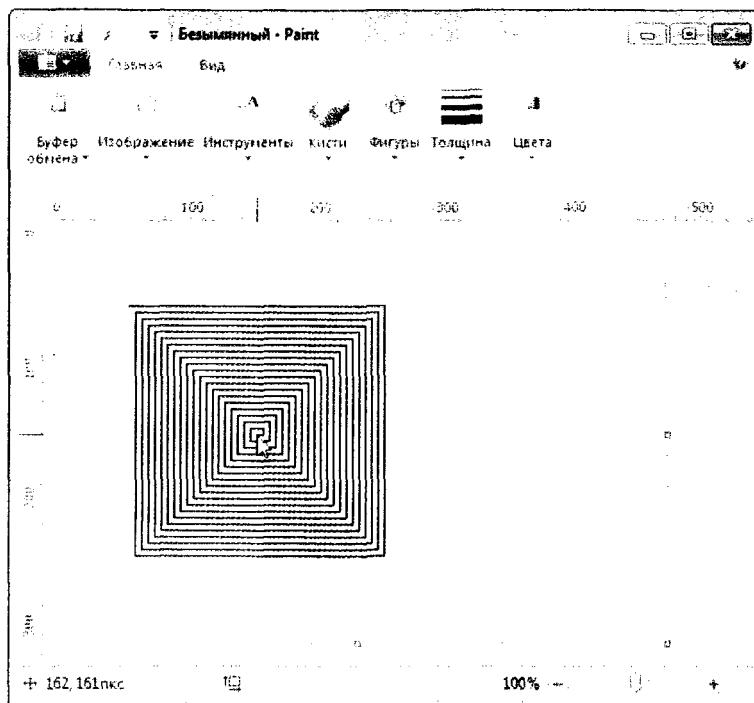


Рис. 18.2. Результат работы примера программы, в которой используется функция `pyautogui.dragRel()`

Вы могли 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 программы GUI-автоматизации не должны вслепую выполнять щелчки и нажимать виртуальные клавиши. 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`. Этую 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 пикселям, 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 посыпать 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!

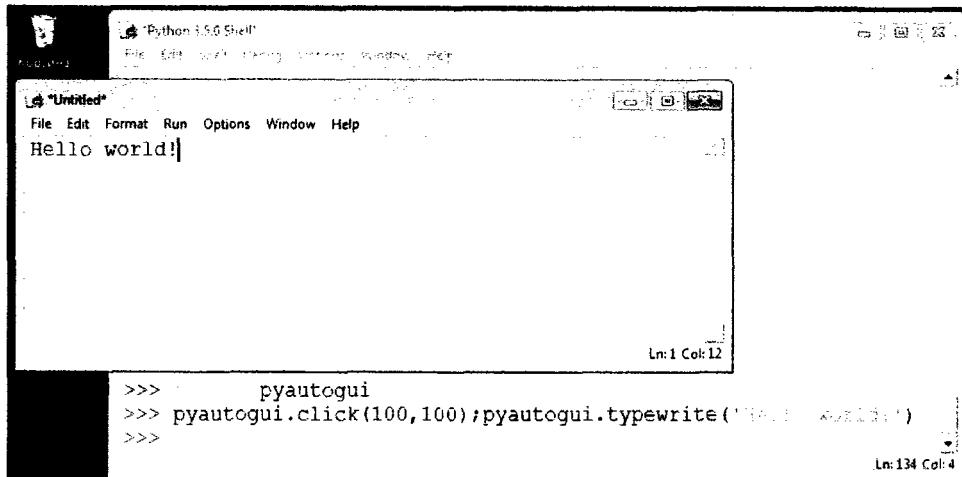


Рис. 18.3. Использование средств PyAutoGUI для выполнения щелчка в окне файлового редактора и ввода в нем текста `Hello world!`

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, выраждающее 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 клавиши `<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 ``, затем – двум нажатиям клавиши `<↔>` и наконец – нажатиям клавищ `<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`. Стока `'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
<code>'a', 'b', 'c', 'A', 'B', 'C', '1', '2', '3', '!', '@', '#'</code> и т.д.	Клавиши одиночных символов
<code>'enter'</code> (а также <code>'return'</code> или <code>'\n'</code>)	Клавиша <code><Enter></code>
<code>'esc'</code>	Клавиша <code><Esc></code>
<code>'shiftleft', 'shiftright'</code>	Левая и правая клавиши <code><Shift></code>
<code>'altright', 'altright'</code>	Левая и правая клавиши <code><Alt></code>
<code>'ctrlleft', 'ctrlright'</code>	Левая и правая клавиши <code><Ctrl></code>
<code>'tab'</code> (или <code>'\t'</code>)	Клавиша <code><Tab></code>
<code>'backspace', 'delete'</code>	Клавиши <code><Backspace></code> и <code><Delete></code>
<code>'pageup', 'pagedown'</code>	Клавиши <code><Page Up></code> и <code><Page Down></code>
<code>'home', 'end'</code>	Клавиши <code><Home></code> и <code><End></code>
<code>'up', 'down', 'left', 'right'</code>	Клавиши <code><↑></code> , <code><↓></code> , <code><↔></code> и <code><→></code>
<code>'f1', 'f2', 'f3' и т.д.</code>	Клавиши от <code><F1></code> до <code><F12></code>
<code>'volumemute', 'volumedown', 'volumeup'</code>	Клавиши отключения, уменьшения и увеличения звука (на некоторых клавиатурах эти клавиши отсутствуют, но операционная система все равно будет распознавать эти имитированные нажатия клавиш)
<code>'pause'</code>	Клавиша <code><Pause></code>

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 отпусканьй 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 отпусканьем клавиши <Shift>. Для ввода строки в текстовом поле лучше подойдет функция `typewrite()`. Но в случае приложений, нуждающихся в одноклавищных 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, Alt-3 comments out a line* (в IDLE нажатие клавиши `<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.

The screenshot shows a Google Form titled "Generic Form". The URL in the address bar is <http://docs.google.com/forms/d/1JGgXWzvDyfjwzqCgkVQYBzgjPjyfLcA/edit>. The form contains the following fields:

- A text input field labeled "Name".
- A text input field labeled "Greatest Fear(s)".
- A question "What is the source of your wizard powers?" followed by a dropdown menu with options: "RoboCop was the greatest action movie of the 1980s", "I'm a wizard, Harry", and "I'm a wizard, Harry".
- A text input field labeled "Additional Comments".

Рис. 18.4. Форма, используемая в этом проекте

Вот что должна делать данная программа при высоконивневом 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 *Robocop* 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.

```
#!/usr/bin/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> <<<')
    ①   time.sleep(5)

    # Дождаться окончания загрузки страницы формы.
    ②   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 ①, 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 ②, что будет свидетельствовать о завершении загрузки формы. Вспомните о том, что информация о координатах x и y

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 больших 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--

❶ # Заполнение поля Source of Wizard Powers.
❷ if person['source'] == 'wand':
    pyautogui.typewrite(['down', '\t'])
❸ elif person['source'] == 'amulet':
    pyautogui.typewrite(['down', 'down', '\t'])
❹ elif person['source'] == 'crystal ball':
    pyautogui.typewrite(['down', 'down', 'down', '\t'])
❺ elif person['source'] == 'money':
    pyautogui.typewrite(['down', 'down', 'down', 'down',
        '\t'])

❻ # Заполнение поля RoboCop.
❼ if person['robocop'] == 1:
    pyautogui.typewrite([' ', '\t'])
❽ elif person['robocop'] == 2:
    pyautogui.typewrite(['right', '\t'])
❾ elif person['robocop'] == 3:
    pyautogui.typewrite(['right', 'right', '\t'])
❿ 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.

```
#!/usr/bin/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 наблюдать, 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 открываящая 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

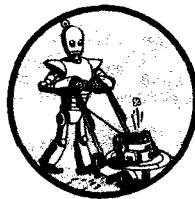
Возможно, вам стоит создать тестовые фиктивные учетные записи, чтобы вы случайно не забросали спамом своих друзей, пока пишете эту программу.

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.)

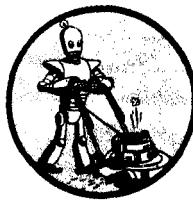
- `pip install send2trash`
- `pip install requests`
- `pip install beautifulsoup4`
- `pip install selenium`
- `pip install openpyxl`
- `pip install PyPDF2`
- `pip install python-docx` (*именно python-docx, а не docx*)
- `pip install imapclient`
- `pip install pyzmail`
- `pip install twilio`
- `pip install 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 .ру 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).

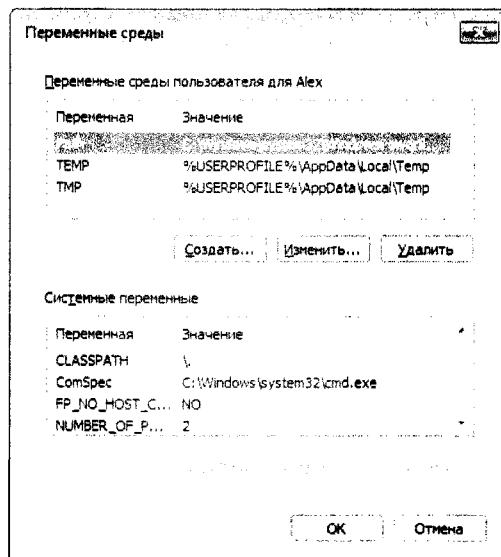


Рис. Б.1. Окно *Переменные среды* в Windows

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*-файл 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 сообщает операционной системе, где искать исполняемый файл интерпретатора Python.

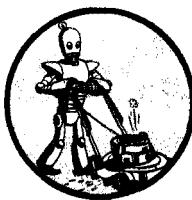
Запуск программ на Python с отключенными утверждениями

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. Стока – `'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 ошибки, 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 списоковый 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 списоковые 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 совпадшему тексту, группа 1 – тексту, совпадшему с выражением в первой паре круглых скобок, группа 2 – тексту, совпадшему 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-zA-Z]
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}(,\d{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. Стока '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` перемещают 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 месток.
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 совпадший элемент в виде объекта `WebElement`. Методы `find_elements_*` возвращают список всех совпадших 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, предназначеными 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 у-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

A

API 413

B

Beautiful Soup 314

C

CMYK 493

CSV 403

G

Google Maps 300

GUI-автоматизация 525

H

HTML 308

I

IDLE 34

IMAP 463

сервер 464, 475

J

JSON 403, 413

P

PDF-документ 373

дешифрование 376

извлечение текста 374

копирование страниц 377

наложение страниц 380

поворот страниц 379

создание 377

шифрование 382

PEP8 50

Pillow 491, 495

R

RGBA 491

S

Selenium 328

SMTP 457

сервер 459

T

TLS 461

Twilio 480

U

Unicode 306

URL-адрес 300

UTC 424

A

Абсолютный путь 226

Аварийное завершение 42, 474

Активный лист 338

Аргумент функции 53, 96

Атрибут 309

B

Бесконечный цикл 81, 83

Бинарный оператор 65

Блок 68

Булево значение 62

Булев оператор 65

V

Веб-скрапинг 299, 414

Вещественное число 45

Виртуальная клавиатура 542

Возвращаемое значение 97

Временная метка Unix 424, 430

Встроенная функция 89

Выражение 42

Г

Глобальная область видимости 101

Горячие клавиши 545

Групповое присваивание 125, 149

Групповой символ 208

Д

Двоичный (бинарный) файл 232

Дерево каталогов 259

Дескриптор HTML 308

Диаграмма 366

Документ Word 386

добавление заголовков 396

добавление изображений 397

запись 394

извлечение текста 389

использование стилей 390

чтение 388

Домен верхнего уровня 216

Ж

Жадный поиск 203, 209
Журнал 288

З

Значение 42

И

Изображение 491
обрезка 499
поворот 504
размеры 504
рисование 515
Именованный аргумент 100
Имя файла 223
Индекс 116
 отрицательный 118
Инициализация 48
Интерактивная оболочка 34, 41
Интерактивная среда разработки 34
Интерпретатор Python 34
Исключение 108, 276
Исходный код 27
Итерация 79

К

Кавычки 166
Канал 199
Капитель 393
Каталог 223
Клавиатура 542
Ключ 145
Код выхода 446
Командная строка 301
Комментарии 53
Конкатенация 46
Координаты 494
Копирование файлов 254
Корневая папка 223
Кортеж 135

Л

Локальная область видимости 101

М

Магическая строка 561
Метод 126

append() 127
center() 177
endswith() 174
findall() 204
get() 150
index() 126
insert() 127
islower() 171
isupper() 171
items() 148
join() 175
keys() 148
ljust() 176
lower() 170
remove() 128
rjust() 176
search() 467
sendmail() 462
setdefault() 150
sort() 129
split() 175
startswith() 174
sub() 211
upper() 170
values() 148
wait() 447
Многопоточность 437
Многострочный блок 167
Модуль
 CSV 404
 datetime 430
 ImageDraw 515
 imapclient 463
 JSON 414
 logging 283
 openpyxl 338
 os.path 227
 pyautogui 526, 538, 546
 PyPDF2 373
 pyperclip 179, 214
 python-docx 386
 pyzmail 463
 re 195
 Requests 303
 send2trash 258
 shelve 237

- shutil 254
threading 438
time 423
webbrowser 300
zipfile 261
- Н**
Нежадный поиск 203, 209
- О**
Обратная трассировка стека вызовов 278
- Объект
 Datetime 436
 Font 359
 Reader 405
 Regex 195
 timedelta 436
 Writer 406
- Округление чисел 426
- Оператор 42
 бинарный 65
 булев 65
 присваивания 47
 комбинированный 125
 сравнения 63
 унарный 66
- Отладка 29, 288
- Относительный путь 226
- П**
Параллелизм 441
Параметр функции 97
Пароль приложения 461
Передача управления 61
Переименование файлов 255, 264
Переменная 47
 инициализация 48
 правила именования 49
- Перемещение файлов 255
- Перетаскивание 534
- Печать 152
- Пиксель 492
- Планировщик 448
- Последовательность 123
 Коллатца 114
- Поток
 выполнения 438
 управления 61, 68
- Приоритет операций 43
- Прокрутка 536
- Протоколирование 283
 отключение 287
- Пункт 363
- Пустая строка 45
- Путь к файлу 223
- Р**
Рабочий каталог 225
Разделитель 408
Расширение имени файла 223
Регулярные выражения 191, 194
 группы 197
 жадные 203
 нежадные 203
 необязательные группы 200
- Резервное копирование 258, 269
- Репликация строк 46
- Рисование
 текста 518
 фигур 516
- С**
Сжатие файлов 261
Символьный класс 205
 инвертированный 206
- Синтаксический анализ 314
- Словарь 145
- Снимок экрана 538
- Список 115
- Срез 118
- Ссылка 137
 передача 139
- Стандартная библиотека 90
- Стек вызовов 278
- Стиль 387
- Строка 45
 сырая 167, 195
- Строковый литерал 165

Т

Таблица истинности 66

Тег 308

Текстовый файл 232

Тип данных 45

булев 62

вещественный 64

неизменяемый 133

строковый 64

целочисленный 64

Том 224

Точка останова 290, 294

У

Удаление

пробелов 178

файлов 257

Указатель мыши 528

перетаскивание 534

Унарный оператор 66

Управляющая инструкция 61

Уровень критичности сообщений 286

Условие 68

Утверждение 279

отключение 282

Ф

Файловый редактор 51

Факториал 283

Форма 547

Функция 95

copy() 140

deepcopy() 140

float() 56

input() 54

int() 56

len() 54

list() 136

locateOnScreen() 541

pformat() 152

Popen() 445, 447

pprint() 152

print() 53

round() 426

scroll() 536

sleep() 425

str() 55, 56

time() 424

tuple() 136

Ц

Целое число 45

Цикл 76

Циркумфлекс 206

Ч

Число с плавающей точкой 45

Щ

Щелчок мышью 533

Э

Экранирование символов 166

Электронная почта 458

отправка 462

Электронная таблица 337

закрепленные области 365

настройка 362

создание 352

стили 358

формулы 360

чтение 339, 345

Элемент списка 116

Элемент HTML 308

Эра Unix 424