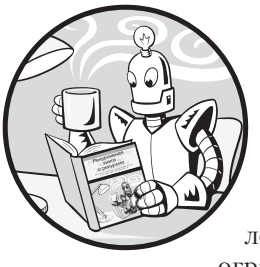
10

Инструмент для поиска файлов



В этой главе вы напишете собственную рекурсивную программу для поиска файлов в соответствии со своими потребностями. Ваш компьютер уже предусматривает несколько команд и приложений для подобных задач, однако зачастую их возможности ограничены. Что, если вам необходимо использовать какой-то специфический критерий поиска, например найти все файлы с четным значением размера в байтах или с именами, содержащими все гласные буквы?

Скорее всего, вам никогда не придется выполнять именно такой поиск, однако рано или поздно может возникнуть потребность в использовании нестандартных критериев поиска. И будет лучше, если вы освоите принципы написания соответствующего кода заранее.

Как уже говорилось, рекурсия больше всего подходит для решения проблем, предусматривающих использование древовидной структуры данных. Файловая система на вашем компьютере подобна дереву (см. рис. 2.6). Каждая папка «разветвляется» на подпапки, которые, в свою очередь, могут разветвляться еще на одни. В данной главе мы напишем рекурсивную функцию для навигации по этому дереву.

ПРИМЕЧАНИЕ

Поскольку JavaScript не имеет доступа к папкам на вашем компьютере, программа для данного проекта будет написана только на языке Python.

Полный код программы для поиска файлов

Начнем с рассмотрения полного исходного кода рекурсивной программы для поиска файлов. В оставшейся части главы каждый ее раздел будет рассмотрен отдельно. Скопируйте исходный код программы в файл с именем `fileFinder.py`:

```
import os

def hasEvenByteSize(fullFilePath):
    """Возвращает True, если значение размера fullFilePath в байтах является четным.
    В противном случае возвращает False."""
    fileSize = os.path.getsize(fullFilePath)
    return fileSize % 2 == 0

def hasEveryVowel(fullFilePath):
    """Возвращает True, если имя fullFilePath содержит буквы a, e, i, o, u,
    иначе возвращает False."""
    name = os.path.basename(fullFilePath).lower()
    return ('a' in name) and ('e' in name) and ('i' in name) and ('o' in name)
    and ('u' in name)


def walk(folder, matchFunc):
    """Вызывает функцию сопоставления для каждого файла в папке
       и ее подпапках. Возвращает список файлов, для которых функция
       сопоставления вернула значение True."""
    matchedFiles = [] # Этот список содержит все совпадения.
    folder = os.path.abspath(folder) # Используйте абсолютный путь к папке.

    # Переберите в цикле все файлы и подпапки в папке:
    for name in os.listdir(folder):
        filepath = os.path.join(folder, name)
        if os.path.isfile(filepath):
            # Вызовите функцию сопоставления для каждого файла:
            if matchFunc(filepath):
                matchedFiles.append(filepath)
        elif os.path.isdir(filepath):
            # Рекурсивно вызовите функцию walk для каждой подпапки
            # и добавьте обнаруженные совпадения в matchedFiles:
            matchedFiles.extend(walk(filepath, matchFunc))
    return matchedFiles
print('All files with even byte sizes:')
print(walk('.', hasEvenByteSize))
print('All files with every vowel in their name:')
print(walk('.', hasEveryVowel))
```

Основой программы для поиска файлов является функция `walk()`, которая «обходит» все файлы в базовой папке и ее подпапках. Она вызывает одну из двух других функций, которые реализуют пользовательские критерии поиска. В контексте данной программы мы будем называть их *функциями сопоставления*. Вызов такой функции возвращает значение `True`, если файл соответствует критериям поиска, в противном случае он возвращает значение `False`.

Задача `walk()` состоит в том, чтобы вызвать функцию сопоставления один раз для каждого файла, содержащегося в каталогах, которые она обходит. Давайте рассмотрим код более подробно.

Функции сопоставления

В Python при вызове функции мы можем передавать в качестве аргументов сами функции. В следующем примере `callTwice()` дважды вызывает функцию, переданную ей в качестве аргумента, будь то `sayHello()` или `sayGoodbye()`:

```
>>> def callTwice(func):
...     func()
...     func()
...
>>> def sayHello():
...     print('Hello!')
...
>>> def sayGoodbye():
...     print('Goodbye!')
...
>>> callTwice(sayHello)
Hello!
Hello!
>>> callTwice(sayGoodbye)
Goodbye!
Goodbye!
```

Функция `callTwice()` вызывает любую функцию, переданную ей в качестве параметра `func`. Обратите внимание, что мы опускаем круглые скобки в аргументе функции и пишем `callTwice(sayHello)` вместо `callTwice(sayHello())`. Это связано с тем, что мы передаем саму функцию `sayHello()`, а не вызываем `sayHello()` и передаем возвращаемое ею значение.

Функция `walk()` принимает в качестве аргумента функцию соответствия, служащую критерием поиска. Это позволяет нам настраивать поведение программы для поиска файлов, не изменяя код `walk()`. Обсудим функцию `walk()` чуть позже. Сначала рассмотрим два примера функций сопоставления.

Поиск файлов с четным значением размера в байтах

Первая функция сопоставления находит файлы с четным значением размера в байтах:

```
import os

def hasEvenByteSize(fullFilePath):
    """Возвращает True, если размер fullFilePath в байтах — четное число.
    В противном случае возвращает False."""
    fileSize = os.path.getsize(fullFilePath)
    return fileSize % 2 == 0
```

Мы импортируем модуль os, который используется в этой программе для получения информации о файлах на компьютере, с помощью таких функций, как getsize(), basename() и т. д. Затем создаем функцию сопоставления с именем hasEvenByteSize(). Все функции сопоставления принимают один строковый аргумент fullFilePath и, если совпадение обнаружено, возвращают True, в противном случае — False.

Функция os.path.getsize() определяет размер файла fullFilePath в байтах. Затем мы используем оператор %, чтобы определить четность полученного числа. Если оно кратно двум, инструкция return возвращает значение True, а если нет, то False. Например, давайте определим размер приложения Блокнот, которое поставляется с ОС Windows (в macOS или Linux попробуйте применить эту функцию к программе /bin/ls):

```
>>> import os
>>> os.path.getsize('C:/Windows/system32/notepad.exe')
211968
>>> 211968 % 2 == 0
True
```

Функция сопоставления hasEvenByteSize() может использовать любую функцию Python для нахождения дополнительной информации о файле fullFilePath. Это дает возможность писать код с учетом любых критериев поиска. В процессе обхода папок и подпапок walk() вызывает функцию сопоставления для каждого файла, содержащегося в этих каталогах. Данная функция сопоставления возвращает True или False, сообщая walk() о том, соответствует ли тот или иной файл критериям поиска.

Поиск имен файлов, содержащих все гласные

Рассмотрим еще одну функцию сопоставления:

```
def hasEveryVowel(fullFilePath):
    """Возвращает True, если имя файла fullFilePath содержит
    гласные a, e, i, o, u, иначе возвращает False."""
    name = os.path.basename(fullFilePath).lower()
    return ('a' in name) and ('e' in name) and ('i' in name) and
      ('o' in name) and ('u' in name)
```

Мы вызываем функцию os.path.basename(), чтобы удалить имена папок из пути к файлу. Python сравнивает строки с учетом регистра, поэтому нам нужно гарантировать, что функция hasEveryVowel() не пропустит ни одной гласной в имени файла из-за того, что она написана в верхнем регистре. Например, вызов os.path.basename('C:/Windows/system32/notepad.exe') возвращает строку notepad.exe. Вызов метода lower() преобразует символы строки в нижний регистр, благодаря чему нам достаточно проверить наличие в ней лишь строчных гласных.

Далее, в разделе «Полезные функции стандартной библиотеки Python для работы с файлами» нам предстоит рассмотреть еще несколько функций для получения информации о файлах.

Мы используем оператор `return` с длинным выражением, которое оценивается как `True`, если `name` содержит `a`, `e`, `i`, `o` или `u`, то есть соответствует критериям поиска. В противном случае оператор `return` возвращает `False`.

Рекурсивная функция walk()

Функции сопоставления проверяют, соответствует ли файл критериям поиска, а функция `walk()` находит все файлы, подлежащие проверке. Рекурсивной функции `walk()` передается имя базовой папки для выполнения поиска и функция сопоставления, которая должна вызываться для каждого содержащегося в этом каталоге файла.

Функция `walk()` также рекурсивно вызывает сама себя для проверки каждой подпапки, находящейся в базовой папке. В ходе рекурсивного вызова эти вложенные папки играют роль базовой.

Ответим на три вопроса относительно текущей рекурсивной функции.

Что представляет собой базовый случай? Базовый случай имеет место, когда функция завершает обработку всех файлов и подпапок в базовом каталоге.

Какой аргумент передается рекурсивной функции при ее вызове? Базовая папка, в которой будет осуществляться поиск, и функция сопоставления для поиска файлов, соответствующих указанным критериям. Для всех папок внутри базовой выполняется рекурсивный вызов, в ходе которого соответствующая подпапка играет роль базового каталога.

Как этот аргумент приближается к базовому случаю? В конечном итоге функция либо завершает проверку всех вложенных папок, либо обнаруживает базовые папки, в которых отсутствуют подпапки.

На рис. 10.1 показан пример файловой системы вместе с рекурсивными вызовами функции `walk()`, которые она совершает при обходе базовой папки `C:\`.

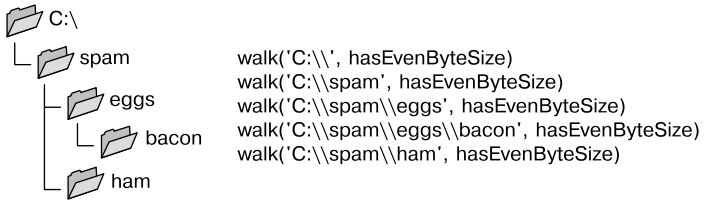


Рис. 10.1. Пример файловой системы и рекурсивных вызовов функции walk()

Давайте проанализируем код функции `walk()`:

```
def walk(folder, matchFunc):
    """Вызывает функцию сопоставления для каждого файла в папке
       и ее подпапках. Возвращает список файлов, для которых функция
       сопоставления вернула значение True."""
    matchedFiles = [] # Этот список содержит все совпадения.
    folder = os.path.abspath(folder) # Используйте абсолютный путь к папке.
```

Функция `walk()` имеет два параметра: `folder` — строка с именем базовой папки, в которой требуется осуществить поиск (мы можем передать значение `'.'`, чтобы указать на тот каталог, из которого запускается программа Python), и `matchFunc` — функция Python, которая принимает имя файла и возвращает `True`, если совпадение найдено. В противном случае функция возвращает `False`.

Следующая часть функции проверяет содержимое `folder`:

```
    # Переберите в цикле все файлы и подпапки в папке:
    for name in os.listdir(folder):
        filepath = os.path.join(folder, name)
        if os.path.isfile(filepath):
```

Цикл `for` вызывает `os.listdir()` для возвращения списка содержимого папки `folder`. Этот список включает все файлы и подкаталоги. Для каждого файла создаем полный абсолютный путь, присоединяя к имени каталога имя файла или папки. Если имя ссылается на файл, вызов функции `os.path.isfile()` возвращает `True`, и мы проверяем данный файл на соответствие критериям поиска:

```
            # Вызовите функцию сопоставления для каждого файла:
            if matchFunc(filepath):
                matchedFiles.append(filepath)
```

Иначе говоря, вызываем функцию сопоставления, передавая ей полный абсолютный путь к текущему файлу цикла `for`. Обратите внимание, что `matchFunc` — это имя одного из параметров функции `walk()`. Когда в качестве аргумента для параметра `matchFunc` передается функция наподобие `hasEvenByteSize()` и `hasEveryVowel()`, имеет место вызов функции `walk()`. Если `filepath` содержит файл, соответствующий критериям поиска, он добавляется в список совпадений:

```
        elif os.path.isdir(filepath):
            # Рекурсивно вызовите функцию walk для каждой подпапки и добавьте
            # обнаруженные совпадения в matchedFiles:
            matchedFiles.extend(walk(filepath, matchFunc))
```

В противном случае, если текущим файлом в цикле `for` является подпапка, вызов функции `os.path.isdir()` возвращает значение `True`. Затем мы передаем этот подкаталог в ходе рекурсивного вызова функции. Такой рекурсивный вызов возвращает

список всех отвечающих критериям файлов в подпапке (и вложенных в нее папках), которые затем добавляются в список совпадений:

```
    return matchedFiles
```

После завершения работы цикла `for` список совпадений должен содержать все отвечающие критериям файлы в этой директории (и во всех ее подпапках). Данный список становится возвращаемым значением для функции `walk()`.

Вызов функции walk()

Теперь, когда реализована функция `walk()` и несколько функций сопоставления, можно запустить процесс поиска файлов. В качестве первого аргумента мы передаем в `walk()` строку `'.'`, указывая, что базовой папкой для осуществления поиска является *текущий каталог*, из которого была запущена программа:

```
print('All files with even byte sizes:')
print(walk('.', hasEvenByteSize))
print('All files with every vowel in their name:')
print(walk('.', hasEveryVowel))
```

Результат выполнения программы зависит от того, какие именно файлы находятся на вашем компьютере. В данном случае он просто демонстрирует способ написания кода для использования нужных вам критериев поиска. Например, вывод программы может выглядеть следующим образом:

```
All files with even byte sizes:
['C:\\Path\\accesschk.exe', 'C:\\Path\\accesschk64.exe',
'C:\\Path\\AccessEnum.exe', 'C:\\Path\\ADExplorer.exe',
'C:\\Path\\Bginfo.exe', 'C:\\Path\\Bginfo64.exe',
'C:\\Path\\diskext.exe', 'C:\\Path\\diskext64.exe',
'C:\\Path\\Diskmon.exe', 'C:\\Path\\DiskView.exe',
'C:\\Path\\hex2dec64.exe', 'C:\\Path\\jpegtran.exe',
'C:\\Path\\Tcpview.exe', 'C:\\Path\\Testlimit.exe',
'C:\\Path\\wget.exe', 'C:\\Path\\whois.exe']
All files with every vowel in their name:
['C:\\Path\\recursionbook.bat']
```

Полезные функции стандартной библиотеки Python для работы с файлами

Рассмотрим операции, которые могут вам пригодиться при написании собственных функций сопоставления. Стандартная библиотека модулей, поставляемая вместе с Python, содержит несколько полезных функций для получения информации о файлах. Многие из них являются частью модулей `os` и `shutil`, поэтому ваша

программа должна будет выполнить команду `import os` или `import shutil`, прежде чем вызвать эти функции.

Поиск информации об имени файла

Полный путь к файлу, передаваемый функциям сопоставления, может быть разбит на базовое имя и имя каталога с помощью функций `os.path.basename()` и `os.path.dirname()`. Вы также можете вызвать `os.path.split()`, чтобы получить эти имена в виде кортежа. Введите следующий код в окне интерактивной оболочки Python (в ОС macOS или Linux попробуйте использовать в качестве имени файла `/bin/ls`):

```
>>> import os
>>> filename = 'C:/Windows/system32/notepad.exe'
>>> os.path.basename(filename)
'notepad.exe'
>>> os.path.dirname(filename)
'C:/Windows/system32'
>>> os.path.split(filename)
('C:/Windows/system32', 'notepad.exe')
>>> folder, file = os.path.split(filename)
>>> folder
'C:/Windows/system32'
>>> file
'notepad.exe'
```

При проверке файла на соответствие критериям поиска к этим строковым значениям можно применить любой строковый метод Python, например, как это было сделано в случае с методом `lower()` в функции сопоставления `hasEveryVowel()`.

Поиск информации о временных метках файла

Файлы предусматривают временные метки, указывающие момент их создания, последнего изменения и последнего обращения к ним. Функции Python `os.path.getctime()`, `os.path.getmtime()` и `os.path.getatime()` возвращают эти временные метки в виде значений с плавающей точкой, указывающих количество секунд, прошедших с начала *эпохи Unix*, которая наступила в полночь 1 января 1970 года по всемирному координированному времени (Coordinated Universal Time, UTC). Введите следующий код в окне интерактивной оболочки:

```
> import os
> filename = 'C:/Windows/system32/notepad.exe'
> os.path.getctime(filename)
1625705942.1165037
> os.path.getmtime(filename)
1625705942.1205275
> os.path.getatime(filename)
1631217101.8869188
```

Программы с легкостью используют значения с плавающей точкой, поскольку они представляют собой обычные числа. Однако для того, чтобы сделать их более простыми для восприятия, потребуются функции из модуля Python `time`. Функция `time.localtime()` преобразует временную метку Unix в объект `struct_time`, ссылающийся на часовой пояс компьютера. Объект `struct_time` имеет несколько атрибутов, имена которых начинаются с `tm_`, для получения информации о дате и времени. Введите следующий код в окне интерактивной оболочки:

```
>>> import os
>>> filename = 'C:/Windows/system32/notepad.exe'
>>> ctimestamp = os.path.getctime(filename)
>>> import time
>>> time.localtime(ctimestamp)
time.struct_time(tm_year=2021, tm_mon=7, tm_mday=7, tm_hour=19,
tm_min=59, tm_sec=2, tm_wday=2, tm_yday=188, tm_isdst=1)
>>> st = time.localtime(ctimestamp)
>>> st.tm_year
2021
>>> st.tm_mon
7
>>> st.tm_mday
7
>>> st.tm_wday
2
>>> st.tm_hour
19
>>> st.tm_min
59
>>> st.tm_sec
2
```

Имейте в виду, что атрибут `tm_mday` обозначает день месяца в диапазоне от `1` до `31`, а `tm_wday` — день недели, причем понедельник соответствует значению `0`, вторник — значению `1` и т. д., вплоть до воскресенья, соответствующего `6`.

Если вы хотите превратить объект `time_struct` в короткую, удобочитаемую строку, передайте его функции `time.asctime()`:

```
>>> import os
>>> filename = 'C:/Windows/system32/notepad.exe'
>>> ctimestamp = os.path.getctime(filename)
>>> import time
>>> st = time.localtime(ctimestamp)
>>> time.asctime(st)
'Wed Jul  7 19:59:02 2021'
```

Функция `time.localtime()` возвращает объект `struct_time`, использующий местное время, а `time.gmtime()` возвращает объект, использующий часовой пояс UTC или среднее время по Гринвичу (GMT). Введите в окне интерактивной оболочки следующий код:

```
>>> import os
>>> filename = 'C:/Windows/system32/notepad.exe'
>>> ctimestamp = os.path.getctime(filename)
>>> import time
>>> ctimestamp = os.path.getctime(filename)
>>> time.localtime(ctimestamp)
time.struct_time(tm_year=2021, tm_mon=7, tm_mday=7, tm_hour=19,
tm_min=59, tm_sec=2, tm_wday=2, tm_yday=188, tm_isdst=1)
>>> time.gmtime(ctimestamp)
time.struct_time(tm_year=2021, tm_mon=7, tm_mday=8, tm_hour=0,
tm_min=59, tm_sec=2, tm_wday=3, tm_yday=189, tm_isdst=0)
```

Взаимодействие между функциями `os.path` (которые возвращают временные метки Unix) и функциями `time` (возвращающими объекты `struct_time`) может сбить с толку. На рис. 10.2 показана цепочка команд, начинающаяся со строки с именем файла и заканчивающаяся получением отдельных фрагментов временной метки.

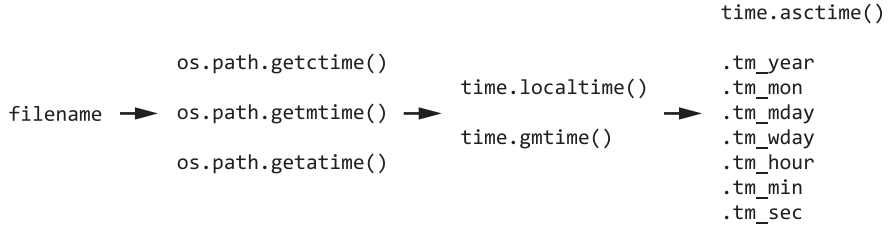


Рис. 10.2. Переход от имени файла к отдельным атрибутам временной метки

Наконец, функция `time.time()` возвращает количество секунд, прошедших с начала эпохи Unix до текущего момента.

Изменение файлов

После того как функция `walk()` возвратит список файлов, соответствующих критериям поиска, можно переименовать, удалить или выполнить над ними другую операцию. Модули `shutil` и `os` стандартной библиотеки Python предусматривают соответствующие функции. Кроме того, модуль `send2trash`, созданный сторонними разработчиками, позволяет отправлять файлы в корзину вместо того, чтобы безвозвратно их удалять.

Чтобы переместить файл, вызовите функцию `shutil.move()`, передав ей в виде аргументов имена перемещаемого файла и папки, в которую его нужно переместить. Например, можете выполнить следующие команды:

```
>>> import shutil
>>> shutil.move('spam.txt', 'someFolder')
'someFolder\\spam.txt'
```

Функция `shutil.move()` возвращает строку, содержащую новый путь к файлу. Имеется возможность переименовать файл в момент его перемещения:

```
>>> import shutil
>>> shutil.move('spam.txt', 'someFolder\\newName.txt')
'someFolder\\newName.txt'
```

Если второй аргумент не содержит имени папки, допускается просто указать новое имя файла, чтобы переименовать его в той директории, в которой он находится:

```
>>> import shutil
>>> shutil.move('spam.txt', 'newName.txt')
'newName.txt'
```

Имейте в виду, что функция `shutil.move()` отвечает как за перемещение, так и за переименование файлов, подобно команде `mv` в Unix и macOS. Отдельной функции `shutil.rename()` не существует.

Чтобы скопировать файл, используйте функцию `shutil.copy()`, передав ей два аргумента — имя копируемого файла и новое имя для его копии. Например, вы можете выполнить следующие команды:

```
>>> import shutil
>>> shutil.copy('spam.txt', 'spam-copy.txt')
'spam-copy.txt'
```

Функция `shutil.copy()` возвращает имя копии. Чтобы удалить файл, вызовите функцию `os.unlink()` и передайте ей имя удаляемого файла:

```
>>> import os
>>> os.unlink('spam.txt')
>>>
```

В данном случае используется слово *unlink* («отвязать»), а не *delete* («удалить»), поскольку с технической точки зрения происходит исключение имени, связанного с файлом. Но так как большинство файлов имеют лишь одно связанное с ними имя, удаление этой связи приводит к удалению самого файла. Не переживайте, если эти концепции кажутся вам непонятными, просто имейте в виду, что функция `os.unlink()` отвечает за удаление файла.

Вызов функции `os.unlink()` навсегда стирает файл, что может представлять опасность в случае наличия ошибки в программе, приводящей к удалению не того файла. Вместо нее вы можете использовать функцию `send2trash()` одноименного модуля, чтобы поместить файл в корзину. Для установки этого модуля введите код `run python -m pip install --user send2trash` в командной строке Windows или `run python3 -m pip install` в окне терминала macOS или Linux. После установки модуля вы сможете импортировать его с помощью команды `import send2trash`.

Введите в окне интерактивной оболочки следующий код:

```
>>> open('deleteme.txt', 'w').close() # Создайте пустой файл
>>> import send2trash
>>> send2trash.send2trash('deleteme.txt')
```

В данном примере создается пустой файл с именем `deleteme.txt`. После вызова `send2trash.send2trash()` (модуль и функция имеют одно и то же имя) этот файл перемещается в корзину.

Резюме

Описанная в главе программа для поиска файлов использует рекурсию для «обхода» содержимого базовой папки и всех ее каталогов. Функция `walk()` рекурсивно обходит эти папки, проверяя содержащиеся в них файлы на предмет соответствия пользовательским критериям поиска. Данные критерии реализованы в виде функций сопоставления, которые передаются функции `walk()`. Благодаря этому для изменения критериев поиска нам достаточно написать новые функции вместо того, чтобы модифицировать код `walk()`.

В нашем проекте были использованы две функции сопоставления для поиска файлов с четным размером в байтах или файлов, содержащих в своем имени гласные буквы. Однако вы можете написать собственные функции, чтобы передать их для `walk()`.

В этом и заключается мощь программирования: вы способны создавать функции, недоступные в коммерческих приложениях, исходя из собственных потребностей.

Дополнительные источники информации

Документацию по встроенной в Python функции `os.walk()` (напоминающей функцию `walk()` из описанной в данной главе программы) можно найти по адресу https://docs.python.org/3/library/os.html#os.walk. Чтобы узнать больше о файловой системе компьютера и функциях Python для работы с файлами, прочитайте главу 9 моей книги «Автоматизация рутинных задач с помощью Python», доступную по адресу https://automatetheboringstuff.com/2e/chapter9.

Модуль `datetime` стандартной библиотеки Python предусматривает множество способов взаимодействия с данными временных меток. Вы можете узнать о них подробнее из главы 17 моей книги «Автоматизация рутинных задач с помощью Python» (https://automatetheboringstuff.com/2e/chapter17).