ГЛАВА 3
Разработка веб-краулеров

До сих пор нам встречались лишь отдельные статические страницы с несколько искусственно законсервированными примерами. В этой главе мы познакомимся с реальными задачами, для решения которых веб-скраперы перебирают несколько страниц и даже сайтов.

Веб-краулеры называются так потому, что они «ползают» (crawl) по Всемирной паутине и собирают данные с веб-страниц. В основе их работы лежит рекурсия. Веб-краулер получает контент страницы по ее URL, исследует эту страницу, находит URL другой страницы, извлекает содержимое *этой другой* страницы и далее до бесконечности.

Однако будьте осторожны: возможность собирать данные в Интернете еще не означает, что вы всегда должны это делать. Веб-скраперы, показанные в предыдущих примерах, отлично работают в ситуациях, когда все необходимые данные находятся на одной странице. Используя веб-краулеры, следует быть предельно внимательными к тому, какую часть пропускной способности сети вы задействуете, и всеми силами постараться определить, есть ли способ облегчить нагрузку на интересующий вас сервер.

Проход отдельного домена

Даже если вам еще не приходилось слышать об игре «Шесть шагов по “Википедии”», вы наверняка знаете о ее предшественнице — «Шесть шагов до Кевина Бейкона». В обеих играх цель состоит в том, чтобы установить взаимосвязь между двумя мало связанными элементами (в первом случае — между статьями «Википедии», а во втором — между актерами, сыгравшими в одном фильме) и построить цепь, содержащую не более шести звеньев (включая начальный и конечный элементы).

Например, Эрик Айдл (Eric Idle) снялся в фильме «Дадли Справедливый» (Dudley Do-Right) с Бренданом Фрейзером (Brendan Fraser), который, в свою очередь, снялся с Кевином Бейконом (Kevin Bacon) в «Воздухе, которым я дышу» (The Air I Breathe)[1]. В этом случае цепь от Эрика Айдла до Кевина Бейкона состоит всего из трех элементов.

В этом разделе мы начнем разработку проекта, который позволит строить цепи для «Шести шагов по "Википедии"»: например, начав со страницы Эрика Айдла (https://en.wikipedia.org/wiki/Eric_Idle), вы сможете найти наименьшее количество переходов по ссылкам, которые приведут вас на страницу Кевина Бейкона (https://en.wikipedia.org/wiki/Kevin_Bacon).

А КАК НАСЧЕТ НАГРУЗКИ НА СЕРВЕР «ВИКИПЕДИИ»?

По данным Фонда Викимедиа (вышестоящей организации, отвечающей в том числе за «Википедию»), за одну секунду происходит примерно 2500 обращений к веб-ресурсам сайта, причем более 99 % из них относятся к «Википедии» (см. раздел Traffic volume на странице Wikimedia in figures, https://meta.wikimedia.org/wiki/Wikimedia_in_figures_-_Wikipedia#Traffic_volume). Из-за большого объема трафика ваши веб-скраперы вряд ли сколько-нибудь заметно повлияют на нагрузку сервера «Википедии». Тем не менее, если вы намерены активно использовать примеры кода, приведенные в этой книге, или будете разрабатывать собственные проекты для веб-скрапинга «Википедии», я призываю вас совершить необлагаемое налогом пожертвование в Фонд Викимедиа (https://wikimediafoundation.org/wiki/Ways_to_Give) — не только в качестве компенсации нагрузки на сервер, но и чтобы помочь сделать образовательные ресурсы более доступными для других пользователей.

Кроме того, имейте в виду: если вы планируете создать большой проект с применением данных из «Википедии», то стоит убедиться, что эти данные еще неоступны через API «Википедии» «(https://www.mediawiki.org/wiki/API:Main_page). «Википедия» часто используется в качестве сайта для демонстрации веб-скраперов и веб-краулеров, поскольку данный сайт имеет простую структуру HTML-кода и относительно стабилен. Однако эти же данные могут оказаться более доступными через API.

Вы уже, вероятно, знаете, как написать скрипт на Python, который бы получал произвольную страницу «Википедии» и создавал список ссылок, присутствующих на этой странице:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
```

[1] Благодарю сайт The Oracle of Bacon (http://oracleofbacon.org/), с помощью которого я удовлетворила свое любопытство относительно этой цепочки.

```
html = urlopen('http://en.wikipedia.org/wiki/Kevin_Bacon')
bs = BeautifulSoup(html, 'html.parser')
for link in bs.find_all('a'):
    if 'href' in link.attrs:
        print(link.attrs['href'])
```

Если вы посмотрите на созданный список ссылок, то заметите, что в него вошли все ожидаемые статьи: Apollo 13, Philadelphia, Primetime Emmy Award и т. д. Однако есть и кое-что лишнее:

```
//wikimediafoundation.org/wiki/Privacy_policy
//en.wikipedia.org/wiki/Wikipedia:Contact_us
```

Дело в том, что на каждой странице «Википедии» есть боковая панель, нижний и верхний колонтитулы с множеством ссылок; также есть ссылки на страницы категорий, обсуждений и другие страницы, которые не содержат полезных статей:

```
/wiki/Category:Articles_with_unsourced_statements_from_April_2014
/wiki/Talk:Kevin_Bacon
```

Недавно мой друг, работая над аналогичным проектом по веб-скрапингу «Википедии», заявил, что написал большую функцию фильтрации, насчитывающую более 100 строк кода, с целью определить, является ли внутренняя ссылка «Википедии» ссылкой на страницу статьи. К сожалению, он не уделил достаточно времени поиску закономерностей между «ссылками на статьи» и «другими ссылками», иначе бы нашел более красивое решение. Если вы внимательно посмотрите на ссылки, которые ведут на страницы статей (в отличие от других внутренних страниц), то заметите, что у всех таких ссылок есть три общие черты:

- они находятся внутри тега `div`, у которого атрибут `id` имеет значение `bodyContent`;
- в их URL нет двоеточий;
- их URL начинаются с `/wiki/`.

Мы можем немного изменить код, включив в него эти правила, и получить только нужные ссылки на статьи, воспользовавшись регулярным выражением `^(/wiki/)((?!:).)*$"`):

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re
```

```
html = urlopen('http://en.wikipedia.org/wiki/Kevin_Bacon')
bs = BeautifulSoup(html, 'html.parser')
for link in bs.find('div', {'id':'bodyContent'}).find_all(
    'a', href=re.compile('^(/wiki/)((?!:).)*$')):
    if 'href' in link.attrs:
        print(link.attrs['href'])
```

Запустив этот код, мы получим список всех URL статей, на которые ссылается статья «Википедии» о Кевине Бейконе.

Скрипт, который находит все ссылки на статьи для одной жестко заданной статьи «Википедии», конечно, интересен, однако с практической точки зрения довольно бесполезен. Вы должны уметь, взяв этот код за основу, преобразовать его примерно в такую программу.

- Отдельная функция `getLinks` принимает URL статьи «Википедии» в формате `/wiki/<Название_статьи>` и возвращает список всех URL, на которые ссылается эта статья, в том же формате.
- Основная функция, которая вызывает `getLinks` с первоначальной статьей, выбирает из возвращенного списка случайную ссылку и снова вызывает `getLinks`, пока пользователь не остановит программу или пока не окажется, что на очередной странице нет ссылок.

Вот полный код программы, которая это делает:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import datetime
import random
import re

random.seed(datetime.datetime.now())
def getLinks(articleUrl):
    html = urlopen('http://en.wikipedia.org{}'.format(articleUrl))
    bs = BeautifulSoup(html, 'html.parser')
    return bs.find('div', {'id':'bodyContent'}).find_all('a',
         href=re.compile('^(/wiki/)((?!:).)*$'))

links = getLinks('/wiki/Kevin_Bacon')
while len(links) > 0:
    newArticle = links[random.randint(0, len(links)-1)].attrs['href']
    print(newArticle)
    links = getLinks(newArticle)
```

Сразу после импорта необходимых библиотек программа присваивает генератору случайных чисел начальное значение, равное текущему системному времени, чем обеспечивает новый и интересный случайный путь по статьям «Википедии» практически при каждом запуске программы.

ПСЕВДОСЛУЧАЙНЫЕ ЧИСЛА И СЛУЧАЙНЫЕ НАЧАЛЬНЫЕ ЗНАЧЕНИЯ

В предыдущем примере был задействован генератор случайных чисел Python для случайного выбора статьи на странице, по ссылке на которую будет продолжен обход «Википедии». Однако случайные числа следует использовать осторожно.

Компьютеры хорошо справляются с вычислением правильных решений, но откровенно слабы, когда нужно придумать что-то новое. По этой причине случайные числа могут стать проблемой. Большинство алгоритмов генерации этих чисел стремятся создать равномерно распределенную и труднопредсказуемую числовую последовательность, однако для запуска работы такого алгоритма необходимо «начальное» число. Если оно каждый раз будет одним и тем же, то алгоритм станет всякий раз генерировать одну и ту же последовательность «случайных» чисел. Именно поэтому я использовала значение системных часов в качестве начального значения для создания новых последовательностей случайных чисел и, следовательно, генерации последовательностей случайных статей. Благодаря этому выполнять программу будет немного интереснее.

Любопытно, что генератор псевдослучайных чисел Python работает по *алгоритму Мерсенна Твистера* (Mersenne Twister). Он генерирует труднопредсказуемые и равномерно распределенные случайные числа, однако несколько загружает процессор. Случайные числа — это хорошо, но за все приходится платить!

Затем программа определяет функцию `getLinks`, которая принимает URL статьи в формате `/wiki/...`, добавляет в начало имя домена «Википедии» `http://en.wikipedia.org` и получает объект `BeautifulSoup` с HTML-кодом, который находится по этому адресу. Затем функция извлекает список тегов со ссылками на статьи, исходя из описанных ранее параметров, и возвращает их.

В основной части программы сначала создается список тегов со ссылками на статьи (переменная `links`), в котором содержатся ссылки, найденные на начальной странице https://en.wikipedia.org/wiki/Kevin_Bacon. Затем программа выполняет цикл и находит тег со случайной ссылкой на статью, извлекает оттуда атрибут `href`, выводит страницу и получает по извлеченному URL новый список ссылок.

Разумеется, решение задачи «Шесть шагов по “Википедии”» не ограничивается созданием веб-скрапера, переходящего с одной страницы на другую. Нам так-

же необходимо хранить и анализировать полученные данные. Продолжение решения этой задачи вы найдете в главе 6.

Обрабатывайте исключения!

По большей части обработка исключений в этих примерах пропущена для краткости, однако следует помнить: здесь может возникнуть множество потенциальных ловушек. Что, если, например, «Википедия» изменит имя тега bodyContent? Тогда при попытке извлечь текст из тега программа выдаст исключение AttributeError.

Таким образом, несмотря на то, что эти сценарии достаточно хороши в качестве тщательно контролируемых примеров, в автономном коде готового приложения требуется обрабатывать исключения гораздо лучше, чем позволяет описать объем данной книги. Для получения дополнительной информации об этом вернитесь к главе 1.

Сбор информации со всего сайта

В предыдущем разделе мы прошли по сайту, переходя от одной случайно выбранной ссылки к другой. Но как быть, если нужно систематизировать сайт и составить каталог или просмотреть все страницы? Сбор информации со всего сайта, особенно большого, — процесс, требующий интенсивного использования памяти. Лучше всего с этим справляются приложения, имеющие быстрый доступ к базе данных для хранения результатов краулинга. Однако мы можем исследовать поведение таких приложений, не выполняя их в полном объеме. Подробнее об их выполнении с применением базы данных см. в главе 6.

ТЕМНЫЙ И ГЛУБОКИЙ ИНТЕРНЕТ

Вероятно, вам часто приходилось слышать о *глубоком*, *темном* или *скрытом Интернете*, особенно в последних новостях. Что имеется в виду?

Глубокий Интернет — это любая часть Сети, выходящая за пределы *видимого Интернета*. Видимой называют ту часть Интернета, которая индексируется поисковыми системами. Несмотря на большой разброс оценок, глубокий Интернет почти наверняка составляет около 90 % всего Интернета. Поскольку Google не способен отправлять формы или находить страницы, на которые не ссылается домен верхнего уровня, а также не выполняет поиск сайтов, для которых это запрещено в файле `robots.txt`, видимый Интернет продолжает составлять относительно небольшую часть Сети.

Темный Интернет, также известный как *Даркнет*, — нечто совершенно иное. Он работает на той же сетевой аппаратной инфраструктуре, но использует браузер Tor или

другой клиент, у которого протокол приложения работает поверх HTTP, обеспечивая безопасный канал для обмена информацией. В темном Интернете, как и на обычном сайте, тоже можно выполнять веб-скрапинг, однако эта тема выходит за пределы данной книги.

В отличие от темного, в глубоком Интернете веб-скрапинг выполняется относительно легко. Многие инструменты, описанные в этой книге, научат вас, как выполнять веб-скрапинг и сбор информации во многих местах, недоступных для ботов Google.

В каких случаях сбор данных со всего сайта полезен, а в каких — вреден? Веб-скраперы, которые перебирают весь сайт, хороши во многих случаях, включая следующие.

- *Формирование карты сайта*. Несколько лет назад мне встретилась задача: важный клиент хотел оценить затраты на редизайн сайта, однако не хотел предоставлять моей компании доступ ко внутренним компонентам существующей системы управления контентом и у него не было общедоступной карты сайта. Я воспользовалась веб-краулером, чтобы пройти по всему сайту, собрать все внутренние ссылки и разместить страницы в структуре папок, соответствующей той, что применялась на сайте. Это позволило мне быстро обнаружить разделы сайта, о которых я даже не подозревала, и точно подсчитать, сколько эскизов страниц потребуется создать и какой объем контента необходимо перенести.
- *Сбор данных*. Другой клиент хотел собрать статьи (истории, посты в блогах, новости и т. п.), чтобы построить рабочий прототип специализированной поисковой платформы. Это исследование сайтов должно было быть не всеобъемлющим, однако достаточно обширным (нам хотелось получать данные лишь с нескольких сайтов). Мне удалось создать веб-краулеры, которые рекурсивно обходили каждый сайт и собирали данные только со страниц статей.

Общий подход к полному сбору данных с сайта заключается в том, чтобы начать со страницы верхнего уровня (например, с начальной страницы) и построить список всех ее внутренних ссылок. Затем обойти все страницы, на которые указывают эти ссылки, и на каждой из них собрать дополнительные списки ссылок, запускающие очередной этап сбора данных.

Конечно же, в такой ситуации количество ссылок стремительно растет. Если на каждой странице есть десять внутренних ссылок, а глубина сайта составляет пять страниц (что довольно типично для сайта среднего размера), то необходимо проверить 10^5, то есть 100 000 страниц, чтобы с уверенностью утверждать:

сайт пройден полностью. Как ни странно, хоть и «пять страниц в глубину и десять внутренних ссылок на каждой странице» — довольно типичный размер сайта, очень немногие сайты действительно насчитывают 100 000 и более страниц. Причина, конечно, состоит в том, что подавляющее большинство внутренних ссылок являются дубликатами.

Во избежание повторного сбора данных с одной и той же страницы крайне важно, чтобы все обнаруженные внутренние ссылки имели согласованный формат и сохранялись в общем множестве, что облегчило бы поиск во время работы программы. *Множество* похоже на список, но его элементы не располагаются в определенной последовательности. Кроме того, в множестве содержатся исключительно уникальные элементы, что идеально подходит для наших целей. Следует проверять лишь те ссылки, которые являются «новыми», и искать дополнительные ссылки только по ним:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

pages = set()
def getLinks(pageUrl):
    global pages
    html = urlopen('http://en.wikipedia.org{}'.format(pageUrl))
    bs = BeautifulSoup(html, 'html.parser')
    for link in bs.find_all('a', href=re.compile('^(/wiki/)')):
        if 'href' in link.attrs:
            if link.attrs['href'] not in pages:
                # мы нашли новую страницу
                newPage = link.attrs['href']
                print(newPage)
                pages.add(newPage)
                getLinks(newPage)
getLinks('')
```

Для демонстрации полной работы этого веб-краулера я ослабила требования к тому, как должны выглядеть внутренние ссылки (из предыдущих примеров). Веб-скрапер не ограничивается только страницами статей, а ищет все ссылки, начинающиеся с `/wiki/`, независимо от того, где они располагаются на странице и содержат ли двоеточия. Напомню: в URL страниц статей отсутствуют двоеточия, в отличие от адресов страниц загрузки файлов, обсуждений и т. п.

Изначально функция `getLinks` вызывается с пустым URL, то есть для начальной страницы «Википедии»: к пустому URL внутри функции добавляется `http://en.wikipedia.org`. Затем функция просматривает каждую ссылку на

первой странице и проверяет, есть ли она в *глобальном множестве* страниц (множестве страниц, с которыми скрипт уже встречался). Если нет, то ссылка добавляется в список, выводится на экран и функция `getLinks` вызывается для нее рекурсивно.

Осторожно с рекурсией

Это предупреждение редко встречается в книгах по программированию, но я считаю, что вы должны знать: если оставить программу работать достаточно долго, то предыдущая программа почти наверняка завершится аварийно.

В Python установлен по умолчанию предел рекурсии (количество рекурсивных вызовов программы), и он равен 1000. Поскольку сеть ссылок «Википедии» чрезвычайно обширна, данная программа в итоге достигнет предела рекурсии и остановится, если только не добавить счетчик рекурсии или что-то еще, призванное помешать этому.

Для «плоских» сайтов глубиной менее 1000 ссылок данный метод обычно — за редким исключением — работает хорошо. Например, однажды мне встретилась ошибка в динамически генерируемом URL, поскольку ссылка на следующую страницу зависела от адреса текущей страницы. Это привело к бесконечно повторяющимся путям, таким как /blogs/blogs.../blogs/blog-post.php.

Однако, как правило, этот рекурсивный метод должен подходить для любого обычного сайта, с которым вы, скорее всего, столкнетесь.

Сбор данных со всего сайта. Веб-краулеры были бы довольно скучными программами, если бы только переходили с одной страницы на другую. Полезными они могут быть, что-то делая на странице, на которую перешли. Рассмотрим пример веб-скрапера, который бы выбирал заголовок, первый абзац контента страницы и ссылку на режим редактирования страницы (если таковая существует).

Как всегда, чтобы выбрать лучший способ сделать это, сначала нужно просмотреть несколько страниц сайта и определить его структуру. При просмотре нескольких страниц «Википедии» (как статей, так и страниц, не являющихся статьями, например страницы с политикой конфиденциальности) становится ясно следующее.

- У любой страницы (независимо от статуса: статья, история редактирования или любая другая страница) есть заголовок, заключенный в теги `h1` → `span`, и это единственный тег `h1` на странице.
- Как уже говорилось, весь основной текст находится внутри тега `div#bodyContent`. Но если вы хотите выделить конкретную часть текста — например, получить доступ только к первому абзацу, — то лучше использовать

`div#mw-content-text` → `p` (выбрать только тег первого абзаца). Это подходит для всех контентных страниц, кроме страниц файлов (например, https://en.wikipedia.org/wiki/File:Orbit_of_274301_Wikipedia.svg), на которых нет разделов основного текста.

- Ссылки для редактирования существуют только на страницах статей. Если такая ссылка есть, то она находится в теге `li#ca-edit`, в `li#ca-edit` → `span` → `a`.

Изменив исходный код нашего веб-краулера, мы можем создать комбинированную программу для краулинга и сбора данных (или по крайней мере для вывода данных на экран):

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
import re

pages = set()
def getLinks(pageUrl):
    global pages
    html = urlopen('http://en.wikipedia.org{}'.format(pageUrl))
    bs = BeautifulSoup(html, 'html.parser')
    try:
        print(bs.h1.get_text())
        print(bs.find(id ='mw-content-text').find_all('p')[0])
        print(bs.find(id='ca-edit').find('span')
             .find('a').attrs['href'])
    except AttributeError:
        print('This page is missing something! Continuing.')

    for link in bs.find_all('a', href=re.compile('^(/wiki/)')):
        if 'href' in link.attrs:
            if link.attrs['href'] not in pages:
                # мы нашли новую страницу
                newPage = link.attrs['href']
                print('-'*20)
                print(newPage)
                pages.add(newPage)
                getLinks(newPage)
getLinks('')
```

Цикл `for` в этой программе, по сути, такой же, как и в первоначальной программе сбора данных (для ясности добавлен вывод дефисов в качестве разделителей контента).

Поскольку никогда нельзя быть полностью уверенными в том, что на каждой странице будут присутствовать все нужные данные, операторы `print` расположены в соответствии с вероятностью наличия этих данных на странице.

Так, тег заголовка `h1` есть на каждой странице (во всяком случае, насколько я могу судить), вследствие чего мы сначала пытаемся получить эти данные. Текстовый контент присутствует на большинстве страниц (за исключением страниц файлов), так что этот фрагмент данных извлекается вторым. Кнопка редактирования есть только на страницах с заголовками и текстовым контентом, и то не на всех.

Разные схемы для различных потребностей

Очевидно, что, когда в блоке обработчика исключений заключено нескольких строк, возникает некая опасность. Прежде всего, вы не можете сказать, какая именно строка вызвала исключение. Кроме того, если по какой-либо причине на странице есть кнопка редактирования, но нет заголовка, то кнопка никогда не будет зарегистрирована. Однако во многих случаях при наличии определенной вероятности возникновения на сайте каждого из элементов этого достаточно и непредвиденное отсутствие нескольких точек данных или ведение подробных журналов не является проблемой.

Вы могли заметить: в этом и во всех предыдущих примерах мы не столько «собирали» данные, сколько «выводили» их. Очевидно, что данными, выводимыми в окно терминала, сложно манипулировать. Подробнее о хранении информации и создании баз данных вы узнаете в главе 5.

ОБРАБОТКА ПЕРЕНАПРАВЛЕНИЙ

Перенаправления позволяют веб-серверу использовать имя текущего домена или URL для контента, находящегося в другом месте. Существует два типа перенаправлений:

- перенаправления на стороне сервера, когда URL изменяется до загрузки страницы;
- перенаправления на стороне клиента, иногда с сообщением наподобие «через десять секунд вы перейдете на другой сайт», когда сначала загружается страница сайта, а потом происходит переход на новую страницу.

С перенаправлениями на стороне сервера вам обычно ничего не нужно делать. Если вы используете библиотеку `urllib` для Python 3.x, то она обрабатывает перенаправления автоматически! В случае применения библиотеки запросов проследите, чтобы флаг `allow_redirects` имел значение `True`:

```
r = requests.get( 'http://github.com' , allow_redirects=True)
```

Просто помните, что иногда URL сканируемой страницы может не совпадать с URL, по которому расположена эта страница.

Подробнее о перенаправлениях на стороне клиента, которые выполняются с помощью JavaScript или HTML, см. в главе 12.