# Содержание

**Предисловие** ..... **10**
**Вступление** ..... **13**

**ЧАСТЬ I. ПОСТРОЕНИЕ СКРАПЕРОВ** ..... **20**
**Глава 1. Ваш первый скрапер** ..... **21**
Соединение с Интернетом ..... 21
Введение в BeautifulSoup ..... 24
Установка BeautifulSoup ..... 24
Запуск BeautifulSoup ..... 26
Как обеспечить надежный скрапинг ..... 28

**Глава 2. Продвинутый парсинг HTML** ..... **31**
Вам не всегда нужен молоток ..... 31
Еще одно применение BeautifulSoup ..... 32
find() и findAll() ..... 34
Другие объекты BeautifulSoup ..... 36
Навигация по дереву синтаксического разбора ..... 37
Работа с дочерними элементами и элементами-потомками ..... 38
Работа с одноуровневыми элементами ..... 39
Работа с родительскими элементами ..... 40
Регулярные выражения ..... 41
Регулярные выражения и BeautifulSoup ..... 46
Получение доступа к атрибутам ..... 47
Лямбда-выражения ..... 48
За рамками BeautifulSoup ..... 48

**Глава 3. Запуск краулера** ..... **50**
Обход отдельного домена ..... 50
Краулинг всего сайта ..... 54
Сбор данных по всему сайту ..... 57
Краулинг Интернета ..... 59
Краулинг с помощью Scrapy ..... 65

**Глава 4. Использование API** ..... **70**
Как работают API ..... 71
Общепринятые соглашения ..... 72
Методы ..... 72

Аутентификация....73
Ответы....74
Вызовы API....75
Echo Nest....76
Несколько примеров....76
Twitter....78
Приступаем к работе....78
Несколько примеров....79
Google API....83
Приступаем к работе....83
Несколько примеров....84
Парсинг JSON-данных....86
Возвращаем все это домой....88
Подробнее о применении API....92

**Глава 5. Хранение данных....94**

Медиафайлы....94
Сохранение данных в формате CSV....97
MySQL....99
Установка MySQL....100
Некоторые основные команды....102
Интеграция с Python....106
Методы работы с базами данных и эффективная практика....109
«Шесть шагов» в MySQL....112
Электронная почта....115

**Глава 6. Чтение документов....117**

Кодировка документа....117
Текст....118
Кодировка текста и глобальный Интернет....119
CSV....124
Чтение CSV-файлов....124
PDF....126
Microsoft Word и .docx....128

**ЧАСТЬ II. ПРОДВИНУТЫЙ СКРАПИНГ....132**

**Глава 7. Очистка данных....133**

Очистка данных на этапе создания кода....133
Нормализация данных....136

Очистка данных постфактум ... 138
OpenRefine ... 139

**Глава 8. Чтение и запись естественных языков ... 144**

Аннотирование данных ... 145
Марковские модели ... 148
Шесть шагов Википедии: заключительная часть ... 152
Natural Language Toolkit ... 156
Установка и настройка ... 156
Статистический анализ с помощью NLTK ... 156
Лексикографический анализ с помощью NLTK ... 160
Дополнительные ресурсы ... 163

**Глава 9. Краулинг сайтов, использующих веб-формы ... 165**

Библиотека requests ... 165
Отправка простой формы ... 166
Радиокнопки, флажки и другие элементы ввода данных ... 168
Отправка файлов и изображений ... 170
Работа с логинами и cookies ... 171
Базовая HTTP-аутентификация ... 173
Другие проблемы при работе с формами ... 174

**Глава 10. Скрапинг JavaScript-кода ... 175**

Краткое введение в JavaScript ... 176
Распространенные библиотеки JavaScript ... 177
Ajax и динамический HTML ... 180
Выполнение JavaScript в Python с помощью библиотеки Selenium ... 181
Обработка редиректов ... 186

**Глава 11. Обработка изображений и распознавание текста ... 189**

Обзор библиотек ... 190
Pillow ... 190
Tesseract ... 191
NumPy ... 192
Обработка хорошо отформатированного текста ... 193
Скрапинг текста с изображений, размещенных на веб-сайтах ... 196

Чтение CAPTCHA и обучение Tesseract ..... 198
Обучение Tesseract ..... 200
Извлечение CAPTCHA и отправка результатов распознавания ..... 204

**Глава 12. Обход ловушек в ходе скрапинга ..... 208**

Обратите внимание на этический аспект ..... 209
Учимся выглядеть как человек ..... 210
Настройте заголовки ..... 210
Обработка cookies ..... 212
Время решает все ..... 214
Общие функции безопасности, используемые веб-формами ..... 215
Значения полей скрытого ввода ..... 215
Обходим «горшочки с медом» ..... 217
Проверяем скрапер на «человечность» ..... 219

**Глава 13. Тестирование вашего сайта с помощью скраперов ..... 221**

Введение в тестирование ..... 222
Что такое модульные тесты? ..... 222
Питоновский модуль unittest ..... 223
Тестирование Википедии ..... 224
Тестирование с помощью Selenium ..... 227
Взаимодействие с сайтом ..... 227
Unittest или Selenium? ..... 231

**Глава 14. Скрапинг с помощью удаленных серверов ..... 233**

Зачем использовать удаленные серверы? ..... 233
Как избежать блокировки IP-адреса ..... 234
Переносимость и расширяемость ..... 235
Tor ..... 236
PySocks ..... 237
Удаленный хостинг ..... 238
Запуск с аккаунта веб-хостинга ..... 238
Запуск из облака ..... 240
Дополнительные ресурсы ..... 241
Заглянем в будущее ..... 242

**Приложение А. Кратко о том, как работает Python ..... 244**

Установка и «Hello, World!» ..... 244

**Приложение B. Кратко о том, как работает Интернет.....248**

**Приложение C. Правовые и этические аспекты веб-скрапинга.....252**

Товарные знаки, авторские права, патенты, о боже!..... 252
Авторское право..... 254
Посягательство на движимое имущество..... 256
Закон о компьютерном мошенничестве и злоупотреблении..... 258
robots.txt и Пользовательское соглашение..... 259
Три нашумевших случая в практике веб-скрапинга..... 263
eBay против Bidder's Edge и посягательство на движимое имущество..... 263
США против Орнхаймера и Закон о компьютерном мошенничестве и злоупотреблении..... 265
Филд против Google: авторское право и robots.txt..... 268

**Об авторе.....269**

**Колофон.....270**

**Предметный указатель.....271**

# Предисловие

В современном цифровом мире данные приобретают все большую и большую ценность. Если мы посмотрим вокруг, то увидим огромное количество различных сервисов, которые пытаются сделать нашу жизнь лучше. Стараются нам помочь, посоветовать, найти нужную информацию. От гигантов типа Google до стартапов и небольших экспериментальных проектов, все эти сервисы работают с данными. В основе любой задачи, которую предстоит сегодня решать машине или человеку, лежат данные.

С другой стороны, на недостаток данных в современном мире жаловаться не приходиться. Данные с большой скоростью генерируются и накапливаются компаниями и устройствами, объемы хранения растут но, конечно, самым перспективным источником данных является сеть Интернет.

Когда-то Интернет был маленькой американской сетью для нескольких сотен человек, где почти все друг друга знали. Теперь это гигантская информационная структура. Здесь практически невозможно контролировать потоки информации. То, что называется «контент, генерируемый пользователями» составляет колоссальный объем данных. Этот объем уже даже невозможно точно измерить. Точно так же как мы когда-то перестали измерять расстояние до звезд в километрах, применяемых на Земле, и стали использовать понятие «световой год», то есть характеристику скорости, также и об интернет-данных теперь пишут в терминах скорости прироста информации, а не ее объема. Так, за одну минуту в Интернете появляются, например, более 3 миллионов новых постов в сети Facebook, более 300 тысяч сообщений в Twitter, более 30 тысяч отзывов о книгах и покупках, не говоря уже об описаниях новых фильмов, товаров и т. д., и т. п.

Все это многообразие информации сейчас активно используется не только людьми, но и организациями для повышения эффективности своей деятельности. Жизнь людей сейчас настолько быстро меняется, что традиционных анкетных данных просто недостаточно для оценки поведения заемщика (если это банк), покупателя (если это розничная сеть). В стремительно меняющемся мире они банально устаревают. И тогда нужно обратиться к «внешним» данным. Произвести обогащение текстовыми данными. Банкам важно анализировать, что пользователи пишут в социальных сетях, на сайтах. Оценить их уровень грамотности, словарный запас, тематику публикуемых сообщений. Магазинам, производителям важно собирать отзывы покупателей об

их товарах, анализировать их тональность, чтобы лучше спланировать рекламную компанию, использовать в ней текстовые формулировки, максимально близкие покупателю. Мы, в своей работе с различными компаниями, видим большое количество таких примеров и перечислять их можно бесконечно.

Тем не менее, для того чтобы эффективно работать с этой информацией, получать из нее пользу и реализовывать задачи, востребованные компаниями и людьми, данные нужно извлекать, обрабатывать, структурировать. То, что мы видим как веб-сайт с отзывом о фильме, для машины представляется сборищем разных «кусков» данных с непонятным назначением. Человек, взглянув на веб-страницу, сразу легко определяет нужный и значимый раздел, но для компьютера понимание того, какой именно текст следует обрабатывать, как отделить этот текст от рекламы, ненужных заголовков, ссылок является довольно сложной задачей.

По мере роста потока информации, возможностей по применению этой информации в прикладных задачах, развиваются технические подходы, объединяемые общим термином «веб-краулинг» или «веб-скрапинг». Они предназначены для сбора информации из сети Интернет и ее подготовки к автоматизированной обработке. И несмотря на то, что не всегда процесс веб-скрапинга «виден» для конечного пользователя, часто именно он является ключевым моментом современных веб-технологий. Возьмем для примера тот же Google. Все мы пользуемся поиском, все мы восхищаемся качеством и релевантностью выдаваемых результатов, но, перед тем как ответить на наши поисковые запросы, сервисы Google проводят огромную работу, они обходят все сайты, скачивают с них страницы, выполняют синтаксический разбор контента, определяют, какой текст на этих страницах является «значимым» и именно его индексируют и выдают пользователю.

Разработка и реализация качественных механизмов сбора информации является залогом успешной ее обработки и в этой книге дается детальное руководство по подходам и методам решения этой задачи с помощью популярного языка программирования Python.

Сейчас Python – стремительно развивающийся язык. В первую очередь его популярность связана с простотой и легкостью освоения. Также развитию популярности способствует и то, что он поставляется под открытой лицензией и является свободно распространяемым. Это обеспечивает наличие огромного количества библиотек, расширений и готовых компонентов, которые можно свободно использовать

в своих проектах, что сильно экономит время разработчика. Python – язык с большой и богатой историей. Он появился более 25 лет назад как высокоуровневый язык программирования – альтернатива Java и C++, на котором всего в несколько строчек можно описать то, что на этих низкоуровневых языках программирования занимает по несколько блоков кода. Python в первую очередь предназначен для написания прикладных приложений, но за годы существования он развился в очень гибкий инструмент, на котором сейчас уже пишутся и очень большие, серьезные и высоконагруженные проекты.

Дополнительным преимуществом данной книги является не только использование Python, но и форма подачи материала. В книге даны не просто примеры кода, весь материал представлен в виде примеров конкретных и практических задач.

Нельзя не отметить и наличие целого учебного сайта, разработанного Райан. На его примере наглядно показана отправка форм, работа капчи, скрапинг JavaScript (использование пауз в выполнении скриптов) и т. д.

Очень приятно, что данная книга выходит в свет на русском языке. Авторы и переводчики проделали огромную и качественную работу. Надеюсь, что распространение знаний и методов, описанных в данной книге, будет направлено разработчиками на создание качественно новых, интеллектуальных сервисов, которые будут приносить пользу людям и организациям в нашей повседневной жизни и, конечно, сделают ее немного лучше.

*Денис Афанасьев*,
генеральный директор компании «CleverDATA»

# Вступление

Для новичков программирование может показаться своего рода магией. И если программирование – это магия, то *веб-скрапинг (web scraping)* является колдовством. Применение магии для решения конкретных впечатляющих и важных задач, да еще без всяких усилий, выглядит чудом.

В самом деле, когда я работала инженером-программистом, я пришла к выводу, что существует очень немного направлений в программировании, которые так же, как веб-скрапинг, одинаково привлекают внимание программистов и обычных пользователей. Хотя написать простую программу-робот, которая соберет информацию, отправит ее в терминал или сохранит в базе данных, несложно, все равно испытываешь определенный трепет, ощущение непредсказуемости результата, независимо от того, сколько раз ты уже проделывал эту процедуру.

Жаль, что когда я разговариваю с другими программистами о веб-скрапинге, возникает много непонимания и путаницы. Некоторые высказывают сомнения по поводу законности веб-скрапинга или спорят по поводу способов обработки современного Интернета со всеми его JavaScript, мультимедийным контентом и cookies. Некоторые смешивают API с веб-скраперами.

Настоящая книга стремится поставить окончательную точку в этих наиболее распространенных вопросах и заблуждениях, сложившихся вокруг веб-скрапинга, дает развернутое руководство по решению наиболее востребованных задач веб-скрапинга.

Начиная с главы 1, я буду периодически приводить примеры программного кода, чтобы продемонстрировать основные принципы веб-скрапинга. Эти примеры являются общественным достоянием, и их можно свободно использовать (тем не менее соблюдение авторских прав всегда приветствуется). Кроме того, все примеры программного кода будут размещены на веб-сайте для просмотра и скачивания.

## Что такое веб-скрапинг?

Автоматизированный сбор данных из Интернета существует столько же, сколько сам Интернет. Несмотря на то что *веб-скрапинг (web scraping)* не является новым термином, раньше это направление было больше известно под названием *анализ экранных или интерфейсных данных (screen scraping)*, *интеллектуальный анализ данных (data mining)*, *сбор веб-данных (web harvesting)*. Похоже, что на сегодняшний

день общее мнение склоняется в пользу термина *веб-скрапинг (web scraping)*, который я и буду использовать на протяжении всей книги, хотя время от времени буду называть программы веб-скрапинга *роботами (bots)*.

В теории веб-скрапинг – это сбор данных с помощью любых средств, кроме программ, использующих API (или человека, использующего веб-браузер). Чаще всего веб-скрапинг осуществляется с помощью программы, которая автоматически запрашивает веб-сервер, запрашивает данные (HTML и другие файлы, которые размещены на веб-страницах), а затем выполняет парсинг этих данных, чтобы извлечь необходимую информацию.

На практике веб-скрапинг охватывает широкий спектр методов и технологий программирования, таких как анализ данных и информационная безопасность. Эта книга посвящена основам веб-скрапинга и краулинга (часть I)[1] и раскрывает некоторые сложные темы (часть II).

## Зачем нужен веб-скрапинг?

Если для вас единственным способом доступа к Интернету является браузер, вы теряете огромный спектр возможностей. Хотя браузеры удобны для выполнения JavaScript, вывода изображений и представления объектов в более удобочитаемом формате (помимо прочего), веб-скраперы удобны для сбора и обработки больших объемов данных (помимо прочего). Вместо однократного просмотра одной страницы на дисплее монитора вы можете просматривать базы данных, которые уже содержат тысячи или даже миллионы страниц.

Кроме того, веб-скраперы могут проникнуть в такие места, куда традиционные поисковые системы проникнуть не могут. Поиск Google по «cheapest flights to Boston» выдаст множество рекламных сайтов и популярных сайтов заказа авиабилетов. Google возвращает лишь то, что эти веб-сайты сообщают на своих страницах, а не точные результаты в ответ на различные запросы, введенные в системе зака-

---

[1] Обратите внимание, Райан проводит разницу между терминами *веб-скрапер* (*web-scraper*) и *веб-краулер* (*web-crawler*). Веб-скрапер – это программа, которая собирает данные и извлекает нужную информацию с одной или нескольких страниц сайта. Веб-краулер – это программа, которая просто обходит или сканирует страницы на одном или нескольких сайтах. В этом плане веб-краулер является синонимом термина *поисковый робот* (*web-spider*). – *Прим. пер.*

за авиабилетов. Тем не менее правильно разработанный веб-скрапер может собрать данные о ценах на авиабилеты до Бостона за определенный временной интервал на различных веб-сайтах и подсказать оптимальное время для покупки авиабилета.

Вы, вероятно, спросите: «Разве сбор данных – это не то, для чего используется API?» (Если вы не знакомы с API, обратитесь к главе 4.) Ну, тогда API – это фантастическое средство, если вы с его помощью сразу найдете то, что искали. Они могут обеспечить передачу потока данных определенного формата с одного сервера на другой. вы можете использовать API для получения различных видов интересующих вас данных, например, твитов или страниц Википедии. В общем случае лучше использовать API (если он есть), а не создавать программу-робот для сбора тех же самых данных. Тем не менее есть несколько причин, в силу которых использовать API не представляется возможным:

- вы собираете данные с сайтов, которые не имеют единого API;
- данные, которые вас интересуют, компактны, поэтому API не нужен;
- источник данных не имеет инфраструктуры или технических возможностей, позволяющих разработать API.

Даже в тех случаях, когда API *уже есть*, объем обрабатываемых запросов, ограничения скорости обработки запросов, тип или формат возвращаемых данных могут не удовлетворить ваши потребности.

Вот именно здесь и начинается сфера применения скрапинга веб-страниц. За некоторыми исключениями, вы можете просмотреть эти страницы в браузере или получить доступ к ним с помощью скрипта Python. Получив доступ к ним с помощью скрипта, вы можете сохранить их в базе данных. Сохранив их в базе данных, вы можете выполнять любые действия с ними.

Очевидно, что существует очень много практических сфер, где требуется доступ к данным практически неограниченного объема. Рыночное прогнозирование рынка, машинный перевод и даже медицинская диагностика уже извлекли огромную пользу, воспользовавшись возможностью собрать и проанализировать данные новостных сайтов, переведенный контент и сообщения на медицинских форумах.

Даже в мире искусства веб-скрапинг уже открыл новые горизонты для творчества. В рамках проекта 2006 года «We Feel Fine» (http://wefeelfine.org/) Джонатан Харрис и Сеп Камвар провели скрапинг англоязычных блогов для поиска фраз, начинающихся с «I feel» или «I am

feeling». Это позволило построить визуализацию данных, описать, как люди в мире чувствуют себя изо дня на день, с минуты на минуту.

Независимо от вашей предметной области, почти всегда есть способ, благодаря которому веб-скрапинг может повысить эффективность бизнес-практик, улучшить производительность или даже открыть совершенно новое направление в бизнесе.

## Об этой книге

Эту книгу можно рассматривать не только как введение в веб-скрапинг, но и как развернутое руководство по скрапингу веб-данных практически любого типа. Хотя в книге используется язык программирования Python и освещаются основные принципы его работы, ее не следует использовать в качестве вводного пособия по Python.

Если вы не являетесь опытным программистом и не знаете Python вообще, чтение этой книги может быть несколько сложной задачей. Однако если вы опытный программист, то сочтете материал книги легким. В приложении А освещаются установка и работа с Python 3.x, который используется в этой книге. Если вы работали только с Python 2.x или у вас не установлен Python 3.x, вы, возможно, захотите ознакомиться с приложением А.

Если вы ищете более подробные ресурсы по изучению Python, книга *Introducing Python* от Билла Любановича является очень хорошим развернутым руководством. Для тех, у кого мало времени, видеокурс *Intoduction to Python* от Джессики МакКеллер является отличным ресурсом.

Приложение С включает в себя кейсы, а также в нем рассматриваются ключевые вопросы, касающиеся правовых аспектов использования скраперов в США и данных, полученных с их помощью.

В технической литературе внимание часто уделяется конкретному языку или технологии, однако веб-скрапинг является относительно разрозненной предметной областью, которая охватывает такие направления, как работа с с базами данных, веб-серверами, HTTP, HTML, интернет-безопасность, обработка изображений, наука о данных (data science) и др. Эта книга стремится осветить все эти направления в том объеме, в каком это требуется для сбора данных в Интернете.

В первой части подробно рассказывается о веб-скрапинге и веб-краулинге, значительное внимание уделено работе библиотек, использующихся в книге. Первую часть книги можно использовать в качестве развернутого справочного пособия по этим библиотекам

и методам (за некоторыми исключениями, когда будут приводиться дополнительные источники).

Во второй части освещаются дополнительные темы, которые могут пригодиться читателю при написании веб-скрапера. К сожалению, эти темы являются слишком широкими, чтобы подробно рассмотреть их в одной главе. Поэтому часто будут приводиться ссылки на другие ресурсы для получения дополнительной информации.

Структура данной книги составлена так, что можно легко перемещаться по главам в поисках необходимой информации. Если какой-то принцип или фрагмент программного кода связан с другим принципом или фрагментом, уже упоминавшимся в предыдущей главе, я обязательно сошлюсь на соответствующий раздел.

## Типографские соглашения

В этой книге применяются следующие типографские соглашения:

*Курсив*
: Используется для обозначения новых терминов, URL-адресов, адресов электронной почты, имен файлов и расширений файлов.

`Моноширинный шрифт`
: Используется для листингов программ, а также внутри параграфов для обозначения элементов программ (названий переменных или функций, баз данных, типов данных, переменных среды, операторов и ключевых слов).

**`Моноширинный жирный шрифт`**
: Обозначает команды или другой текст, который должен вводиться пользователем.

*`Моноширинный курсив`*
: Обозначает текст, который должен замещаться фактическими значениями, вводимыми пользователем или определяемыми из контекста.


Этот элемент означает совет или подсказку.


Этот элемент означает примечание.


Этот элемент означает предупреждение или предостережение.

## Использование примеров программного кода

Все примеры программного кода и упражнения, что приводятся в этой книге, доступны для скачивания по адресу http://pythonscraping.com/code/.

Данная книга призвана оказать вам помощь в решении задач, связанных с веб-скрапингом. Вы можете свободно использовать примеры программного кода из этой книги в своих программах и документации. Вам не нужно обращаться в издательство за разрешением, если вы не собираетесь воспроизводить существенные части программного кода. Например, если вы разрабатываете программу и используете в ней несколько фрагментов программного кода из книги, вам не нужно обращаться за разрешением. Однако в случае продажи или распространения компакт-дисков с примерами из этой книги вам необходимо получить разрешение от издательства O'Reilly. Если вы отвечаете на вопросы, цитируя данную книгу или примеры из нее, получения разрешения не требуется. Но при включении значительного объема программного кода из этой книги в Вашу документацию необходимо получить разрешение издательства.

Мы приветствуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN. Например: *Web Scraping with Python* by Ryan Mitchell (O'Reilly). Copyright 2015 Ryan Mitchell, 978-1-491-91029-0.

Если вы считаете, что использование вами примеров программного кода выходит за разрешенные рамки, присылайте свои вопросы на нашу электронную почту permissions@oreilly.com.

## Благодарности

Подобно тому, как наилучшие продукты появляются благодаря постоянной обратной связи с пользователями, так и выход этой книги в той или иной форме был бы невозможен без участия большого числа коллег, вдохновителей и редакторов. Спасибо сотрудникам O'Reilly за всестороннюю поддержку этой несколько необычной темы, а также моим друзьям и семье, которые давали советы и вынуждены были слушать мои импровизированные чтения, моих коллегам из компании LinkeDrive, которой я, вероятно, задолжала уйму рабочего времени.

Я выражаю благодарность, в частности, Эллисону Макдональду (Allyson MacDonald), Брайану Андерсону (Brian Anderson), Мигелю Гринбергу (Miguel Grinberg) и Эрику ВанВику (Eric VanWyk) за обратную связь, рекомендации и иногда жесткую критику из лучших побуждений. Довольно большое количество разделов и примеров программного кода явилось прямым результатом их вдохновляющих предложений.

Я выражаю благодарность Йейлу Шпехту (Yale Specht), изначально сподвигнувшему меня на этот проект и вносившему стилистические поправки по мере написания книги, за его безграничное терпение на протяжении последних девяти месяцев. Без него эта книга была бы написана в два раза быстрее, но не была бы столь полезной.

И наконец, я выражаю благодарность Джиму Вальдо (Jim Waldo), с которого все, в общем-то, и началось, когда он отправил по почте молодому и впечатлительному подростку компьютер с Linux и книгу *The Art and Science of C*.

# ЧАСТЬ I

# ПОСТРОЕНИЕ СКРАПЕРОВ

В этой части книги основное внимание уделено основным принципам работы веб-скраперов: как использовать Python, чтобы получить информацию с веб-сервера, как выполнить обработку ответа сервера и как начать работать с веб-сайтом в автоматическом режиме. В итоге вы будете с легкостью собирать данные по всему Интернету, создавая скраперы, которые могут переходить с одного домена на другой, собирать информацию и сохранять ее для дальнейшего использования.

Честно говоря, веб-скрапинг – это фантастическое поле деятельности, которое обязательно нужно освоить, если вы хотите получить огромный выигрыш при сравнительно небольших первоначальных вложениях. По всей вероятности, 90% проектов по веб-скрапингу, с которыми вы столкнетесь, будут опираться на методы, используемые лишь в первых шести главах. В этой части освещаются темы, которые чаще всего возникают в головах людей (хотя и технически подкованных) при упоминании слова «веб-скрапер»:

- сбор HTML-данных с домена;
- парсинг данных с целью получения интересующей информации;
- хранение извлеченной информации;
- перемещение на другую страницу для повторения процесса (опционно).

Эта часть книги составит прочную основу ваших знаний, прежде чем вы перейдете к более сложным проектам, рассмотренным во второй части. Не обманывайте себя, думая, что проекты, приведенные в первой части, не столь важны, в отличие от некоторых сложных проектов, рассмотренных во второй части. При разработке скраперов вы будете регулярно использовать информацию, изложенную в первой части книги.

# Глава 1

# Ваш первый скрапер

Приступая к веб-скрапингу, начинаешь ценить все мелкие настройки, kotcisesopые выполняют браузеры за нас. Веб, лишенный HTML-форматирования, CSS-стилей, JavaScript и рендеринга изображений, может поначалу немного напугать своим видом, но в этой главе, а также в следующей мы расскажем, как форматировать и интерпретировать данные без помощи браузера.

Эта глава сначала расскажет, как отправить GET-запрос к веб-серверу на сканирование конкретной страницы, считав HTML-вывод и выполнив извлечение данных так, чтобы собрать только интересующий нас контент.

## Соединение с Интернетом

Если вы не занимались организацией сетей или сетевой безопасностью, то работа Интернета может показаться вам немного таинственной. Мы не задумываемся о том, что, собственно, сеть делает каждый раз, когда мы открываем браузер и переходим на http://google.com, да и сейчас это нам не нужно. На самом деле я бы назвала фантастикой тот факт, что компьютерные интерфейсы достигли такого совершенства, что большинство пользователей Интернета не имеют ни малейшего представления о том, как он работает.

Однако скрапинг следует рассматривать не только как веб-интерфейс, лишь на уровне браузера (в плане обработки всех этих HTML, CSS и JavaScript), он также связан с типом сетевого соединения.

Чтобы дать вам некоторое представление об инфраструктуре, которая используется для загрузки информации в ваш браузер, приведем следующий пример. У Алисы есть веб-сервер. Боб использует настольный компьютер, который пытается подключиться к серверу Алисы. Когда одна машина хочет подсоединиться к другой, происходит следующий обмен:

1. Компьютер Боба посылает последовательность битов, представленных в виде низкого и высокого напряжений. Запрос Боба

разбит на фрагменты, к каждому фрагменту добавлен заголовок со служебной информацией (этим заведует протокол TCP). Передачей отдельных фрагментов от компьютера Боба до компьютера Алисы заведует протокол IP.

2. Локальный маршрутизатор Боба получает эту последовательность и интерпретирует ее как пакет с помощью собственного MAC-адреса и направляет на IP-адрес Алисы. Маршрутизатор заменяет в заголовке пакета обратный адрес на свой и посылает пакет дальше.
3. Пакет Боба проходит несколько промежуточных серверов, которые направляют его по правильному физическому/проводному пути на сервер Алисы.
4. Сервер Алисы получает пакет на свой IP-адрес.
5. Сервер Алисы считывает порт назначения пакета (почти всегда это порт 80 для веб-приложений, это что-то вроде «номера квартиры» в пакетной передаче данных, где IP-адрес является «улицей») в заголовке и передает его в соответствующее приложение – приложение веб-сервера.
6. Веб-сервер принимает поток данных от серверного процессора. Эти данные говорят что-то вроде:
   - это GET-запрос;
   - запрашивается следующий файл: index.html.
7. Веб-сервер находит соответствующий HTML-файл, записывает его в новый пакет для отправки Бобу и посылает его через свой локальный маршрутизатор обратно на компьютер Боба точно таким же вышеописанным способом.

И вуаля! У нас есть Интернет.

Итак, где в этом обмене задействован браузер? Абсолютно нигде. На самом деле браузеры – это относительно недавнее изобретение в истории Интернета, первый браузер Nexus появился в 1990 году.

Да, веб-браузер – очень полезная программа для создания пакетов информации, их отправки и интерпретации в виде красивых картинок, звуков, видео и текста. Однако веб-браузер – это просто код, а код можно разбить на части, выделить основные компоненты, перезаписать, повторно использовать и вообще сделать все, что угодно. Веб-браузер сообщает процессору о том, что нужно отправить некоторые данные в приложение, которое использует беспроводной (или проводной) интерфейс, однако библиотеки, предлагаемые различными языками программирования, могут выполнить то же самое.

Теперь покажем, как это делается в Python:

```
from urllib.request import urlopen
html = urlopen("http://pythonscraping.com/pages/page1.html")
print(html.read())
```

Вы можете сохранить этот код как *scrapetest.py* и запустить его в своем терминале с помощью команды:

```
$python scrapetest.py
```

Обратите внимание, если у вас дополнительно установлен Python 2.x, вам, возможно, потребуется явно вызывать Python 3.x с помощью команды:

```
$python3 scrapetest.py
```

Она выведет полный HTML-код страницы http://bit.ly/1QjYgcd. Если говорить более точно, она выводит HTML-файл *page1.html*, найденный в каталоге *<web root>/pages* на сервере, расположенном в домене http://pythonscraping.com.

В чем заключается разница? Большинство современных веб-страниц содержат файлы различных форматов. Это могут быть файлы изображений, файлы JavaScript, CSS-файлы или любой другой контент, который содержит запрашиваемая страница. Когда веб-браузер находит тег, например `<img src="cuteKitten.jpg">`, он отправляет еще один запрос на сервер, чтобы получить данные из файла *cuteKitten.jpg* для корректного отображения страницы. Имейте в виду, что наш скрипт Python пока не может вернуться обратно и запросить несколько файлов, он может прочитать только один запрошенный нами HTML-файл.

Итак, как это сделать? Строка

```
from urllib.request import urlopen
```

делает то, что написано, – импортирует функцию `urlopen` из модуля `request` библиотеки `urllib`.

**urllib или urllib2?**

Используя библиотеку urllib2 в Python 2.x, вы, возможно, заметили некоторые отличия между urllib2 и urllib. В Python 3.x urlib2 была переименована в urllib и разбита на несколько подмодулей: `urllib.request`, `urllib.parse` и `urllib.error`. Несмотря на то что названия функций преимущественно остались теми же, вам, возможно, стоит узнать, какие функции включены в подмодули новой urllib.

`urllib` – стандартная библиотека Python (то есть вам не нужно устанавливать ничего лишнего, чтобы запустить этот пример) и со-

держит функции для запроса данных в сети, обработки cookies и даже изменения метаданных (заголовков и пользовательского агента). На протяжении всей книги мы будем использовать `urllib` довольно часто, поэтому рекомендуем вам прочитать документацию Python по этой библиотеке (*http://bit.ly/1FncvYE*).

Функция `urlopen` используется, чтобы открыть удаленный объект в сети и прочитать его. Поскольку она имеет достаточно широкое применение (она может с легкостью прочитать HTML-файлы, файлы изображений или любой другой поток файлов), мы будем использовать ее довольно часто на протяжении всей книги.

# Введение в BeautifulSoup

*«Красивый суп, столь густой и зеленый,*
*Ожидающий в горячем глубоком блюде!*
*Кто для таких лакомств не наклонился бы?*
*Вечерний суп, красивый суп!»*

Библиотека BeautifulSoup была названа в честь одноименного стихотворения, входящего в сказку Льюиса Кэролла «Приключения Алисы в Стране чудес». В сказке это стихотворение произносит персонаж по имени Черепаха Квази (Mock Turtle)[1].

Как и ее тезка, библиотека BeautifulSoup пытается придать смысл бессмыслице, она помогает отформатировать и систематизировать грязные интернет-данные, исправляя плохо размеченные HTML-страницы и выводя их в виде легко обрабатываемых объектов Python, представляющих собой XML-структуры.

## Установка BeautifulSoup

Поскольку библиотека BeautifulSoup не является библиотекой Python по умолчанию, ее нужно установить. На протяжении всей книги мы будем использовать библиотеку BeautifulSoup 4 (также известную как BS4). Полные инструкции по установке BeautifulSoup 4 можно найти на Crummy.com. Однако для Linux основным методом будет:

```
$sudo apt-get install python-bs4
```

а для Mac:

```
$sudo easy_install pip
```

---

[1] Mock Turtle Soup – название блюда-подделки, популярного в Викторианскую эпоху, которое имитировало дорогой черепаший суп, но готовилось из телятины.

Эта команда устанавливает питоновский менеджер пакетов *pip*. Затем выполните следующее:

```
$pip install beautifulsoup4
```

чтобы установить библиотеку.

И снова обратите внимание, что если вы используете на вашем компьютере Python 2.x и 3.x, вам, возможно, потребуется явно вызвать python3:

```
$python3 myScript.py
```

Убедитесь, что используете `python3` при установке пакетов или пакеты можно установить под Python 2.x, а не под Python 3.x:

```
$sudo python3 setup.py install
```

При использовании pip вы можете также вызвать `pip3`, чтобы установить пакеты для Python 3.x:

```
$pip3 install beautifulsoup4
```

Установка пакетов в Windows практически идентична установке пакетов для Mac и Linux. Загрузите последнюю версию BeautifulSoup 4, воспользовавшись вышеприведенной ссылкой, зайдите в распакованную папку и выполните:

```
>python setup.py install
```

Вот и все! BeautifulSoup теперь используется в качестве библиотеки Python на вашем компьютере. Вы можете проверить это, открыв терминал Python и импортировав библиотеку:

```
$python
> from bs4 import BeautifulSoup
```

Импорт должен быть выполнен без ошибок.

Кроме того, есть exe-инсталлятор для *pip* под Windows, поэтому вы можете легко установить пакеты и работать с ними:

```
>pip install beautifulsoup4
```

**Сохранение библиотек с помощью виртуальных окружений**

Если вам необходимо работать одновременно с несколькими проектами Python, или вам нужен способ, с помощью которого можно легко связать проекты с использованными библиотеками, или вы хотите избежать возможных конфликтов между уже имеющимися библиотеками, можно установить виртуальную среду Python, чтобы хранить проекты по отдельности и с легкостью управлять ими.

Если вы устанавливаете библиотеку Python без виртуального окружения, вы устанавливаете ее *глобально*. Обычно для этого надо иметь права администратора или root-пользователя. К счастью, создать виртуальное окружение довольно просто:

```
$ virtualenv scrapingEnv
```

Эта команда создает новое окружение *scrapingEnv*, которое нужно активировать для использования:

```
$ cd scrapingEnv/
$ source bin/activate
```

После активации окружения вы увидите, что название окружения появилось в командной строке, напоминая, что теперь вы работаете с ним. Любые установленные библиотеки или запущенные скрипты теперь будут храниться и выполняться только в данном виртуальном окружении.
Работая в только что созданном окружении scrapingEnv, я могу установить и использовать библиотеку BeautifulSoup, например:

```
(scrapingEnv)ryan$ pip install beautifulsoup4
(scrapingEnv)ryan$ python
> from bs4 import BeautifulSoup
>
```

Я могу покинуть окружение с помощью команды деактивации, после чего у меня уже не будет доступа к библиотекам, установленным внутри этого окружения:

```
(scrapingEnv)ryan$ deactivate
ryan$ python
> from bs4 import BeautifulSoup
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: No module named 'bs4'
```

Изолирование библиотек для каждого отдельного проекта позволяет заархивировать все окружение в папку и отправить ее кому-то еще. Если у этих пользователей на их компьютерах установлена та же самая версия Python, они смогут работать с вашим кодом в данной виртуальной среде, не прибегая к инсталляции других библиотек.
Несмотря на то что в рамках этой книги мы не будем явно рассказывать, как использовать виртуальное окружение, имейте в виду, что вы можете воспользоваться виртуальным окружением в любой момент, заранее активировав его.

## Запуск BeautifulSoup

Наиболее часто используемым объектом в библиотеке BeautifulSoup является, соответственно, объект BeautifulSoup. Давайте посмотрим на него в действии, изменив пример, приведенный в начале этой главы:

```
from urllib.request import urlopen
from bs4 import BeautifulSoup
html = urlopen("http://www.pythonscraping.com/pages/page1.html")
bsObj = BeautifulSoup(html.read());
print(bsObj.h1)
```

Вывод выглядит так:

```
<h1>An Interesting Title</h1>
```

Как и в примере выше, мы импортируем библиотеку `urlopen` и вызываем функцию `html.read()`, чтобы получить HTML-контент страницы. Затем этот HTML-контент мы преобразуем в объект BeautifulSoup со следующей структурой:

- **html** → <html><head>...</head><body>...</body></html>
  - **head** → <head><title>A Useful Page<title></head>
    - **title** → <title>A Useful Page</title>
  - **body** → <body><h1>An Int...</h1> Lorem ip... </body>
    - **h1** → <h1>An Interesting Title</h1>
    - **div** → Lorem Ipsum dolor...

Обратите внимание, что тег `<h1>`, который мы извлекли из страницы, был вложен в два слоя внутри объекта BeautifulSoup (html → body → h1). Однако после излечения тега h1 из объекта мы обращаемся к нему напрямую:

```
bsObj.h1
```

Фактически любой вызов функции из нижеприведенного списка сгенерирует один и тот же вывод:

```
bsObj.html.body.h1
bsObj.body.h1
bsObj.html.h1
```

Мы надеемся, что этот небольшой пример использования `BeautifulSoup` дал вам представление о возможностях и простоте использования этой библиотеки. Практически любую информацию можно извлечь из любого файла HTML (или XML), при условии что она заключена в определенный идентифицирующий тег или расположена рядом с ним. В главе 3 мы подробнее разберем вызов более сложных функций BeautifulSoup, а также рассмотрим регулярные выражения и покажем, как их можно использовать вместе с `BeautifulSoup` для извлечения информации с веб-сайтов.

## Как обеспечить надежный скрапинг

Веб не идеален. Данные плохо отформатированы, веб-сайты падают, закрывающие теги отсутствуют. Самая неприятная ситуация, которая случается при использовании веб-скрапера, – вы пошли спать, запустив скрапер в надежде, что на следующий день он соберет все данные в базу, однако, проснувшись, видите, скрапер выдал ошибку при обработке данных неизвестного формата и прекратил свою работу вскоре после того, как вы оторвали свой взгляд от монитора. В таких ситуациях вы, возможно, проклинаете разработчика веб-сайта (и странно отформатированные данные), но на самом деле обвинять вы должны самого себя, изначально не надеясь на счастливый случай!

Давайте взглянем на первую строку нашего скрапера (расположена после операторов импорта) и выясним, как обработать сгенерированные исключения:

```
html = urlopen("http://www.pythonscraping.com/pages/page1.html")
```

Есть две основные ошибки, которые могут возникнуть здесь:

- страница не найдена на сервере (или есть какая-то ошибка при ее получении);
- сервер не найден.

В первом случае будет возвращена ошибка HTTP. Ошибкой HTTP может быть «404 Page Not Found», «500 Internal Server Error» и т. д. Во всех этих случаях функция `urlopen` генерирует общее исключение «HTTPError». Мы можем обработать это исключение следующим образом:

```
try:
    html = urlopen("http://www.pythonscraping.com/pages/page1.html")
except HTTPError as e:
    print(e)
    #возвратить null, прервать или выполнять операции по "Плану Б"
else:
    #программа продолжает работу. Примечание: если возвращаете или
    прерываете в #exception catch, оператор "else" использовать
    не нужно
```

Если вернулся код ошибки HTTP, программа выводит сообщение об ошибке, и оставшаяся часть программы для оператора `else` не выполняется.

Если сервер не найден вообще (например, http://www.pythonscraping.com упал или URL-адрес набран неправильно), функция `urlopen` возвращает объект `None`. Этот объект аналогичен значению `null`, исполь-

зуемому в других языках программирования. Дополнительно мы можем проверить, является ли возвращенная html-страница объектом None:

```
if html is None:
    print("URL is not found")
else:
    #программа продолжает работу
```

Конечно, даже если страница успешно получена с сервера, остается еще вопрос, связанный с тем, что контент страницы может совершенно не соответствовать нашим ожиданиям. Каждый раз, когда вы обращаетесь к тегу в объекте BeautifulSoup, полезно удостовериться в том, что этот тег существует на самом деле. Если вы попытаетесь обратиться к тегу, который не существует, BeautifulSoup вернет объект None. При попытке получить тег объекта None будет сгенерировано исключение AttributeError.

Следующая строка (где nonExistentTag – это выдуманный тег, а не название действительной функции BeautifulSoup):

```
print(bsObj.nonExistentTag)
```

возвращает объект None. Этот объект вполне пригоден для обработки и проверки. Проблема возникнет, если вы, вместо того чтобы проверить объект, продолжите работу и попытаетесь вызвать некоторые другие функции для объекта None, как показано в следующей строке:

```
print(bsObj.nonExistentTag.someTag)
```

которая вернет исключение:

```
AttributeError: 'NoneType' object has no attribute 'someTag'
```

Итак, как мы можем избежать этих двух ситуаций? Самый простой способ – явно протестировать возможность возникновения обеих ситуаций:

```
try:
    badContent = bsObj.nonExistingTag.anotherTag
except AttributeError as e:
    print("Tag was not found")
else:
    if badContent == None:
       print ("Tag was not found")
    else:
        print(badContent)
```

Проверка и обработка каждой ошибки кажется поначалу трудоемкой, однако этот код легко реорганизовать, сделать простым в написании (и, что более важно, менее трудным для чтения). Например, код, приведенный ниже, – это тот же самый наш скрапер, записанный немного по-другому:

```
from urllib.request import urlopen
from urllib.error import HTTPError
from bs4 import BeautifulSoup
 def getTitle(url):
    try:
        html = urlopen(url)
    except HTTPError as e:
        return None
    try:
        bsObj = BeautifulSoup(html.read())
        title = bsObj.body.h1
    except AttributeError as e:
        return None
    return title
title = getTitle("http://www.pythonscraping.com/pages/page1.html")
if title == None:
    print("Title could not be found")
else:
    print(title)
```

В этом примере мы создали функцию `getTitle`, которая возвращает либо название страницы, либо объект `None`, если с извлечением страницы возникла какая-то проблема. Как и в предыдущем примере, внутри `getTitle` мы проверяем возможность получения `HTTPError`, а также инкапсулируем две строки BeautifulSoup в оператор `try`. `AttributeError` может быть сгенерирована любой из этих строк (если сервер не существует, `html` будет объектом `None`, а `html.read()` сгенерирует `AttributeError`). На самом деле мы могли бы включить в оператор `try` столько строк, сколько нам нужно, или вызвать вообще другую функцию, которая может сгенерировать `AttributeError` в любой момент.

При написании скрапера важно позаботиться об общей структуре Вашего кода, чтобы обработать исключения и сделать его читаемым. Кроме того, вы, вероятно, захотите повторно использовать код. С помощью функций `getSiteHTML` и `getTitle` (в сочетании с тщательной обработкой исключений) скрапинг веб-сайтов становится быстрым и надежным.