

Методы сбора и обработки данных при помощи Python

Scrapy

# На этом уроке

1. Познакомимся с фреймворком Scrapy.
2. Научимся работать с ним, используя разные шаблоны для парсинга.

**Оглавление**

[Scrapy](#_2et92p0)

[Как устроен Scrapy?](#_kaep9waov0d6)

[Установка](#_tyjcwt)

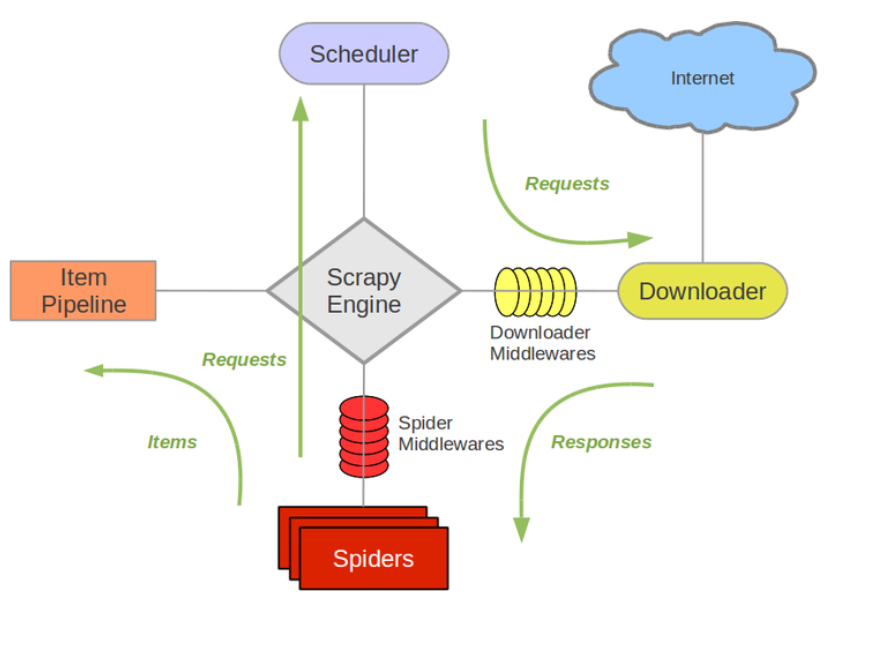
[Создание поискового робота](#_3dy6vkm)

[Домашнее задание](#_1t3h5sf)

[Используемая литература](#_4d34og8)

# Scrapy

Scrapy — один из наиболее популярных и производительных фреймворков Python для получения данных с веб-страниц. Он включает в себя большинство общих функциональных возможностей. Вам не придётся самостоятельно прописывать многие функции. Scrapy позволяет быстро и без труда создать «веб-паука».



## Как устроен Scrapy?

**Движок Scrapy (Scrapy Engine)** — отвечает за обработку элементов после их извлечения или очистки «пауками». Типичные задачи включают очистку, проверку и сохранение. Например, сохранение элемента в базе данных. Движок отвечает за координацию между всеми компонентами.

**Планировщик (Scheduler)** —получает запросы от движка и ставит их в очередь для передачи позже (также в движок), когда движок их запрашивает. По сути, планировщик определяет порядок операций.

**Загрузчик (Downloader)** — отвечает за получение веб-страниц и передачу их движку, который, в свою очередь, передаёт их поисковым роботам.

**Пауки (Spiders)** —это настраиваемые классы, написанные пользователями Scrapy для парсинга данных. Каждый паук может обрабатывать определённый домен или группу доменов.

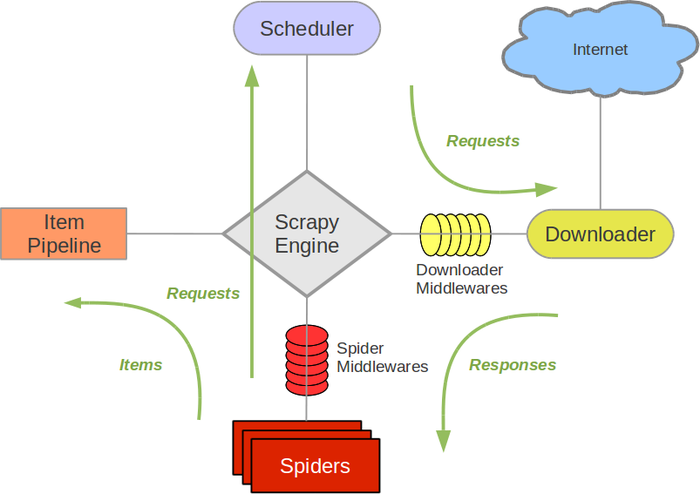
**Конвейер элементов (Item Pipelines)** — отвечает за обработку элементов после их извлечения или очистки «пауками». Типичные задачи включают очистку, проверку и сохранение. Например, сохранение элемента в базе данных.

**Промежуточное программное обеспечение загрузчика (Downloader middlewares)** —отвечают за загрузку разметки сайта. Они предоставляют удобный механизм расширения функциональности Scrapy путём добавления пользовательского кода.

**Промежуточное программное обеспечение паука (Spider middlewares)** — отвечают за возврат данных. Они предоставляют удобный механизм для расширения функциональности Scrapy путём добавления пользовательского кода. В Spider middlewares находятся пользовательские заголовки (headers), а также проксирование.

**Поток данных в Scrapy** контролируется движком и выглядит следующим образом:

1. Движок открывает домен, находит паука, который обрабатывает этот домен, и запрашивает у паука первые URL-адреса для сканирования.
2. Движок получает от паука первые URL-адреса для обхода и планирует их в планировщике как запросы.
3. Движок запрашивает у планировщика следующие URL-адреса для сканирования.
4. Планировщик возвращает следующие URL-адреса для сканирования в движок, а движок отправляет их в загрузчик, проходя через downloader middleware.
5. Как только страница завершает загрузку, загрузчик генерирует ответ (с этой страницей) и отправляет его в движок, проходя через downloader middleware.
6. Движок получает ответ от загрузчика и отправляет его на обработку пауку, проходя через spider middleware.
7. Паук обрабатывает ответ и возвращает очищенные элементы и новые запросы (для последующих) в движок.
8. Движок отправляет очищенные элементы (возвращённые пауком) в конвейер элементов и запросы (возвращённые пауком) в планировщик.
9. Процесс повторяется (начиная с шага 2) до тех пор, пока не перестанут поступать запросы от планировщика, и движок не закроет домен.

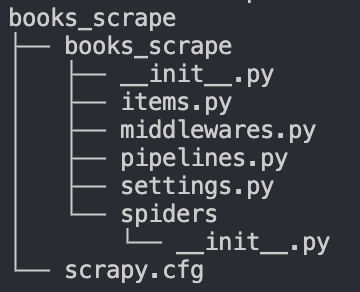
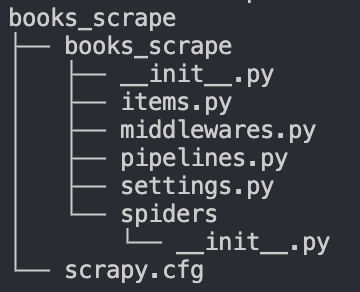
officeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt objectofficeArt object

## Установка

Пакет Scrapy можно найти в PyPI (Python Package Index, также известен как pip) — поддерживаемом сообществом репозитории для всех вышедших пакетов Python.

| pip install scrapy |
| --- |

После установки создадим проект scrapy. Для этого введем scrapy startproject и название проекта books\_scrape — будем парсить книги с уже знакомого нам сайта. У нас создалась папка books\_scrape с вот такой структурой:

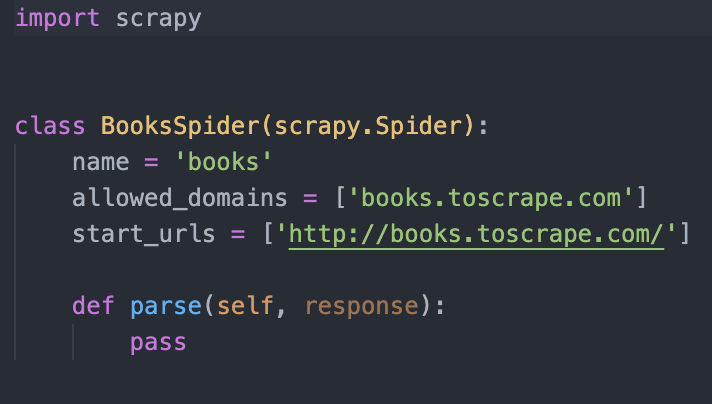


# Создание поискового робота

Затем нам надо создать паука, который будет парсить сайт. Для этого внутри созданной папки books\_scrape (корневой для следующей папки books\_scrape и файла scrapy.cfg) мы должны ввести команду “scrapy genspider”, потом имя паука, пусть будет “books”, а затем — указать ссылку на сайт, который мы хотим спарсить. Обратите внимание, что ссылка должна быть без *https* и последней слеш в конце, например, [books.toscrape.com](http://books.toscrape.com).

В итоге ввод будет выглядеть так: “scrapy genspider books [books.toscrape.com](http://books.toscrape.com)”.

Внутри папки books, а затем spiders, у нас появился файл books.py со следующей структурой:



Где **name** — это имя паука, которое мы будем использовать для его запуска. В **allowed\_domains** лежит домен, который паук сможет парсить. Если вы сделаете ссылку [toscrape.com](http://toscrape.com), то паук books сможет парсить и сайт [quotes.toscrape.com](http://quotes.toscrape.com). Запомните, что в allowed domains никогда нельзя оставлять обозначение протоколов http и https. В **start\_urls** лежит ссылка, с которой будет начат парсинг. Так как у нас сайт с протоколом https, добавим “s" к этой ссылке. В методе **parse()** вернётся response — то, что будет спарсено из start\_urls.

Давайте спарсим первую страницу уже известного нам сайта и запустим паука. Для начала получим название, обложку, цену и наличие каждой книги на первой странице. В итоге у нас должен получиться список из двадцати объектов.

Создадим переменную books и получим из response все книги сразу. Сначала найдём, где у нас лежат книги. Каждая книга лежит в теге <li>, а все эти теги вложены в тег <ol> с классом row. Давайте пропишем это в нашем пауке. Для извлечения будем использовать метод xpath:

**books = response.xpath(«//ol[@class='row']/li")**

То есть получаем все теги <li>, вложенные в тег <ol> с классом row. Теперь у нас есть список книг, по которому мы должны пройти, чтобы извлечь информацию о каждой книге.

Создадим цикл:

**for book in books:**

И сделаем сразу вывод данных:

**yield {**

**'image': book.xpath(),**

**'title': book.xpath(),**

**'price': book.xpath(),**

**'instock': book.xpath()**

**}**

Смотрите, мы обращаемся к каждой книге, то есть проваливаемся в html-разметку конкретно одной книги в цикле for. Теперь запишем пути для каждой переменной. Обратите внимание, что начинать путь надо с точки, таким образом мы говорим scrapy, что нас интересует конкретно этот элемент разметки и его дочерние элементы. Если точку не поставить, Scrapy будет искать xpath по всему сайту и, в данном случае, выведет название, цену, обложку и наличие одной и той же книги 20 раз.

Поищем, где у нас лежат все нужные нам переменные. Ссылка на изображение у нас лежит в теге <div> с классом image\_container.Внутри — тег <а>, а внутри него — <img>. Из этого тега <img> нам надо достать атрибут src. Запишем xpath. Чтобы получить содержимое атрибута после метода xpath, надо использовать метод get(). Есть ещё метод getall(), к нему мы вернемся чуть позже.

**'image': book.xpath(«.//div[@class='image\_container']/a/img/@src").get(),**

Как мы помним и видим сейчас на сайте, ссылка на картинку у нас не полная. В предыдущих уроках мы добавляли корневую ссылку руками. В Scrapy есть специальная функция для этого. Чтобы к извлеченной ссылке добавить [books.toscrape.com](http://books.toscrape.com), достаточно способ извлечения обернуть в response.urljoin(), который добавит ссылку из allowed\_domains к тому, что мы извлекли:

**'image': response.urljoin(book.xpath(«.//div[@class=‘image\_container’]/a/img/@src»).get()),**

Далее нам надо получить название. Оно лежит в теге a, который находится внутри тега h3. Из тега а нам надо получить атрибут title.

**'title': book.xpath(«.//h3/a/@title").get(),**

Теперь получаем цену. Она лежит в теге p с классом price\_color и нам надо получить текст этго тега. К тексту мы обращаемся с использованием круглых скобок:

**'price': book.xpath(«.//p[@class='price\_color']/text()").get()**

Последнее, что нам надо получить — это информацию о наличии в магазине. Она лежит в теге p с классом instosk и классом available. И нам надо получить текст этого тега. Давайте запишем:

**'instock': book.xpath(«.//p[contains(@class, ‘instock’)]/text()»).get()**

Теперь у нас написано всё, чтобы получить список всех 20 книг с первой страницы. Давайте запустим паука. Для этого нам надо, находясь в той же директории, что и файл scrapy.cfg написать команду для запуска парсера scrapy crawl и имя паука, в данном случае — books.

Мы видим, что парсер запустился. Вы даже можете видеть, как проскочили какие-то данные. Scrapy хорош тем, что в него уже встроена асинхронность, так что парсинг занимает существенно меньше времени, чем с помощью того же requests. После завершения работы паука мы видим вывод, в котором содержится много разной информации. Тут есть и время завершения работы, и время начала. Основное, что нас интересует — это данные **item\_scraped\_count**.Как видите, тут стоит цифра 20, то есть наш парсер собрал 20 элементов.

Теперь давайте поднимемся выше и посмотрим, что же у нас собралось. Мы видим словари, везде вроде бы всё правильно, кроме значения instock. Тут у нас символ новой строки вместо текста “in stock”. Вернёмся на сайт и посмотрим, в чём же может быть дело. Как видите, текст состоит не только из букв, но и из пустых строчек. Чтобы собрать всё это, надо использовать метод getall() вместо get(). В таком случае нам будет возвращён словарь. Мы сможем этот словарь объединить и удалить переносы строк. Давайте исправим наш код. Сделаем это в отдельной переменной, чтобы возвращать уже чистую.

**instock = ''.join(book.xpath(".//p[contains(@class, ‘instock’)]/text()").getall()).strip()**

**‘instock': instock**

Снова запускаем паука. Смотрим — собрано 20 элементов, и теперь в ключе instock находится верное значение. Так, отлично. Теперь сохраним данные в файл. Для этого при запуске паука достаточно указать опцию *-o* и имя файла, в который мы хотим сохранить данные. Давайте сначала сохраним наши данные в csv:

**scrapy crawl books -o books.csv**

Проверяем — всё правильно. Теперь сохраним данные в json.

**scrapy crawl books -o books.json**

Смотрим файл. Видите, у нас тут вместо знака доллара стоит неправильный символ юникода. Чтобы избежать таких ошибок, идём в файл settings.py и добавляем там поддержку юникода:

**FEED\_EXPORT\_ENCODING = ‘UTF-8’**

Удалим json и запустим паука ещё раз. Проверяем. Теперь знак доллара указан верно. У нас в файле 20 книг с первой страницы сайта. Отлично.

Давайте я вам расскажу ещё немного про файл settings.py. Во-первых, тут можно сделать так, чтобы scrapy не следовал указаниям файла robots.txt сайта. Мы об этом файле говорили на прошлых лекциях. Так как мы с вами парсим сайт, у которого нет этого файла, то ROBOTSTXT\_OBEY = True

в настройках нам никак не мешает. Но чаще всего вы будете сталкиваться с тем, что эта настройка будет вам мешать: Scrapy не будет парсить страницы, которые не разрешено парсить в файле robots.txt. Чтобы заставить Scrapy парсить то, что нужно вам, измените значение на False.

Ещё есть два параметра:

CONCURRENT\_REQUESTS = 32

DOWNLOAD\_DELAY = 3

Первый позволяет ограничить число одновременных запросов к сайту — это к вопросу об этичном парсинге. Второй делает паузу между запросами на столько секунд, на сколько вы укажете. Также в настройках можно менять user agent, включать и отключать куки и многое другое. Подробнее об остальных параметрах и их значениях вы можете почитать, конечно же, в документации.

Теперь давайте поговорим о том, как спарсить все книги со всех страниц. Давайте внутри уже существующего паука создадим переменную next\_page. В неё положим ссылку на следующую страницу. Она лежит внутри тега а с текстом “next”, внутри тега <li> c классом next.

**next\_page = response.xpath("//li[@class='next']/a[contains(text(), ‘next')]/@href").get()**

Если следующая страница существует, будем переходить по ней, используя метод Request, и вызывать метод parse, который мы с вами только что создали.

**if next\_page:**

**next\_page\_link = response.urljoin(next\_page)**

**yield scrapy.Request(url=next\_page\_link, callback=self.parse)**

Запустим парсер. Надо немного подождать. Как видите, у нас собрана 1000 книг. На каждой странице по 0 книг, всего 50 страниц — всё правильно.

Ещё у нас на сайте есть возможность провалиться внутрь каждой книги, если перейти по ссылке, которая лежит в названии. Давайте посмотрим, что там внутри. Как видите, тут есть не только название, но и много других разных данных о книге. Очень часто бывает, что надо спарсить всю информацию о товаре, а значит, переходить по ссылке на каждый товар. И никто не отменял того, что надо собрать всю тысячу товаров, а не первые 20. В таком случае нам на помощь придёт специальный шаблон Scrapy.

Давайте создадим нового паука и теперь в опции укажем *-t* (template) и название шаблона — crawl, — а затем имя паука и снова ссылку на главную страницу:

**scrapy genspider -t crawl pages** [**books.toscrape.com**](http://books.toscrape.com)

У нас в папке spiders появился ещё один паук. Открываем. Как видите, у него немного другая структура. Тут появилась переменная rules, в которой у нас содержатся разные правила. Мы будем использовать эти правила, чтобы переходить на следующие страницы и проваливаться внутрь каждой книги.

Для начала напишем правило перехода на следующую страницу. Для этого внутри объекта Rules запишем xpath для кнопки на следующую страницу. Мы можем её скопировать из предыдущего кода. Меняем **allow** на **restrict\_xpaths**, внутри прописываем путь до ссылки **"//li[@class=‘next']/a"** и на этом заканчиваем. Судя по этому правилу, парсер будет находить ссылку на следующую страницу, извлекать её самостоятельно — нам не надо извлекать атрибут href и использовать метод get() — и переходить по этой ссылке до тех пор, пока ссылка на странице существует.

Теперь напишем правило, по которому мы будем искать на странице ссылку на каждую книгу и переходить по этой ссылке. Создаём новый объект Rule и указываем, где ссылки лежат. Они лежат в теге <article> с классом productt\_pod, затем тег <h3> и тег <a>. После извлечения ссылки нам надо парсить страницу книги. Делать это мы будем в уже созданной функции parse\_item, так что пишем callback=parse\_item, и указываем, что follow=True, то есть парсер должен пройти по этой ссылке и извлечь из неё html.

**Rule(LinkExtractor(restrict\_xpaths="//article[@class='product\_pod']/h3/a"), callback='parse\_item', follow=True)**

Теперь давайте в функции parse\_item будем доставать название книги. Итак, переходим на страницу любой книги. Смотрим, где у нас лежит название. Оно лежит внутри тега <div> с классами col-sm-6 product\_main — в теге <h1>. Запишем:

**title = response.xpath(«//div[contains(@class,‘product\_main’)]/h1/text()»).get()**

Воспользуемся уже созданным словарём, запишем туда: **item[‘title’] = title** и сделаем **yield item**. Теперь всё готово для запуска парсера. Потребуется какое-то время, чтобы парсер обработал все страницы и каждую книгу. В итоге мы видим, что item\_scraped\_count равняется тысяче: именно столько книг у нас на сайте. Пробежимся по извлечённым книгам — везде есть название. Парсер всё правильно собрал.

# Домашнее задание

Доработать паука в имеющемся проекте, чтобы по каждой книге была собрана вся информация с её страницы: Title, Price, In stock, Product Description, UPC, Product Type, Price (excl. tax), Price (incl. tax), Tax, Availability, Number of reviews.

# Используемая литература

1. [Документация Scrapy](https://docs.scrapy.org/en/xpath-tutorial/index.html).
2. [Работа с файлами и фото на Scrapy](https://docs.scrapy.org/en/latest/topics/media-pipeline.html).
3. [Web scraping с помощью Scrapy и Python](https://www.8host.com/blog/web-scraping-s-pomoshhyu-scrapy-i-python-3/).