



Методы сбора и обработки данных при помощи Python

Парсинг HTML. Beautiful Soup

**На этом уроке**

1. Познакомимся с библиотекой BeautifulSoup.
2. Напишем первый парсер сайта с использованием этой библиотеки.

**Оглавление**

[Beautiful Soup для сбора данных в HTML](#_2et92p0)

[Установка и начало работы](#_tyjcwt)

[Дерево синтаксического разбора](#_3dy6vkm)

[Атрибуты тегов](#_4d34og8)

[Навигация по дереву синтаксического разбора](#_2s8eyo1)

[parent](#_17dp8vu)

[contents](#_3rdcrjn)

[string](#_26in1rg)

[next\_sibling и previous\_sibling](#_lnxbz9)

[next и previous](#_35nkun2)

[Глоссарий](#_1ksv4uv)

[Дополнительные материалы](#_44sinio)

[Домашнее задание](#_2jxsxqh)

[Используемая литература](#_wfnq91unob6c)

# Beautiful Soup для сбора данных в HTML

## Установка и начало работы

Парсить HTML мы будем с применением библиотеки Requests, а обрабатывать полученную структуру — с помощью библиотеки Beautiful Soup. Установим их, введя команды в терминале:

| pip install bs4 |
| --- |

Импортируем библиотеки:

| import requests  from bs4 import BeautifulSoup as bs |
| --- |



Чтобы посмотреть основные возможности Beautiful Soup, попробуем собрать все книги с первой страницы известного нам сайта [Books to Scrape](http://books.toscrape.com).

Для начала сделаем GET-запрос на сайт:

| from bs4 import BeautifulSoup as bs import requests response = requests.get('http://example.com') |
| --- |

Теперь передадим полученный ответ в Beautiful Soup и укажем, какой парсер нам нужен. Создадим объект «‎суп»:

| soup = bs(response.content, ‘html.parser’) |
| --- |

## Дерево синтаксического разбора

Дерево синтаксического разбора — структуры данных Beautiful Soup, которые создаются по мере синтаксического разбора документа. Объект парсера (экземпляр класса BeautifulSoup) обладает большой глубиной вложенности связанных структур данных, соответствующих структуре документа XML или HTML. Объект парсера состоит из объектов двух других типов:

* объектов **Tag**, которые соответствуют тегам, к примеру, <div> и <p>;
* объекты **NavigableString**, соответствующие таким строкам, как In stock или Sapiens: A Brief History ....

## Атрибуты тегов

У тегов HTML есть атрибуты: например, у каждого тега <img> в приведённом выше примере HTML есть атрибуты *class*, *src* и *alt*. К атрибутам тегов можно обращаться таким же образом, как если бы объект Tag был словарём:

| images = soup.find\_all('img')  for image in images:  img\_src = image['src']  img\_alt = image['alt']  print(img\_src, img\_alt) |
| --- |

Атрибуты есть только у объектов **Tag**. Объекты **NavigableString** не имеют атрибутов.

## Навигация по дереву синтаксического разбора

**Важно!** *Все объекты* ***Tag*** *содержат элементы, перечисленные ниже (тем не менее* ***фактическое значение элемента может равняться None****). Объекты NavigableString имеют все из них за исключением contents и string.*

### parent

В примере выше родителем объекта <HEAD> Tag будет объект <HTML> Tag. Родитель объекта <HTML> Tag — сам объект парсера BeautifulSoup.

| soup.head.parent.name  # 'html'  soup.head.parent.parent.\_\_class\_\_.\_\_name\_\_  # 'BeautifulSoup'  soup.parent == None  # True |
| --- |

### contents

Parent перемещает нас вверх по дереву синтаксического разбора. С помощью **contents** мы перемещаемся вниз по дереву синтаксического разбора. **Сontents** — **упорядоченный список**, состоящий — из объектов **Tag** и **NavigableString**, содержащихся в элементе страницы (**page element**).

### string

Если у тега есть только один дочерний узел, который будет строкой, этот узел будет доступен через tag.string точно так же, как и через tag.contents[0].

### next\_sibling и previous\_sibling

Эти элементы позволяют переходить к следующему или предыдущему элементу на этом же уровне дерева синтаксического разбора. Эти способы перемещения довольно часто используются, когда у вас есть одинаковые элементы с абсолютно одинаковыми атрибутами, и вы точно знаете, что вам нужен именно второй.

### next и previous

Эти элементы позволяют передвигаться по элементам документа в том порядке, в котором они были обработаны парсером, а не в порядке появления в дереве.

Спарсим информацию о каждой книге с первой страницы. Для начала нам надо понять, где находится каждая книга. Нажимаем вот на этот значок в левом углу нашего инспектора и наводим на книгу. Видим, что вся информация о книге находится в теге *article*, родителем которого является тег *li*. Давайте посмотрим, сколько у нас всего элементов с тегом *article* на странице, для этого в строку поиска вводим *article* и нажимаем enter. Поиск показывает 20 элементов, и на самом верху страницы мы видим, что нам показано 20 элементов. Таким образом, чтобы выбрать все блоки с книгами с первой страницы, нам надо написать следующий код:

| books = soup.find\_all(‘article’)  books2 = soup.select(‘article’)  print(len(books), len(books2)) |
| --- |

Мы использовали оба метода, и итоговая длина списков у нас получилась одинаковой. Давайте посмотрим на первый элемент списка books — *print(books[0])*. Как видите, это элемент HTML, то есть, обращаясь к нему, мы можем применять все методы, которые доступны для супа.

Сохраним его в переменную *book* и протестируем для него получение названия, обложки, цены и наличия. Будем обращаться к элементу *book* и вызывать для него методы, доступные для супа. Рассмотрим разные способы получения информации.

Первое, что нам надо получить — название. Мы внутри тега *article*, так что всё, что выше, нас не интересует. Название находится в теге *а*, который вложен в тег *h3*. Возьмём этот тег за основу и спустимся вниз. Обратите внимание: если текст мы возьмём у тега *a*, то получим неполное название, так что нам надо брать атрибут title. Итак:

| title = book.find('h3').find('a')['title'] |
| --- |

Теперь найдём цену. Возвращаемся на сайт. Цена находится в теге *p* с классом *price\_color*, который вложен в тег *div*. Внутри этого же тега лежит и информация о наличии книги в магазине. Сперва получим информацию о наличии товара. Мы можем сделать это двумя способами.

Первый — используя метод *find*:

| instock = book.find('p', attrs={'class':['instock', ‘availability']}).text  print(instock)  print(instock.strip()) |
| --- |

Метод *strip* мы используем, чтобы избавиться от лишних пробелов и энтеров у текста.

Второй — используя метод *select\_one* и указание неполного класса. Этот метод удобен, когда вы точно знаете, что внутри html находится единственный тег с указанным классом или вам нужен первый тег из множества. Метод *select\_one* выберёт первый элемент, который попадёт под ваше описание. Давайте проверим, что у нас на странице ровно 20 элементов с классом *instock*. Для этого в поиске пишем точкаi nstock. Да, всё верно, элементов ровно 20. В этом случае используем второй способ получения статуса о наличии.

| instock = book.select\_one('div[class=product\_price] p[class\*=instock]’).text  print(instock)  print(instock.strip()) |
| --- |

Вы видите, что мы указали и родительский элемент. А для тега *p* — класс со звёздочкой, так как на странице у него есть ещё один класс — *availability*. Вообще, указывать точный путь до элементов довольно нудно и долго, но иногда только так вы можете получить нужную информацию.

Теперь попробуем получить цену. Самое простое решение — найти тег *p* с классом *price\_color*, но лучше разобрать и другие методы BeautifulSoup. Так что давайте воспользуемся методом *find\_previous\_sibling()*, который позволяет искать предыдущие теги одного уровня, а *find\_next\_sibling()* ищет следующие теги одного уровня. Внутри мы указываем, какой тег хотим найти. Это удобно в тех случаях, когда много одинаковых тегов, а классы ничем не отличаются. Если вы добавите *s* в конце — find\_next\_siblings() и find\_previous\_siblings() — то вам вернётся список. Итак, ищем цену. Копируем уже найденную сущность *instock*, убираем получение текста и добавляем *find\_previuos\_sibling()*. Внутри указываем тег *p*. Смотрим, что получилось.

| instock = book.select\_one('div[class=product\_price] p[class\*=instock]')  price = instock.find\_previous\_sibling(‘p')  print(price) |
| --- |

Как видите, мы нашли цену. Осталось найти изображение обложки. Оно лежит в теге *img*, вложенном в тег *а*, родителем которого является тег *div* с классом *image\_container*. У тега *img* нам надо получить атрибут *src*. Давайте всё это запишем.

| image = soup.find('div', attrs={'class': 'image\_container'}).find('img')['src']  print(image) |
| --- |

Обратите внимание, что ссылка у нас неполная. Желательно сохранять полные ссылки, чтобы вы могли воспользоваться ими без необходимости открывать сайт и вспоминать, как именно выглядит ссылка. Чтобы понять, какая ссылка полная, можно кликнуть на картинку и посмотреть, что будет в адресной строке. Копируем недостающую часть. Теперь сохраним переменную *image\_link* таким образом.

| image\_link = ‘<https://books.toscrape.com>/’+image  print(image\_link) |
| --- |

Теперь у нас есть полная ссылка на изображение обложки.

Последнее, что нам осталось сделать, чтобы спарсить все книги с первой страницы сайта, — создать цикл, в котором мы будем идти по каждой книге, извлекать название, цену, обложку и наличие, сохранять всё это в словарь, а словарь сохранять в список. Таким образом, в итоге у нас должен получиться список из 20 книг с первой страницы сайта.

Давайте объединим все участки кода, которые мы уже написали и добавим совсем немного.

| books\_list = []  for book in books:  title = book.find('h3').find('a')['title']  instock = book.select\_one('div[class=product\_price] p[class\*=instock]').text.strip()  price = book.select\_one('div[class=product\_price] p[class\*=instock]').find\_previous\_sibling('p').text.strip()  image = soup.find('div', attrs={'class': 'image\_container'}).find('img')['src']  image\_link = '<https://books.toscrape.com/'+image>    book\_dict = {  'Image': image\_link,  'Title': title,  'Price': price,  'Instock': instock  }  books\_list.append(book\_dict)  print(len(books\_list))  pprint(books\_list) |
| --- |

В нашем итоговом списке 20 книг, как и должно быть. При просмотре этого списка мы видим, что он состоит из 20 словарей, в каждом из которых находится вся нужная информация о книге.

Так же в Beautiful Soup есть много других полезных методов: например, *find\_parent(), find\_next() и find\_previous()*, которые помогают найти родительский тег, следующий или предыдущий тег соответственно. При работе с любыми библиотеками полезно пользоваться их документацией или искать ответы на вопросы на сайте StackOverflow по тегу BeautifulSoup.

# Глоссарий

**HTML** (HyperText Markup Language) — язык гипертекстовой разметки. Интерпретируется браузерами, в результате чего форматированный текст отображается на экране. HTML-страницы передаются от сервера к клиенту по протоколу HTTP в виде обычного или зашифрованного текста.

**HTML-теги** — используются для разграничения начала и конца элементов в разметке.

**Атрибуты** — свойства тега*,* дающие дополнительные возможности форматирования текста.

**DOM** (Document Object Model) — не зависящий от платформы и языка программный интерфейс, позволяющий программам и скриптам получить доступ к содержимому HTML-, XHTML- и XML-документов, а также изменять содержимое, структуру и оформление таких документов.

**Библиотека Beautiful Soup** — парсер lxml, который преобразует наш HTML-код в DOM и обрабатывает полученную структуру.

**Дерево синтаксического разбора** — структуры данных Beautiful Soup, которые создаются по мере синтаксического разбора документа.

# Дополнительные материалы

1. [Документация BeautifulSoup](https://www.crummy.com/software/BeautifulSoup/bs4/doc/)

# Домашнее задание

Спарсить цитаты с сайта [Quotes to Scrape](https://quotes.toscrape.com/). Необходимо извлечь саму цитату, автора и список тегов.

Задание со звёздочкой: спарсить все цитаты на сайте, а не только с первой страницы. При парсинге желательно использовать разные методы для получения необходимой информации.

# Используемая литература

1. [Базовая структура HTML](https://www.htmlgoodies.com/html5/markup/working-with-html-styling-blocks.html).
2. [Элементы HTML/CSS](https://www.w3schools.com/htmL/).
3. [Объектная модель документа](https://ru.wikipedia.org/wiki/Document_Object_Model).
4. [Стандарт HTML5](https://ru.wikipedia.org/wiki/HTML5).
5. [Документация Beautifull Soup](https://www.crummy.com/software/BeautifulSoup/bs4/doc/).
6. [Документации/BeautifulSoup](http://wiki.python.su/%25D0%2594%25D0%25BE%25D0%25BA%25D1%2583%25D0%25BC%25D0%25B5%25D0%25BD%25D1%2582%25D0%25B0%25D1%2586%25D0%25B8%25D0%25B8/BeautifulSoup).