

Парсинг HTML. Beautiful Soup

Сбор и разметка данных



Оглавление

Словарь терминов	3
Введение	3
Как работает веб-скрейпинг?	6
Законен ли веб-скрейпинг?	7
Введение в BeautifulSoup	10
Парсинг HTML с помощью BeautifulSoup	11
Скрейпинг веб-страницы	15
Заключение	26
Что можно почитать еще?	26

Словарь терминов

Веб-скрейпинг (web scraping, скрепинг, скрапинг) — это технология получения веб-данных путем извлечения их со страниц веб-ресурсов.

Парсинг — это процесс получения данных в одном формате и преобразования их в другой формат.

Введение

Интернет — огромный источник данных. Мы можем получить к ним доступ с помощью веб-скрейпинга и Python.

Некоторая информация недоступна в удобных форматах типа CSV, некоторую можно получить с помощью API. В то же время веб-сайты — часто ценный источник данных.

На прошлой лекции мы говорили, что API — это набор протоколов и инструментов, который можно использовать для доступа и манипулирования данными с сервера или базы данных.

Веб-скрейпинг — это техника извлечения данных с веб-страниц путем парсинга HTML-кода. Процесс включает в себя отправку автоматических запросов на веб-страницы и извлечение данных из HTML-кода страницы.

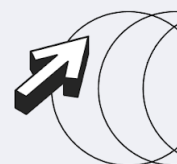
В некоторых случаях веб-скрейпинг может стать альтернативой API. Например, когда API недоступны или не предоставляют нужных данных. Если у веб-сайта нет API для доступа к данным, с помощью веб-скрейпинга можно извлечь нужные данные непосредственно с веб-сайта.

Взаимодействие «клиент-сервер», парсинг API и веб-скрейпинг связаны между собой — все они подразумевают доступ к данным с сервера и манипулирование ими. Однако они отличаются по подходу и уровню контроля над доступом к данным.

Подумайте, какую информацию для анализа можно получить с веб-страниц?



**Ответ будет ниже. Переходите к нему,
когда поразмыслите над вопросом.**



Можно провести анализ данных спортивных мероприятий, анализ цен магазинов и сервисов, анализ обзоров и отзывов на товары, новостей и так далее. На самом деле можно выполнить скрейпинг любой информации с веб-сайта: текста, изображений, видео и так далее.

Сейчас существует множество инструментов для веб-скрейпинга. Говоря об инструментах, мы имеем в виду:

- Полноценные фреймворки. Самый популярный из них — Scrapy.
- Модули. Популярные — BeautifulSoup и lxml. Эти два модуля называются парсерами. Они не предназначены для больших и комплексных задач, но позволяют выполнить скрейпинг некоторых страниц.

Сегодня будем говорить о том, как выполнить веб-скрейпинг и парсинг с помощью библиотеки BeautifulSoup в Python, а также выполним проект по созданию датасета из информации, полученной с веб-сайта.

Предположим, нам нужно проанализировать розничный рынок лекарств. Информацию о ценах и ассортименте мы можем найти на просторах интернета, но проблема в том, что ее не предоставляют нам в готовом табличном формате. Более того, на разных ресурсах она может быть представлена по-разному.

Давайте возьмем для примера сайт www.apteka.de¹. Он содержит информацию о лекарственных препаратах, ценах и так далее, но эти данные недоступны в формате CSV или через API.

Представим, что мы хотим проанализировать эти данные или загрузить их. Для этого можем нанять специально обученного человека, который будет кропотливо и долго путем copy-paste переносить информацию в Excel. Но нас, скорее всего, такой вариант не устроит.


Веб-скрейпинг позволяет нам использовать программирование для выполнения тяжелой работы. Мы напишем код, который получит информацию с сайта, возьмет только те данные, с которыми мы хотим работать, и выведет их в нужном нам формате.

В качестве рабочего примера попробуем извлечь информацию о препаратах для лечения аллергии. В меню на сайте выбираем: Все категории → Аллергия и аллергический ринит. Заходим и смотрим, какая информация нам нужна: например, название препарата и цена.

Сортировка:

Актуальность

▼



-39 %¹


ALLERGO-MOMELIND von DoppelherzPharma

18 g

450,56 € / 1 kg

13,45 €

8,11 €



-52 %¹


MometaxHEXAL Heuschnupfenspray

18 g

588,33 € / 1 kg

22,24 €

10,59 €



-50 %¹

Lorano akut Tabletten

20 St

10,02 €

10,02 €

4,99 €

Каждое из изображений лекарств — это ссылка, пройдя по которой мы можем извлечь еще больше информации. Кликнув на изображение препарата, попадаем на его страницу. Здесь можем найти производителя, форму выпуска, инструкцию и так далее.

¹ Иногда ссылки не открываются. Так бывает — сайты меняются или перестают работать. Главное для нас — понять принцип работы с данными.



-39 %¹

ALLERGO-MOMELIND von DoppelherzPharma 18 g

Zur symptomatischen
Behandlung der Beschwerden
eines Heuschnupfens. Für
Erwachsene.

Форма выпуска: Nasendosierspray
Содержание: 18 g
Базовый тариф: 450,56 € / 1 kg
Фармацевтический номер:
16665753
Производитель: Queisser Pharma
GmbH & Co. KG

8,11 €

вместо 13,45 € UVP

Вкл. НДС" при необходимости включая
стоимость доставки

● ● ● в наличии

📄 1008248087

★ [Оценки](#)

Прежде чем мы начнем извлекать данные, давайте разберемся, как работает веб-скрейпинг.

Как работает веб-скрейпинг?

Когда мы скрейпируем веб-страницы, мы пишем код, который посылает запрос на сервер, на котором находится указанная нами страница, а сервер возвращает для запрошенной страницы исходный код, как правило, HTML.

Пока мы делаем то же, что и веб-браузер: посылает серверу запрос с определенным URL и просим сервер вернуть код для этой страницы.

Но, в отличие от веб-браузера, наш код веб-скрейпинга не будет интерпретировать исходный код страницы и отображать ее визуально. То есть мы получим чистый HTML, и нам нужно будет выполнить парсинг. Для этого мы напишем собственный код, который будет фильтровать исходный код страницы в поисках определенных элементов, указанных нами, и извлекать то содержимое, которое нам нужно.

Таким образом, последовательность действий при скрейпинге веб-страницы будет такой:

1. Запросить содержимое (HTML-код) определенного URL с сервера.
2. Загрузить полученное содержимое.
3. Определить элементы страницы, содержащие нужную нам информацию.
4. Извлечь и при необходимости выполнить парсинг элементов страницы в датасет.

Важно отметить: с точки зрения сервера, запрос страницы с помощью веб-скрейпинга — это то же самое, что и загрузка страницы в браузере.

🔥 Когда мы используем код для отправки таких запросов, мы можем «загружать» страницы гораздо быстрее, чем обычный пользователь, и тем самым быстро расходовать ресурсы сервера владельца сайта.

Законен ли веб-скрейпинг?

К сожалению, однозначного ответа нет. Некоторые сайты прямо разрешают веб-скрейпинг. Другие прямо запрещают его. Многие сайты не дают четких указаний.

Прежде чем приступить к скрейпингу сайта, нужно найти страницу с условиями и положениями, чтобы узнать, есть ли там четкие правила, касающиеся скрейпинга. Если они есть, мы должны следовать им. Если нет, то это решение, которое вы принимаете, основываясь только на личном опыте и чувствах, потому что нет правильного или простого ответа.

В российском законодательстве напрямую о парсинге не говорится ничего. С одной стороны, информация в интернете общедоступная, поэтому ее можно свободно скрейпить. С другой, есть ряд подводных камней, которые нужно иметь в виду.

Советуем прочитать статью [«Скрапинг интернет-ресурсов: критерии законности»](#), а также статью на Хабре — [«Парсинг — это законно?»](#).

Правовые нормы, с которыми можно столкнуться (даже случайно), выполняя скрейпинг:

1. Правила о гражданско-правовой ответственности и о причинении вреда имуществу.
2. Уголовная ответственность за преступления в сфере компьютерной информации.
3. Нормы договорного права.
4. Право интеллектуальной собственности.
5. Правила о персональных данных.

Что делать, чтобы случайно не нарушить закон? Прежде чем приступить к скрейпингу, нужно заглянуть в подвал сайта и прочитать то самое «Пользовательское соглашение», которое обычно никто не читает. На некоторых платформах есть раздел, прямо указывающий на правила скрейпинга или на его запрет.

Давайте посмотрим на примере Авито. Вот выдержка из [Условий использования Авито](#), где явно указан запрет автоматического сбора информации:

Без согласия Компании запрещено использовать технические средства для взаимодействия с сервисом в обход обычного порядка использования баз данных и программ для ЭВМ. В том числе запрещено использовать автоматизированные скрипты для сбора информации на Авито, а также для автоматической регистрации профилей.

Другой способ узнать ограничения сайта — просмотр файла robots.txt. Файл robots.txt находится в корневом каталоге сайта. Например, на сайте www.example.com он находится по адресу www.example.com/robots.txt.

В файле robots.txt содержатся инструкции, которые говорят поисковым роботам, какие URL им разрешено обрабатывать. С его помощью ограничивают количество запросов на сканирование и тем самым снижают нагрузку на сайт.

Этот файл содержит три важные инструкции. Первая — это **user agent**, который представляет собой идентификатор скрейпера.

Агент пользователя — это часть программного обеспечения, которая выступает в качестве посредника между веб-браузером или другим клиентским приложением и веб-сервером. Когда клиентское приложение делает запрос к веб-серверу, оно включает в HTTP-заголовок строку user-agent, которая идентифицирует клиентское программное обеспечение и номер его версии. Эта информация используется веб-сервером для определения того, как ответить на запрос, включая то, какое содержимое предоставить и как его оформить.

User agent может включать информацию об операционной системе, типе устройства и языковых предпочтениях пользователя, что позволяет веб-серверу дополнительно адаптировать ответ на основе этих факторов.

User agent может быть настроен или модифицирован для маскировки идентификации клиентского приложения или для имитации поведения других типов клиентов. Это используется в веб-скрейпинге, когда может потребоваться имитировать поведение определенного пользовательского агента, чтобы избежать обнаружения или обеспечить совместимость с конкретным веб-сайтом.

Важно отметить, что изменение агента пользователя в целом законно, но потенциально может нарушать условия обслуживания некоторых веб-сайтов или веб-служб.

Вторая инструкция — **allow**, определяет веб-страницы, которые роботу разрешено скрейпить. Также есть инструкция **disallow**, которая определяет запрещенные для скрейпинга веб-страницы.

Пример простого файла robots.txt с двумя правилами:

```
User-agent: Googlebot
Disallow: /nogooglebot/

User-agent: *
Allow: /

Sitemap: http://www.example.com/sitemap.xml
```

Агенту пользователя с названием Googlebot запрещено сканировать любые URL, начинающиеся с `http://example.com/nogooglebot/`.

Любым другим агентам пользователя разрешено сканировать весь сайт. Это правило можно опустить, и результат будет тем же. По умолчанию агенты пользователя могут сканировать сайт целиком.

Файл Sitemap этого сайта находится по адресу `http://www.example.com/sitemap.xml`.

Подробнее о файле robots.txt: [Create and Submit a robots.txt File | Google Search Central | Documentation](#).

Важно помнить, что веб-скрейпинг потребляет ресурсы сервера принимающего сайта. Если мы скрейпируем одну страницу один раз, это не вызовет проблем. Но если наш код скрейпит 1000 страниц раз в десять минут, процесс может быстро стать дорогостоящим для владельца сайта.

Таким образом, в дополнение к соблюдению всех явных правил о веб-скрейпинге, размещенных на сайте, также полезно следовать лучшим практикам:

- Никогда не делайте скрейп чаще, чем это необходимо.
- Рассмотрите возможность кэширования содержимого, которое вы скрейпите, чтобы оно загружалось только один раз.
- Встраивайте в код паузы (для этого есть функция `time.sleep()`), чтобы не перегружать сервер слишком большим количеством запросов.

В нашем случае данные apteka.de — общественное достояние. Их условия не запрещают веб-скрейпинг, поэтому мы можем продолжать.

Введение в BeautifulSoup

Beautiful Soup — это библиотека Python для веб-скрейпинга. Ее используют для парсинга документов HTML и XML, она известна своей гибкостью и простотой использования. BeautifulSoup позволяет разработчикам извлекать информацию из веб-страниц, манипулировать HTML-кодом и преобразовывать его в структурированный формат.

Ключевые особенности BeautifulSoup:

- **Парсинг документов HTML и XML.**
- **Навигация по дереву парсинга** (parse tree). BeautifulSoup предоставляет инструменты для навигации и поиска определенных элементов в HTML-коде.
- **Доступ к тегам и атрибутам.** BeautifulSoup предоставляет простые способы доступа к тегам и атрибутам в HTML-коде.
- **Изменение HTML-кода.** BeautifulSoup позволяет разработчикам изменять HTML-код путем добавления или удаления тегов и атрибутов.
- **Поддержка Unicode.** BeautifulSoup поддерживает символы Unicode, что упрощает работу с символами, отличными от символов ASCII, в HTML-документах.

Parsing tree также называется синтаксическим деревом. Это структура данных, которая представляет синтаксическую структуру строки текста в соответствии с правилами формальной грамматики.

В контексте парсинга HTML дерево парсинга представляет собой иерархическую структуру HTML-документа. Дерево состоит из узлов, которые представляют элементы HTML, такие как теги, атрибуты и текстовые узлы. Узлы соединены ребрами, которые представляют собой отношения «родитель-потомок» между узлами.

Например, рассмотрим HTML-документ:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <h1>Welcome to my website!</h1>
    <p>This is a paragraph of text.</p>
  </body>
</html>
```

Дерево парсинга этого HTML-документа имеет корневой узел `<html>`, представляющий весь документ, и дочерние узлы, представляющие `<head>`, `<title>`, `<body>`, `<h1>`, `<p>` и текстовые узлы. Дерево будет иметь ребра, соединяющие узлы, чтобы представить иерархические отношения между ними.

Дерево парсинга полезно для извлечения конкретной информации из HTML-документа, например, для поиска всех ссылок или изображений на веб-странице. Оно также полезно для выявления структурных или семантических закономерностей в документе, таких как общая компоновка или организация содержимого.

Парсинг HTML с помощью BeautifulSoup

Парсинг HTML-документов — это процесс извлечения данных из HTML-кода веб-страницы. BeautifulSoup предоставляет простой и мощный API для парсинга HTML-документов.

Чтобы выполнить парсинг HTML с помощью BeautifulSoup, сперва нужно создать объект BeautifulSoup. Этот объект представляет собой результат парсинга HTML-документа и предоставляет методы для навигации по дереву и извлечения из него информации.

Конструктор BeautifulSoup принимает два аргумента: HTML-документ в виде строки и используемый парсер. Парсер может быть встроенным парсером HTML или внешним парсером, таким как LXML.

```
from bs4 import BeautifulSoup
import requests

url = "https://example.com"
response = requests.get(url)
html = response.content

soup = BeautifulSoup(html, 'html.parser')

print(soup.title.string)
```

В этом примере мы используем библиотеку requests для загрузки веб-страницы по адресу `https://example.com`.

Возвращаемый объект ответа содержит HTML-содержимое веб-страницы, которое мы извлекаем с помощью атрибута `content`. Атрибут `content` возвращает двоичное представление содержимого HTML, которое может быть декодировано в строку.

Мы передаем HTML-содержимое в конструктор `BeautifulSoup` вместе с парсером `html.parser`. Полученный объект `BeautifulSoup`, `soup`, представляет собой результат парсинга HTML.

Затем мы можем использовать методы, которые предоставляет `BeautifulSoup`, для навигации по дереву и извлечения из него информации. В этом случае мы выводим заголовок веб-страницы, используя атрибут `title.string` объекта `soup`.

В `Beautiful Soup` вы можете получить доступ к тегам HTML и их атрибутам, используя различные методы. Давайте разберем некоторые из них.

1. **find()** — метод возвращает первый совпадающий тег или элемент в HTML-документе. Вы можете использовать его для поиска тегов по имени, классу, ID или другим атрибутам.

Например, чтобы найти первый тег `<a>` в документе HTML, вы можете использовать следующий код:

```
a = soup.find('a')
```

2. **find_all()** — метод, аналогичный предыдущему, но возвращает список *всех* соответствующих тегов или элементов в документе HTML.

3. **Доступ к атрибутам.** Когда у вас есть объект тега, который вы получили с помощью метода `find()`, вы можете получить доступ к его атрибутам, используя нотацию в стиле словаря.

Например, чтобы получить значение атрибута `href` тега `<a>`, вы можете использовать следующий код:

```
link = soup.find('a')
href = link['href']
```

4. **`get()`** — метод похож на доступ к атрибутам с использованием нотации в стиле словаря, но он предоставляет значение по умолчанию, если атрибут не найден.

Например, чтобы получить значение атрибута `href` тега `<a>` со значением по умолчанию `None`, если атрибут не найден, вы можете использовать следующий код:

```
link = soup.find('a')
href = link.get('href', None)
```

5. **`string` и `text`** — эти атрибуты объекта тега позволяют получить доступ к текстовому содержимому тега без форматирования HTML.

Например, чтобы получить текстовое содержимое тега `<a>`, вы можете использовать следующий код:

```
link = soup.find('a')
text = link.text
```

Перейдем к группе методов для навигации по дереву парсинга. Вот некоторые из тех, которые используют чаще всего:

- **`contents`** — атрибут объекта тега возвращает список прямых дочерних объектов тега в виде тегов и текстовых объектов.

Например, чтобы получить список всех прямых дочерних объектов тега `<html>`, вы можете использовать следующий код:

```
html = soup.find('html')
children = html.contents
```

- **descendants** — метод объекта тега возвращает генератор, который рекурсивно выдает всех потомков тега.

Например, чтобы получить список всех потомков тега `<html>`, вы можете использовать следующий код:

```
html = soup.find('html')
descendants = html.descendants
```

- **parent** и **parents** — эти атрибуты объекта тега позволяют получить доступ к родителю тега или ко всем его предкам в дереве.
- **next_sibling** и **previous_sibling** — атрибуты объекта тега позволяют получить доступ к следующему или предыдущему родственнику тега соответственно.
- **find_all()** и **find()** — методы объекта тега позволяют искать теги в пределах поддерева, корнем которого является тег.

```
div = soup.find('div')
links = div.find_all('a')
```

Атрибуты `contents` и `descendants` объекта тега `Beautiful Soup` предоставляют доступ к узлам дерева, но делают это по-разному.

Атрибут `contents` возвращает список всех прямых дочерних элементов тега. Этими дочерними объектами могут быть как теги, так и текстовые объекты. Например, рассмотрим следующий HTML:

```
<div>
  <p>This is a <b>bold</b> statement.</p>
  <ul>
    <li>Item 1</li>
    <li>Item 2</li>
  </ul>
</div>
```

Если с помощью `Beautiful Soup` найти тег `<div>`, а затем обратиться к его атрибуту `contents`, мы получим список всех прямых дочерних элементов тега `<div>`:

```
div = soup.find('div')
children = div.contents
```

Список дочерних элементов будет содержать четыре элемента: объект тега <p>, текстовый объект, содержащий строку «This is a », объект тега и еще один текстовый объект, содержащий строку «statement.».

Напротив, атрибут descendants возвращает генератор, который выдает все узлы в дереве разбора, рекурсивно, начиная с объекта tag. Например, если мы используем BeautifulSoup для поиска тега <div>, а затем обращаемся к его атрибуту descendants, мы получаем генератор, который выдает все узлы в дереве под тегом <div>:

```
div = soup.find('div')
descendants = div.descendants
```

Генератор потомков выдаст все узлы в дереве под тегом <div>. Сюда входят все дочерние элементы тега <div>, а также их дочерние элементы и так далее.

Скрейпинг веб-страницы

Приступим к веб-скрейпингу. Мы начнем с получения данных со страницы «Аллергия и аллергический ринит»:

<https://apteka.de/online-shop/allergie-heuschnupfen>

Первое, что нужно сделать, — загрузить данные веб-страницы. Будем работать в Google Colab. Начнем с импорта необходимых библиотек:

```
from bs4 import BeautifulSoup
import requests
import pandas as pd
```

Импортируем BeautifulSoup.

Библиотека requests позволяет отправлять GET-запрос серверу с помощью Python, который возвращает HTML-содержимое веб-страницы.

Pandas — библиотека Python для анализа и обработки данных.

Создадим переменную, содержащую URL требуемой страницы:

```
website="https://apteka.de/online-shop/allergie-heuschnupfen "
```

С помощью метода get библиотеки requests отправляем запрос на сервер:

```
page = requests.get(website)
```

Можно проверить, что запрос успешно обработан с помощью метода `status_code`, который возвращает число, указывающее на статус (200 — OK, 404 — Not Found).

```
page.status_code
```

```
>>> 200
```

Что представляет собой результат команды `requests.get`? HTTP-запрос возвращает объект `Response Object` со всеми данными ответа (содержимое, кодировка, статус и так далее).

Итак, мы загрузили HTML-документ. Теперь можем использовать библиотеку `BeautifulSoup` для парсинга этого документа.

Создаем экземпляр класса `BeautifulSoup` для парсинга нашего документа. Для этого передаем два аргумента в `BeautifulSoup`: первый — это контент `Response Object`, второй — `html.parser`.

```
soup = BeautifulSoup(page.content, 'html.parser')
```

Теперь, если мы выведем содержимое этого объекта, увидим весь HTML-код запрошенной страницы:

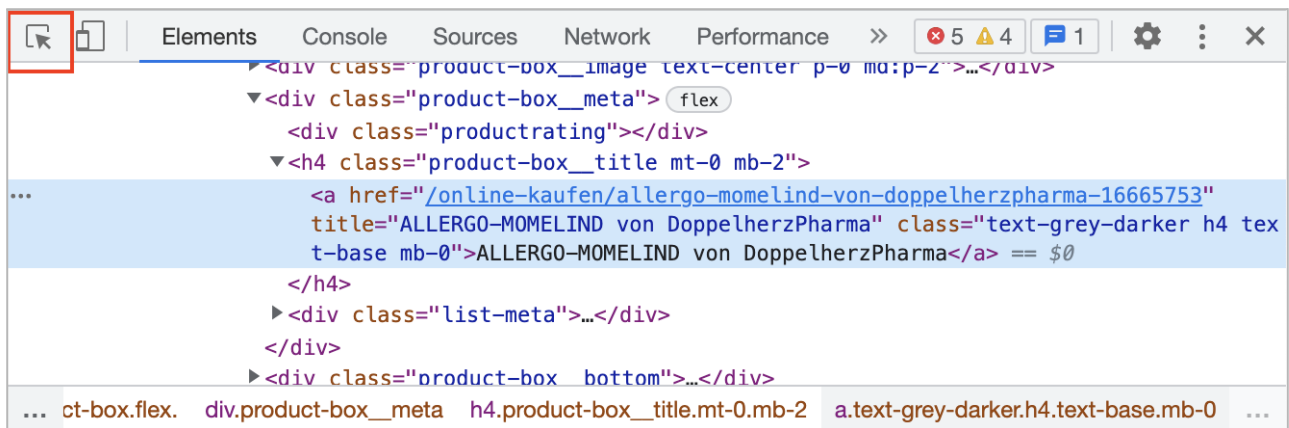
▶ soup

```
<!DOCTYPE html>

<html lang="ru">
<head>
<script>
  let cookiesAccepted = document.cookie.indexOf("cookie-policy-accepted") != -1;
  let month = new Array("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
  let days = new Array("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat");
  let date = new Date();
  let expiration = new Date(date.setDate(date.getDate() + 365));
```

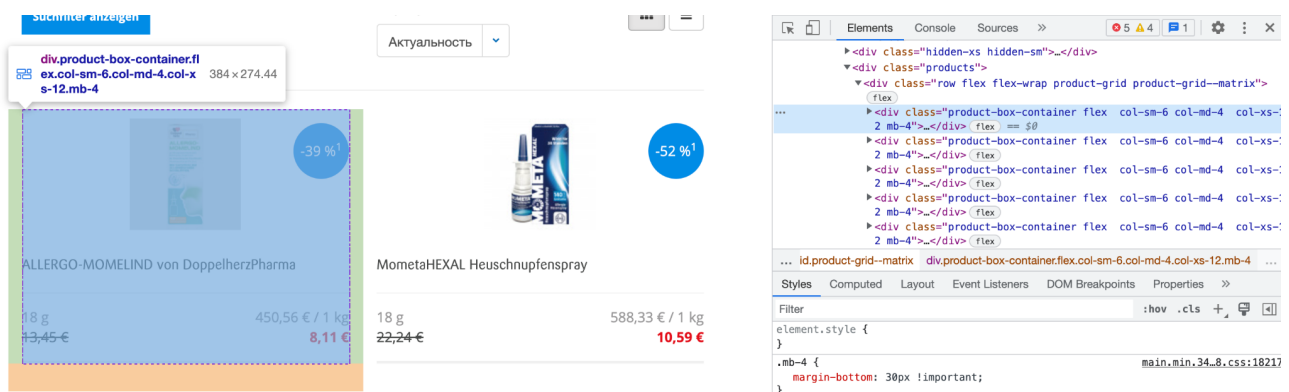
Теперь создадим контейнер `result`, который будет содержать всю нужную нам информацию со страницы. Для этого нужно выполнить поиск по объекту `soup`. В браузере Chrome кликаем правой кнопкой мыши в любом месте страницы и во всплывающем меню выбираем «Посмотреть код». В правой части браузера открывается окно `DevTools`, где будет подсвечена соответствующая часть HTML-кода.

Здесь нужно выполнить поиск тегов с помощью инструмента поиска, чтобы определить теги, соответствующие нужному нам элементу на странице.



В нашем случае это тег `<div class="product-box-container flex col-sm-6 col-md-4 col-xs-12 mb-4">`.

Это значит, что внутри этого тега содержится вся информация о продукте, которую мы видим на странице, в том числе ссылка на продукт. Ссылка понадобится для извлечения информации, расположенной на странице продукта.



Сейчас нам нужно найти некий общий элемент для каждого продукта внутри тега `<div>`. В данном случае это часть значения атрибута `class="product-box-container"`. Используем метод `find_all`, чтобы найти все соответствующие элементы:

```
result = soup.find_all('div', ('class', 'product-box-container'))
```

Напомню, метод `find_all()` объекта `soup` используется для поиска всех вхождений определенного тега в HTML-документе. В данном случае тег, который мы ищем, — это тег `<div>`.

Первый аргумент `find_all()` — это имя тега, который мы ищем (в данном случае `'div'`). Второй аргумент — это словарь атрибутов для поиска, где ключ — имя атрибута, а значение — искомое значение. В этом случае искомый атрибут — `'class'`, а искомое значение — `'product-box-container'`.

Метод `find_all()` возвращает список объектов `Tag`, соответствующих заданным критериям. Каждый объект `Tag` представляет собой одно вхождение указанного тега в HTML-документ и предоставляет доступ к атрибутам и содержимому тега.

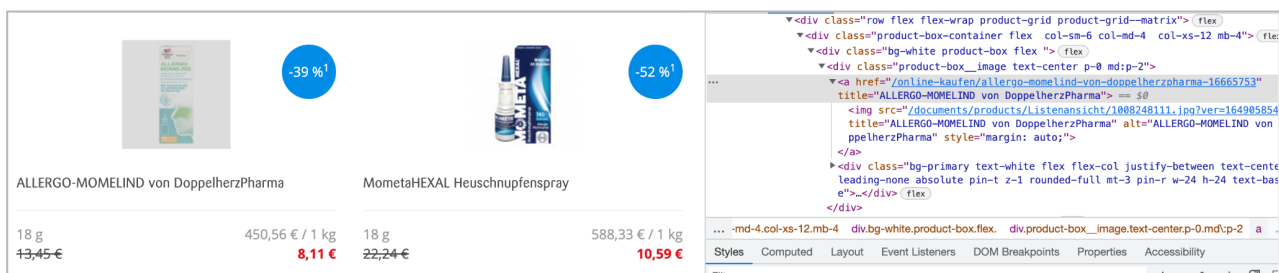
Проверим количество элементов в объекте `result`:

```
len(result)

>>> 24
```

Можно проверить, что это число соответствует количеству товаров на странице.

Следующая задача — извлечь ссылки из каждого элемента, чтобы получить доступ к информации, имеющейся в описании каждого товара. Находим тег ссылки `<a>` и атрибут `<href>`.



ALLERGO-MOMELIND von DoppelherzPharma	Mometax HEXAL Heuschnupfenspray
18 g 450,56 € / 1 kg 13,45 € 8,11 €	18 g 588,33 € / 1 kg 22,24 € 10,59 €

Здесь мы видим проблему: в коде указан относительный путь к странице. Абсолютный путь содержит все элементы URL, как в адресной строке браузера, то есть в нашем случае:

<https://apteka.de/online-kaufen/allergo-momelind-von-doppelherzpharma-16665753>

Мы же видим только часть адреса, то есть относительный путь:

```
href="/online-kaufen/allergo-momelind-von-doppelherzpharma-16665753"
```

Значит, нам нужно объединить:

- первую часть HTTP-запроса — `https://www.apteka.de/`
- и вторую — `"/online-kaufen/allergo-momelind-von-doppelherzpharma-16665753"`

Создадим переменную, содержащую первую часть запроса:

```
url_1 = 'https://www.apteka.de'
```

Теперь возвращаемся в браузер в DevTools и ищем блок, внутри которого находится вторая часть запроса. В нашем случае это `<div class="product-box__image text-center p-0 md:p-2">`.

```
..      ▼<div class="product-box__image text-center p-0 md:p-2"> == $0
      ▼<a href="/online-kaufen/allergo-momelind-von-doppelherzpharma-16665753"
        title="ALLERGO-MOMELIND von DoppelherzPharma">
```

Чтобы извлечь адрес, создадим два цикла: во внешнем цикле проходим по HTML-документу, во внутреннем находим тег `<div>`, содержащий атрибут `<class='product-box__image'>`, далее находим тег ссылки `<a>` и присоединяем результат в список `url_2`.

```
url_2 = []
for i in result:
    for link in i.find_all('div', ('class', 'product-box__image')):
        url_2.append(link.find('a').get('href'))
```

Проверим результат — получаем список всех относительных путей на странице.

```
[20] url_2

['/online-kaufen/allergo-momelind-von-doppelherzpharma-16665753',
 '/online-kaufen/mometahexal-heuschnupfenspray-11697286',
 '/online-kaufen/lorano-akut-tabletten-07222502',
 '/online-kaufen/cetirizin-axicur-10-mg-filmtabletten-14293520',
 '/online-kaufen/levocamed-kombipackung-0-5-mg-ml-augentropfen-nasenspray-15624835',
 '/online-kaufen/desloratadin-ratiopharm-5-mg-filmtabletten-15397612',
 '/online-kaufen/tavegil-tabletten-16791883',
 '/online-kaufen/allergodil-akut-forte-1-5-mg-ml-nasenspray-loesung-17510656',
```

Теперь нам нужно объединить первую и вторую части ссылок. Для этого импортируем модуль `urllib.parse`, который позволяет объединить относительный URL в абсолютный. Создадим новый список `url_joined` и в цикле объединим части наших URL.

```
import urllib.parse
url_joined = []

for link in url_2:
    url_joined.append(urllib.parse.urljoin(url_1, link))
```

Проверяем, что мы получили список путей к каждому продукту на странице. Чтобы убедиться, что все работает, можно открыть любую ссылку в браузере.

url_joined

```
[ 'https://www.aptekaru.de/online-kaufen/allergo-momelind-von-doppelherzpharma-16665753',  
  'https://www.aptekaru.de/online-kaufen/mometahexal-heuschnupfenspray-11697286',  
  'https://www.aptekaru.de/online-kaufen/lorano-akut-tabletten-07222502',  
  'https://www.aptekaru.de/online-kaufen/cetirizin-axicur-10-mg-filmtabletten-14293520',  
  'https://www.aptekaru.de/online-kaufen/levocamed-kombipackung-0-5-mg-ml-augentropfen-nasenspray-15624835',  
  'https://www.aptekaru.de/online-kaufen/desloratadin-ratiopharm-5-mg-filmtabletten-15397612',  
  'https://www.aptekaru.de/online-kaufen/tavegil-tabletten-16791883',  
  'https://www.aptekaru.de/online-kaufen/allergodil-akut-forte-1-5-mg-ml-nasenspray-loesung-17510656',
```

Следующая задача — получить нужную информацию по ссылке конкретного товара. Для начала выполним парсинг только для одной страницы по первой ссылке в списке. Зайдем на страницу первого товара и посмотрим, какая информация нам нужна. Для примера извлечем:

- название товара,
- цену,
- содержание,
- форму выпуска,
- производителя.

Первое, что нужно сделать, — выполнить скрейпинг страницы. Эта процедура ничем не отличается от того, что мы уже делали.

```
first_link = url_joined[0]  
response = requests.get(first_link)  
soup = BeautifulSoup(response.content, 'html.parser')
```


Проверяем результат:

soup

```
<!DOCTYPE html>  
  
<html lang="ru">  
<head>  
<script>  
  let cookiesAccepted = document.cookie.indexOf("cookie-policy-accepted") != -1;  
  let month = new Array("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");  
  let days = new Array("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat");  
  let date = new Date();  
  let expiration = new Date(date.setDate(date.getDate() + 365));
```

Далее выполняем поиск тега в HTML-коде, в котором находится название товара в DevTools. Видим, что название товара находится в теге ``, но тег не уникальн. Если вы посмотрите код, увидите несколько тегов ``, а нам нужен конкретный. В таком случае нужно найти некий уникальный идентификатор, для этого смотрим родительские теги. Находим, что уровнем выше тег `<h1>` содержит атрибут `<class>` с уникальным значением.

Стартовая страница > Аллергия и аллергический ринит > ALLERGO-MOMELIND von DoppelherzPharma



-39 %!

ALLERGO-MOMELIND von DoppelherzPharma 18 g

Zur symptomatischen

8,11 € ВМЕСТО

```

<div class="col-sm-6">
  <div class="bg-grey-light">
    <div class="p-4 pb-3 sm:pb-4">
      <h1 class="h2 mt-0 sm:mb-6">
        <span>ALLERGO-MOMELIND von DoppelherzPharma</span> == $0
      </h1>
    </div>
    <div class="hidden-lg hidden-md hidden-sm mb-2"></div>
    <div class="flex flex-wrap -mx-3"></div>
  </div>
</div>

```

Мы можем использовать функцию `find`, чтобы найти идентификатор сначала родительского уровня, а затем внутри него (опять же с помощью функции `find`) тег ``, содержащий название товара.

```
soup.find('h1', ('class', 'h2 mt-0 sm:mb-6'))
```

Для извлечения текста воспользуемся методом `text`.

```
soup.find('h1', ('class', 'h2 mt-0 sm:mb-6')).text
```

Теперь извлечем форму выпуска товара. Алгоритм отличается от предыдущего. С помощью инструмента поиска DevTools находим тег, в котором находится форма выпуска. Здесь мы сталкиваемся с проблемой, что нет уникального тега, чтобы однозначно идентифицировать форму выпуска. Можно, конечно, прописать весь путь до нужного нам значения, но на разных страницах этот путь может быть разным. Поэтому воспользуемся другим приемом. Вот код:

```
import re
form = soup.find('div', text=re.compile('Форма
выпуска:')).get_text(strip=True)[15:]
```

Импортируем `re` — это встроенный модуль в Python, который расшифровывается как «регулярные выражения». Он предоставляет набор функций и классов, которые позволяют выполнять поиск и манипулировать строками с помощью сопоставления шаблонов.

С помощью `re` вы можете определить шаблон регулярного выражения (regex), который описывает определенный образец поиска. Затем вы можете использовать этот шаблон для поиска совпадений в строке, извлечения определенных частей строки или замены частей строки новыми значениями.

Изучение регулярных выражений не входит в задачи нашего курса. Если вы не знакомы с ними, рекомендую прочитать статью по ссылке в конспекте лекции.

Давайте разберем код:

1. `soup.find('div', text=re.compile('Форма выпуска:'))` ищет в HTML-документе первый элемент `div`, который содержит точный текст «Форма выпуска:». Параметр `text` представляет собой регулярное выражение, которое позволяет находить частичное совпадение с текстом строки.
2. `.get_text(strip=True)` извлекает текстовое содержимое элемента `div`, найденного на шаге 1, удаляя все пробелы. Для этого установлен параметр `strip` в `True`.
3. `[15:]` разрезает полученную строку, чтобы исключить первые 15 символов, а именно фразу «Форма выпуска:».

Аналогично выполняем парсинг «содержания» и «производителя».

Перейдем к извлечению цены:

```
price = soup.find('p', ('class', 'price-details text-grey-darker text-4xl font-sans font-bold mb-0')).find('span').text
price = price.replace(',', '.')
price = re.sub(r'^\d.+', '', price)
float(price)
```

1. `soup.find('p', ('class', 'price-details text-grey-darker text-4xl font-sans font-bold mb-0'))` ищет первый элемент `p` в HTML-документе, который имеет атрибуты класса `price-details text-grey-darker text-4xl font-sans font-bold mb-0`.
2. `.find('span')` находит первый элемент `span`, который является дочерним элементом элемента `p`, найденного на шаге 1.
3. `.text` извлекает текстовое содержимое элемента `span`, найденного в шаге 2, которое должно быть значением цены в виде строки.
4. `price = price.replace(',', '.')` заменяет все символы запятой в строке цены на точку. Это делается потому, что для конвертации строки (`string`) в числовое значение (`float`) нужно заменить запятую на точку.
5. `price = re.sub(r'^\d.+', '', price)` использует модуль `re` для выполнения регулярного выражения, которое удаляет из строки цены любые символы валюты или другие нечисловые символы.
6. `float(price)` преобразует полученную строку в число с плавающей точкой.

Наконец, объединим все, что мы сделали, и выполним скрейпинг информации по каждому продукту.

Создаем пустые списки, которые будут содержать данные соответствующих полей.

```
name = []
price = []
form = []
content = []
manufacturer = []
```

В цикле проходим по ссылкам на товары и выполняем скрейпинг с помощью BeautifulSoup.

```
for i in url_joined:
    response = requests.get(i)
    soup = BeautifulSoup(response.content, 'html.parser')
```

В следующем блоке кода выполняем парсинг так, как мы разбирали выше. В дополнение обрабатываем исключения с помощью try-except на случай, если какая-то информация не будет найдена. В случае появления исключения в список добавим пустую строку, то есть name.append(''). Все собранные данные записываем в словарь output = {}.

```
# Цикл по списку ссылок на товары
for i in url_joined:
    response = requests.get(i)
    soup = BeautifulSoup(response.content, 'html.parser')

# Парсинг названия товара. Обработка исключения: добавляем пустую строку.
    try:
        name.append(soup.find('h1', ('class', 'h2 mt-0 sm:mb-6')).text)
    except:
        name.append('')

# Парсинг цены товара.
    try:
        p = soup.find('p', ('class', 'price-details text-grey-darker text-4xl font-sans font-bold mb-0')).find('span').text
        p = p.replace(',', '.', '')
        p = float(re.sub(r'^\d.]+', '', p))
        price.append(p)
    except:
        price.append('')
```

```

# Парсинг формы выпуска товара.
try:
    form.append(soup.find('div', text=re.compile('Форма
выпуска:'))).get_text(strip=True)[15:])
except:
    form.append('')

# Парсинг содержания (веса) товара.
try:
    cont = soup.find('div',
text=re.compile('Содержание:')).get_text(strip=True)
    cont = int(re.findall(r'\d+', cont)[0])
    content.append(cont)
except:
    content.append('')

# Парсинг производителя товара.
try:
    manufacturer.append(soup.find('div',
text=re.compile('Производитель:')).get_text(strip=True)[15:])
except:
    manufacturer.append('')

# Записываем данные в словарь
output = {'Name' : name, 'Price' : price, 'Form' : form, 'Content' :
content, 'Manufacturer' : manufacturer}

```

Наконец, мы можем трансформировать наш словарь в датафрейм:

```

df = pd.DataFrame(output)
df

```

	Name	Price	Form	Content	Manufacturer
0	ALLERGO-MOMELIND von DoppelherzPharma	8,11	Nasendosierspray	18	Queisser Pharma GmbH Co KG
1	MometaHEXAL Heuschnupfenspray	10,59	Nasenspray	18	Hexal AG
2	Lorano akut Tabletten	4,99	Tabletten	20	Hexal AG
3	Cetirizin axicur 10 mg Filmtabletten	9,14	Filmtabletten	100	axicorp Pharma GmbH

Мы получили все нужные данные с одной страницы. Переходим к следующему этапу — скрейпингу данных из нескольких страниц. Процесс скрейпинга информации из нескольких страниц называется **pagination**.

Beautiful Soup не имеет встроенной функциональности для выполнения пагинации. Для решения проблемы можно использовать встроенные библиотеки Python — requests и urllib.

Разберем основные элементы кода для сбора данных с нескольких страниц с помощью BeautifulSoup:

```
url_1 = 'https://www.apteka.de'
url = 'https://apteka.de/online-shop/allergie-heuschnupfen'
```

url_1 — это базовый URL, а url — начальный URL первой страницы, которую нужно скрейпить.

```
while True:
    page = requests.get(url)
    soup = BeautifulSoup(page.content, 'html.parser')
    next_page_link = soup.find('a', ('class', 'arrow next'))
    result = soup.find_all('div', ('class', 'product-box-container'))
```

Цикл while True будет выполняться до тех пор, пока не закончатся новые страницы для скрейпинга. Метод requests.get() используется для получения HTML-содержимого текущей страницы.

Метод BeautifulSoup() используется для парсинга HTML-содержимого текущей страницы. Переменная next_page_link используется для поиска ссылки на следующую страницу (если она существует) с помощью метода find() и поиска тега (<a>) с классом arrow next. Переменная result используется для поиска всех элементов HTML, содержащих данные, которые нужно получить.

Далее код дублирует то, что мы делали для скрейпинга одной страницы. В завершении следующие строки кода используются для прерывания цикла, когда больше нет следующей страницы. Если следующая страница есть, то переменная url устанавливается в URL следующей страницы, который является базовым URL (url_1) плюс значение атрибута 'href' из next_page_link.

```
if not next_page_link:
    break
url = url_1 + next_page_link['href']
```

Полный код приведен в блокноте Google Colab.

Заключение

В этой лекции мы рассмотрели основы веб-скрейпинга и сбора данных. Сосредоточились на использовании BeautifulSoup в качестве библиотеки Python для парсинга HTML-документов.

Мы поговорили о BeautifulSoup как об объектно-ориентированной библиотеке, которую можно использовать для навигации по дереву HTML-документа, поиска определенных элементов и атрибутов, извлечения текста и данных.

Мы рассмотрели, как использовать конструктор BeautifulSoup() для создания объекта BeautifulSoup. Мы также разобрались, как получать доступ к тегам и атрибутам в HTML-документах, перемещаться по дереву и искать определенные элементы с помощью find() и find_all().

В заключение лекции мы обсудили несколько реальных примеров использования веб-скрейпинга и то, как работать с пагинацией при скрейпинге нескольких страниц данных.

Beautiful Soup — это простая, но мощная библиотека для скрейпинга и парсинга HTML-документов и извлечения данных с веб-сайтов. Это важный инструмент для сбора данных, который будет полезен в самых разных сценариях.

До встречи на следующей лекции!

Что можно почитать еще?

1. [Документация BeautifulSoup](#)
2. [Библиотека requests](#)
3. [Скрапинг интернет-ресурсов: критерии законности](#)
4. [Парсинг — это законно?](#)
5. [Модуль re Python – осваиваем регулярные выражения Python](#)