МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Пензенский государственный университет» (ПГУ)

А. Г. Петренко, С. Н. Катков

ПАРСИНГ ВЕБ-САЙТА НА PYTHON

Учебное пособие

Пенза
Издательство ПГУ
2026

УДК 004(075.8)
ББК 65ся7
П30

Р е ц е н з е н т ы :
кандидат экономических наук,
заведующий кафедрой «Информационные системы и цифровые технологии»
Пензенского казачьего института технологий (филиала)
Московского государственного университета технологий и управления
имени К. Г. Разумовского (Первый казачий университет)»
А. А. Тусков;

кандидат технических наук, доцент кафедры
«Вычислительная техника»
Пензенского государственного университета
Н. В. Слепцов

Петренко, Александр Григорьевич.

П30 Парсинг веб-сайта на Python : учеб. пособие / А. Г. Петренко, С. Н. Катков. – Пенза : Изд-во ПГУ, 2026. – 68 с

ISBN 978-5-908090-29-2

Приводится описание процессов краулинга, парсинга и скрапинга. Подробно разбирается процесс парсинга на Python, который включает анализ структуры HTML и выделение необходимых данных. Рассматриваются примеры для практической работы с этими программными средствами.

Издание подготовлено на кафедре «Цифровая экономика» ПГУ и предназначено для обучающихся по направлению подготовки 38.03.05 «Бизнес-информатика, а также 38.03.01 «Экономика», 38.03.04 «Государственное и муниципальное управление», 41.03.05 «Международные отношения» и другим смежным направлениям.

УДК 004(075.8)
ББК 65ся7

ISBN 978-5-908090-29-2

© Пензенский государственный университет, 2026

СОДЕРЖАНИЕ

1. КРАУЛИНГ, ПАРСИНГ И СКРАПИНГ 4
1.1. Краулинг 5
1.2. Парсинг 5
1.3. Скрапинг 6
1.4. Процесс парсинга (скрапинг) 7
1.5. Выбор инструментов для веб-скрапинга 9
2. ОСНОВЫ СТРУКТУРЫ ВЕБ-САЙТОВ 11
2.1. Язык HTML 11
2.2. Основные концепции CSS для веб-парсинга 15
3. ПАРСИНГ САЙТОВ НА PYTHON 19
3.1. Библиотеки для веб-парсинга на Python 19
3.2. Реализация процесса обработки веб-сайта 22
3.3. Использование Beautiful Soup для анализа и извлечения данных 40
3.4. Создание парсера 47
3.5. Парсинг одной веб-страницы 50
3.6. Обработка многостраничного контента 53
4. ПАРСИНГ ТАБЛИЦ ИЗ САЙТА НА PYTHON 60
4.1. Преимущества сбора табличных данных 60
4.2. Извлечение таблиц с веб-сайтов с помощью Python 61
СПИСОК ЛИТЕРАТУРЫ 65

1. КРАУЛИНГ, ПАРСИНГ И СКРАПИНГ

Одной из главных задач современного бизнеса является регулярное извлечение информации из Интернета для дальнейшего анализа. Информация в Интернете обычно представлена в неструктурированном или полуструктурированном виде, плохо пригодном для дальнейшей обработки.

Веб-скрапинг – это технология извлечения данных с веб-сайтов и сохранения их в структурированном формате для дальнейшего анализа или использования. Она включает автоматическое получение информации с веб-сайтов и преобразование ее в структурированный формат, который можно легко анализировать и хранить.

Процесс веб-скрапинга включает:

– сбор HTML-данных с домена;

– парсинг данных с целью получения интересующей информации;

– хранение извлеченной информации;

– перемещение на другую страницу для повторения процесса (опционно).

Современный Интернет имеет сложную техническую и логическую структуру, состоящую из нескольких взаимосвязанных уровней.

Основу организации логической (информационной) структуры составляют следующие объекты:

Веб-сайт – совокупность связанных между собой веб-страниц, объединенных под одним доменным именем и общей тематикой или дизайном.

Веб-страница – отдельный документ, доступный в Интернете и отображаемый браузером. Обычно написан на языке HTML и может содержать текст, изображения, видео, аудио и другие мультимедиа элементы.

Контент – информационное содержимое веб-страницы или сайта.

Гиперссылка – элемент веб-страницы, позволяющий осуществлять переход к другой странице или ресурсу.

Для эффективной работы с данными, связанной с поиском и обработкой информации, содержащейся в веб-сайтах, выделяют три основных процесса:

– краулинг;

– парсинг;

– скрапинг.

1.1. Краулинг

Веб-краулинг представляет собой процесс автоматического обхода веб-сайтов с целью сбора информации о страницах для индексации поисковыми системами. Также в задачу краулинга входит обход защиты от парсеров, которую часто устанавливают на серверах, например, краулер использует прокси-сервера для обхода блокирования по IP и т.д.

Основной целью краулинга является создание поисковых индексов, которые позволяют находить нужную информацию в сети. Этот процесс может быть масштабным и часто включает миллионы веб-страниц.

Таким образом, краулинг предназначен для перебора страниц Интернета с целью занесения информации о них в базу данных поисковых систем, таких как Google, Bing и Yahoo, основное назначение которых – индексация миллионов веб-страниц с целью предоставления пользователям результатов поиска в виде:

Веб-архив. Некоторые организации сканируют и сохраняют копии веб-страниц для создания веб-архивов, которые могут использоваться в исследованиях или для доступа к старой информации.

Цена и анализ конкурентоспособности. Компании могут использовать веб-краулинг для мониторинга цен на товары, а также анализа конкурентов и рынка.

Медиамониторинг. Медиакомпании и аналитики используют веб-краулинг для отслеживания новостей, обсуждений и социальных медиа в реальном времени.

Сбор данных и исследования. Исследователи и аналитики могут выполнять веб-краулинг для сбора данных, анализа трендов и исследований в разных областях.

1.2. Парсинг

Изначально программы для нахождения и извлечения информации из различных источников назывались **парсерами** от английского слова “parsing”, означающее грамматический разбор. Во время парсинга интернет – страницы программа-парсер сканирует HTML-документ, находит нужную информацию и возвращает найденные данные в формате электронной таблицы.

Метод **веб-парсинга** используется для извлечения HTML-контента с целевой веб-страницы и его анализа.

Парсинг веб-страниц на Python или любом другом языке может быть утомительным. Нет двух веб-сайтов, организованных одинаково, а HTML часто бывает беспорядочным. Более того, сайты со временем меняются. Веб-скраперы, которые работают сегодня, не обязательно будут работать в следующем году или на следующей неделе.

В основном данные на странице содержатся в неструктурированном формате, который необходимо преобразовать в структурированный после выполнения предварительной обработки.

Таким образом, процедура парсинга в изначальном понимании подразумевает обработку одного документа (веб-страницы).

1.3. Скрапинг

Термин «скрапинг» стал объединять в себе два этих этапа, т.е. выполняет функции **краулера** и **парсера.**

Вот какое определение веб-скрапинга дается в Википедии:

Веб-скрапинг – это технология получения веб-данных путем извлечения их со страниц веб-ресурсов. Веб-скрапинг может быть сделан вручную пользователем компьютера, однако термин обычно относится к автоматизированным процессам, реализованным с помощью кода, который выполняет GET-запросы на целевой сайт.

Веб-скрапинг используется для синтаксического преобразования веб-страниц в более удобные для работы формы. Веб-страницы создаются с использованием текстовых языков разметки (HTML и XHTML) и содержат множество полезных данных в коде. Однако большинство веб-ресурсов предназначено для конечных пользователей, а не для удобства автоматического использования, поэтому была разработана технология, которая «очищает» веб-контент.

В ходе развития ИТ-технологий структура Интернета усложняется, резко возрастают объем хранимой информации и количество пользователей, а сам процесс обработки начали разделять на два независимых этапа: краулинг и сам скрапинг.

Среди российских пользователей Интернета название скрапинг не прижилось, и весь процесс обхода и сбора данных с веб-сайтов обычно называется парсингом, в то время как среди англоязычных используется термин «скрапинг». В дальнейшем в этой работе будет употребляться термин «парсинг» для обозначения процесса обработки информации из Интернета.

Парсинг – это преобразование одних данных в другие, из неструктурированных или неподходящих в структурированные и подходящие.

Также парсинг, известный как веб-скрейпинг, определяется как автоматизированный сбор данных по Сети.

Таким образом, веб-парсинг (скрапинг) представляет собой процесс извлечения конкретных данных с веб-сайтов для анализа, сохранения или дальнейшего использования. В отличие от краулинга, который ориентирован на обширное извлечение информации, парсинг фокусируется на конкретных данных. Например, парсинг может использоваться для извлечения цен на товары из онлайн-магазинов, новостей с медиапорталов или данных о продуктах с сайтов конкурентов. Все же можно выявить сходства и различия.

Сходства:

– **автоматизация**. Оба процесса основаны на автоматизированном извлечении данных с веб-сайтов, что позволяет сэкономить время и усилия;

– **использование HTTP**. Как краулинг, так и парсинг (скрапинг) используют протокол HTTP для взаимодействия с веб-серверами и получения данных.

***Различия*:**

– **цели**. Краулинг направлен на индексацию веб-сайтов для поисковых систем, в то время как парсинг ориентирован на извлечение конкретных данных для анализа и других целей;

– **объем данных**. Краулинг работает с большими объемами данных и может индексировать миллионы веб-страниц, в то время как парсинг часто работает с ограниченным объемом данных;

– **частота запросов**. Краулинг часто выполняется автоматически и может быть непрерывным процессом, обновляющим индексы поисковых систем, в то время как парсинг может быть однократной операцией или выполняться периодически в соответствии с потребностями пользователя;

– **защита сайтов**. Если многие сайты ограничивают и блокируют доступ парсеров для скачивания данных, то для краулинга ситуация обратная. Владельцы сайтов обычно заинтересованы в рекламе своих сайтов и увеличении количества ссылок на них.

1.4. Процесс парсинга (скрапинг)

Типовой процесс парсинга обычно состоит из нескольких этапов:

1) сбор нужных url-адресов;

2) обход защиты от парсинга, если она присутствует на сайте;

3) отправка GET или POST запросов на сайты;

4) поиск необходимой информации в разметке;

5) форматирование собранной информации;

6) экспорт результатов парсинга.

Основные инструменты и процессы веб-парсинга:

1) **HTML:** язык разметки гипертекста (HTML) используется для создания веб-страниц. Веб-парсинг основывается на понимании структуры и элементов HTML для извлечения данных;

2) **CSS-селекторы:** CSS-селекторы используются для идентификации и нахождения конкретных элементов на веб-странице;

3) **HTTP-запросы:** веб-парсеры имитируют веб-браузер, отправляя HTTP-запросы на целевой веб-сайт для доступа и получения HTML-контента страницы;

4) **парсинг:** после получения HTML-контента выполняется процесс парсинга, который включает анализ структуры HTML и выделение необходимых данных;

5) **извлечение и сохранение данных:** после парсинга HTML, веб-парсеры могут извлекать нужные данные и сохранять их в различных форматах, таких как CSV, JSON или Excel.

Правила проведения веб-парсинга

Веб-скрапинг может вызывать некоторые этические и юридические вопросы, которые важно учитывать.

Правила и условия – ознакомьтесь с правилами сайта перед парсингом. Парсите только те данные, на использование которых у вас есть разрешение.

Конфиденциальность данных – будьте осторожны при извлечении личных данных, таких как адреса электронной почты и имена пользователей.

Атрибуция – при публикации собранных данных указывайте авторство и ссылайтесь на исходный источник.

Загрузка сервера – ограничьте частоту и объем запросов, чтобы не перегружать сервер.

Блокировка – используйте прокси и случайные IP-адреса, чтобы избежать попадания в черный список.

Авторское право – ознакомьтесь с правилами владения данными и добросовестного использования при повторной публикации собранного контента.

Веб-парсинг должен проводиться ответственно и этично. Важно убедиться, что веб-сайт разрешает парсинг, и получить соответствующее разрешение. Законность зависит от множества факторов – действуйте осторожно и с уважением к владельцам данных.

1.5. Выбор инструментов для веб-скрапинга

Существуют различные инструменты и подходы к парсингу данных с веб-сайтов, отличающихся по сложности и предоставляемым возможностям.

Приложение Excel (Power Query)

Excel имеет встроенный инструмент Power Query (в версиях 2010 и выше доступен как надстройка “Power Pivot и Power Query”, в более поздних версиях интегрирован). Power Query позволяет импортировать данные из различных источников, включая веб-страницы. Он обладает достаточно понятным интерфейсом и позволяет выполнять базовый парсинг, например, извлечение таблиц с веб-страниц. Это хороший вариант для простых задач и пользователей, не обладающих навыками программирования.

Внешние инструменты и программы

Существует множество специализированных программ и онлайн-сервисов для парсинга данных. Они предлагают более продвинутые функции, такие как обход блокировок сайтов, работа с динамическим контентом, распознавание изображений и многое другое. Некоторые из них имеют графический интерфейс, другие требуют написания простых скриптов. Примеры таких инструментов: Octoparse, ParseHub, WebHarvy.

Расширения для браузера

Парсеры сайтов, выполненные в виде расширений для браузера, удобны с точки зрения использования, установка минимальная – нужен всего лишь браузер, захват данных визуальный не требует программирования. Примеры некоторых расширений:

Scrape.it – расширение для браузера Chrome для сбора данных с сайтов с визуальным Point-Click интерфейсом.

Web Scraper.io – инструмент для парсинга сайтов, выполненный в виде расширения для Chrome, сервис с большим набором опций и возможностью визуального программирования парсинга.

Data miner – расширение для Google Chrome и Microsoft Edge помогает собирать данные с сайтов с помощью простого визуального интерфейса.

Все эти расширения платные, но предоставляют возможность бесплатного использования с ограниченными возможностями по срокам использования или по объему обрабатываемой информации.

Языки программирования (VBA, Python)

Для самых сложных задач и высокой степени кастомизации можно использовать языки программирования, такие как VBA (встроенный в Excel) или Python. Это требует определенных навыков программирования, но предоставляет максимальную гибкость и контроль над процессом парсинга. Существуют специальные библиотеки для Python, такие как Beautiful Soup и Scrapy, которые значительно упрощают процесс веб-парсинга.

2. ОСНОВЫ СТРУКТУРЫ ВЕБ-САЙТОВ

Для успешного веб-парсинга понимание структуры языка HTML и CSS-классов на веб-странице является ключевым фактором.

Необходимо иметь представление о структуре веб-страницы, построенной на языке HTML, знании его основных элементов и их взаимосвязях.

Также, чтобы организовать эффективный поиск и загрузку требуемых данных, нужно иметь представление о селекторах CSS-языка стилей для оформления веб-страниц,

2.1. Язык HTML

HTML (*HyperText Markup Language*) – это язык разметки, используемый для создания веб-страниц. HTML состоит из различных элементов, которые формируют структуру и содержание страницы. Веб-парсинг основывается на понимании структуры и элементов HTML для извлечения данных.

Основные понятия:

Теги – это основные строительные блоки HTML-документа, которые выделяются угловыми скобками < >. Они имеют открывающий и закрывающий теги, которые обрамляют содержимое элемента. Например, – это открывающий тег для абзаца, а – закрывающий тег. Теги определяют тип и структуру элемента на странице.

Теги обрамляют элементы HTML и указывают, как они должны отображаться в браузере. Примеры тегов: <html>, <head>, <title>, <body>, <h1>, , <a>.

***Элемент HTML*:** состоит из открывающего тега, текста и заканчивается закрывающим тегом, где перед именем элемента предшествует косая черта.

Примеры элементов:

```
<h1> Это заголовок</h1>
<p> Это абзац </p>
```

В этом примере используются теги ** ** и ** **. Тег ** ** является начальным тегом абзаца, а ** ** – закрывающим тегом этого же абзаца, а вся строка ** Это абзац ** является элементом абзаца.

Некоторые HTML-элементы не нужно закрывать, например, элементы **<img>**, **<hr>** и **
**.

Атрибуты используются для определения характеристик HTML-элемента и помещаются внутри открывающего тега элемента и отделяются пробелом от имени тега. Атрибуты содержат дополнительную информацию об элементах HTML.

Пример: `<a href="https://example.com">Example</a>` – здесь href является атрибутом тега <a>.

Все атрибуты состоят из двух частей – это имя и значение.

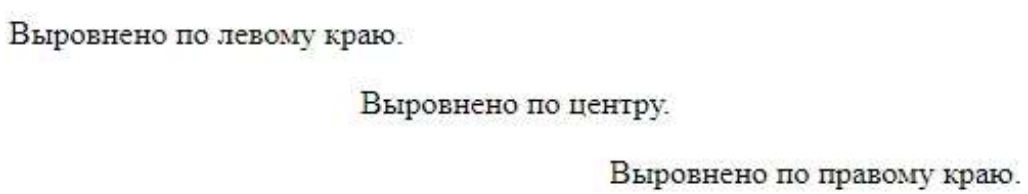
Имя – это свойство, которое нужно установить. Например, элемент абзаца <p>, в примере ниже, содержит атрибут **align**, который вы можете использовать для указания выравнивания абзаца на странице.

Значение – это значение, которое нужно установить для свойства. Значение атрибута всегда помещается в кавычки. Например, для атрибута align могут быть три возможных значения: **left**, **center** и **right**.

Пример:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Пример атрибута align</title>
  </head>
  <body>
    <p align="left">Выровнено по левому краю.</p>
    <p align="center">Выровнено по центру.</p>
    <p align="right">Выровнено по правому краю.</p>
  </body>
</html>
```

Результат:



Текст – это содержимое между открывающим и закрывающим тегами, которое отображается на странице.

2.1.1. Базовая структура документа

Простая веб-страница состоит из «базового» набора тегов:

<html> – указывает, что это HTML-код;

<head> – содержит техническую информацию, которую не видит пользователь, например, заголовок страницы, подключаемые стили и скрипты, мета-информацию и т.д.;

<title> – содержит название веб-страницы, которое будет отображаться во вкладке браузера;

<body> – «тело» или основное содержимое документа, которое видит пользователь.

Все базовые теги имеют закрывающие теги и являются вложенными друг в друга (рис. 2.1).

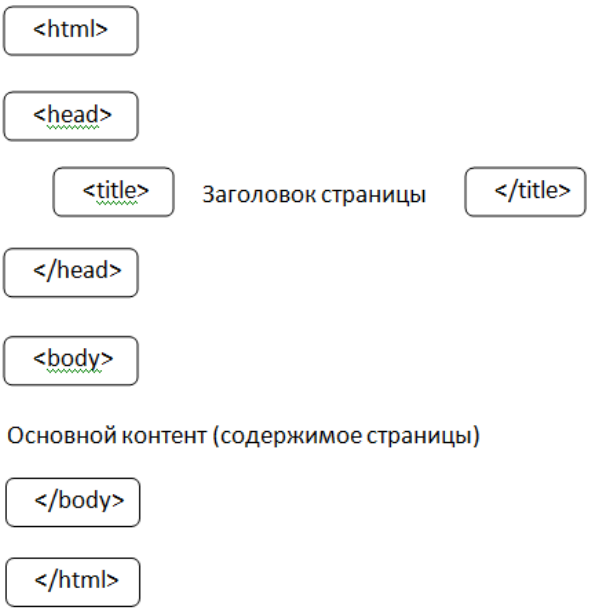


Рис. 2.1. Структура простого HTML-документа

2.1.2. Основные теги HTML

Основными тегами HTML являются:

<html> – корневой элемент, который содержит весь HTML-код страницы. Определяет начало и конец HTML-документа;

<head> – содержит метаданные и информацию о документе, такие как заголовок страницы, ссылки на стили и другие настройки;

<title> – задает заголовок страницы, который отображается в заголовке браузера;

<body> – содержит основной контент страницы, такой как текст, изображения и ссылки и другие элементы, видимые для пользователя;

<h1> – **<h6>** – заголовки различных уровней;

<p> – используется для создания абзацев текста;

<a> – создает гиперссылки на другие страницы или ресурсы;

<img> – загружает изображение.

2.1.3. Дополнительные теги HTML

Количество тегов в различных версиях языка HTML больше 100. Из них в данной работе используются:

<ul> – маркированный список;

<li> – элемент списка;

** ** – блоковый элемент, используемый для группировки других элементов;

**** – строчный элемент, используемый для стилизации части текста.

Теги Div и Span

Большинство тегов HTML предназначены для придания контенту некоего значения (тег <p> создает параграф, тег <h1> – заголовок и т.д.). Теги и никакого значения не имеют, но используются очень активно совместно с технологией CSS.

Они используются для того, чтобы группировать области HTML кода и затем подключать к этой группе определенные стили CSS. Это осуществляется при помощи атрибутов *class* и *id*, ассоциирующих данные элементы с селекторами класса или идентификатора CSS.

Разница между тегом **** и тегом ** ** заключается в том, что элемент **** является строчным и обычно используется для группирования небольших областей строчного HTML кода. Элемент ** ** является блоковым (что выражается в наличие перевода строки до и после этого элемента) и используется для группирования более крупных областей кода, например:

```
<div id="scissors">
   <p>Это <span class="paper">здорово</span></p>
</div>
```

2.1.4. Атрибуты HTML-тегов

Атрибуты тегов – это дополнительные сведения или параметры, которые могут быть добавлены к HTML-элементам для определения их характеристик, поведения или внешнего вида. Атрибуты помогают уточнить, как должен вести себя или отображаться элемент на веб-странице. Атрибуты всегда указываются внутри открывающего тега элемента и обычно имеют вид имя="значение".

1. Атрибут href (hypertext reference)

Используется в теге <a> для создания гиперссылок и указания URL-адреса, на который следует перейти при клике на ссылку.

Пример: `<a href="https://www.example.com">Ссылка на пример</a>`

2. Атрибут class

Указывает **класс элемента** и позволяет задать одно и то же свойство стиля для нескольких элементов веб-страницы. Одинаковое значение class можно задавать одному или нескольким HTML-элементам. Используется для применения стилей из таблицы стилей (CSS) или для выборки элементов.

Например, может быть несколько абзацев с классом .content или несколько ссылок с классом .external.

Пример: `<div class="highlighted">Выделенный текст</div>`

3. Атрибут id

Указывает **идентификатор элемента**, который должен быть уникальным во всем документе. Его цель – однозначно идентифицировать элемент при ссылке на него, написании сценариев или стилизации (с помощью CSS). Значение id задается лишь одному HTML-элементу, у нескольких элементов не может быть одинакового значения id в рамках одной отдельной страницы.

Пример: `<p id="unique-paragraph">Уникальный абзац</p>`

Таким образом, основное различие между атрибутами class и id в том, что class используется для группы элементов, а id – для отдельного элемента**.**

2.2. Основные концепции CSS для веб-парсинга

Хотя основное назначение CSS – это стилизация веб-страниц, однако для целей веб-парсинга он также имеет важное значение. Можно рассматривать CSS как инструмент для анализа структуры страницы и выбора элементов.

Понимая, как элементы страниц стилизованы и структурированы с помощью CSS, можно точнее выбрать нужные данные и извлечь их. Рассмотрим, как подключение CSS к HTML, использование селекторов, а также атрибуты style, class, id и name помогают в работе со структурой веб-страниц для парсинга.

Рассмотрим ключевые концепции CSS, которые важны для парсинга.

Селекторы – CSS-селекторы используются для идентификации и нахождения конкретных элементов на веб-странице. Их использование помогает точно идентифицировать нужные данные.

Атрибуты class, id и name являются уникальными идентификаторами, помогающими выделять и различать элементы. Для парсинга они особенно полезны, так как помогают выделить нужные элементы, что упрощает извлечение данных.

2.2.1. CSS-селекторы, свойства, значения

Каскадная таблица стилей состоит из CSS-правил. Каждое CSS-*правило* состоит из *селектора* и *блока декларации* (рис. 2.1).



Рис. 2.1. Структура CSS (правила)

CSS-*селекторы* очень похожи на HTML-*теги*. Селектор указывает на HTML-элемент, для которого определяется визуальный стиль.

Блок деклараций содержит одну или несколько деклараций свойств, разделенных точкой с запятой. Блок деклараций заключают в фигурные скобки.

Каждая декларация состоит из имени *свойства* и его *значения*, записанных через двоеточие.

Свойства – это ключевые слова, такие как color, font-weight или background-color, которым присвоены определенные значения:

```
body {
  font-size: 0.8em;
  color: navy;
}
```

В этом примере создается CSS-селектор body, который соотносится с HTML-тегом <body>. В этом селекторе определяется два свойства – font-size и color, которым присваиваются соответствующие значения. Таким образом, если этот стиль подключить к HTML-

документу, то текст внутри элемента <body> (что на самом деле весь контент в основном окне браузера) будет отображаться шрифтом размером 0.8em и темно-синим цветом.

2.2.2. Типы CSS-селекторов

Селекторы CSS используются для выбора HTML-элементов на веб-страницах, позволяя применять стили на основе их **идентификаторов**, **классов**, атрибутов типа и многого другого. CSS селекторы обычно делят на несколько типов или категорий.

Рассмотрим категорию основных или общих селекторов (элементы отбираются по тегу, .классу или # идентификатору для базовых нужд в области стилизации).

CSS-селектор элемента

Селектор элемента, или HTML-селектор – это простой селектор, который отбирает HTML-элементы по их тегу.

В следующем примере отбираются все существующие на веб-странице элементы <p>. Их текст будет красного цвета и выровнен по центру:

```
p {
    text-align: center;
    color: red;
}
```

CSS-селектор идентификатора

Селектор идентификатора для отбора заданного элемента использует атрибут id HTML-элемента.

Идентификатор элемента уникален в пределах веб-страницы, таким образом, селектор идентификатора используется для выбора одного уникального элемента. Чтобы определить селектор идентификатора, нужно написать символ решетки (#) и имя идентификатора элемента.

В следующем примере определяются стили для HTML-элемента с идентификатором id="para1":

```
#para1 {
  text-align: center;
  color: red;
}
```

Важно! Имя идентификатора не может начинаться с цифры!

CSS-селектор класса

Селектор класса для отбора заданного элемента использует атрибут class HTML-элемента. Чтобы определить селектор класса, нужно написать символ точки (.) и имя класса элемента.

В следующем примере все HTML-элементы с классом class="center" будут отображать текст красного цвета, который выровнен по центру:

```
.center {
  text-align: center;
  color: red;
}
```

Так как один и тот же класс может присваиваться нескольким элементам, вы можете указать на какой именно HTML-элемент должен влиять селектор класса.

В следующем примере только элементы <p> с классом class="center" будут отображать текст красного цвета, который выровнен по центру:

```
p.center {
  text-align: center;
  color: red;
}
```

Также HTML-элементы могут иметь больше одного класса.

В следующем примере элемент <p> будет принимать стили, соответствующие селекторам класса class="center" и class="large":

```
<style>
p.center {
  text-align: center;
  color: red;
}
p.large {
  font-size: 300%;
}
</style>
<p class="center large">Этот параграф определен по двум классам.</p>
```

Важно! Имя класса не может начинаться с цифры!

3. ПАРСИНГ САЙТОВ НА PYTHON

Веб-парсинг – извлечение данных из сайтов в Интернете. Для этого используется программа парсер-инструмент, позволяющий осуществлять веб-парсинг.

Основной функцией парсеров является получение нужной информации с веб ресурсов. В основном парсеры используются для:

– поиска и извлечения данных;

– преобразования информации в другие форматы;

– мониторинга изменений на сайтах.

Python является одним из самых популярных языков программирования и содержит большое количество библиотек и фреймворков для веб-парсинга. Это делает его идеальным вариантом для парсинга сайтов. Скрипт для веб-парсинга на Python требует относительно небольшого количества кода.

Python широко используется в веб-парсинге благодаря своим преимуществам:

– простой синтаксис: интуитивно понятный синтаксис Python позволяет быстро писать код для парсинга;

– встроенные библиотеки: Python имеет встроенные библиотеки и модули наподобие urllib и lxml, помогающие в парсинге;

– качественные библиотеки парсинга: библиотеки наподобие Beautiful Soup и Scrapy упрощают парсинг любых масштабов;

– универсальность: Python можно использовать для создания готовых конвейеров данных на основе парсинга;

– совместимость: Python хорошо интегрируется с другими языками и имеет достаточную производительность.

3.1. Библиотеки для веб-парсинга на Python

Можно создать скрипт веб-парсера с нуля с помощью операторов языка Python, но это не идеальное решение. Более продуктивным является использование библиотек и фреймворков на основе Python для веб-парсинга.

Среда Python предоставляет большой выбор инструментов для парсинга веб-сайтов с разными характеристиками: простотой освоения, быстродействием и т.д.

Одной из основных важных характеристик библиотеки является ее способность работать со статическими или динамическими страницами.

Статические страницы содержат фиксированный HTML. Его код не меняется после загрузки. По этой причине статические страни-

цы проще парсить. Достаточно отправить HTTP-запрос и будет получен весь HTML-документ, из которого можно извлечь целевые теги. С этой задачей успешно справляются любые библиотеки для парсинга: Requests, BeautifulSoup, Urllib3 и др.

Динамические страницы формируют окончательный контент с помощью JavaScript после загрузки основного HTML. Контент может подгружаться через API, изменяться при взаимодействии пользователя или обновляться автоматически.

Простой HTTP-запрос не парсит динамически загруженные сайты. Для этого нужны инструменты, эмулирующие работу браузера, например, Selenium WebDriver. Он запускает браузер, выполняет JavaScript-код и парсит финальное DOM-дерево с динамическим контентом.

Для веб-парсинга с использованием инструментов Python часто используются библиотеки:

Requests. Библиотека Requests позволяет выполнять HTTP-запросы на языке Python. Она упрощает отправку HTTP-запросов, особенно по сравнению со стандартной библиотекой Python HTTP. Requests играет ключевую роль в проекте для веб-парсинга на Python. Это связано с тем, что для сбора данных, содержащихся на странице, необходимо сначала получить их с помощью HTTP-запроса GET. Кроме того, возможно, придется выполнить и другие HTTP-запросы к серверу целевого сайта.

Beautiful Soup. Python-библиотека Beautiful Soup упрощает сбор информации со страниц, но работает только со статическими страницами.

Основные возможности Beautiful Soup включают:

– *преобразование полученного HTML-контента*. Полученный в результате запроса HTML-контент из URL-адреса преобразуется в объект Beautiful Soup;

– *навигация по дереву документа*. Изучение структуры объекта с целью настройки поиска нужных элементов по тегам, атрибутам, тексту;

– *поиск элементов*. Методы find(), find_all(), select() и select_one() позволяют находить один или все элементы, соответствующие заданным критериям;

– *извлечение данных*. Получение текста и атрибутов найденных элементов;

– *модификация дерева*. Добавление, удаление и изменение элементов (иногда нужно для определенных задач);

– *поддержка различных парсеров*. Использование html.parser, lxml, html5lib для обработки HTML и XML.

В частности, Beautiful Soup работает с любым HTML- или XML-парсером и предоставляет все необходимое для итерации, поиска и модификации абстрактного синтаксического дерева. Обратите внимание, что Beautiful Soup можно использовать вместе с html.parser – парсером, входящим в стандартную библиотеку Python и позволяющим парсить текстовые HTM-файлы. Также Beautiful Soup помогает обходить DOM и извлекать из него нужные данные.

BeautifulSoup относительно прост для понимания для новичков в программировании и позволяет решать небольшие задачи за короткий срок.

Scrapy – это специализированная платформа для веб-скрапинга на Python. Она предоставляет экосистему инструментов для запросов, парсинга, кэширования, прокси-серверов, автоматизации и экспорта собранных данных.

Более сложный и объемный фреймворк для веб-парсинга, который предоставляет множество настроек и функций для полноценного извлечения данных.

Если BeautifulSoup – это в принципе просто парсер HTML и XML, требующий дополнительных библиотек, таких как requests и urlib2 для открытия ссылок и сохранения результатов, то Scrapy – это самый полный набор инструментов для загрузки веб-страниц, их обработки и сохранения в файлы и базы данных.

Selenium – бесплатная (с открытым исходным кодом) платформа для автоматизированной проверки и тестирования веб-приложения, которую можно использовать в различных браузерах и системах. Кроме того, можно разрабатывать тестовые сценарии Selenium на различных языках программирования, таких как Java, C#, Python и других подобных языках.

Selenium использует настоящий веб-браузер, такой как Chrome или Firefox, для отображения страниц. Это позволяет парсить сложные сайты, управляемые JavaScript, которые не загружаются полностью для запросов и BeautifulSoup

Подходит для взаимодействия с динамическими и JavaScript-генерируемыми страницами. С помощью него можно даже управлять браузером, нажимать кнопки и заполнять формы.

Если контент на странице подгружается динамически с помощью JavaScript, requests не сможет его загрузить. В этом случае используется Selenium, которая позволяет автоматизировать взаимодействие с браузером, отрисовывать страницу и получать полный HTML-код.

Selenium предоставляет все необходимое для создания парсера, не требуя использования других библиотек.

3.2. Реализация процесса обработки веб-сайта

Процесс обработки, в общих чертах, выглядит так:

1. Поиск и загрузка HTTP-контента из указанного URL-адреса веб-сайта.

Это может быть заранее заданный один или список URL-адресов. Также можно рекурсивно переходить по ссылкам на страницах, чтобы охватить весь сайт.

2. Преобразование полученного сырого контента в структурированный вид с формированием дерева DOM;

3. Изучение структуры загруженного HTML-сайта (синтаксический анализ структуры полученного HTTP-документа на основе построенной модели DOM);

4. Извлечение нужных элементов HTTP-документа (заголовков, строк таблицы, ссылок или абзацев и т.д.), их обработка и сохранение в базе данных или в выходном файле в структурированном формате, таком как JSON или CSV.

Третий и четвертый этапы схожи между собой по используемым инструментам и методам, но цели у них разные.

Если изучение структуры предназначено для определения места точного расположения и извлечения нужной информации желательно без дополнительных затрат на последующую очистку, связанную с дублированием и удалением ненужных избыточных данных, то четвертый этап использует эти данные для непосредственного извлечения.

Рассмотрим эти этапа подробнее.

3.2.1. Поиск и загрузка HTTP-контента

HTTP означает «протокол передачи гипертекста» (*HyperText Transfer Protocol*). Это основной интернет-протокол современного веб-интернета. В основе его работы лежат запросы и их методы, определяющие, что именно будет происходить с данными на серверах, т.е. это список правил, по которым компьютеры обмениваются данными в интернете.

Основным объектом манипуляции в HTTP является ресурс, на который указывает его адрес URL в запросе клиента. Обычно такими ресурсами являются хранящиеся на сервере файлы.

Физически интернет-протокол можно представить как взаимодействие двух объектов: клиента и сервера. Клиент отправляет запросы и ждет ответа (данных) от сервера, а сервер ждет, когда ему придет очередной запрос, обрабатывает его и возвращает ответ клиенту (рис. 3.1) .

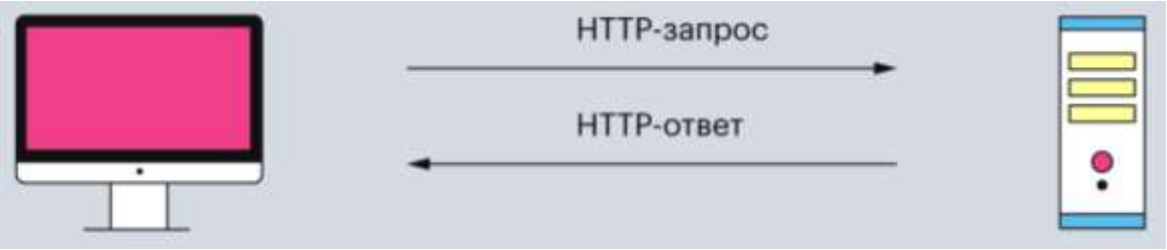


Рис. 3.1. Взаимодействие клиента и сервера

Обычно эта связь между клиентом и сервером имеет посредников в виде прокси-серверов. Они нужны для разных операций – например, для безопасности и конфиденциальности, кэширования или распределения нагрузки на серверы. Чаще всего прокси-серверы используют для кэширования и сжатия данных, обхода ограничений и анонимных запросов. Как правило, между клиентом и основным сервером находится один или несколько таких прокси-серверов.

Поэтому типичная процедура отправки HTTP-запроса от клиента выглядит так (рис. 3.2).

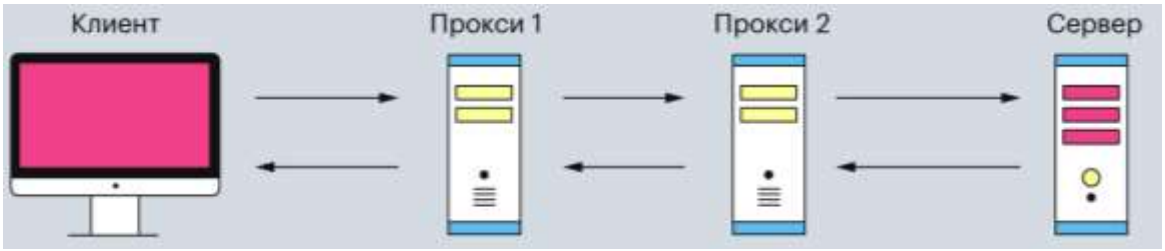


Рис. 3.2. Использование прокси-серверов

Метод – это действие, которое клиент ждет от сервера в ответ на посланный запрос. Например, размещение отправленной серверу HTML-страницы сайта или скачивания документа. Протокол HTTP не ограничивает количество разных методов. Ключевые из них – GET и POST:

– **GET** – получить данные с сервера;

– **POST** – отправить данные на сервер.

Процесс передачи HTTP-запроса серверу и получения результата для клиента содержит несколько шагов и его реализация достаточно трудоемкая.

Библиотека Requests для Python облегчает эту задачу и позволяет работать с HTTP-запросами любого уровня сложности, используя простой синтаксис. Это помогает не тратить время на написание кода, а быстро взаимодействовать с серверами.

Преимущества выбора Requests

Python Requests – это библиотека, которая создана для быстрой и простой работы с запросами. Стандартные HTTP-библиотеки Python, например, та же Urllib3 часто требуют значительно больше кода для выполнения одного и того же действия, а это затрудняет работу.

Библиотека Requests является стандартным инструментом для составления HTTP-запросов в Python и значительно облегчает трудоемкий процесс создания и выполнения запросов.

Библиотека Requests позволяет выполнять HTTP-запросы на языке Python. Она упрощает отправку HTTP-запросов, особенно по сравнению со стандартной библиотекой Python HTTP. Requests играет ключевую роль в проекте для веб-парсинга на Python. Это связано с тем, что для сбора данных, содержащихся на странице, необходимо сначала получить их с помощью HTTP-запроса GET. Кроме того, возможно, придется выполнить и другие HTTP-запросы к серверу целевого сайта.

GET-запрос

Методы HTTP, такие как GET и POST, определяют, какое действие требуется выполнить при отправке HTTP-запроса. Помимо GET и POST, существует еще несколько распространенных методов.

Одним из наиболее распространенных методов HTTP является GET. Метод GET указывает, что вы пытаетесь получить данные из указанного ресурса. Чтобы выполнить запрос GET, надо вызвать метод requests.get().

Например, можно сделать запрос GET к сайту Института экономики и управления, вызвав get() со следующим URL-адресом:

```
requests.get(‘https://ieu.pnzgu.ru/‘)
```

В ответ конструктор создает объект класса Response, который используется в различных контекстах, например, в веб-приложениях, HTTP-клиентах и веб-API:

```
<Response [200]>
```

Созданный объект используется для проверки и обработки результатов запроса. Повторим тот же запрос, но на этот раз сохраним возвращаемое значение в переменной, чтобы можно было рассмотреть его атрибуты и поведение:

```
response = requests.get(‘https://ieu.pnzgu.ru/‘)
```

В этом примере ответ на запрос, возвращаемый методом get(), который является экземпляром класса Response, сохраняем в переменной с именем response. Теперь можно использовать response, чтобы просмотреть различную информацию о результатах запроса GET.

Структура объекта Response

Объект Response содержит информацию об ответе, которая включает:

– код состояния (например, 200, если запрос выполнен успешно);

– заголовки (объект Headers);

– содержимое ответа (строка, буфер, поток).

Первая информация, которую можно получить из Response, – это код состояния. Код состояния информирует о статусе запроса.

Необходимо всегда проверяйте код состояния (response.status_code), чтобы быть уверенными, что ваш запрос выполнен успешно. Некорректное использование методов find() и find_all() может привести к ошибкам, если вы не учтете структуру HTML-страниц.

Например, статус 200 OK означает, что запрос был успешным, тогда как статус 404 NOT FOUND означает, что ресурс, который искали, не найден. Существует также множество других возможных кодов статуса, которые дают конкретное представление о том, что произошло с запросом.

Пример проверки результата выполнения запроса:

```
import requests
url = 'http://example.com'  # Заменить на реальный URL
response = requests.get(url)
# Убедитесь, что получили успешный ответ
if response.status_code == 200:
    html_content = response.text
else:
    print(f" Не удалось загрузить веб-страницу: {response.status_code}")
```

GET является одним из самых популярных HTTP-методов. В библиотеке Requests функции для формирования HTTP-запросов называются так же, как и сами запросы. Например, для выполнения GET-запроса используем функцию get().

Метод GET указывает на то, что происходит попытка извлечь данные из определенного ресурса. Для того, чтобы выполнить запрос GET, используется функция requests.get().

Пример:

```
import requests

url = "https://www.kdnuggets.com/"
res = requests.get(url)
print(res)
```

Результат:

```
<res [200]>
```

Функция get() вернула ответ HTTP-сервера с кодом 200. Значит, запрос был обработан успешно.

HTTP-коды состояний

Коды состояний имеют вид трехзначных чисел от 100 до 500. Чаще всего встречаются следующие:

- 200 – «OK». Запрос прошел успешно, и получен ответ.
- 400 – «Плохой запрос». Его получаем тогда, когда сервер не может понять запрос, отправленный клиентом. Как правило, это указывает на неправильный синтаксис запроса, неправильное оформление сообщения запроса и т.д.
- 401 – «Unauthorized». Для выполнения запроса необходимы актуальные учетные данные.
- 403 – «Forbidden». Сервер понял запрос, но не может его выполнить. Например, у используемой учетной записи нет достаточных прав для просмотра содержимого.
- 404 – «Не найдено». Сервер не нашел содержимого, соответствующего запросу.

Получение содержимого страницы

Для получения содержимого страницы используется метод content. Он позволяет получить информацию в виде байтов, т.е. в итоге у нас будет вся информация, не только строковая:

Пример:

```
import requests
url = "https://www.kdnuggets.com/"
```

```
res = requests.get(url)
htmlData = res.content
print(htmlData)
```

Результат:

```
b'<!DOCTYPE html><html lang="en-
US"><head><title>Just a moment...</title><meta http-
equiv="Content-Type" content="text/html; charset=UTF-
8"><meta http-equiv="X-UA-Compatible" con-
tent="IE=Edge"><meta name="robots" con-
tent="noindex,nofollow"><meta name="viewport" con-
tent="width=device-width,initial-scale=1"><style
...
```

Этот скрипт извлекает весь сырой необработанный HTML-контент из URL-адреса. Эти необработанные данные содержат все тексты, абзацы, теги привязки, элементы div и т.д. Следующая задача – проанализировать эти данные и извлечь все тексты и теги отдельно.

3.2.2. Преобразование полученного HTML-контента

Парсинг веб-сайта предназначен для решения двух задач:

1) анализ веб-сайта с целью изучения его структуры;

2) извлечение данных из веб-страниц сайта.

После загрузки веб-страниц нужно привести полученный сырой HTML-код к структурированному виду, чтобы легче было провести анализ и понимать, где находятся необходимые данные и как их извлечь.

Преобразование HTML-кода исходного документа

Любой HTML- или XML-документ записывается в определенной кодировке, такой как ASCII или UTF-8. Но при передаче конструктору BeautifulSoup преобразуется в универсальный код Unicode. Затем Beautiful Soup анализирует документ с помощью указанного доступного парсера. Он будет использовать парсер HTML, если не указать другой.

Beautiful Soup поддерживает синтаксический анализатор (парсер) HTML, включенный в стандартную библиотеку Python, а также ряд сторонних синтаксических анализаторов Python, например парсеры lxml, html5lib. Выбор конкретного парсера зависит от требований к параметрам обработки: быстродействии, качества анализа.

Beautiful Soup преобразует сложный HTML-документ в сложное дерево объектов Python. Но на деле придется иметь дело только с четырьмя *видами* объектов: **BeautifulSoup, Tag**, **NavigableString** и **Comment**. Эти объекты представляют собой HTML-элементы, из которых состоит страница:

1. Объект **Beautiful Soup**.

Он представляет анализируемый документ в целом. Это полный HTML-документ, обработанный указанным парсером (html.parser).

Зададим простой документ в виде текстовой стоки:

```
soup = BeautifulSoup("<h1> Welcome to KDnuggets!
</h1>", "html.parser")
print(type(soup))
```

Результат:

После обработки и присвоения получаем переменную soup типа Beautiful Soup:

```
<class 'bs4.BeautifulSoup'>
```

2. Объект **Tag**.

Объект тега соответствует определенному тегу в документе HTML. Он может извлечь тег из всего документа и вернуть первый найденный тег, если в DOM присутствует несколько тегов с одинаковым именем.

Из объекта soup извлечем тег <h1>:

```
soup = BeautifulSoup("<h1> Welcome to KDnuggets!
</h1>", 'html.parser')
print(type(soup.h1))
```

Результат:

```
<class 'bs4.element.Tag'>
```

3. Объект **NavigableString**.

Он содержит текст внутри тега в строковом формате. Beautiful Soup использует объект NavigableString для хранения текстов тега:

```
soup = BeautifulSoup("<h1> Welcome to KDnuggets!
</h1>", "html.parser")
print(soup.h1.string)
print(type(soup.h1.string))
```

Результат:

```
Welcome to KDnuggets!
<class 'bs4.element.NavigableString'>
```

4. Объект **Comment**.

Он читает комментарии HTML, присутствующие внутри тега. Это особый тип NavigableString.

```
soup = BeautifulSoup("<h1><!–Это комментарий --
></h1>", "html.parser")
print(soup.h1.string)
print(type(soup.h1.string))
```

Результат:

```
Это комментарий
<class 'bs4.element.Comment'>
```

Навигация по дереву документа

Одной из основных функций библиотеки Beautiful Soup является анализ и выявление некорректного HTML-кода. Также в ней есть множество других функций, включая возможность определения кодировки страницы, что еще больше повышает точность данных, извлекаемых из HTML-файла.

Переданный библиотеке сырой неструктурированный HTML-код обрабатывается и преобразуется в модель DOM в виде древовидной структуры, более удобной для последующего анализа. При перемещении по ветвям DOM-модели можно находить целевые теги и объекты. Также к полученной структуре применим метод prettify(), который улучшает внешний вид дерева.

Пример:

```
import requests
from bs4 import BeautifulSoup

url = "https://www.kdnuggets.com/"
res = requests.get( url )
htmlData = res.content
parsedData = BeautifulSoup( htmlData,
"html.parser" )
print( parsedData.prettify() )
```

Результат:

```
<!DOCTYPE html>
<html lang="en-US">
 <head>
  <title>
```

```
        Just a moment...
       </title>
       <meta content="text/html; charset=utf-8" http-
equiv="Content-Type"/>
       <meta content="IE=Edge" http-equiv="X-UA-
Compatible"/>
       <meta content="noindex,nofollow" name="robots"/>
       <meta content="width=device-width,initial-
scale=1" name="viewport"/>
       <style>
        *{box-sizing:border-
box;margin:0;padding:0}html{line-height:1.15;-webkit-
text-size-adjust:100%;color:#313131;font-
family:system-ui,-apple-system,BlinkMac…
     }
       </style>
       <meta content="360" http-equiv="refresh"/>
      </head>
      <body>
       <div class="main-wrapper" role="main">
        <div class="main-content">
         <noscript>
          <div class="h2">
           <span id="challenge-error-text">
            Enable JavaScript and cookies to continue
           </span>
          </div>
         </noscript>
        </div>
       </div>
       <script>
        (function(){
     …}());
       </script>
      </body>
     </html>
```

В приведенном результате можно видеть, что Beautiful Soup представляет код HTML в структурированном виде с соответствующими уровню вложения отступами. Функция BeautifulSoup() прини-

мает два аргумента: один – входной HTML-контент, а другой – анализатор. В данном случае используется html.parser, но есть и другие анализаторы, например lxml или html5lib. У всех из них есть свои плюсы и минусы. Одни из них менее требовательные, а другие – более быстрые. Выбор парсера полностью зависит от выбора пользователя.

Метод **prettify()** в Beautiful Soup предназначен для создания удобочитаемого строкового представления HTML-дерева. Он добавляет отступы и переносы строк, чтобы визуально отобразить структуру документа. Ожидается, что при вызове prettify() для объекта Beautiful Soup результатом будет красиво отформатированный HTML-код. Однако в некоторых случаях prettify() может не работать так, как ожидается, и вместо этого выдавать неформатированный или частично отформатированный вывод.

Если prettify() не работает, убедитесь, что Beautiful Soup установлен правильно, HTML-код не содержит ошибок, кодировка документа указана верно, и используется подходящий парсер. Также при необходимости можно использовать альтернативные методы форматирования HTML.

Структура HTML-документа

Страница на HTML состоит из тегов, вложенных в друг друга (рис. 3.3). Самый общий тег – это HTML. В него вкладываются два дочерних тега **head** и **body.**

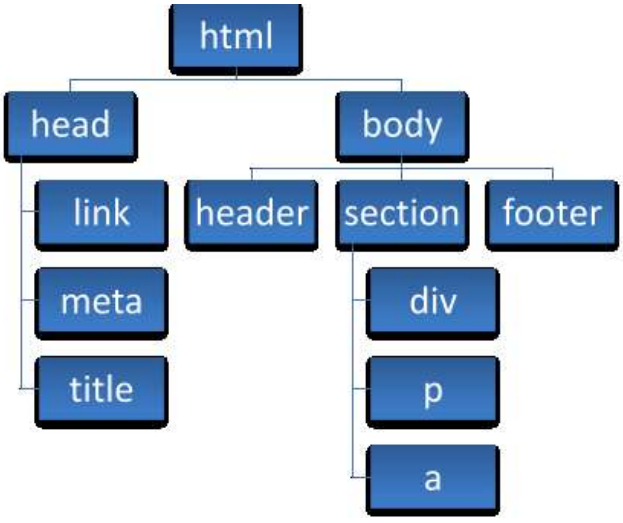


Рис. 3.3. Структура веб-документа

Тег **head** используется для подключения информации, которая не будет отображаться непосредственно на странице, но будет исполь-

зоваться для подключения важных файлов, например, подключение CSS-файлов, подключенные шрифты, название сайта, язык, кодировка, скрипты, которые должны выполняться в первую очередь, иконка сайта и т.д.

В **body** находится значимое содержимое. Обычно в **body** выделяют три части: шапка сайта, основное содержимое и подвал. В шапке обычно содержится верхнее меню сайта, за это отвечает тег **header**. Для основного содержимого нет определенного тега, но обычно используется **section**. Для подвала используется **footer**, там обычно содержатся контактная информация, ссылки на ключевые страницы сайта и копирайт. Теги **header** и **footer** должны быть единственными на странице, а **section** может бесконечно повторяться.

Как строится DOM-дерево

DOM-модель (дерево) – это представление HTML-документа в виде древовидной структуры объектов. Каждый элемент HTML становится узлом этого дерева.

Для описания структуры DOM потребуются термины: *корневой, родительские* и *дочерние элементы*. Корневой элемент находится в основании всей структуры и не имеет родительского элемента. Внутри родительских элементов находятся дочерние и наследуют различные свойства от них.

Дерево HTML представляет собой иерархическое представление информации. Корневым узлом является тег, который может иметь родительские, дочерние и одноуровневые узлы. Тег head и тег body следуют за тегом HTML. Тег head содержит метаданные и заголовок, а тег body содержит элементы div, абзацы, заголовок и т.д.

Ниже на рисунке (рис. 3.4) изображено DOM-дерево.

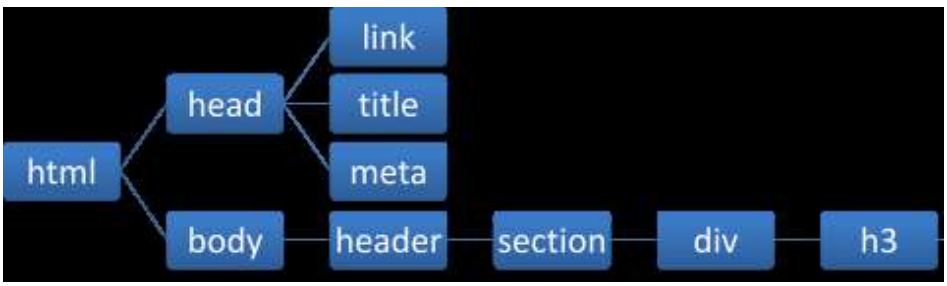


Рис. 3.4. Графическое представление элементов HTML-страницы

Корневой элемент здесь html – без него страница не будет скомпилирована. Он не имеет родительского (вышестоящего) элемента, но имеет два наследника или дочерних элемента – head и body.

По отношению друг к другу элементы head и body являются **элементами, имеющими общего родителя**. В каждый из них можно вложить еще много дочерних элементов. Например, в head обычно находятся link, meta, script или title.

Все эти теги не являются уникальными, и в одном документе может быть по несколько экземпляров каждого из них.

Вот так может выглядеть фрагмент HTML:

```
<div class="article">
    <h2 id="title">Заголовок</h2>
    <p class="content">Это текст статьи...</p>
    <a href="https://example.com"
class="link">Читать далее</a>
</div>
```

Блок div является родительским элементом для h2, p и a. Каждый из них имеет свои атрибуты и содержимое.

В body могут находиться разнообразные элементы. Например, в родительском body – дочерний элемент header, в элементе header – дочерний элемент section, в родительском section – дочерний div, в div – элемент h3. В этом случае h3 не имеет дочерних элементов, но их можно добавить в любой момент.

Если преобразовать дерево на этом рисунке в код, то получится так:

```
<HTML>
    <head>
      <link>
      <title>
      </title>
      <meta>
    </head>
    <body>
      <header>
         <section>
            <div>
               <h3>
               </h3>
            </div>
        </section>
      </header>
    </body>
</HTML>
```

Эту схему важно понимать, чтобы разобраться с темой наследования свойств. Элементы могут наследовать не все, но многие свойства своих родителей, например, цвет, шрифт, видимость и т.д.

Таким образом, чтобы задать стиль шрифта на всей странице, потребуется не прописывать цвет для каждого элемента, а задать его только для body. А чтобы изменить наследуемое свойство у дочернего элемента, нужно прописать только ему новые свойства. Наследование удобно для создания единообразной страницы.

3.2.3. Изучение структуры HTML-сайта

Всегда необходимо анализировать HTML-документ перед тем, как начинать извлечение данных. Для этого можно использовать следующие инструменты:

– **Dev Tools** – инструменты разработчика, встроенные в браузер;

– **BeautifulSoup** – популярная библиотека Python для анализа HTML и XML. Позволяет легко извлекать данные с помощью CSS-селекторов или условий.

Инструменты разработчика Dev Tools

Google Chrome DevTools – это встроенный набор инструментов разработчика, доступный в браузере Chrome, который позволяет исследовать и анализировать веб-страницы. Для веб-парсера DevTools становится незаменимым инструментом, так как позволяет глубже понять структуру страниц, определять селекторы, анализировать динамический контент и сетевые запросы.

В любом браузере есть инструменты, с помощью которых можно отобразить веб-страницу в виде DOM-дерева для удобного анализа ее HTML-структуры. Если выбрать какой-либо объект в структуре, он будет подсвечен на странице. При этом для него будут выведены свойства CSS, которые работают сейчас, и те, которые не задействованы.

Чтобы посмотреть DOM в браузере, нужно зайти в инструменты разработчика **Dev Tools**. В большинстве браузеров для этого надо нажать на F12 – тогда откроется дополнительная панель с вкладками и зонами.

Также можно воспользоваться командой «Исследовать элемент»:

1. Открытие инструментов разработчика. Откройте инструменты разработчика браузера (рис. 3.5), щелкнув правой кнопкой мыши на любом элементе страницы и выбрав «Исследовать элемент» (или с помощью комбинации клавиш Ctrl + Shift + I для Windows или Cmd + Option + I для Mac).

2. Навигация к вкладке Elements. Перейдите на вкладку Elements, чтобы увидеть HTML-код страницы.

3. Использование инструмента селектора. Найдите инструмент селектора (иконка курсора или цели в левом верхнем углу окна Элементы), чтобы выбрать конкретные элементы на странице и подсветить соответствующий HTML-код.

4. Анализ структуры элементов. Изучите структуру и атрибуты выбранного элемента. Определите уникальные идентификаторы, такие как имена классов или ID, которые помогут в процессе парсинга.

5. Рассмотрите родительские и дочерние элементы. Понимание иерархии HTML-элементов необходимо для точного извлечения данных.

6. Анализ формата данных. Обратите внимание на формат данных в HTML-структуре, чтобы определить конкретные элементы, которые нужно парсить.

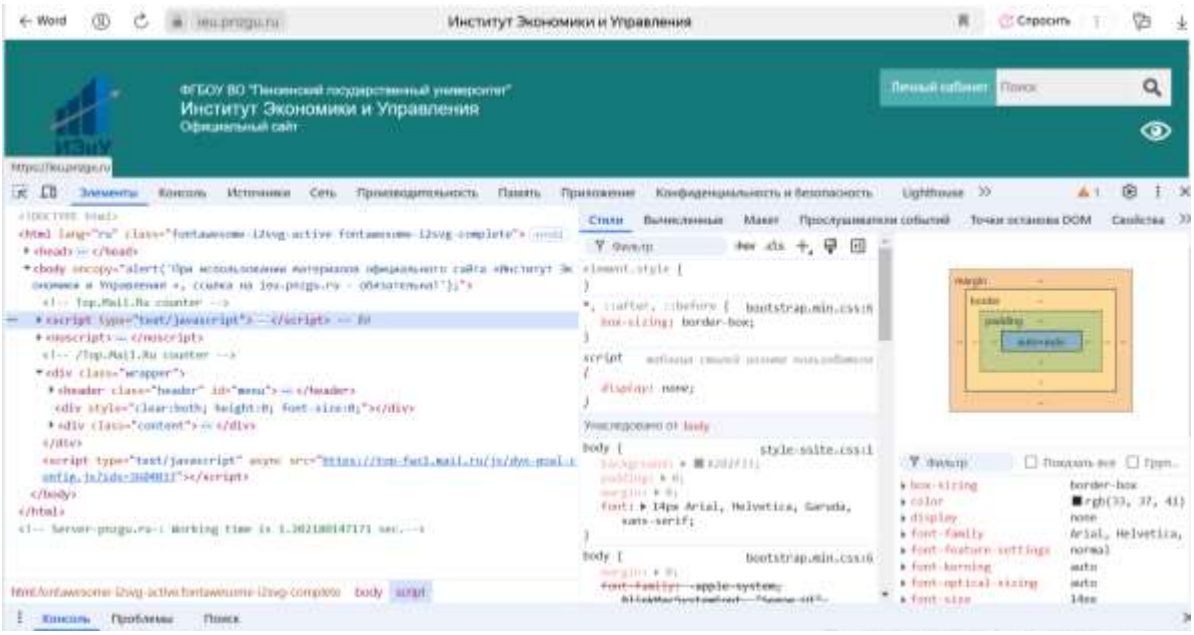


Рис. 3.5. Окно «Инструменты разработчика»
для сайта ИЭиУ ПГУ

Библиотека Beautiful Soup

Библиотека Beautiful Soup – это Python-библиотека, предназначенная для парсинга HTML и XML-документов.

Beautiful Soup принимает сырой текст HTML и опциональный парсер, например, lxml или встроенный html.parser и превращает за-

путанный HTML в дерево парсинга, отражающее DOM-структуру тегов, атрибутов и текста. Можно использовать CSS-селекторы и методы обхода для быстрого выделения нужных нам данных из этого дерева.

Python-библиотека Beautiful Soup упрощает сбор информации со страниц. В частности, Beautiful Soup работает с любым HTML- или XML-парсером и предоставляет все необходимое для итерации, поиска и модификации абстрактного синтаксического дерева. Обратите внимание, что Beautiful Soup можно использовать вместе с html.parser – парсером, входящим в стандартную библиотеку Python и позволяющим парсить текстовые HTM-файлы. Также Beautiful Soup помогает обходить DOM и извлекать из него нужные данные.

Первым аргументом конструктора Beautiful Soup является текстовая строка или документ из файла, содержащий HTML-контент – источник разметки, которую требуется проанализировать. В качестве второго указывается парсер, который будет выполнять синтаксический разбор.

Если нужно просто проанализировать HTML-код, обычно достаточно передать контент в конструктор Beautiful Soup без дополнительных аргументов. Beautiful Soup выберет парсер по умолчанию и проанализирует данные. Но есть несколько дополнительных аргументов, которые можно передать в конструктор, чтобы изменить используемый парсер.

В этом случае для извлечения данных из HTML-кода лучше использовать конструктор Beautiful Soup() с дополнительными аргументами (обычно два аргумента): разметку (HTML-код) и анализатор (необходим для обработки HTML-кода). Версия BS4 поддерживает различные библиотеки для синтаксического анализа, включая стандартные html.parser, а также более быстрые, такие как lxml и html5lib.

Перед тем как приступать к извлечению данных с сайта, важно понять структуру HTML-сайта. Это позволит более эффективно извлекать данные.

Определение конкретных данных для извлечения является важным этапом в процессе парсинга. Это включает определение структуры сайта, понимание его контента и выявление элементов, содержащих нужную информацию:

1) определите цель: четко определите цель извлечения данных, чтобы понять, какая информация нужна;

2) изучите сайт: ознакомьтесь со структурой и организацией целевого сайта. Обратите внимание на отображение данных и иерархию элементов. Важно выделить ключевые теги и атрибуты;

3) проанализируйте исходный код: изучите исходный код сайта, чтобы определить HTML-элементы, содержащие интересующие вас данные. Например, если веб-разработчики разметили данные в своей странице, используя атрибут class="headline" для заголовка, это поможет при извлечении;

4) выявите паттерны данных: проанализируйте контент сайта и выявите повторяющиеся шаблоны в данных, которые нужно извлечь;

5) проведите тесты: перед началом извлечения данных проведите небольшие тесты, чтобы убедиться в точности и надежности вашего подхода.

Для правильного понимания структуры кода страницы в качестве инструмента просмотра можно использовать библиотеку Beautiful Soup, которая предоставляет много методов для поиска различных элементов HTML-контента.

Также для анализа структуры целевого веб-сайта и выявления элементов, содержащих нужные данные, можно использовать инструменты разработчика Dev Tools, которые есть в любом современном веб-браузере.

3.2.4. Извлечение элементов HTTP-документа и использование регулярных выражений

После анализа HTML-документа и определения местоположения необходимой информации можно переходить к непосредственному ее извлечению.

Полный процесс парсинга данных с веб-сайта, кроме предыдущих этапов, также включает написание кода, который должен эффективно извлекать нужную информацию, для чего необходимо выполнить следующее:

1. Создайте скрипт или функцию для парсинга сайта. Используйте уникальные атрибуты для извлечения нужных данных из HTML-структуры, чтобы не увеличивать объем выборки из-за ненужных данных.

2. Обработайте вызовы и ошибки. Парсинг может включать вызовы к динамическому контенту или входу в систему. Используйте соответствующие методы для решения этих задач.

3. Извлеките данные о продукте. Используйте операторы Python и методы BeautifulSoup для извлечения и вывода данных о продукте из HTML-кода страницы.

Для извлечения необходимой информации при необходимости также используют механизм регулярных выражений, который применяется во многих языках программирования и в приложениях для обработки текста.

Регулярные выражения

Для более гибкого извлечения данных можно использовать регулярные выражения.

Регулярные выражения – это шаблоны, которые можно использовать для поиска текста в строке. Python поддерживает регулярные выражения через модуль **re** стандартной библиотеки.

Регулярные выражения состоят из литералов (букв и цифр) и метасимволов.

Примечание. Регулярные выражения не являются особенностью Python. Они представляют собой общую концепцию программирования и поддерживаются многими языками программирования.

Для работы с регулярными выражениями необходимо импортировать модуль re:

import re

В регулярных выражениях используются специальные символы, называемые **метасимволы**, для обозначения различных шаблонов. Например, символ звездочки (*) обозначает ноль или более экземпляров всего, что идет непосредственно перед звездочкой. Для экранирования спецсимволов применяют обратные слэши (\), или же заключают выражение в r-строку. Такой шаблон, к примеру, можно использовать для валидации email-адреса:

```
r'^[a-zA-Z0-9._-]+@[a-zA-Z-.]+$'
```

Этот шаблон – один из простейших, регулярные выражения для проверки email-адресов могут выглядеть гораздо сложнее.

Примеры применения Regex методов

re.match() – проверяет, начинается ли строка с нужного фрагмента:

```
import re
s = 'ой, мороз, мороз, не морозь меня'
print(re.match('мороз', s))
None
```

Чтобы найти первое вхождение подстроки в текст, используют **re.search()**, при необходимости – с флагом re.I для игнорирования регистра:

```
s = 'Синий, синий иней лег на провода'
print(re.search('синий', s, re.I))
<re.Match object; span=(0, 5), match='Синий'>
```

string возвращает строку, содержащую искомый фрагмент:

```
st = 'Дракула Брэма Стокера'
print(re.search('Сток', st).string)
Дракула Брэма Стокера
```

group вернет подстроку, совпадающую с запросом:

```
st = 'пример домашнего хищника: кот'
print(' текст найден - ',
re.search(r'хищника:\s\w\w\w', st).group())
текст найден - хищника: кот
```

Пример использования Regex для парсинга HTML:

```
import re
html_content = '<p>ScrapingAnt is a web scraping
service that allows you to scrape data from websites
and APIs.</p>'
# pattern = r'<p>(.*?)</p>'
pattern = r'<p>(.*?)</p>'
regex = re.compile(pattern)
results = regex.findall(html_content,
re.IGNORECASE | re.DOTALL)
for result in results:
    print(result)
```

В этом примере регулярное выражение соответствует всем тегам <p> и извлекает текст внутри каждого тега. Функция findall находит все совпадения шаблона в HTML-документе, а извлеченный текст выводится в консоль.

Регулярные выражения в поиске

Регулярные выражения (RegEx) – мощный инструмент для поиска и манипуляции текстом, позволяющий создавать сложные критерии поиска.

В Beautiful Soup можно использовать регулярные выражения вместе с методами поиска для более гибкой фильтрации текста.

Пример:

```
import re
soup.find_all(string=re.compile("регулярное выра-
жение"))
```

Применение в парсинге Википедии

Для поиска всех ссылок, содержащих слова, начинающиеся на «Р», можно использовать следующий код:

```
from bs4 import BeautifulSoup
import requests
import re
url = 'https://ru.wikipedia.org/wiki/Python'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
links = soup.find_all('a',
string=re.compile("^P"))
print(links)
```

Реализация регулярных выражений полностью – довольно сложная задача. Используйте регулярные выражения только тогда, когда это действительно необходимо. Старайтесь писать простые и понятные регулярные выражения. Сложные регулярные выражения могут замедлить процесс парсинга. Важно стремиться к их оптимизации.

3.3. Использование Beautiful Soup для анализа и извлечения данных

После преобразования HTML-контента выполняется следующие этапы парсинга, которые включают анализ структуры HTML и выделение необходимых данных.

Как было упомянуто выше, этапы изучения структуры и извлечения нужных элементов HTML-документа используют одни и те же методы библиотеки Beautiful Soup, хотя и для разных целей.

Библиотека Beautiful Soup содержит большое количество простых и удобных функций для поиска и обработки HTML-документов и предоставляет их для навигации по дереву элементов, поиска и извлечения требуемых данных. В контексте интернет-маркетинга и веб-программирования Beautiful Soup незаменима для сбора данных с веб-сайтов и автоматизации задач, связанных с обработкой HTML.

Библиотека Beautiful Soup предоставляет методы обработки HTML-контента веб-сайта, которые можно условно разделить на две группы:

– использующие теги HTML;

– использующие селекторы CSS.

3.3.1. Использование тегов HTML

Beautiful Soup предлагает удобные методы для навигации по дереву и извлечения данных по их тегам. Метод **find()** находит первый элемент, соответствующий заданным критериям:

```
# Находим первый абзац
paragraph = soup.find('p').text
print('Первый абзац:', paragraph)
```

Метод **find_all()** находит все элементы, соответствующие заданным критериям. Возвращает список:

```
# Находим все элементы списка
list_items = soup.find_all('li')
for item in list_items:
   print('Элемент списка:', item.text)
```

В зависимости от параметров, передаваемых этим двум методам, они будут искать элементы на странице по-разному. В частности, можно выбрать следующие элементы HTML:

По тегу:

```
# найти все <h1> элементы на странице
h1_elements = soup.find_all('h1')
```

По id:

```
# найти элемент с id="main-title"
main_title_element = soup.find(id='main-title')
```

По тексту:

```
# найти  footer элемент
# based on the text it contains
footer_element = soup.find(text={'Powered by Word-
Press'})
```

По атрибуту:

```
# найдите элемент ввода электронной почты через
его атрибут "name"
email_element = soup.find(attrs={'name': 'email'})
```

По классу:

```
# найдите все центрированные элементы на странице
centered_element   =   soup.find_all(class_='text-
center')
```

Объединение этих методов позволяет извлекать данные из любого HTML-элемента на странице:

```
# извлечь все  "li" элементы в элементе "navbar"
soup.find(class_='navbar').find_all('li')
```

Примеры применения основных функций и методов

Найти первый абзац:

```
first_paragraph = soup.find('p')
```

Найти все абзацы:

```
all_paragraphs = soup.find_all('p')
```

Один элемент с классом 'content':

```
content = soup.find(class_='content')
```

Все элементы с классом 'content':

```
all_content = soup.find_all(class_='content')
```

Поиск ссылки по href:

```
link = soup.find('a', href='https://example.com')
```

Поиск по нескольким атрибутам:

```
element = soup.find('div', attrs={'id': 'main', 'class': 'content'})
```

Найти все абзацы с определенным классом:

```
paragraphs = soup.find_all('p', class_='text-block')
```

Найти все ссылки внутри div с определенным id:

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

Использование атрибутов для фильтрации элементов

Можно найти элементы с использованием атрибутов, таких как id и class:

```
# Находим первый элемент с определенным id
title = soup.find(id="title")
print(title.text)  # Вывод: Заголовок
# Находим все элементы с классом "content"
content_paragraphs = soup.find_all(class_="content")
```

```
    for p in content_paragraphs:
        print(p.text)  # Вывод: Это текст статьи...
```

Важно! Используется class_ вместо class – это чтобы избежать конфликта с зарезервированным словом Python.

Иногда элементы имеют одинаковые теги, но могут различаться по атрибутам, например class или id:

```
    element = soup.find('tag_name',
class_='class_name')
    elements = soup.find_all('tag_name',
id='id_value')
```

Здесь также используется class_, поскольку class это зарезервированное ключевое слово в Python.

Пример:

Нужно извлечь ссылки и тексты заголовков из повторяющегося HTML-массива статей:

```
    html_doc = """<div class="articles">
                    <div class="article">
                        <h2  class="title">Первая  ста-
тья</h2>
                        <a
href="https://example.com/1"   class="read-more">Читать
далее</a>
                    </div>
                    <div class="article">
                        <h2 class="title">Вторая ста-
тья</h2>
                        <a
href="https://example.com/2" class="read-more">Читать
далее</a>
                    </div>
                  </div>"""
    soup = BeautifulSoup(html_doc, 'html.parser')
    articles = soup.find_all('div', class_='article')
    for article in articles:
        title = article.find('h2',
class_='title').text
        link = article.find('a', class_='read-
more')['href']
        print(f"Заголовок: {title}, Ссылка: {link}")
```

Результат:

```
    # Заголовок: Первая статья, Ссылка:
https://example.com/1
    # Заголовок: Вторая статья, Ссылка:
https://example.com/2
```

3.3.2. Использование текстовых фильтров

Фильтрация текста в Beautiful Soup может быть выполнена с использованием различных методов, таких как параметр string и регулярные выражения.

Параметр string методов find() и find_all() позволяет фильтровать элементы на основе их текстового содержимого:

```
soup.find_all(string="Текст для поиска")
```

Этот параметр особенно полезен, когда нужно найти элементы, содержащие точный текст или фрагмент текста.

Пример использования в парсинге Википедии

Предположим, нужно найти все ссылки, содержащие слово "PyPI":

```
from bs4 import BeautifulSoup
import requests

url = 'https://ru.wikipedia.org/wiki/Python'
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

links = soup.find_all('a', string='PyPI')

print(links)
```

Результат:

```
    [<a href="/wiki/PyPI" title="PyPI">PyPI</a>, <a
href="/wiki/PyPI" title="PyPI">PyPI</a>, <a
href="/wiki/PyPI" title="PyPI">PyPI</a>, <a
href="/wiki/PyPI" title="PyPI">PyPI</a>]
```

3.3.3. Использование CSS-селекторов

CSS отвечает за стилизацию веб-страниц, однако для целей веб-парсинга можно рассматривать CSS как инструмент для понимания

структуры и выбора элементов. CSS-селекторы могут быть использованы для поиска элементов, которые имеют общие стили, что делает парсинг более гибким и точным. CSS-селекторы позволяют обращаться к элементам HTML и помогают определить, какие элементы на странице нужно обработать.

Ключевые концепции CSS, которые важны для парсинга:

– **селекторы** – это правила, которые указывают на конкретные HTML-элементы. Их использование помогает точно идентифицировать нужные данные;

– **атрибуты class, id, и name** являются уникальными идентификаторами, помогающими выделять и различать элементы. Для парсинга они особенно полезны, так как помогают изолировать нужные элементы, что упрощает извлечение данных.

CSS-селекторы

Тег: p – выбирает все <p> элементы (абзацы).

Класс: .classname – выбирает все элементы с заданным классом.

ID: #idname – выбирает элемент с определенным ID. Символ (#) используется в CSS для выбора тегов по их id-атрибутам.

Комбинации: div > p – выбирает все <p>, которые являются непосредственными дочерними элементами .

Примеры выборки элементов на веб-странице:

– div – выбрать все элементы;

– #container – выберите элемент с идентификатором «container»;

– .item – выберите элементы с классом «item»;

– a[href^="http"] – выберите теги привязки с href, начинающимся с http://.

Методы select() и select_one()

Метод select() возвращает список всех элементов, которые соответствуют указанному селектору. А select_one() найдет самый первый элемент, который соответствует селектору, как поисковик, который возвращает именно то, что нужно, а не километровый список бессмысленных ресурсов.

Beautiful Soup позволяет использовать CSS-селекторы через метод select. Например:

```
# Выборка всех ссылок
links = soup.select('a')
for link in links:
    print('Ссылка:', link['href'])
```

Можно также комбинировать селекторы для более точной выборки. Например, soup.select('div.article h2') выберет все h2 внутри div с классом article.

Примеры извлечения с CSS-селекторами

Рассмотрим несколько конкретных примеров использования CSS-селекторов для сбора данных.

Получение элементов по имени тега

Выделите все гиперссылки на странице:

```
links = soup.select('a')
```

Это позволит сопоставить любые элементы с тегом привязки <a>.

Выборка элемента по идентификатору

Получите форму на странице с определенным идентификатором:

```
login_form = soup.select_one('#loginForm')
```

Префикс # указывает на соответствие атрибуту id. Это позволит выбрать элемент с id="loginForm".

Идентификаторы должны быть уникальными в пределах страницы, поэтому всегда будет возвращаться только один элемент.

Получение элементов по имени класса

Выберите все элементы страницы с определенным классом:

```
products = soup.select('.product-item')
```

Префикс (.) обозначает селектор классов. Это позволит выбрать все элементы с class="product-item".

Обратите внимание, что классы можно использовать повторно, поэтому это может соответствовать нескольким элементам.

Выборка по значению атрибута

Извлечение полей ввода на основе их атрибута типа:

```
text_inputs = soup.select('input[type="text"]')
```

Синтаксис [attribute="value"] позволяет выполнять поиск по конкретным значениям атрибутов.

Выборка и вывод результата

Выберите все абзацы с классом highlight и выведите их текст:

```
# Извлечение всех абзацев с классом 'highlight'
```

```
    highlighted_paragraphs =
soup.select('p.highlight')
    for para in highlighted_paragraphs:
        print('Выделенный абзац:', para.text)
```

3.4. Создание парсера

Процесс создания своего парсера может показаться достаточно сложным и запутанным, особенно, если нужно писать программу с чистого листа. Ситуацию меняет наличие готовых профильных библиотек, таких как Beautiful Soup. Это библиотека для Python, предназначенная для ускорения написания собственных утилит парсинга веб-страниц. Она преобразует HTML-код в аналог индексированного словаря и предоставляет простые методы для навигации, поиска и извлечения данных на основе использования DOM-дерева.

Пример парсера

Создайте файл sample.html в своей папке с этим текстом HTML-документа:

```
    <!DOCTYPE html>
    <html>
     <head>
        <title>Это заголовок страницы, который отобра-
жается в результатах поисковой выдачи, а также при
наведении курсора на вкладку браузера</title>
           <meta charset="utf-8">
      </head>
      <body>
         <h1>А это заголовок, который показывается в
теле страницы</h1>
           <p style = "background-color: yellow; font-
size: 40px;">
        Это просто абзац, внутри может быть какой-то
текст с описанием чего-то полезного…
           </p>
          <h2>Подзаголовок, пусть будет название списка
«Типы прокси»:</h2>
            <ul id="thebestlist">
              <li>Резидентные прокси</li>
              <li>Мобильные прокси</li>
```

```
            <li>Общие прокси</li>
            <li>Датацентровые прокси</li>
            <li>Приватные прокси</li>
          </ul>
      </body>
    </html>
```

Запустите Python-парсер.

Пример:

```
    # Как парсить с Beautiful Soup: гайд для начинающих
    # https://blog.froxy.com/ru/scrape-with-beautiful-
soup
    # ЛР 8. Парсинг сайтов на Python
    # 3. Выполнение HTTP-запросов
    from bs4 import BeautifulSoup
    # D:\Петренко\МЕТОДИЧКИ\Веб-парсинг на Python. УП\
sample.html
    with open('D:\Петренко\МЕТОДИЧКИ\Веб-парсинг на
Python. УП\sample.html', 'r') as file:
        content = file.read()
        soup = BeautifulSoup(content, "html.parser",
from_encoding='windows-1251')
    for child in soup.descendants:
        if child.name:
            print(child.name)
```

Результат:

```
    html
    head
    title
    meta
    body
    h1
    p
    h2
    ul
    li
    li
    li
    li
    li
```

Найдем в файле sample.html список (тег ul) с идентификатором «thebestlist», а затем выведем все элементы списка (они имеют тег li).

При чтении указываем параметр «encoding='cp1251'» в операторе open для нормального отображения символов кириллицы.

Кодировку «cp1251» выбрали намеренно для Windows.

Пример:

```
from bs4 import BeautifulSoup
# Открываем HTML-файл и передаем содержимое в переменную htmlcontent
with open('D:\Петренко\МЕТОДИЧКИ\Веб-парсинг на Python. УП\sample.html', 'r', encoding='cp1251') as file:
        htmlcontent = file.read()
        # print(htmlcontent)
        soup = BeautifulSoup(htmlcontent, "html.parser")
# Ищем тег ul с атрибутом id=thebestlist. Будет выведен только первый список, но так как список только один, он будет выведен весь, вместе с дочерними элементами
print(soup.find('ul', attrs={'id': 'thebestlist'}))
# В цикле перебираем все найденные теги li (то есть элементы списка)
for tag in soup.find_all('li'):
    print(tag.text)
```

Результат:

```
<ul id="thebestlist">
<li>Резидентные прокси</li>
<li>Мобильные прокси</li>
<li>Общие прокси</li>
<li>Датацентровые прокси</li>
<li>Приватные прокси</li>
</ul>
Резидентные прокси
Мобильные прокси
Общие прокси
Датацентровые прокси
Приватные прокси
```

Задание 1. Создание парсера

1. Создать файл HTML, содержащий список из нескольких студентов и значением суммы баллов по двум контрольным точкам ($R^{тек}$), в виде:

Дисциплина: <название дисциплины>
Номер группы: <номер группы>
ФИО; Сумма баллов: <список из нескольких студентов>

2. Провести парсинг этого файла и сформировать экзаменационную ведомость в виде:

ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ

Дисциплина _____
Группа _____
Дата _____
№ п/п Фамилия, инициалы студента $R^{тек}$
1…
2…
…

3.5. Парсинг одной веб-страницы

Прежде чем приступить к написанию кода, необходимо изучить веб-страницу, которую нужно парсить, и полученный HTML-код, чтобы определить наилучшую стратегию парсинга. Для этого удобно воспользоваться инструментами разработчика Dev Tools, которые есть в браузере.

Пример парсинга веб-страницы

Для примера будем использовать сайт Quotes to Scrape, который содержит постраничный список цитат. Надо извлечь из него данные: текст цитаты и автора. Созданный парсер Python извлечет все цитаты и автора на каждой странице, и предоставит их в виде данных CSV.

Сначала необходимо определить интересующие HTML-элементы. В частности, важно определить стратегию выбора тех, которые содержат требующие извлечения данные.

Для этого можно воспользоваться предлагаемыми браузером инструментами для разработчиков. В браузере Chrome щелкните правой кнопкой мыши на интересующем HTML-элементе и выберите опцию «Просмотреть код». В данном случае это нужно сделать на элементе цитаты (class=”text”) (рис. 3.6).

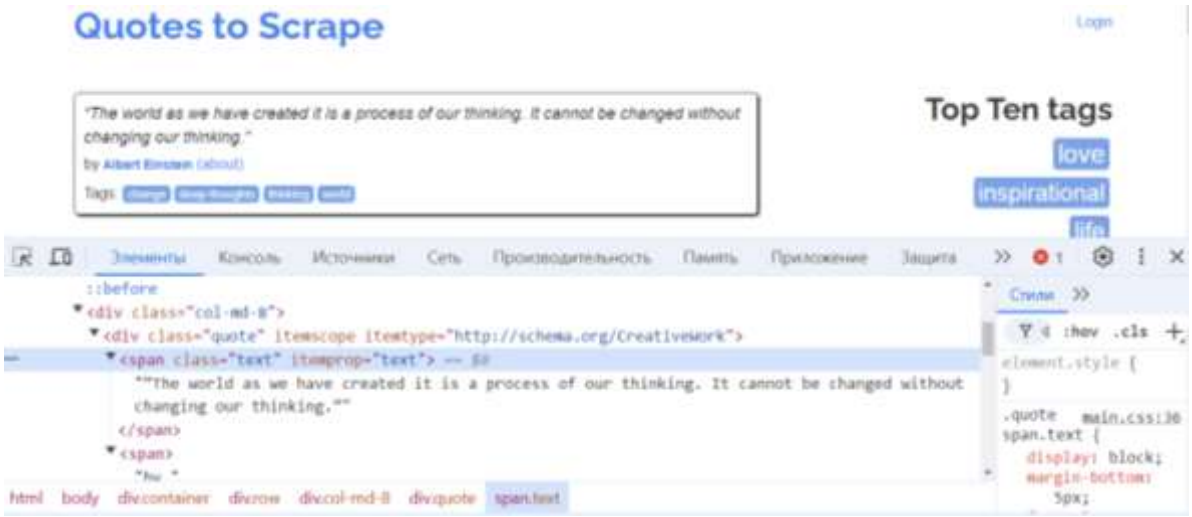


Рис. 3.6. Сайт Quotes to Scrape

Процесс парсинга будет включать следующие шаги:

Шаг 1. Подключение к целевому URL. Первое, что необходимо сделать в веб-парсере – подключиться к целевому сайту. Для этого необходимо получить полный URL-адрес страницы из браузера:

```
page = requests.get('https://quotes.toscrape.com'
```

Шаг 2. Парсинг содержимого HTML-документа. Чтобы выполнить парсинг HTML-документа, возвращенного сервером после GET-запроса, передайте конструктору Beautiful Soup() файл page.text:

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

Второй параметр задает парсер, который будет использовать Beautiful Soup.

Шаг 3. Выделение HTML-элементов с помощью Beautiful Soup. Можно воспользоваться предлагаемыми браузером инструментами для разработчиков. В браузере Chrome щелкните правой кнопкой мыши на интересующем вас HTML-элементе и выберите опцию «Просмотреть код».

Шаг 4. Извлечение данных из элементов. С помощью Soup извлекаем из DOM элементы цитаты, применив определенный ранее CSS-селектор .quote:

```
    quote_elements = soup.find_all('div',
class_='quote')
```

Метод find_all() вернет список всех HTML-элементов:

```
    for quote_element in quote_elements: # extract the
text of the quote text = quote_element.find('span',
class_='text').text
```

Полный код:

```
    import requests
    from bs4 import BeautifulSoup
    page = requests.get('https://quotes.toscrape.com')
    soup = BeautifulSoup(page.text, 'html.parser')
    quote_elements = soup.find_all('div',
class_='quote')
    for quote_element in quote_elements:
        # extract the text of the quote
        text = quote_element.find('span',
class_='text').text
        print(text)
        # extract the author of the quote
        author = quote_element.find('small',
class_='author').text
        print(author)
```

Результат:

```
    “The world as we have created it is a process of
our thinking. It cannot be changed without changing
our thinking.”
    Albert Einstein
    “It is our choices, Harry, that show what we truly
are, far more than our abilities.”
    J.K. Rowling
    “There are only two ways to live your life. One is
as though nothing is a miracle. The other is as though
everything is a miracle.”
    Albert Einstein
    “The person, be it gentleman or lady, who has not
pleasure in a good novel, must be intolerably stupid.”
    Jane Austen
```

```
    “Imperfection is beauty, madness is genius and
it's better to be absolutely ridiculous than absolute-
ly boring.”
    Marilyn Monroe
    “Try not to become a man of success. Rather become
a man of value.”
    Albert Einstein
    “It is better to be hated for what you are than to
be loved for what you are not.”
    André Gide
    “I have not failed. I've just found 10,000 ways
that won't work.”
    Thomas A. Edison
    “A woman is like a tea bag; you never know how
strong it is until it's in hot water.”
    Eleanor Roosevelt
    “A day without sunshine is like, you know, night.”
    Steve Martin
    In [ ]:
```

Задание 2. Обработка веб-страницы

На сайте ПГУ найти и извлечь всех студентов своей группы. Для изучения структуры сайта ввести запрос «группа <номер группы заглавными буквами> ПГУ» в браузере и определить, как выделить элемент с фамилией. При необходимости и для упрощения записи условий для поиска используйте регулярные выражения.

3.6. Обработка многостраничного контента

Веб-сайт обычно состоит из нескольких страниц, объединенных гиперссылками. Обычно на сайте есть страница-источник, которая содержит ссылки на остальные страницы. Таких страниц может быть несколько сотен и перебирать их вручную в поисках нужной информации – задача сложная и трудоемкая. Поэтому для автоматизации задачи используется процесс парсинга сайтов.

Процесс сбора данных с сайта содержит следующие этапы:

1. Изучить целевой сайт. Нужно понять, содержит ли он динамический контент (например, загружается через JavaScript). Если

нет, подойдут библиотеки Beautiful Soup и Requests. Если контент динамический – потребуется другой подход (например, использование capmonster.cloud).

2. Выяснить. способ пагинации (переход по страницам) и структуру кода сайта. Пагинация – распределение информации постранично, как в поисковиках Google и Яндекс. Каждая страница пронумерована и на ней всегда фиксированное количество информации.

3. Написать логику парсинга. Она может включать:

– цикл для перебора страниц. Например, если на сайте есть ссылка «Следующая страница», нужно следовать по ней и извлекать контент итеративно, пока не будет достигнута последняя страница;

– извлечение данных с каждой страницы. Можно использовать методы Beautiful Soup для поиска элементов на странице и извлечения текста или атрибутов.

4. Сохранить полученные данные. Например, можно сохранить информацию в нужном формате (например, в CSV).

Важно помнить, что парсинг может нарушать правила сайта, поэтому перед написанием кода рекомендуется изучить его политику.

Одна из самых сложных вещей в парсинге – получить список адресов всех нужных страниц. Для этого можно использовать:

– карту сайта;

– внутренние рубрикаторы;

– разделы на сайте;

– готовые страницы со всеми ссылками.

Пример кода

Пример простого парсера, который собирает заголовки статей с главной страницы блога, с использованием библиотек Requests и Beautiful Soup представлен ниже.

Получение HTML-кода страницы с помощью Requests:

```
import requests
url = 'https://example.com/blog/'
response = requests.get(url)
```

Парсинг HTML с помощью Beautiful Soup:

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(response.text, 'html.parser')
```

Поиск данных с помощью CSS-селекторов:

```
titles = soup.select('h2.blog-title') # Находим
заголовки статей в тегах <h2> с классом blog-title
```

Извлечение данных – получение текста заголовков с помощью метода get_text():

```
for title in titles:
    print(title.get_text()) # Выводим текст заго-
ловков
```

Сохранение данных в текстовый файл:

```
with open('titles.txt', 'w') as f:
for title in titles:
    f.write(title.get_text() + '\n') # Сохраняем
заголовки в файл
 mate.academy
```

Ошибки и их решение

При парсинге многостраничного сайта на Python возможны ошибки, например:

– неправильный выбор селекторов**.** Нужно проверить структуру сайта и убедиться, что нужный элемент существует в HTML-коде;

– ошибки соединения с сайтом**.** Рекомендуется проверять статус запроса (200 – запрос выполнен успешно, 404 – страница не найдена и т.д.);

– проблемы с кодировкой. Нужно проверить кодировку HTML-кода и при необходимости скорректировать код.

Также важно учитывать, что не все сайты разрешают парсинг, и перед разработкой парсера нужно проверить политику сайта.

Рассмотрим варианты проведения парсинга многостраничных сайтов.

Пример 1. Перебор номеров страниц

На большинстве веб-сайтов страницы пронумерованы от 1 до N. Это позволяет легко просматривать эти страницы и извлекать из них данные, поскольку эти страницы имеют схожую структуру. Например (рис. 3.8, 3.9).

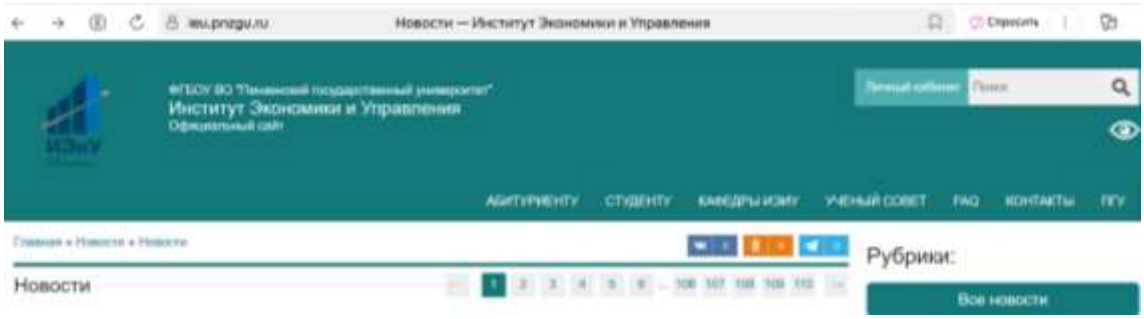


Рис. 3.7. Номера страниц в верхней части
сайта новостей ИЭиУ ПГУ



Рис. 3.8. Адрес веб-страницы

В конце URL-адреса указан номер страницы /p/1/. Используя эту информацию, можно создать цикл for, который будет перебирать столько страниц, сколько необходимо (поместив *page/(i)/* в строку URL-адреса и перебирая *i до N*), и извлечь из них все полезные данные. Есть базовый URL-адрес сайта (‘https://ieu.pnzgu.ru/news/p/’), к которому добавляем номер веб-страницы.

Для выделения заголовка статьи можно использовать значение ‘b’ селектора стиля, так как оно выделяет текст заголовка статьи (рис. 3.9).

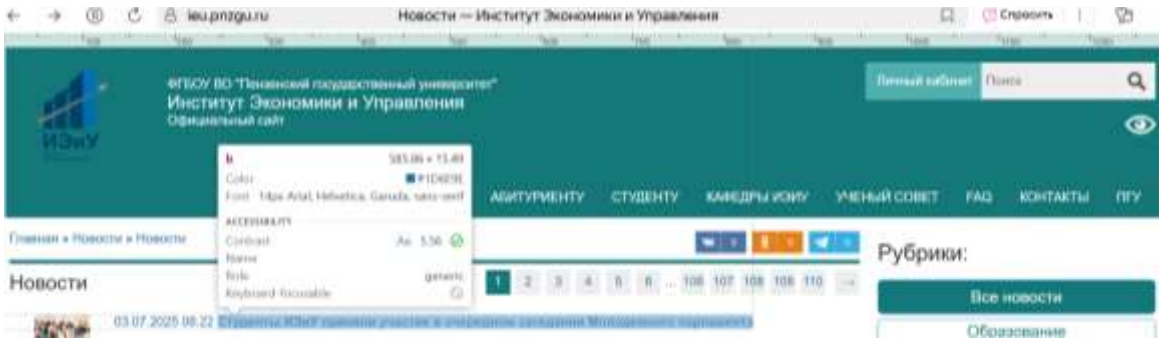


Рис. 3.9. Окно Краткая справка по стилям

Пример:

Чтобы не перебирать все страницы, обработаем только первые три страницы и из каждой выберем два первых заголовка новостей:

```
import requests
from bs4 import BeautifulSoup as bs

for i in range(1, 4):
    URL = "https://ieu.pnzgu.ru/news/p/" + str(i)
    print('
Страница ', i, '\n')
    req = requests.get(URL)
    soup = bs(req.text, 'html.parser')
    # Текст заголовка новости находим по тегу <b>
    # Тег <b> служит для присвоения полужирного
стиля тексту после него
    # На сайте «Новости ИЭиУ» так выделены заго-
ловки новостей
    news = soup.find_all('b')
    print(news[1].text, '\n')
    print(news[2].text, '\n')
```

Результат:

```
Страница  1

Студенты ИЭиУ приняли участие в очередном заседа-
нии Молодежного парламента

Рабочая встреча с управляющим отделением Пенза
Волго-Вятского ГУ банка России

Страница  2

Обучающимся начального курса "Таможенное дело"
вручены сертификаты

Студенты ИЭиУ - участники Всероссийской научно-
методической конференции «Инновации в науке, образова-
нии и бизнесе»
```

Пример 2. Цикл по списку различных URL-адресов

Приведенный выше метод хорошо работает. Но если нужно собрать данные с разных страниц с неизвестными заранее адресов? В этом случае придется собирать данные с разных URL-адресов по одному, вручную подставляя URL-адрес очередной страницы в код для парсинга.

Вместо этого можно просто составить список этих URL-адресов и пройтись по ним. Затем перебирая URL-адреса в списке, можно получить заголовки этих страниц без необходимости писать код для каждой страницы.

Пример:

```
page = requests.get("http://required-
site.ru/page") # Передаем GET-запрос, результат пишем
в page
variable = BeautifulSoup(page.text, "html.parser") #
Ключевая строка, скачивающая весь html-код
for result in variable.find_all("a"): # перебираем
циклом все теги <a>
print(result.get("href")) # извлекаем и распечатываем
атрибуты href=»»
```

Пример 3. Использование списка страгниц

Предположим, что существует список страниц, которые нужно посетить и выбрать оттуда заголовки.

Объявляем массив со статьями:

```
url = [
«https://thecode.media/is-not-defined-jquery/»,
«https://thecode.media/arduino-projects-2/»,
«https://thecode.media/10-raspberry/»,
«https://thecode.media/easy-css/»,
«https://thecode.media/to-be-front/»,
«https://thecode.media/cryptex/»,
«https://thecode.media/ali-coders/»,
«https://thecode.media/po-glandy/»,
]
```

Перебираем все элементы этого массива в цикле:

```
import requests
from bs4 import BeautifulSoup
url = [
"https://thecode.media/is-not-defined-jquery/",
"https://thecode.media/arduino-projects-2/",
"https://thecode.media/10-raspberry/",
"https://thecode.media/easy-css/",
"https://thecode.media/to-be-front/",
"https://thecode.media/cryptex/",
"https://thecode.media/coffee/",
]
# перебираем все адреса из списка
for x in url:
    # получаем исходный код очередной страницы из
списка
    # html_code = str(urlopen(x).read(),'utf-8')
    req = requests.get(x)
    # отправляем исходный код страницы на обработ-
ку в библиотеку
    soup = BeautifulSoup(req.text, "html.parser")
    # находим название страницы с помощью метода
find()
    s = soup.find('title').text
    # выводим его на экран
    print(s)
```

Результат:

```
$ is not defined в jQuery: что это значит и что делать – Журнал «Код» программирование без снобизма
10 интересных проектов, которые можно сделать на Arduino
10 проектов на Raspberry Pi: что можно на нем сделать – Журнал «Код»
10 полезных инструментов для создания роскошного CSS – Журнал «Код» программирование без снобизма
Сколько зарабатывает фронтенд и как им стать – Журнал «Код»
5 классных головоломок с Алиэкспресса – Журнал «Код» программирование без снобизма
Сколько стоит капучино? – Журнал «Код» программирование без снобизма
```

Задание 3. Обработка веб-сайта

1. Используя цикл for, выбрать все заголовки новостей с первых двух страниц сайта новостей ИЭиУ ПГУ.
2. Загрузить все изображения с пятой страницы этого сайта.

4. ПАРСИНГ ТАБЛИЦ ИЗ САЙТА НА PYTHON

HTML-таблицы обычно используются для организации и представления информации в понятном и доступном формате, что упрощает пользователям интерпретацию и сравнение информации. Некоторые популярные применения табличных данных включают финансовые данные, статистическую информацию, списки продуктов электронной коммерции и отчеты об академических исследованиях.

4.1. Преимущества сбора табличных данных

Структурированные и реляционные данные

Таблицы представляют данные в структурированном формате со строками, столбцами и общими полями для каждой записи. Это делает парсинг таблиц идеальным способом получения чистых, согласованных наборов данных, готовых к импорту в базы данных и хранилища данных.

Формат строк и столбцов также упрощает анализ и извлечение реляционных данных, в которых различные атрибуты связаны друг с другом и имеет преимущества.

Табличные данные чище, чем неструктурированный текст. В отличие от парсинга длинных текстов или отзывов, при извлечении структурированных полей из HTML-таблицы потребность в сложном NLP-анализе снижается. Данные уже структурированы. Конечно, все равно придется решать такие проблемы, как перекрывающиеся строки и столбцы, отсутствующие значения, повторяющиеся записи и т.д. Но в целом парсинг таблиц будет проще, чем парсинг текста произвольной формы со страниц.

Данные в таблицах представлены в упорядоченном виде, обычно ранжированы, подготовлены для сравнения и индексированы.

Авторы часто представляют сводные данные, рейтинги и индексы в табличном формате, чтобы улучшить читабельность. Примерами могут служить финансовые индексы, рейтинги учебных заведений, таблицы ставок и т.д. Такие табличные данные идеально подходят для парсинга.

Таблица HTML структурирована с использованием тега <table>, который служит контейнером для содержимого таблицы. В этой табли-

це строки определяются с использованием тега <tr> (представляющего «строку таблицы»), а в каждой строке ячейки создаются с использованием тега <td> (обозначает «стандартные данные таблицы») или тега <th> (обозначает «заголовки таблицы»).

Пример:

```
<table>
  <thead>
    <tr>
      <th>Страна</th>
      <th>Население</th>
      <th>Дата</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Россия</td>
      <td>146 171 015</td>
      <td>1 января 2021</td>
    </tr>
    <tr>
      <td>Румыния</td>
      <td>19 257 520</td>
      <td>1 января 2020</td>
    </tr>
  </tbody>
</table>
```

4.2. Извлечение таблиц с веб-сайтов с помошью Python

Для простого извлечения данных из таблиц, отображаемых на веб-сайте, вы можете использовать библиотеки Beautiful Soup и Pandas в Python. Beautiful Soup помогает извлекать данные из HTML-документа, а Pandas помогает анализировать и обрабатывать извлеченные данные.

Извлечение простых таблиц

Извлечем данные HTML-таблицы из страницы Википедии (рис. 4.1)



Рис. 4.1. Пример таблицы из Википедии

Таблица содержит три столбца и одиннадцать строк. Первая строка таблицы – это строка заголовка с тегами столбцов заголовка таблицы (<th>). Остальные строки содержат простые столбцы с данными таблицы (<td>). Задача состоит в том, чтобы извлечь эту таблицу с исходной страницы, очистить и сохранить ее данные.

Шаг 1. Импортируйте библиотеки. Создайте новый файл Python и начните с импорта следующих библиотек:

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

Шаг 2. Сделайте запрос. После импорта необходимых модулей отправьте HTTP-запрос на целевую веб-страницу, чтобы получить ее HTML-контент:

```
        # Step 2: Send a request to the Wikipedia page.
        url =
"https://en.wikipedia.org/wiki/List_of_highest-
grossing_films"
        response = requests.get(url)
```

Шаг 3. Очистите данные HTML-таблицы и проанализируйте их. Полученную переменную response проанализируйте с помощью библиотеки Beautiful Soup и извлеките таблицу с помощью соответствующего селектора CSS. Затем преобразуйте ее в DataFrame, как показано в последнем операторе следующего фрагмента:

```
    # Шаг 3.1: Парсинг HTML-содержимое с помощью Beau-
tifulSoup
```

```
    soup = BeautifulSoup(response.content,
"html.parser")

    # Шаг 3.2: Поиск конкретной таблицы (в данном слу-
чае это вторая таблица на этой странице)
    table = soup.find_all("table", {"class": "wikita-
ble"})[1]

    # Шаг 3.3: Чтение таблицы в DataFrame и вывод ее с
использованием pandas
    df = pd.read_html(str(table))[0]
    print(df.to_string(index=False))
```

Функция str() преобразует HTML-таблицу в текстовый поток, который затем преобразуется в текстовый поток в памяти с помощью StringIO. Полученный текстовый поток в памяти обрабатывается методом read_html() как объект файла, который преобразует его в DataFrame.

Шаг 4. Сохраните результаты в CSV-файл. Теперь можно просто отобразить его на экране, чтобы просмотреть извлеченные данные напрямую.

Окончательный вариант:

```
    # Шаг 1: Импорт модулей
    import requests
    import pandas as pd
    from bs4 import BeautifulSoup

    # Шаг 2: Отправка запроса на страницу Википедии
    url =
"https://en.wikipedia.org/wiki/List_of_highest-
grossing_films"
    response = requests.get(url)

    # Шаг 3.1: Парсинг HTML-содержимое с помощью Beau-
tifulSoup
    soup = BeautifulSoup(response.content,
"html.parser")

    # Шаг 3.2: Поиск конкретной таблицы (в данном слу-
чае это вторая таблица на этой странице)
    table = soup.find_all("table", {"class": "wikita-
ble"})[1]

    # Шаг 3.3: Чтение таблицы в DataFrame и вывод его
использованием pandas
```

```
df = pd.read_html(str(table))[0]
print(df.to_string(index=False))
```

Результат:

Rank	Title	Worldwide gross(2024 $)	Year
1	Gone with the Wind	GW$4,450,000,000	1939
2	Avatar	A1$4,056,000,000	2009
3	Titanic	T$3,769,000,000	1997
4	Star Wars	$3,652,000,000	1977
5	Avengers: Endgame	AE$3,357,000,000	2019
6	The Sound of Music	$3,059,000,000	1965
7	E.T. the Extra-Terrestrial	ET$2,990,000,000	1982
8	The Ten Commandments	$2,827,000,000	1956
9	Doctor Zhivago	$2,680,000,000	1965
10	Star Wars: The Force Awakens	TFA$2,642,000,000	2015

Задание 4. Извлечение табличных данных

1. Создать HTML-таблицу, содержащую данные с оценками по нескольким студентам и записать ее в файл:

№ п/п	Группа	ФИО	Математика	Экономика
1				
2				
...				

2. Разработать парсер на языке Python для считывания и вывода этих данных.

3. Из сайта «Табло Пенза-1» (https://penza-zhd.ru/raspisanie/online-tablo) вывести таблицу прибытия поездов на сегодня.

СПИСОК ЛИТЕРАТУРЫ

1. Основы HTML. Как создать веб-сайт. URL: https://victor-komlev.ru/osnovy-html-kak-sozdat-veb-sajt/

2. Что такое извлечение данных? Подробное руководство по веб-скрапингу на Python. URL: https://webscrapingsite.com/guide/web-data-extraction/

3. Извлечение данных с веб-сайтов в Excel: пошаговое руководство. URL: https://parsingmaster.com/blog-post/extracting-data-from-websites-in-excel-guide/

4. Начало начал и основа основ: секреты парсинга. URL: https://www.seonews.ru/analytics/nachalo-nachal-i-osnova-osnov-sekrety-parsinga/?utm_source=yandex.ru&utm_medium=organic&utm_campaign=yandex.ru&utm_referrer=yandex.ru

5. В чем различие парсинга и скрепинга. URL: https://allrival.com/post/articles/v-chem-razlichie-parsinga-i-skrepinga

6. Классификация программ и инструментов для парсинга. URL: https://idatica.com/blog/parser-obzor-i-klassifikaci/

7. Span и Div. URL: https://msiter.ru/tutorials/html-srednego-urovnya/span-i-div

8. HTML-атрибуты. URL: https://proglang.su/html/attributes

9. Знакомство с CSS. URL: https://javarush.com/quests/lectures/ru.javarush.python.core.lecture.level22.lecture05

10. Изучение HTML и CSS с нуля : пошаговое руководство. URL: https://sky.pro/wiki/html/izuchenie-html-i-css-s-nulya-poshagovoe-rukovodstvo/

11. CSS-селекторы, свойства, значения. URL: https://msiter.ru/tutorials/css-nachalnogo-urovnya/selektory-svoistva-znacheniya

12. CSS-селекторы. URL: https://www.geeksforgeeks.org/css/css-selectors/

13. Основные селекторы в CSS. URL: https://javarush.com/quests/lectures/ru.javarush.python.core.lecture.level22.lecture06

14. Применение эффективного асинхронного web-парсинга при работе с Big Data (библиотека Scrapy). URL: https://habr.com/ru/companies/sberbank/articles/748406/

15. Скрапинг веб-сайтов на Python : пошаговое руководство. URL: https://ru-brightdata.com/blog/how-tos-ru/web-scraping-with-python

16. Requests в Python. Примеры выполнения HTTP запросов. URL: https://python-scripts.com/requests

17. Самоучитель по Python для начинающих. Ч. 16: Регулярные выражения. URL: https://proglib.io/p/samouchitel-po-python-dlya-nachinayushchih-chast-16-regulyarnye-vyrazheniya-2023-02-20

18. Как извлекать данные из таблиц с помощью Beautiful Soup? URL: https://webscrapingsite.com/guide/how-to-scrape-tables-with-beautifulsoup/

19. Освоение веб-скрапинга и автоматизации с помощью Python. URL: https://blog.enterprisedna.co/mastering-web-scraping-and-automation-with-python/

20. Как устроен парсинг сайтов. URL: https://skillbox.ru/media/code/kak-parsit-sayty-i-materialy-smi-s-pomoshchyu-javascript-i-nodejs/?utm_source=media&utm_medium=link&utm_campaign=all_all_media_links_links_articles_all_all_skillbox

21. Как парсить новости с помощью Python и ИИ. URL: https://ru-brightdata.com/blog/web-data-ru/how-to-scrape-news-articles

22. Как парсить несколько веб-страниц с помощью Python. URL: https://www.freecodecamp.org/news/how-to-scrape-multiple-web-pages-using-python/

23. Парсинг на Python с Beautiful Soup. URL: https://pythonru.com/biblioteki/parsing-na-python-s-beautiful-soup

24. Учимся писать парсеры на примере парсера новостных сайтов. URL: https://www.diggernaut.ru/blog/uchimsya-pisat-parsery-na-primere-parsera-novostnyh-saytov/

25. Как создать HTML-таблицы с помощью Python: шаг за шагом. URL: https://oxylabs.io/blog/python-scrape-tables

26. Руководство по веб-скрейпингу на Python. URL: https://habr.com/ru/companies/ruvds/articles/796885/

27. Парсинг таблицы с сайта на Python : пошаговое руководство. URL: https://idatica.com/blog/parsing-tablitsy-s-sayta-na-python-poshagovoye-rukovodstvo/

Учебное издание

Петренко Александр Григорьевич,
Катков Сергей Николаевич

ПАРСИНГ ВЕБ-САЙТА НА PYTHON

Редактор *В. В. Чувашова*
Технический редактор *М. А. Борисов*
Компьютерная верстка *М. А. Борисова*
Дизайн обложки *О. И. Жаровой*

Подписано в печать 02.11.2026.
Формат 60×84$^{1}/_{16}$. Усл. печ. л. 3,95
Тираж 43. Заказ № 513.

Издательство ПГУ
440026, г. Пенза, ул. Красная, 40
Тел.: (8412) 66-60-49, 66-67-77; e-mail: iic@pnzgu.ru

Вниманию авторов!

Издательство ПГУ выпускает учебную, научную и художественную литературу, презентационную и акцидентную продукцию, а также полноцветные юбилейные и мемориальные издания в соответствии с ГОСТ 7.60–2003.

Издательство ПГУ принимает к изданию рукописи, подготовленные с использованием текстового редактора Microsoft Word for Windows версий **2003 и выше**. Формат – А4, основной шрифт – Times New Roman, 14–16 pt через одинарный интервал (минимальный размер шрифта в таблицах и сносках – 12,5 pt). Тип файла в электронном виде – doc, docx.

Работа должна содержать индекс УДК, аннотацию.

Аннотация (ГОСТ 7.86–2003, ГОСТ 7.9–1995) включает характеристику основной темы, проблемы объекта, цели работы и ее результаты. В аннотации указывают, что нового несет в себе данный документ в сравнении с другими, родственными по тематике и целевому назначению. Аннотация может включать сведения о достоинствах произведения. Текст аннотации начинают фразой, в которой сформулирована главная тема документа. Заканчивается аннотация читательским адресом.

Рисунки и таблицы должны быть размещены в тексте после ссылки на них (растровые рисунки представляются в виде отдельных файлов в формате jpg, BMP с разрешением 300 dpi, векторные рисунки в формате Corel Draw с минимальной толщиной линии 0,75 pt. Рисунки должны быть доступны для правки!). Они должны сопровождаться подрисуночными подписями, на все рисунки и таблицы в тексте должны быть ссылки.

Формулы в тексте выполняются только в редакторе формул **MathType версий 5.0** и выше. Символы греческого и русского алфавита должны быть набраны прямо, нежирно; латинского – курсивом, нежирно; обозначения векторов и матриц – прямо, жирно; цифры – прямо, нежирно. Наименования химических элементов набираются прямо, нежирно. Эти же требования необходимо соблюдать и в рисунках.

В списке литературы **нумерация источников** должна соответствовать очередности ссылок на них в тексте ([1], [2], …). Номер источника указывается в квадратных скобках. Требования к оформлению списка литературы, русских и иностранных источников (ГОСТ 7.0.5–2008): для книг – фамилия и инициалы автора, название, город, издательство, год издания, том, количество страниц; для журнальных статей, сборников трудов – фамилия и инициалы автора, название статьи, полное название журнала или сборника, серия, год, том, номер, страницы; для материалов конференций – фамилия и инициалы автора, название статьи, название конференции, город, издательство, год, страницы.

К материалам **должна** прилагаться следующая информация: фамилия, имя, отчество, контактные телефоны.

Контакты Издательства ПГУ: (8412) 66-60-49, 66-67-77. E-mail: iic@pnzgu.ru