

Конспект по теме «Извлечение данных из веб-ресурсов»

Что такое Web Mining

Когда предоставленных компанией данных мало для решения задачи, аналитик добывает информацию самостоятельно. Такое обогащение данных позволяет не только учесть больше факторов, но и выявить новые закономерности, прийти к неожиданным выводам. Аналитики обогащают имеющуюся информацию в интернете. Сперва находят ценные для исследования веб-ресурсы, а затем извлекают из них нужные данные. Этот процесс называют **Web Mining**, или **парсинг**.

Что аналитик должен знать об интернете? Браузер. HTML. HTTP.

Интернет — сеть компьютеров, которые обмениваются данными. В интернете действуют концепции, которые утверждают правила *представления информации в интернете (учитывая как компьютер их отображает)* и *обмена информации в интернете*. Для этого был придуман язык, на котором можно создавать документы в интернете — **HTML**, написана программа для просмотра этих документов — браузер и сформулированы единые правила, по которым документы передаются — транспортный протокол **HTTP**.

Что такое транспортный протокол

Интернет — сеть из компьютеров, которые обмениваются информацией. Чтобы это было возможным, нужны правила: в каком виде один компьютер высылает данные другому. Обмен данными в интернете построен на принципе «запрос — ответ»: браузер формирует запрос, сервер его анализирует и отправляет ответ. Правила, по которым нужно формулировать запросы и ответы, определяет транспортный протокол — **HTTP**.

Сегодня большинство сайтов применяют более совершенный протокол передачи информации — **HTTPS**. Это защищённая версия HTTP-протокола. Он гарантирует, что все коммуникации между вашим браузером и веб-сайтом зашифрованы.

Когда вы заходите на сайт, браузер отправляет HTTP-запрос на сервер, а тот в свою очередь формирует ответ: HTML-код нужной страницы. Запрос, который формирует браузер, может включать в себя:

- *HTTP-метод*: он определяет операцию, которую нужно совершить. Есть несколько методов, самые популярные из них: GET — запрос данных с сервера — и POST — отправление данных на сервер.
- *Путь до ресурса*: это часть адреса без имени сайта.
- *Версию HTTP-протокола*, который используется для отправки запроса
- *Заголовки запроса*, в них можно передать серверу дополнительную информацию.
- *Тело запроса* есть не у всех запросов.

Ответ может включать:

- *Версию HTTP-протокола*.
- *Код и сообщение ответа*.
- *Заголовки*, содержащие дополнительную информацию для браузера.
- *Тело ответа*.

Введение в HTML

Чтобы получить необходимую информацию из веб-страниц, нужно заполучить код страницы и контент внутри него. Для этого нужно проанализировать HTML-код.

В HTML каждый объект страницы размечается для корректного представления на сайте. Разметка состоит в том, что блоки информации заключают в управляющие конструкции — *теги*. Такие «бирки» указывают браузеру, как отобразить то, что в них «обёрнуто».

HTML-элемент состоит из *тегов* и размещённого между ними содержания — *контента*. У любого тега **HTML** есть имя и угловые скобки. В начале HTML-элемента ставят *открывающий тег* с именем тега, а в конце — *закрывающий тег*, где имя будто перечеркнуто косой чертой. Созданный элемент называют по имени тега.

Типовая структура страницы HTML выглядит так:

1. `<html> ... </html>`

Тег `<html>` открывает каждый HTML-документ и определяет его начало, а `</html>` означает его конец. Внутри этого тега хранится заголовок `<head>` и тело `<body>` HTML-документа.

2. `<head> ... </head>`

Эта пара тегов обозначает заголовок документа. Внутри заголовка помещаются теги для названия документа `<title>` и дополнительной (мета) информации `<meta>`.

3. `<body> ... </body>`

Тег `<body>` показывает, где начинается тело HTML-страницы. Внутри тела помещают всё наполнение HTML-страницы: заголовки, абзацы текста, таблицы, изображения.

Чтобы в разметке легче было разобраться, в коде веб-страницы разработчики оставляют комментарии внутри специальных тегов `<!-- -->`.

Таблицы в HTML обычно помещаются в элемент-контейнер *table* между тегами `<table>` и `</table>`. Внутри контейнера содержимое таблицы делится на строки тегами `<tr>`, а строки — на ячейки тегами `<td>`. Верхняя, первая строка вместо ячеек обычно содержит заголовки столбцов в тегах `<th>`.

Текст часто помещают в элемент *p*. Абзац текста располагается между открывающим `<p>` и закрывающим `</p>`.

Распространён тег блоков `<div>`. Это обёртка для других элементов.

Контейнер *div* удобен, что может включить в себя любое число разнородных элементов и определить им общие свойства или поведение.

Внутри тегов можно указывать **атрибуты**. Они служат для передачи дополнительных сведений в элемент. Для разных сведений — разные атрибуты. Имя атрибута говорит браузеру, какой признак он определяет, а значение — каким этот признак должен стать.

Чаще всего вам будут нужны атрибуты `id` и `class`. Атрибут `id` — идентификатор с уникальным именем. Значение атрибута `class` — имя, которое могут носить несколько элементов, как разные члены семьи носят общую фамилию.

Инструменты разработчика

В каждом современном браузере есть «швейцарский нож» программиста — **панель инструментов разработчика**. Здесь можно посмотреть код всей страницы или конкретного элемента, изучить стили каждого элемента страницы и даже изменить их отображение на своём компьютере. В браузерах панель инструментов разработчика вызывают комбинацией `Ctrl + Shift + I`.

Ваш первый get-запрос

Чтобы получить данные с сервера, вам понадобится метод **`get()`**. Для отправления HTTP-запросов подключают библиотеку **Requests**:

```
import requests
```

Метод `get()` библиотеки *Requests* выступает в роли браузера. Он принимает ссылку на сайт в качестве аргумента. Метод отправит get-запрос на сервер, обработает полученный оттуда результат и вернёт объект **`response`**. *Response* — специальный объект, содержащий ответ сервера на HTTP-запрос:

```
req = requests.get(URL) # сохраняем объект Response в переменную req
```

Объект *Response* содержит ответ сервера: код состояния, содержание запроса и код самой HTML-страницы. Атрибуты объекта *Response* позволяют возвращать не все данные с сервера, а только нужные для анализа. Например, объект *Response* с атрибутом *text* «отдаст» лишь текстовое содержание запроса:




```
print(req.text) # название атрибута пишут после объекта Response, разделяя точкой
```

Атрибут *status_code* отвечает на вопрос: отправил сервер ответ или возникла какая-то ошибка:

```
print(req.status_code)
```

К сожалению, не все запросы возвращаются с данными. Иногда результатом запроса бывает ошибка: в зависимости от типа её обозначают специальным кодом. Вот коды ошибок, которые чаще всего возникают:

Коды ошибок

 Код ошибки	 Название	 Что означает?
<u>200</u>	OK	Всё отлично
<u>302</u>	Found	Расположение ресурса изменилось
<u>400</u>	Bad Request	Синтаксическая ошибка в запросе
<u>404</u>	Not Found	Ресурс не найден
<u>500</u>	Internal Server Error	Внутренняя ошибка сервера
<u>502</u>	Bad Gateway	Ошибка при обмене данных между серверами
<u>503</u>	Server Unavailable	Сервер временно не может обрабатывать запросы





Регулярные выражения



Для поиска строк с больших текстах используется мощный инструмент — регулярные выражения. **Регулярное выражение** — правило для поиска подстрок (фрагментов текста внутри строк). Регулярные выражения позволяют создавать сложные правила, так что одно выражение вернёт несколько подстрок.

Для работы с регулярными выражениями в Python импортируют библиотеку **re**. Дальше поиск ведётся в два этапа. Сначала создают шаблон регулярного выражения. Это алгоритм, по которому нужно искать строку в тексте. Затем готовый шаблон передают специальным методам библиотеки *re*, которые ищут, заменяют и удаляют нужные символы. Таким образом, шаблон определяет, что и как искать, а метод — что с этим потом делать.

В таблице приведены простейшие шаблоны регулярных выражений. Сложные регулярные выражения состоят из их комбинаций.

Синтаксис регулярных выражений

 Регулярное выражение	 Описание	 Пример	 Пояснение
[.]	Один из символов в скобках	[a-]	a или -
[^...]	Отрицание	[^a]	любой символ кроме «a»
-	Интервал	[0-9]	интервал: любая цифра от 0 до 9
.	Один любой символ, кроме перевода строки	a.	as, a1, a_
\d (аналог [0-9])	Любая цифра	a\d a[0-9]	a1, a2, a3
\w	Любая буква, цифра или _	a\w	a_, a1, ab
[A-z]	Любая латинская буква	a[A-z]	ab
[А-я]	Любая буква кириллицы	a[А-я]	ая
?	Ноль или одно вхождение	a?	a или ничего
+	Одно и более вхождений	a+	a или aa, или aaa
*	Ноль и более вхождений	a*	ничего или a, или aa
^	Начало строки	^a	a1234, abcd

 Регулярное выражение	 Описание	 Пример	 Пояснение
\$	Конец строки	a\$	1a, ba

Самые распространённые задачи аналитика:

- найти подстроку в строке
- разбить строки на подстроки на основании шаблона
- заменить части строки на другую строку

Вот какие методы библиотеки *re* для этого понадобятся:

1. **search(pattern, string)** ищет шаблон *pattern* в строке *string*. Хотя *search()* ищет шаблон во всей строке, возвращает он только первую найденную подстроку:

```
import re
print(re.search(pattern, string))
```

Метод *search()* возвращает объект типа **match**. Параметр *span* указывает диапазон индексов, подходящих под шаблон. В параметре *match* указано само значение подстроки.

Если нам не нужны дополнительные сведения о диапазоне, выведем только найденную подстроку методом *group()*:

```
import re
print(re.search(pattern, string).group())
```

2. **split(pattern, string)** разделяет строку *string* по границе шаблона *pattern*.

```
import re
print(re.split(pattern, string))
```

Строка разделена на несколько частей. Границы деления строки проходят там, где метод встретил указанный в аргументе шаблон. Количеством

делений строки можно управлять. За это отвечает параметр **maxsplit** метода *split()* (по умолчанию равен 0).

```
import re
print(re.split(pattern, string, maxsplit = num_split))
```

3. **sub(pattern, repl, string)** ищет подстроку по шаблону *pattern* в строке *string* и заменяет её на подстроку **repl**.

```
import re
print(re.sub(pattern, repl, string))
```

4. Метод **findall(pattern, string)** возвращает список *всех подстрок* в *string*, удовлетворяющих шаблону *pattern*. А не только первую подходящую подстроку, как *search()*.

```
import re
print(re.findall(pattern, string))
```

Метод **findall()** удобен тем, что можно сразу посчитать количество повторяющихся подстрок в строке функцией *len()*:

```
import re
print(len(re.findall(pattern, string)))
```

Парсинг HTML

Достать данные из строки, которая содержит код страницы, вручную сложно. Чтобы решить проблему, обратимся к возможностям библиотеки **BeautifulSoup**. Методы библиотеки *BeautifulSoup* превращают HTML-файл в

древовидную структуру. После этого нужный контент можно отыскать по тегам и атрибутам.

```
from bs4 import BeautifulSoup
soup = BeautifulSoup(req.text, 'lxml')
```

Первый аргумент — это данные, из которых будет собираться древовидная структура. Второй аргумент — синтаксический анализатор, или парсер. Он отвечает за то, как именно из кода веб-страницы получается «дерево». Парсеров много, они создают разные структуры из одного и того же HTML-документа. За высокую скорость работы мы выбрали анализатор **lxml**. Есть и другие, например, *html.parser*, *xml* или *html5lib*.

После превращения кода страницы в дерево, можно искать данные различными методами. Первый метод поиска называется **find()**. В HTML-документе он находит первый элемент, имя которого ему передали в качестве аргумента, и возвращает его весь, с тегами и контентом.

```
tag_content = soup.find(tag)
```

Чтобы посмотреть контент без тега, вызывают метод **text**. Результат возвращается в виде строки:

```
tag_content.text
```

Существует и другой метод поиска — **find_all**. Этот метод находит *все* вхождения определённого элемента в HTML-документе и возвращает список:

```
tag_content = soup.find_all(tag)
```

Методом *text* вычленим только контент из тегов:

```
for tag_content in soup.find_all(tag):
    print(tag_content.text)
```

У методов `find()` и `find_all()` есть дополнительный фильтр поиска элементов страницы — параметр **`attrs`**. Он используется для поиска по идентификаторам и классам. Их имена уточняют в панели разработчика.

Параметру `attrs` передают словарь с именами и значениями атрибутов:





```
soup.find(tag, attrs={"attr_name": "attr_value"})
```

API

Чтобы получать данные из внешних ресурсов, устроенных намного сложнее обычной HTML-страницы, аналитики отправляют GET-запросы к сторонним сервисам через специальный интерфейс передачи данных — API.

API позволяет разработчику взаимодействовать с системой, не беспокоясь о том, как именно она реализована. Для этого API предоставляет «инструкцию» — набор методов с определёнными параметрами. Например:

Параметры метода `forecasts`

 Имя параметра	 Описание	 Свойства	 Значения
<u><code>city</code></u>	город	обязательный	Moscow
<u><code>limit</code></u>	срок прогноза в днях	необязательный	от 1 до 7
<u><code>hours</code></u>	наличие почасового прогноза	необязательный	true / false
<u><code>extra</code></u>	подробный прогноз осадков	необязательный	true / false

Библиотека `requests` позволяет передавать параметры в URL. Ключевому слову `params` в GET-запросе передают словарь с параметрами.

```
import requests

BASE_URL = "https://weather.data-analyst.praktikum-services.ru/v1/forecast"
# URL метода
```

```
params = { # словарь с параметрами запроса
    "city" : "Moscow", # определяем город
    "hours" : True # указываем, что нужен почасовой прогноз
}

response = requests.get(BASE_URL, params=params)
print(response.text)
```

JSON

Отвечая на запрос, сервер возвращает структурированные данные в одном из специальных форматов. Самый распространённый из них — JSON (JavaScript Object Notation). Вот как выглядят данные в формате JSON:

```
[
  {
    "name": "Генерал Слокам",
    "date": "15 июня 1904 года"
  },
  {
    "name": "Каморта",
    "date": "6 мая 1902 года"
  },
  {
    "name": "Норье",
    "date": "28 июня 1904 года"
  }
]
```

JSON начинается с фигурных скобок, если содержит один объект, или с квадратных — если список объектов. Каждый элемент JSON записан в фигурных скобках, внутри которых пары `ключ : значение`.

Значениями ключей могут быть строки, числа, булевы значения, null, массивы, объекты. JSON не может содержать функций, переменных или комментариев. Обратите внимание, что ключи взяты в двойные кавычки — в JSON это обязательно.

В Python есть встроенный модуль для работы с данными в формате JSON. Его метод `json.loads()` конвертирует строки в формате JSON:

```
import json

x = '[{"name": "Генерал Слокам", "date": "15 июня 1904 года"}, {"name": "Каморта", "date": "6 мая 1902 года"}]'
y = json.loads(x)

for i in y:
    print('Name : {0}, date : {1}'.format(i['name'], i['date']))
```

```
Name : Генерал Слокам, date : 15 июня 1904 года
Name : Каморта, date : 6 мая 1902 года
```

Метод `json.dumps()`, наоборот, конвертирует данные из Python в формат JSON. При работе с буквами кириллицы указывают аргумент `ensure_ascii=False`:

```
out = json.dumps(y, ensure_ascii=False)
print(out)
```

```
[{"name": "Генерал Слокам", "date": "15 июня 1904 года"}, {"name": "Каморта", "date": "6 мая 1902 года"}]
```