

Python-12

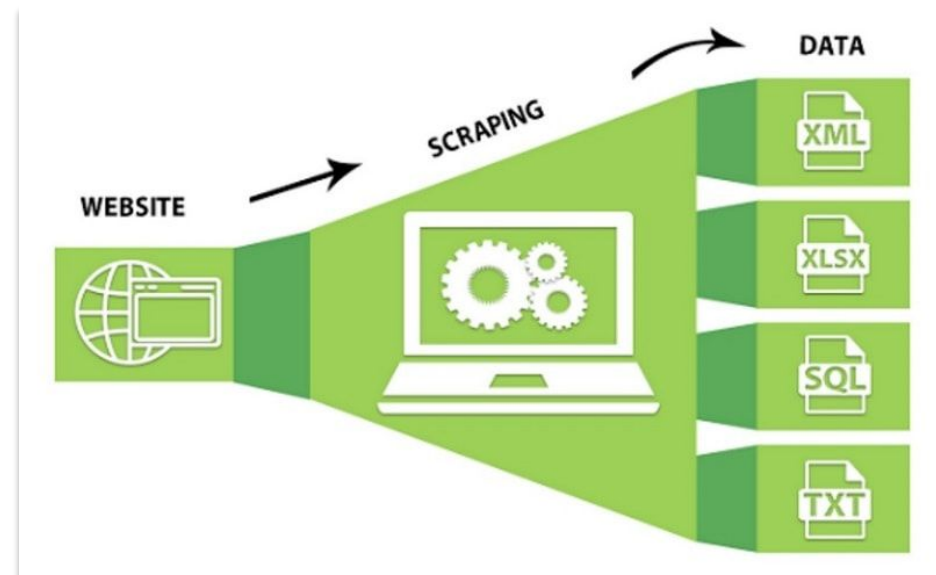
Парсинг данных.

Что такое парсинг данных?

Парсинг данных - это процесс анализа и извлечения нужной информации из какого-то источника данных.

Например, при парсинге веб-страницы можно извлекать текст, ссылки, изображения или другую информацию.

Парсинг помогает компьютеру понять структуру данных и извлечь необходимые элементы для дальнейшего использования. Это часто используется при написании программ, сценариев или ботов для автоматического сбора данных из интернета или других источников.



Язык HTML.

HTML - это язык разметки веб-страниц. Он состоит из тегов и атрибутов которые определяют структуру веб-страницы.

Например.

`<h1>`, `<h2>`, ... `<h6>` - это теги разных уровней заголовков.

`<p>` - тег абзаца (параграф)

`<a>` - тег ссылок

`<button>` - тег кнопка

`<title>` - заголовок веб-страницы

`<form>` - тег формы какой то

`<input>` - тег для ввода данных (чаще исп-ся в `<form>` совместно с `<label>`)

`
` - тег переноса строки

`<hr>` - тег горизонтальная линия на веб-странице

`<div>`/`` блочные элементы

``/`` - жирный шрифт, ``/`<i>` - курсив

`src`, `href`, `width`, .. - это все атрибуты



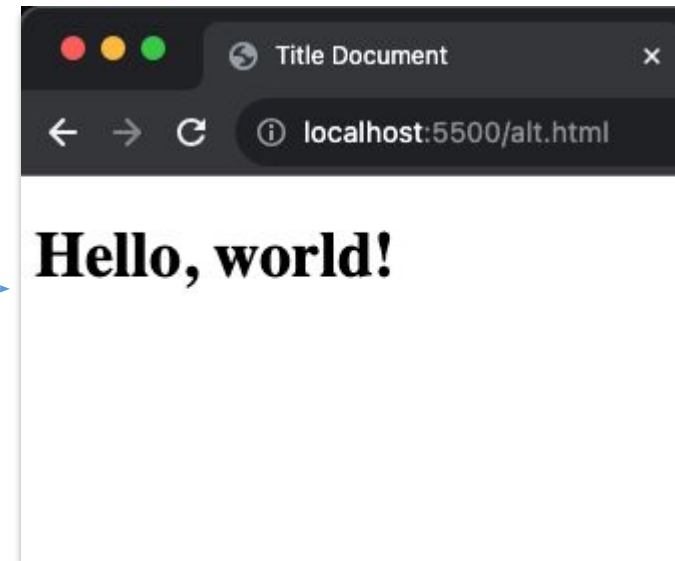
Простой HTML код.

DOCTYPE - тип документа

<meta charset="UTF-8"> - для установки кодировки сайта исп-ся

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Title Document</title>
</head>
<body>
  <h1>Hello, world!</h1>
</body>
</html>
```

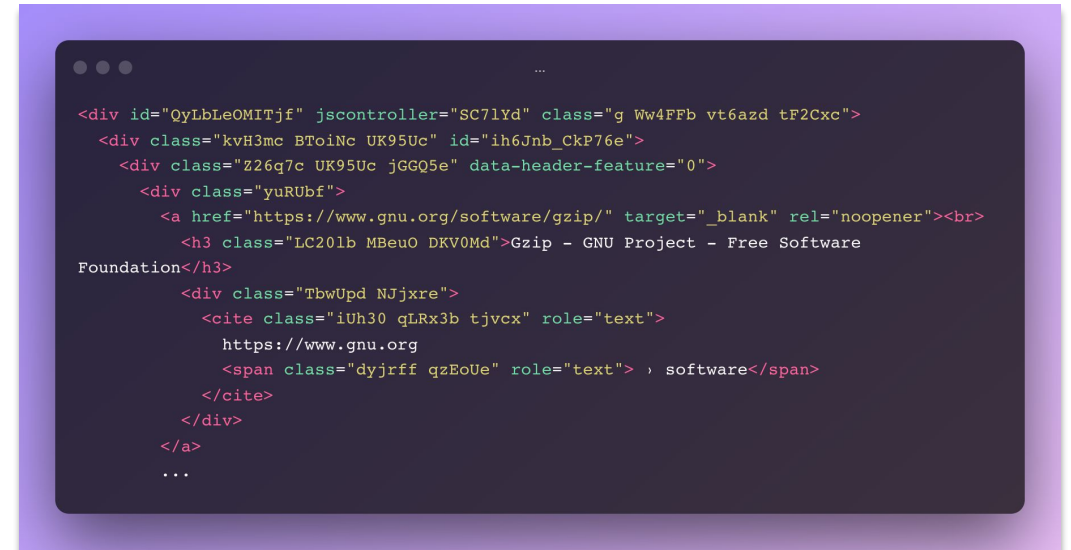
выведет
следующее



*<body> - основная часть сайта тут.

Классы и ID. HTML.

В HTML, классы и идентификаторы (ID) являются атрибутами, которые используются для стилизации и идентификации элементов на веб-странице. Они предоставляют возможность применять стили CSS, а также обеспечивают уникальность для идентификации элементов среди других. Так например, класс - позволяет использовать одни и те же стили для нескольких элементов, а ID - должен быть уникальным для каждого элемента веб-страницы. При этом необязательно указывать классы или ID.



```
<div id="QyLbLeOMITjf" jscontroller="SC7lYd" class="g Ww4FFb vt6azd tF2Cxc">
  <div class="kvH3mc BToiNe UK95Uc" id="ih6Jnb_CkP76e">
    <div class="Z26q7c UK95Uc jGGQ5e" data-header-feature="0">
      <div class="yuRUbf">
        <a href="https://www.gnu.org/software/gzip/" target="_blank" rel="noopener"><br>
          <h3 class="LC201b MBeuO DKV0Md">Gzip - GNU Project - Free Software
Foundation</h3>
        <div class="TbwUpd NJjxre">
          <cite class="iUh30 qLRx3b tjvcx" role="text">
            https://www.gnu.org
          <span class="dyjrff qzEoUe" role="text"> , software</span>
          </cite>
        </div>
      </a>
    </div>
  </div>
  ...
```

Чем поможет знание классов и id при парсинге данных?

Если на веб-странице определен класс используется для выделения текста или других элементов, вы можете использовать его в селекторах для точного извлечения данных.

Если элемент имеет уникальный идентификатор, это может быть также использовано для точного нахождения этого элемента.



```
!DOCTYPE html PUBLIC "-//W3C//DTD
head>
<meta http-equiv="Content-Type" co
<title>CSS3</title>
<link href="style.css" rel="styles
</head>
<body>
<div id="container"> <!-- Main Con
<div class="menu"> <!-- Menu here
<div class="article"><h2>CSS3 Samp
<div class="article"><h2>CSS3 & HTML5 are so good
```

Какие библиотеки/модули суц-т для парсинга?

Beautiful Soup (самая популярная)

является библиотекой Python для извлечения данных из HTML и XML-документов. Он предоставляет удобный интерфейс для навигации и поиска элементов на веб-страницах.

Установка: `pip install beautifulsoup4`

lxml

это библиотека для обработки XML и HTML в Python. Она предоставляет высокоэффективные и гибкие инструменты для парсинга.

Установка: `pip install lxml`

Requests

не является библиотекой для парсинга, но это популярный модуль для отправки HTTP-запросов. Он часто используется в сочетании с Beautiful Soup или lxml для загрузки веб-страниц перед их парсингом.

Установка: `pip install requests`

Scrapy

это фреймворк для извлечения данных с веб-сайтов. Он предоставляет удобные инструменты для создания пауков (spiders) для автоматического парсинга данных.

Установка: `pip install scrapy`

BS4

Импорт в проект.

*(Обычно также импортируют модуль requests, если требуется получение HTML-кода веб-страницы)

```
from bs4 import BeautifulSoup
import requests
```

Если вы работаете с веб-страницей, вам нужно получить ее HTML-код. Это можно сделать с использованием библиотеки requests:

```
url = 'https://...' # веб-страница (сайт)
response = requests.get(url)
html_content = response.text
```

Создайте объект BeautifulSoup, передавая ему HTML-код и указывая парсер (в данном случае, html.parser):

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

Используйте методы BS4 для навигации по дереву HTML и поиска интересующих вас элементов. Например, для поиска всех тегов <a> (ссылок) можно использовать метод find_all():

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


<продолжение>

Извлеките необходимые данные из найденных элементов. Например, для получения текста ссылок:

```
for link in links:  
    print(link.text)
```

Если вам нужно извлечь ссылки на изображения из HTML-кода веб-страницы с использованием BeautifulSoup, вы можете использовать тег `` и атрибут `src`, который содержит URL изображения.

```
# Найти все теги <img> на странице  
img_tags = soup.find_all('img')  
image_urls = []  
for img in img_tags:  
    image_urls.append(img['src'])  
  
# Вывести все URL изображений  
for url in image_urls:  
    print(url)
```

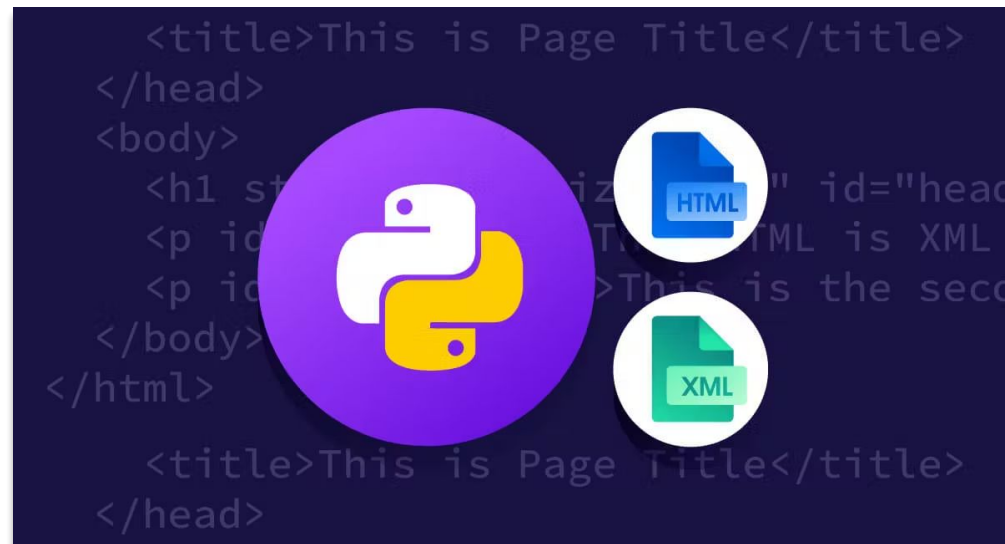
BS4. Основные методы.

find(tag, attrs, recursive, text, kwargs) - находит первый элемент, соответствующий указанным параметрам.
find_all(tag, attrs, recursive, text, limit, kwargs) - находит все элементы, соответствующие указанным параметрам.
select(css_selector) - выбирает элементы с использованием CSS-селектора.
get_text(separator, strip) - извлекает текст из элемента, объединяя его в строку.
find_parent(name, attrs, kwargs) - находит первого родителя, соответствующего параметрам.
find_parents(name, attrs, limit, kwargs) - находит всех родителей, соответствующих параметрам.
find_next_sibling(name, attrs, kwargs) - находит следующего брата, соответствующего параметрам.
find_next_siblings(name, attrs, limit, kwargs) - находит всех следующих братьев, соответствующих параметрам.
find_previous_sibling(name, attrs, kwargs) - находит предыдущего брата, соответствующего параметрам.
find_previous_siblings(name, attrs, limit, kwargs) - находит всех предыдущих братьев, соответствующих параметрам.

DOC BS4: <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

Lxml.

Lxml - это библиотека для работы с XML и HTML в языке программирования Python. Она предоставляет высокоэффективный и гибкий инструментарий для парсинга, обработки и создания XML-документов. (но BS4 круче..)



Lxml. Преимущества.

- высокая производительность, особенно при работе с большими XML-файлами;
- поддержка мощного языка запросов XPath, который позволяет выразительно выбирать и манипулировать элементами XML-структуры;
- позволяет удобно добавлять, изменять и удалять элементы в XML-дереве;
- универсальность исп-я инструмента;



Lxml. Основные методы.

fromstring(xmlstring) и **ElementTree(xmlroot)** - позволяют создавать XML-дерево из строки или корня.

Element и **SubElement** - создают новые элементы в XML-дереве.

Element.get(tag) и **Element.set(tag, value)** - получение и установка атрибутов элемента.

Element.find(path) и **Element.findall(path)** - выполнение поиска элементов с использованием XPath.

Element.text - получение или установка текстового содержимого элемента.

Element.getchildren(), **Element.iter()**, **итераторы** - получение дочерних элементов или использование итераторов для обхода элементов.



Lxml. Пример кода.

```
from lxml import etree

# Создание XML-дерева из строки
xml_string = '<root><element1>Value1</element1><element2>Value2</element2></root>'
root = etree.fromstring(xml_string)

# Получение значения атрибута
attribute_value = root.get('attribute_name')

# Изменение значения атрибута
root.set('attribute_name', 'new_value')

# Использование XPath для поиска элементов
element_list = root.xpath('//element1')

# Создание нового элемента
new_element = etree.Element('new_element')
new_element.text = 'New Value'
root.append(new_element)

# Итерация по дочерним элементам
for child in root.getchildren():
    print(child.tag, child.text)
```

xml_string =
'<root><element1>Value1</element1><element2>Value2</element2></root>' -
создание строки с XML-разметкой.

root = etree.fromstring(xml_string) -
создание XML-дерева из строки.

attribute_value = root.get('attribute_name') -
получение значения атрибута

'attribute_name'.
root.set('attribute_name', 'new_value') -
изменение значения атрибута
'attribute_name' на 'new_value'.

****потом там происходит**

1) создание нового элемента с тегом 'new_element', установка текстового содержимого нового элемента с добавлением в root;

2) вывод тега и текстового содержимого каждого дочернего элемента.