Введение в bs4.

- `pip install beautifulsoup4`
- `pip3 install beautifulsoup4`

```
from bs4 import BeautifulSoup
```

```
response.encoding = 'utf-8'
```

Когда мы парсим веб-сайты, серверы могут блокировать наш IP-адрес из-за слишком многих запросов. Для избежания этой проблемы, мы можем сначала сохранить HTML-страницу на наш компьютер и затем анализировать ее локально. Это особенно полезно, если сервер очень строгий и блокирует нас часто. Сохранение страницы и анализ на компьютере помогает избежать блокировок.

Введение в bs4.

Парсер	Типичное использование	Преимущества	Недостатки
html.parser	`BeautifulSoup(markup, "html.parser")`	• *Мягкий	• Не такой быстрый, как lxml, менее *мягкий, чем html5lib.
lxml	`BeautifulSoup(markup, "lxml")`	• Очень быстрый • *Мягкий	• Внешняя зависимость
xml	`BeautifulSoup(markup, "xml")`, --- `BeautifulSoup(markup, "lxml-xml")`	• Очень быстрый • Единственный в настоящее время поддерживаемый парсер XML	• Внешняя зависимость
html5lib	`BeautifulSoup(markup, "html5lib")`	• *Мягкий • Разбирает страницы так же, как веб-браузер • Создает действительный HTML5	• Очень медленно • Внешняя зависимость Python

Объекты Tag.

1. **Объекты Tag**: В BeautifulSoup элементы HTML и XML представлены объектами типа `Tag`. Эти объекты соответствуют XML или HTML тегам в исходном документе.

2. **Атрибуты Tag**: У объектов `Tag` есть множество атрибутов и методов для доступа к данным исходного документа:

   - `name`: Получает или задает имя тега.
   - `attrs`: Словарь, содержащий атрибуты тега.
   - Методы навигации, такие как `.find()`, `.find_all()` и др., для поиска других тегов внутри или вокруг данного тега.

3. **Манипуляции с Tag**: Объекты `Tag` являются изменяемыми, что означает, что вы можете изменять их имя, их атрибуты и их содержимое, как вам захочется.

```
from bs4 import BeautifulSoup

html = "<div class='myclass'>Hello, world!</div>"
soup = BeautifulSoup(html, 'html.parser')

tag = soup.div

print(type(tag))   # <class 'bs4.element.Tag'>
print(tag.name)    # div
print(tag.attrs)   # {'class': ['myclass']}
print(tag.string)  # Hello, world!
```

Таким образом, когда вы работаете с объектом `<class 'bs4.element.Tag'>`, вы работаете с конкретным тегом из вашего HTML- или XML-документа и можете проводить с ним различные операции: читать его содержимое, изменять атрибуты, искать внутри него другие теги и так далее.

.text

`.text` — это свойство в BeautifulSoup, которое позволяет получить весь текст внутри данного элемента, включая его дочерние элементы. Оно возвращает весь текст как одну непрерывную строку. Это свойство просто в использовании, вы можете получить текст элемента просто написав `element.text`.
`.text` менее гибкое по сравнению с `.get_text()`, так как не предоставляет дополнительных параметров для настройки извлечения текста.

```
from bs4 import BeautifulSoup

# Пример HTML-строки
html_string = """
<div>
    <p>Первый абзац.</p>
    <p>Второй абзац <span>со вложенным</span> текстом.</p>
</div>
"""

# Создание объекта BeautifulSoup
soup = BeautifulSoup(html_string, 'html.parser')

# Использование .text для извлечения всего текста из div
div_text = soup.find('div').text

print(div_text)
```

Вывод:

```
Первый абзац.
Второй абзац со вложенным текстом.
```

.get_text()

`.get_text()` — это метод, который также возвращает весь текст внутри элемента, но он предоставляет больше опций для настройки того, как текст извлекается и объединяется.
Вы вызываете этот метод, например, *element*`.get_text()` . Однако у вас есть возможность указать разделитель для текста из дочерних элементов, например `element.get_text(separator=' ')` .
`.get_text()` более гибок, предоставляя параметры, такие как `separator` для указания разделителя между текстами из разных дочерних элементов, `strip` для удаления начальных и конечных пробелов в каждой строке текста и другие.
Синтаксис:

```
.get_text(separator=" ", strip=False, types=default)
```

- `separator=" "` - этот параметр определяет строку, которая будет использоваться для разделения текста из разных тегов. Если вы укажете `separator` , например, как `" | "` , то текст из каждого дочернего элемента будет разделен этой строкой. Важно понимать, что разделитель вставляется между каждым "блоком" текста, который был извлечен из различных тегов, включая текст, который может быть окружен пробельными символами или переносами строк.

- `strip=False` - если этот параметр установлен в `True` , то BeautifulSoup будет удалять все начальные и конечные пробелы в каждом блоке текста перед их объединением. Это полезно для удаления нежелательных пробелов и переносов строк в тексте.

- Комбинация методов (`separator` **и** `strip`): `get_text(separator=" | ", strip=True)` вернет `Пример | Текст`

```
<div>
    <p>Пример</p>
    <p>Текст</p>
</div>
```

- `types=default` - этот параметр позволяет ограничить типы объектов, из которых извлекается текст. Обычно он используется редко и в особых случаях, когда нужно извлекать текст только из определенных типов тегов.

    - **Пример использования:** Не так часто используется в обычных задачах парсинга, но может быть полезен в сложных сценариях, когда нужно фильтровать определенные типы элементов.

.text и .get_text()

```
from bs4 import BeautifulSoup

# Тот же пример HTML-строки
html_string = """
<div>
    <p>Первый абзац.</p>
    <p>Второй абзац <span>со вложенным</span> текстом.</p>
</div>
"""

# Создание объекта BeautifulSoup
soup = BeautifulSoup(html_string, 'html.parser')

# Использование .get_text() с параметром separator
div_text = soup.find('div').get_text(separator=" | ")

print(div_text)
```

Вывод:

```
 | Первый абзац. |
 | Второй абзац  | со вложенным |  текстом. |
```

Основные отличия:

- **Настраиваемость:** `.get_text()` предлагает больше параметров и возможностей для настройки извлечения текста, в то время как `.text` просто возвращает весь текст как одну строку.
- **Способ использования:** `.text` — это свойство, а `.get_text()` — это метод. Это означает, что `.get_text()` может принимать параметры для более точного контроля над результатом.

.find()

`soup.find()` - Это метод, который ищет первый элемент, удовлетворяющий заданным критериям, в дереве элементов HTML DOM. Если такой элемент не найден, возвращается `None`.

Синтаксис:

```
soup.find(name, attrs={}, recursive=True, string, **kwargs)
```

Параметры:

Для поиска элементов с помощью метода `.find()` можно использовать различные параметры.

- `name('div')` - имя тега элемента, который вы хотите найти. Это может быть строкой или регулярным выражением.
- `attrs` - словарь атрибутов элемента, которые вы хотите найти.
- `recursive` - если установлено в `False`, метод ищет только в первых дочерних элементах, иначе поиск происходит во всех подэлементах. По умолчанию равен `True`.
- `string` - строка, которую вы хотите найти.

.find()

Поиск по ID:

```
# Здесь 'tag' - это имя тега, а 'your_id' - значение атрибута id.

soup.find('tag', id='your_id')
```

Поиск по классу:

```
# Заметьте, что используется class_ вместо class, так как class является зарезервированным словом в Python.

soup.find('tag', class_='your_class')
```

Поиск по пользовательскому атрибуту:

```
# Здесь 'data-attribute' - это имя пользовательского атрибута, а 'value' - его значение.

soup.find('tag', {'data-attribute': 'value'})
```

Поиск по двойному пользовательскому атрибуту:

```
# Здесь у тега одновременно должны быть два атрибута с заданными значениями.

soup.find('tag', {'data-attribute1': 'value1', 'data-attribute2': 'value2'})
```

.find()

Поиск по двойному классу:

```
# Указываются оба класса, разделенные пробелом.

soup.find('tag', class_='class1 class2')
```

Поиск по динамическому классу:

```
# Если класс формируется динамически и вы знаете только часть названия, можно использовать регулярные выражения:

import re
soup.find('tag', class_=re.compile('your_dynamic_part'))
```

Поиск по динамическому пользовательскому атрибуту:

```
# 'your_dynamic_part' - часть имени класса.

soup.find('tag', {'data-attribute': re.compile('dynamic_part')})
```

Поиск по тегу без атрибутов:

```
# Просто укажите имя тега:
# Это найдет первый тег с указанным именем без учета его атрибутов.

soup.find('tag')
```

.find_all()

`soup.find_all()` - это метод Beautiful Soup, который ищет все элементы в HTML-документе, соответствующие переданным параметрам. Возвращает список элементов Beautiful Soup, которые удовлетворяют условиям поиска.

Синтаксис:

```
soup.find_all(name, attrs={}, recursive=True, string=None, limit=None, **kwargs)
```

Параметры:

- `name` - имя тега HTML, который нужно найти. Может быть строкой с именем тега, регулярным выражением, списком имен тегов и другими типами фильтров. Опциональный параметр.
- `attrs={}` - словарь атрибутов и их значений, которые нужно найти. Опциональный параметр.
- `recursive=True` - определяет, будут ли поиск проводиться рекурсивно по всем вложенным элементам или только среди прямых потомков. Если `True`, поиск будет проводиться по всем дочерним элементам, если `False` — только среди прямых потомков. По умолчанию установлено значение `True`. Опциональный параметр.
- `string=None` - параметр используется для поиска элементов с определенным текстом. Может быть строкой, регулярным выражением и другими типами фильтров. Если параметр `string` не установлен или установлен в `None`, то текстовое содержимое элементов не учитывается при поиске. Опциональный параметр.
- `limit=None` - максимальное количество элементов, которые будут найдены. Например, если установлено `limit=2`, `find_all` вернет не более двух элементов. `None` - поиск будет происходить без ограничений на количество возвращаемых элементов. Опциональный параметр.

Если нужно найти элементы, которые имеют хотя бы один из указанных классов.

```
elements = soup.find_all('div', class_=['class1', 'class2'])

# Найдём элементы, имеющие хотя бы один из искомых классов

[<div class="class1 class2">Элемент 1</div>, <div class="class1 class3">Элемент 2</div>, <div class="class2
class3">Элемент 3</div>]
```

.select()

`soup.select()` - это метод Beautiful Soup, который позволяет выполнять поиск элементов HTML с помощью CSS селекторов. Он возвращает список тегов, удовлетворяющих заданным условиям.

Вы можете использовать CSS селекторы, такие как тег, класс, идентификатор и другие атрибуты, чтобы найти нужные элементы в документе HTML. Например, вы можете использовать селектор `p.text-class`, чтобы найти все теги `<p>` с классом `text-class`. Или вы можете использовать селектор `p#text-id`, чтобы найти тег `<p>` с идентификатором `text-id`.

Обратите внимание, что `.select()` всегда возвращает список тегов, даже если вы ищете один тег. Чтобы получить один тег, вы можете использовать индекс **[0]** или метод `select_one()`.

Основные отличия `.find()` от `.select()`:

- **Метод поиска:** `.find()` ищет элементы на основе имени тега, атрибутов или текста, в то время как `.select()` использует CSS-селекторы.
- **Результат:** `.find()` возвращает первый найденный элемент, который соответствует критериям, в то время как `.select()` возвращает список всех элементов, соответствующих CSS-селектору.
- **Гибкость:** `.select()` предоставляет более широкие возможности для сложных запросов, особенно при работе с вложенными или специфическими структурами HTML.

Синтаксис:

```
soup.select(selector, namespaces=None, limit=None, **kwargs)
```

.select()

Параметры:

- `selector(str)` - Это основной параметр, куда передается строка CSS-селектора. Селекторы могут быть различной сложности, включая теги, классы, идентификаторы, вложенность и т.д. Например, `"div.content p"` выберет все `<p>`, которые находятся внутри `<div>` с классом `content`.
    - **Пример использования:** `soup.select("div.content p")`
- `namespaces(dict, optional)` - Позволяет указать пространства имен для тегов XML. Это полезно, если вы работаете с XML-документами, которые используют пространства имен.
    - **Пример использования:** Если у вас есть XML с пространствами имен, вы можете использовать `namespaces={"ns": "http://example.com"}` и затем использовать CSS-селекторы, например, `soup.select("ns|tag")`.
- `limit=(int, optional)` - Определяет максимальное количество элементов, которые будут возвращены. Если `limit` не указан, будут возвращены все найденные элементы. Если указан, возвращается не более указанного количества первых найденных элементов.
    - **Пример использования:** `soup.select("p", limit=2)` вернет не более двух элементов `<p>`.

.select()

Примеры:

- Тег: `select("p")`
- Класс: `select(".class")`
- Идентификатор: `select("#id")`
- Атрибут: `select("[attribute=value]")`
- Несколько селекторов: `select("p.class")`
- Вложенные элементы: `select("div span")`
- Непосредственные дочерние элементы: `select("div > p")`
- Элементы с несколькими классами: `select(".class1.class2")`
- Элементы с определенными атрибутами: `select("[data-custom]")`
- Псевдоклассы CSS: `select("p:first-of-type")` или `select("p:last-of-type")`
- Соседние элементы: `select("h1 + p")`
- Элементы, содержащие определенный текст:

```
elements = soup.select("p")
elements_with_text = [elem for elem in elements if "определенный текст" in elem.text]
for elem in elements_with_text:
    print(elem.text)
```

.select_one()

`soup.select_one()` - это метод, который позволяет выбрать один элемент на основе CSS-селектора. Он возвращает первый найденный элемент, удовлетворяющий условию, или `None`, если ничего не было найдено. Этот метод полезен, когда вы знаете, что только один элемент должен удовлетворять условию.

`.select_one()` принимает тот же аргумент, что и `.select()`, т.е. CSS-селекторы.

Разница между ними в том, что `.select_one()` возвращает только первый найденный тег, в то время как `.select()` возвращает список всех найденных тегов.

Синтаксис:
`select_one(selector)`

Параметры:
Метод `.select_one()` принимает один аргумент - CSS-селектор `(selector)` в виде строки. Он используется для поиска первого элемента в HTML или XML документе, который соответствует этому селектору.

Примеры:

- Тег: `select_one("p")`
- Класс: `select_one(".class")`
- Идентификатор: `select_one("#id")`
- Атрибут: `select_one("[attribute=value]")`
- Несколько селекторов: `select_one("p.class")`
- Вложенные элементы: `select_one("div span")`
- Непосредственные дочерние элементы: `select_one("div > p")`
- Элементы с несколькими классами: `select_one(".class1.class2")`
- Элементы с определенными атрибутами: `select_one("[data-custom]")`
- Псевдоклассы CSS: `select_one("p:first-of-type")` или `select_one("p:last-of-type")`
- Соседние элементы: `select_one("h1 + p")`

Навигация по структуре HTML

BeautifulSoup создает объект из HTML-дерева, по которому мы можем осуществлять необходимую нам навигацию и поиск элементов.

Самые простые и понятные методы, которыми мы пользуемся, когда пишем наши парсеры, это:

- `soup.find()` - этот метод возвращает только первый найденный элемент или узел HTML. Он идеален для ситуаций, когда вам нужно быстро найти конкретный элемент на странице.
- `soup.find_all()` - этот метод возвращает **список всех найденных элементов**. Он чрезвычайно полезен, когда необходимо получить все элементы определенного типа. Например, если вы хотите извлечь все ссылки с веб-страницы. Часто `soup.find_all()` используется в комбинации с `soup.find()`, чтобы сначала определить определенный родительский элемент, а затем искать внутри него.

Описание строки `div = soup.find('div', 'item').find_all('li')`:

- `soup.find('div', 'item')`: Этот код ищет первый элемент `<div>` с классом `item` на веб-странице. Если такой элемент найден, он возвращается в виде объекта типа `<class 'bs4.element.Tag'>`.
- `soup.find_all('li')`: Этот метод вызывается для ранее найденного элемента `<div>`. Он ищет и извлекает все вложенные элементы `<li>` внутри этого `<div>`. Возвращаемый результат - это список элементов `<li>`, представленных в виде объектов `<class 'bs4.element.Tag'>`.

Получение значения атрибута у HTML-тега с помощью метода .get()

`tag.get(key, default=None)` - в BeautifulSoup используется для получения значения атрибута у HTML-тега. Этот метод предлагает безопасный и удобный способ доступа к атрибутам, возвращая `None`, если атрибут отсутствует, что помогает избежать ошибок. Метод `.get()` применяется к объектам типа `bs4.element.Tag` и позволяет получить значение указанного атрибута.
Синтаксис:

```
tag.get('attribute_name')
```

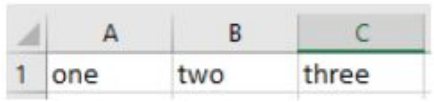
где, `'attribute_name'` - имя атрибута, значение которого вы хотите получить.

Пример 1: Получение ссылки из тега `<a>`

```
from bs4 import BeautifulSoup

html_doc = '<a href="https://example.com">Visit example.com</a>'
soup = BeautifulSoup(html_doc, 'html.parser')

a_tag = soup.find('a')
href_value = a_tag.get('href')

print("Href value:", href_value)  # Href value: https://example.com
```

Сохранение результата в csv

```
import csv

lst = ['one', 'two', 'three']

with open('res.csv', 'w', newline='', encoding='utf-8-sig') as file:
    writer = csv.writer(file, delimiter=';')
    writer.writerow(lst)
```



	A	B	C
1	one	two	three

- `newline=''` - необходимо указывать всегда. Если не указать, то новые строки могут интерпретироваться неправильно и весь документ “сползет”;
- `encoding='utf-8-sig'` - `open()` использует для открытия `.csv` по умолчанию кодировку **unicode**. Чтобы получить файл с необходимой нам кодировкой, нужно явно указывать ее.

`writer = csv.writer(file, delimiter=';')` - в этой строке мы создали экземпляр класса **csv** и применили к нему метод `writer()`. У `writer()` есть метод `writerow()`, с помощью которого можно записывать список в соответствующий формат построчно. `delimiter=';'` указывает, каким будет разделитель между элементами списка, мы можем указать любой.