Парсинг: основные библиотеки | Я.Шпора

Библиотека requests-cache

Загружает HTTP-ответ от сервера, кеширует этот ответ и передаёт его на обработку другим библиотекам.

Установка

```
pip install requests-cache
```

Пример использования

```
# Импортируем библиотеку.
import requests_cache

# Запускаем сессию, кеширующую результаты загрузки страниць
session = requests_cache.CachedSession()
# Загружаем веб-страницу при помощи HTTP-метода get().
response = session.get('https://адрес_веб-страницы_для_парс
```

Команда очистки кеша

```
session.cache.clear()
```

Вывод списка закешированных страниц

```
print(session.cache.urls)
```

Больше информации о библиотеке `requests-cache` ищите в документации.

Библиотека tqdm

Библиотека, с помощью которой в Python можно реализовать прогресс-бар.

Установка

```
(venv) ...$ pip install tqdm
```

Пример использования

```
from time import sleep
# Импортируем функцию tqdm из модуля tqdm.
from tqdm import tqdm

# Передаём в функцию tqdm() итерируемый объект.
for i in tqdm(range(10000)):
    sleep(0.003)
```

Как проанализировать вывод в терминале

Больше информации о библиотеке `tqdm` ищите в документации.

Библиотека Beautiful Soup

Библиотека для парсинга HTML и XML-документов. Она может находить нужные теги внутри содержимого, переходить между тегами, удалять и модифицировать их.

Установка bs4

```
(venv) ...$ pip install beautifulsoup4
```

Установка модуля lxml

```
(venv) ...$ pip install lxml
```

Модуль `lxml` ускоряет программный разбор DOM-дерева, что позволяет значительно ускорить парсинг большого объёма данных. Когда нужно обработать сотни и тысячи страниц, разница в общем времени работы парсера будет существенной.

Пример использования bs4

```
import requests
# Импортируем модуль.
from bs4 import BeautifulSoup
response = requests.get('https://адрес_веб-страницы_для_парсинга')
# Создаём для дальнейшей работы объект soup из веб-страницы.
soup = BeautifulSoup(response.text, features='lxml')
```

Метод prettify()

Задача метода — отформатировать полученный HTML-код и вывести его в читаемом виде. Метод можно применять не только к веб-странице целиком, но и к отдельным тегам.

Пример использования

```
import requests
from bs4 import BeautifulSoup
response = requests.get('https://адрес_веб-страницы_для_парсинга')
soup = BeautifulSoup(response.text, features='lxml')
# Печатаем весь "суп".
print(soup.prettify())
```

```
# Печатаем содержимое только тега <body>.
print(soup.html.body.prettify())
```

Метод find_all()

Применяется, чтобы найти все теги с одинаковыми именами и атрибутами.

Как работает

Метод `find_all()` всегда возвращает список, даже если найден только один элемент. Если в коде нет запрошенных элементов — `find_all()` вернёт пустой список.

Параметры метода

У `find_all()` нет обязательных параметров, и если в коде программы написать `all_tags = soup.find_all()`, то будут найдены вообще все теги, которые есть в «супе».

Когда нужно найти что-то конкретное, чаще всего применяются параметры:

- `name` — название тега;
- `attrs` — название атрибута и его значение.

Параметр `name` принимает названия тегов, которые нужно найти в «супе». Эти названия можно передать в разных форматах:

```
...
# Строкой: ищем в коде все теги <b>.
result = soup.find_all('b')

# Списком: ищем в коде все теги <title> и <b>.
result = soup.find_all(['title', 'b'])
...
```

В параметр `attrs` передаются атрибуты искомого тега:

```
...
# Ищем в коде все элементы с классом 'title'.
result = soup.find_all(attrs={'class': 'title'})
```

```
...
```

Можно использовать комбинированный поиск — по тегам и атрибутам одновременно:

```
...
# Ищем в коде все элементы <a class='hero'>.
result = soup.find_all('a', attrs={'class': 'hero'})

...
```

В качестве ключевого аргумента можно передать класс тега, но при этом важно следить за синтаксисом. Так как слово *class* зарезервировано в Python, после него нужно дополнительно прописать нижнее подчёркивание, чтобы получилось `class_`:

```
...
# Ищем в коде все элементы class_='antihero'.
result = soup.find_all(class_='antihero')

...
```

Метод `find_all()` находит список из тегов, внутри которых можно продолжать поиск. Но искать по всему списку нельзя. Нужно извлечь конкретный элемент — через цикл или по индексу:

```
...
# Ищем в HTML-коде все теги <p class='story'>.
all_stories = soup.find_all('p', class_='story')
# Берём только первый тег с индексом 0.
first_story = all_stories[0]
# Ищем внутри найденного тега id='link2'.
link2 = first_story.find_all(id='link2')
...
```

Метод find()

Метод `find()` возвращает не список, а отдельный элемент, первый, найденный в коде:

```
...
# Ищем в коде первый элемент <a class="hero">.
result = soup.find('a', attrs={'class': 'hero'})

# Будет найден только один элемент,
# хотя под критерии поиска может подходить несколько.
```

Можно продолжить поиск внутри найденного тега:

```
...
# Ищем в коде первого элемента <p class='story'>.
first_story = soup.find('p', class_='story')
# Продолжаем поиск внутри найденного тега элемента <id='l
ink2'>
link2 = first_story.find(id='link2')
...
```

Атрибут text

Позволяет извлечь текстовое содержимое тега:

```
...
first_story = soup.find('p', class_='story')
link2 = first_story.find(id='link2')

# Обращаемся к тексту тега.
print(link2.text)
```

К атрибутам найденных тегов можно обращаться как к ключам словаря:

```
...
first_story = soup.find('p', class_='story')
link2 = first_story.find(id='link2')

# Обращаемся к атрибуту тега, в котором содержится ссылк
```

```
a.
print(link2['href'])
```

Ссылки на документацию:

- библиотека Beautiful Soup,
- метод find_all(),
- метод find().

Практикум