Веб-запросы

Процесс получения/извлечения информации с веб-ресурсов в интернете называется **web-scraping** (рус. **веб-скрейпинг/веб-скрапинг**). Веб-скрапинг может быть проделан вручную пользователем компьютера, однако этот термин обычно связывают с автоматизированными процессами, реализованными с помощью кода.

Отправляя запросы от компьютера-клиента к компьютеру-серверу, в процессе мы можем получать от различную информацию, например:

- цены на товары конкурентов для оптимизации своей стратегии ценообразования;
- сообщения в социальных медиа, по которым можно отслеживать тренды в той или иной области;
- отзывы на товары/услуги компании на различных площадках, которые можно впоследствии анализировать;
- контактные данные пользователей из соцсетей или форумов для дальнейшего взаимодействия с ними;
- и т.д.

При работе с запросами по протоколу **HTTP** наиболее часто используются **GET**- и **POST**-методы.

Библиотека requests

```
# Устанавливаем библиотеку requests
!pip install requests
# Импортируем библиотеку requests
import requests
# Определяем значение переменной url, содержащей адрес страницы для запроса курса валют в формате JSON
url = 'https://www.cbr-xml-daily.ru/daily_json.js'
# Делаем GET-запрос к ресурсу и результат ответа сохраняем в переменной response
response = requests.get(url)
# Выводим значение response на экран как объект
print(response)
# Выводим числовое значение response на экран
print(response.status_code)
```

Парсинг сайтов

HTML (англ. HyperText Markup Language — рус. Язык Гипертекстовой Разметки) — стандартизированный язык разметки документов в интернете. Большинство веб-страниц содержат описание разметки на языке HTML.
Язык HTML интерпретируется браузерами. Полученный в результате интерпретации текст отображается на экране монитора компьютера или мобильного устройства.

Пример страницы в HTML-разметке

```
<h2> Это заголовок второго уровня </h2>
<div> А это обычный текст </div>
<!DOCTYPE html>
<html lang="ru">
   <header>
       <title>Название страницы</title>
       <meta charset="UTF-8">
   </header>
   <body>
       <h1> Это заголовок страницы </h1>
       <p> Какой-то текст </p>
   </body>
</html>
```

Для успешного парсинга сайтов необходимо понимать, что:

- существуют теги с разными именами;
- у тегов бывают атрибуты, например такие как **`class`** и **`id`**;
- теги образуют иерархическую структуру, то есть одни теги вложены в другие.

Библиотека BeautifulSoup

```
# Устанавливаем библиотеку BeautifulSoup
!pip install beautifulsoup4
# Импортируем библиотеку BeautifulSoup
from bs4 import BeautifulSoup
# Импортируем библиотеку requests
import requests
```

```
# Определяем значение переменной url, содержащей адрес страницы для запроса
url = 'https://nplus1.ru/news/2021/10/11/econobel2021'
# Выполняем GET-запрос, содержимое ответа присваивается переменной response
response = requests.get(url)
# Создаём объект BeautifulSoup, указывая html-парсер
page = BeautifulSoup(response.text, 'html.parser')
# Получаем тег title, отображающийся на вкладках браузера, и выводим полученное значение на экран
print(page.title)
# Выводим текст из полученного тега, который содержится в атрибуте text
print(page.title.text)
# Применяем метод find() к объекту и выводим результат на экран
print(page.find('h1').text)
# Выводим на экран содержимое атрибута text тега time
print(page.find('time').text)
# Выводим содержимое атрибута text тега div класса body
print(page.find('div', class_='body').text)
# Ищем ссылку по тегу <a> и выводим её на экран
print(page.find('a'))
# Ищем все ссылки на странице и сохраняем в переменной links
links = page.find_all('a')
```

Работа с API

> Для того чтобы начать работать с API, обычно необходимо получить сервисный ключ авторизации — **токен**.
> **Токен** — это средство идентификации пользователя или отдельного сеанса работы в компьютерных сетях и приложениях.

Пример работы с API ВКонтакте через строку браузера

(вместо слова **TOKEN** необходимо указывать свой сервисный токен вида 8b3341297d3341297d334129917d5e4be377d337d33412920ba951f8120284cd53ff3bd)

```
# GET-запрос передаётся серверу через адресную строку браузера
https://api.vk.com/method/users.get?user_id=1&v=5.95&access_token=TOKEN
```

Примеры работы с API ВКонтакте из кода

(в коде при запросе необходимо указывать свой сервисный токен вида 8b3341297d3341297d334129917d5e4be377d337d33412920ba951f8120284cd53ff3bd)

```
# Импортируем модуль requests
import requests
# Указываем свой сервисный токен
token = ' ... '
# Указываем адрес страницы, к которой делаем запрос
url = 'https://api.vk.com/method/users.get'
# Перечисляем параметры запроса в словаре params
params = {'user_id': 1, 'v': 5.95, 'fields': 'sex,bdate',
'access_token': token, 'lang': 'ru'}
# Отправляем запрос
response = requests.get(url, params=params)
# Выводим текст ответа на экран
print(response.text)
# Извлекаем из словаря по ключу response информацию о первом
пользователе
user = response.json()['response'][0]
# Выводим дату рождения первого пользователя на экран
print(user['bdate'])
# Формируем строку, содержащую информацию о поле id первых трёх
пользователей
ids = ",".join(map(str, range(1, 4)))
# Формируем строку параметров
params = {'user_ids': ids, 'v': 5.95, 'fields': 'bday',
'access_token': token, 'lang': 'ru'}
# Посылаем запрос, полученный ответ в формате JSON-строки преобразуем
в словарь и выводим на экран его содержимое, используя функцию
pprint()
pprint(requests.get(url, params=params).json())
```

Основные ограничения при работе с API, которые свойственны практически всем системам:

- ограничение на количество выгружаемых строк за один запрос;
- ограничение на количество вызовов в единицу времени.

Для решения проблемы первого ограничения необходимо использовать цикл — выгружать строки партиями, последовательно, в ограниченном количестве. Т.е. выгружается по N строк за запрос до тех пор, пока не получим все строки.

Пример кода: выгрузка информации о 20 пользователях, за один цикл выгружаем информацию о 5 пользователях.

```
# Импортируем модуль requests
import requests
# Указываем свой сервисный токен
token = ' ... '
# Указываем адрес обращения
url = 'https://api.vk.com/method/groups.getMembers'
# Указываем количество запросов за цикл
count = 5
# Указываем значение первоначального смещения
offset = 0
user_ids = []
# Указываем количество пользователей, информацию о которых необходимо получить
max_count = 20
while offset < max_count:
   # Будем выгружать по count=5 пользователей,
   # начиная с того места, где закончили на предыдущей итерации (offset)
   print('Выгружаю {} пользователей с offset = {}'.format(count, offset))
   params = {'group_id': 'vk', 'v': 5.95, 'count': count, 'offset': offset, 'access_token': token}
   response = requests.get(url, params = params)
   data = response.json()
   user_ids += data['response']['items']
   # Увеличиваем смещение на количество строк, которое мы уже выгрузили:
   offset += count
print(user_ids)
```

Для решения проблемы второго ограничения (частота запросов) необходимо после каждого запроса делать паузу. В этом случае даже если код будет выполняться на самом быстром компьютере, мы не нарушим установленное ограничение, т.к. периодичность отправки запросов будет искусственно замедлена.

Пример кода: после каждого запроса делаем паузу в 0.5 сек.

```
# Импортируем модуль requests
import requests
```

```
# Импортируем модуль time
import time
# Указываем свой сервисный токен
token = ' ... '
# Указываем адрес обращения
url = 'https://api.vk.com/method/groups.getMembers'
count = 1000
offset = 0
user_ids = []
while offset < 5000:
   params = {'group_id': 'vk', 'v': 5.95, 'count': count, 'offset':
offset, 'access_token': token}
   response = requests.get(url, params = params)
   data = response.json()
   user_ids += data['response']['items']
   offset += count
   # Делаем паузу в 0.5 сек.
   print('Ожидаю 0.5 секунды...')
   time.sleep(0.5)
print('Цикл завершён, offset =',offset)
```

Автоматический запуск

```
# Устанавливаем библиотеку schedule
!pip install schedule
# Импортируем модуль schedule
import schedule
# Функция, которая будет выполняться каждые 15 минут - выводить на
экран сообщение 'Hello! I am a task!'
def task():
   print('Hello! I am a task!')
   return
# Установка расписания запуска
schedule.every(15).minutes.do(task)
# Импортируем модуль time
import time
# Организуем бесконечный цикл, внутри которого проверяем, есть ли
функции для запуска
while True:
      schedule.run_pending()
      time.sleep(1)
```