3. Beautiful Soup и работа с API

Цель занятия

В результате обучения на этой неделе вы:

- узнаете методы модуля BS;
- научитесь пользоваться модулем для подгрузки данных из интернета для решения собственных задач;
- узнаете, что такое WEB-API;
- научитесь пользоваться WEB-API.

План занятия

1. Обзор методов модуля Beautiful Soup
2. Сложный поиск и изменение с Beautiful Soup
3. Взаимодействие с API
4. Практический пример работы с API

Используемые термины

BeautifulSoup — модуль, который призван облегчить извлечение данных из XML и HTML документов.

Время загрузки (Load time) – время, которое затрачивается на то, чтобы получить весь контент от сервера.

Время рендера (Render time) – сколько времени нужно, чтобы отрисовать все элементы и «собрать» страницу из загруженного контента

Конспект занятия

1. Обзор методов модуля Beautiful Soup

Модуль BeautifulSoup

Регулярные выражения – не лучший метод обработки HTML страницы. Если есть какая-то проблема, и вы решаете её с помощью регулярных выражений, то проблемы уже две.

BeautifulSoup — модуль, который призван облегчить извлечение данных из XML и HTML документов. Существует в двух версиях:

- bs3 — более не поддерживается
- bs4 — актуальная версия

Установка:

```
pip3 install beautifulsoup4
```

Алгоритм работы с модулем примерно следующий:

1. Получаем HTML-код страницы
2. Загружаем его в bs4 (создаём «суп»)
3. Извлекаем нужные элементы:
    a. по тегу
    b. по осям (parent/child, next/previous sibling)
    c. извлечение текста из html

Работа со всем файлом

Рассмотрим в качестве примера следующий HTML файл:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <title>The Project Gutenberg eBook of Ulysses, by James Joyce</title>
    <link href="images/cover.jpg" rel="icon" type="image/x-cover" id="id-3532245686325162110">
    <link rel="schema.dc" href="http://purl.org/dc/elements/1.1/">
    <link rel="schema.dcterms" href="http://purl.org/dc/terms/">
    <link rel="dcterms.isFormatOf" href="http://www.gutenberg.org/ebooks/4300">
```

```
</head>
<body>
    <h1>
        Ulysses
    </h1>
    <h2 class="no-break">by James Joyce</h2>
    <hr>
    <div class="chapter">
        <h2><a id="part01"></a>— I —</h2>
    </div>
    <p>
        <!--end chapter-->
    </p>
    <div class="chapter">
        <h3><a id="chap01"></a>[ 1 ]</h3>
        <p>
            Stately, plump Buck Mulligan came from the stairhead, bearing a bowl of lather
            on which a mirror and a razor lay crossed. A yellow dressinggown, ungirdled,
            was sustained gently behind him on the mild morning air. He held the bowl aloft
            and intoned:
        </p>
        <p>
            —<i>Introibo ad altare Dei</i>.
        </p>
        <p>
            Halted, he peered down the dark winding stairs and called out coarsely:
        </p>
        <p>
            —Come up, Kinch! Come up, you fearful jesuit!
        </p>
```

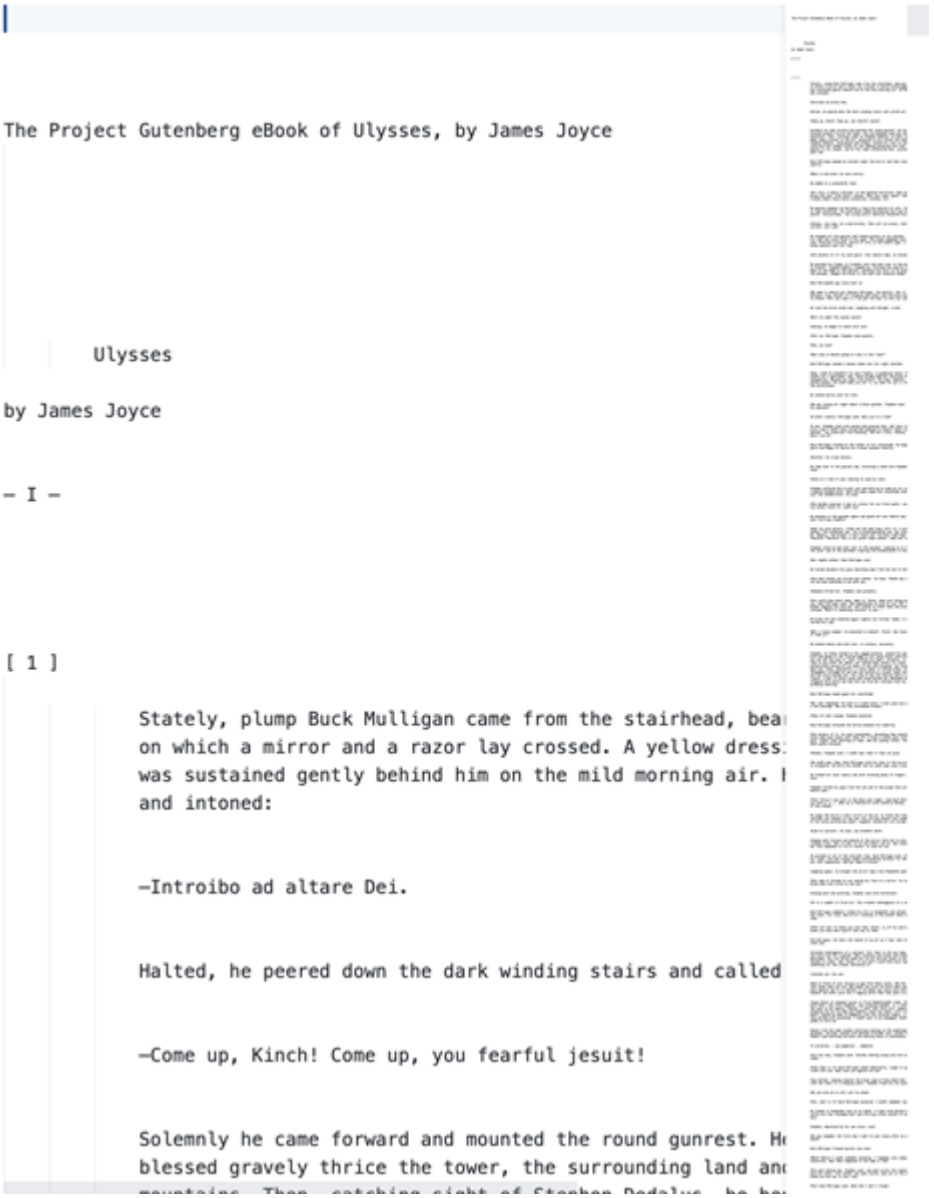
Файл содержит в себе роман Джеймса Джойса “Улисс”. Файл взят с сайта gutenberg, Сайт хранит в открытом доступе электронные версии различных книг, которые уже не подпадают под закон об авторском праве. Такие книги можно спокойно скачивать, и пользоваться ими.

Попробуем для начала оставить только текст и убрать из него все теги.

Функция **get_text()** — оставить только текст, без тегов.

```
from bs4 import BeautifulSoup
with open("ulysses.html", "r", encoding="utf-8") as f:
   html_doc = f.read()
soup = BeautifulSoup(html_doc, "html.parser")
with open("ulysses.txt", "w", encoding="utf-8") as↓f:
   f.write(soup.get_text())
```

Получаем текст в таком виде:



Видим, что в тексте очень много пустых пробелов. Связано это с особенностью работы функции get_text(). Если внутри тегов нет никакого текста, то остается пустая строка. Поэтому текст нужно дообработать, убрав из него пустоты.

Еще один вариант работы: функция **prettify()** — красивое форматирование HTML. Она берет любой HTML код и красиво его выводит.

```
from bs4 import BeautifulSoup
with open("ulysses.html", "r", encoding="utf-8") as f:
```

```
    html_doc = f.read()
soup = BeautifulSoup(html_doc, "html.parser")
with open("ulysses_p.html", "w", encoding="utf-8") as f:
    f.write(soup.prettify())
```

Получаем текст без лишних пробелов и пустых строк, видим все уровни вложенности.

```
<!DOCTYPE html>
<html lang="en">
 <head>
  <meta charset="utf-8"/>
  <title>
   The Project Gutenberg eBook of Ulysses, by James Joyce
  </title>
  <link href="images/cover.jpg" id="id-3532245686325162110" rel="icon"
  <link href="http://purl.org/dc/elements/1.1/" rel="schema.dc"/>
  <link href="http://purl.org/dc/terms/" rel="schema.dcterms"/>
  <link href="http://www.gutenberg.org/ebooks/4300" rel="dcterms.isFo
 </head>
 <body>
  <h1>
   Ulysses
  </h1>
  <h2 class="no-break">
   by James Joyce
  </h2>
  <hr/>
  <div class="chapter">
   <h2>
    <a id="part01">
    </a>
    - I -
   </h2>
  </div>
  <p>
   <!--end chapter-->
  </p>
  <div class="chapter">
   <h3>
    <a id="chap01">
    </a>
    [ 1 ]
   </h3>
   <p>
    Stately, plump Buck Mulligan came from the stairhead, bearing a b
        on which a mirror and a razor lay crossed. A yellow dress
        was sustained gently behind him on the mild morning air. H
        and intoned:
   </p>
```

Обе функции находят свое применение.

Работа с элементами
Есть две функции:

одинарный поиск – **find()**, работает до первого срабатывания (до первого поиска),

поиск всех – **find_all()**, возвращает все элементы, которые подходят введенным условиям.

Условия можно записать по-разному. При обработке HTML чаще всего говорят об обработке и поиске каких-то тегов.

Варианты поиска:

- Тег как атрибут: **soup.p**
- тег как аргумент поиска: **soup.find_all('p')**
- поиск по аргументам:

```
soup.find_all(
    attrs={
        "name": "email"
    }
)
```

2. Сложный поиск и изменение с Beautiful Soup

Поработаем с библиотекой Beautiful Soup и попробуем применить её методы к реально существующей странице. В качестве примера возьмем сайт МФТИ, раздел новости. Посчитаем, сколько предложений на странице. Будем рководствоваться идеей, что предложение – любой набор символов, разделенный точкой (упрощение задачи). Для получения информации, сколько предложений, нужно извлечь текст, выкинуть из него все теги и служебные пометки.

```
from bs4 import BeautifulSoup
import requests

base_url = "https://mipt.ru/news/"
response = requests.get(base_url)
if response.status_code == 200:
      soup = BeautifulSoup(response.text, "html.parser")
      # сколько предложений?
      news_text = soup.get_text()
      print(len(news_text.split(".")))
else:
```

```
      print(“request error”)
```

>>> 118

Чтобы взаимодействовать с интернет-страницей, можно открыть консоль. Для этого нажать правой кнопкой мыши по любому месту на странице и нажать “inspect” или “просмотреть код”.

Если развернуть на странице МФТИ теги, то можно увидеть много тегов <li> (элемент списка – list-item)). Посчитаем, сколько всего таких элементов есть. Класс в рамках HTML верстки – нечто, называемое “атрибут”. Атрибуты со значениями показывают, какая записана информация. Они предоставляют некоторое дополнительное описание. Использование классов – частая методика в современном фронтенде для того, чтобы разграничивать разные элементы между собой. Часто классы разграничивают то, какие стили применяются к тому или иному элементу.

В нашем примере классов и элементов много. Нужно разобраться, сколько их. Нужно подумать, какой метод нужно использовать.

```
from bs4 import BeautifulSoup
import requests

base_url = “https://mipt.ru/news/”
response = requests.get(base_url)
if response.status_code == 200:
      soup = BeautifulSoup(response.text, “html.parser”)
      # сколько предложений?
      news_text = soup.get_text()
      print(len(news_text.split(“.”)))
      # сколько элементов меню?
      menu_items = soup.find_all(“li”, attrs={“class”: “list-item”})
      print(f“На странице {len(menu_items)} элементов меню”)
else:
      print(“request error”)
```

Сконцентрируемся теперь на новостных сюжетах, которые есть на странице нашего сайта https://mipt.ru/news/ . Консоль браузера позволяет посмотреть, как сверстан тот или иной элемент. Мы можем собрать частотный словарь, в котором ключами будут месяцы выхода статьи, а значениями будут все ссылки на статьи, которые выходили в этом месяце. Обратим внимание, что дата и ссылка находятся на одном уровне вложенности. Надо собрать все элементы уровнем выше. А затем у каждого из них можно найти дату и ссылку.

```
from bs4 import BeautifulSoup
```

```
import requests

base_url = “https://mipt.ru/news/”
response = requests.get(base_url)
if response.status_code == 200:
      soup = BeautifulSoup(response.text, “html.parser”)
      # сколько предложений?
      news_text = soup.get_text()
      print(len(news_text.split(“.”)))
      # сколько элементов меню?
      menu_items = soup.find_all(“li”, attrs={“class”: “list-item”})
      print(f“На странице {len(menu_items)} элементов меню”)
      # словарь: ключ – месяц, значение – список ссылок со статьями в
      этом месяце
      monthly_articles = {}
      for a_title in soup.find_all(“div”, attrs={“class”: “title-block”
      }):
            row_date_str = a_title.find(“span”, attrs={“class”:
            “date”}).get_text().strip(“\n”)
            try:
                  month =”.”.join(row_date_str.split(“.”)[1:])
                  link = a_title.find(“a”)
                  if link.has_attr(“href”):
                        href = link[“href”]
                        if month in monthly_articles.keys():
                              monthly_articles[month].append(href)
                        else:
                              monthly_articles[month] = []
                              monthly_articles[month].append(href)
            except:
                  print(“Error in: {month}”)
      for key in monthly_articles:
            print(key)
            print(“\n”.join(monthly_articles[key]))
else:
      print(“request error”)
```

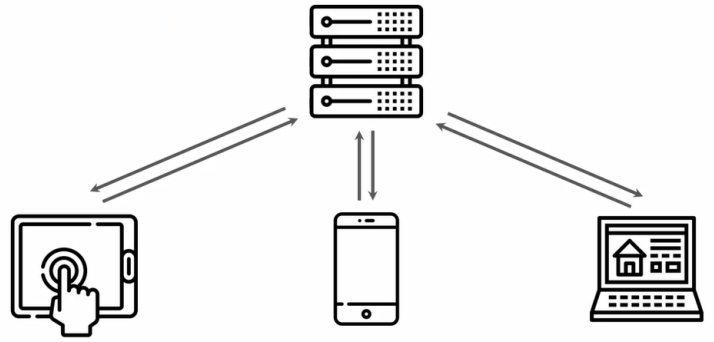
Когда мы что-то получаем из web, нам не гарантирована консистентность данных. Где-то возможно не так написана дата, где-то не так записан атрибут, где-то поехала верстка, и т.д. Поэтому применяем оператор **try except**. Но за счет количества мы можем пренебречь случаем какой-то некорректной верстки.

3. Взаимодействие с API

Зачем нужен API

Ранее мы получали HTML код и каким-то образом его обрабатывали. Рассмотрим новый способ передачи данных.

Повторим, как работает клиент-серверная архитектура.



Есть много разных устройств: мобильные телефоны, планшеты, компьютеры, умные чайники, умные пылесосы, и т.д. – множество других приборов, у которых есть выход в интернет. Все они одинаково отправляют запросы серверу и получают какой-то ответ. Контент, который мы получаем от сервера, может быть очень разным:

- Заголовки
- Контент:
    - HTML-вёрстка
    - CSS-таблицы
    - статический контент (картинки)
    - скрипты (javascript)

Весь контент требует времени на загрузку. Рассмотрим два термина:

- **время загрузки (Load time)** – время, которое затрачивается на то, чтобы получить весь контент от сервера,
- **время рендера (Render time)** – сколько времени нужно, чтобы отрисовать все элементы и «собрать» страницу из загруженного контента,
- **time to interaction** – сколько времени нужно от момента, как пользователь нажал на ссылку, до момента, когда пользователь может взаимодействовать с сайтом.

Эти времена могут кардинально отличаться, особенно если сайт плохо оптимизирован.

Умным устройствам не принципиально, что там за картинки, скрипты на сайте. Им важно получить только данные.

Для обмена данными придуман интерфейс API.

API = Application Programming Interface – интерфейс, который по сути доступен приложениям.

- API нужен, чтобы получить данные без графической обёртки;
- сохраняются базовые типы данных: числа, строки, списки;
- данные сохраняются в формате JSON .

Как обратиться к API:

- из браузера,
- отправить запрос из скрипта,
- клиенты типа Swagger и Postman,
- командная строка (утилиты **curl** и **wget**).

Ответ будет в формате JSON.

Формат JSON

JSON = JavaScript Object Notation. В JavaScript есть такой тип данных как объект. В Python к нему ближе всего тип данных словарь. Объект в JavaScript примерно объясняется, как набор некоторых пар из описания и значения этого описания (аналог – ключ-значение).

- JSON – строковый формат обмена данными
- легко читается людьми. Сравните с HTML:

```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Konfabulator \
        "name": "main_window",
        "width": 500,
        "height": 500
    },
    "image": {
        "src": "Images/Sun.png",
        "name": "sun1",
        "hOffset": 250,
        "vOffset": 250,
        "alignment": "center"
    },
```

```
    "text": {
        "data": "Click Here",
        "size": 36,
        "style": "bold",
        "name": "text1",
        "hOffset": 250,
        "vOffset": 100,
        "alignment": "center",
        "onMouseUp": "sun1.opacity = (
    }
}}
```

- На данный момент JSON – де факто стандарт обмена данными в веб-разработке
- В Python для обработки JSON файлов есть специальная системная библиотека **json.**

Пример: агрегация погодных данных

Задача. Open-Meteo.com — открытое API для некоммерческого использования.

Запросим график погоды для Москвы (широта=55.75, долгота=37.62), с температурой по часам, с 1 по 7 января 2023 года.

```
import requests
weather =
requests.get("https://api.open-meteo.com/v1/forecast?latitude=55.75&longi
tude=37.62&hourly=temperature_2m&timezone=Europe%2FMoscow&start_date=2023
-01-01&end_date=2023-01-07")
```

Получаем температуру в градусах Цельсия.

```
{
  "latitude": 55.75,
  "longitude": 37.625,
  "generationtime_ms": 0.3120899200439453,
  "utc_offset_seconds": 10800,
  "timezone": "Europe/Moscow",
  "timezone_abbreviation": "MSK",
  "elevation": 152.0,
  "hourly_units": { "time": "iso8601", "temperature_2m": "°C" },
  "hourly": {
```

```
    "time": [
      "2023-01-01T00:00",
      "2023-01-01T01:00",
      "2023-01-07T20:00",
      --- тут были ещё часы ---
      "2023-01-07T21:00",
      "2023-01-07T22:00",
      "2023-01-07T23:00"
    ],

    "temperature_2m": [
      -0.2, 0.0, 0.2, 0.5, 0.7, 0.7, 0.8, 0.9, 0.9, 1.4, 1.6, 1.8, 2.1, 2.2,
      2.4, 2.6, 2.8, 3.0, 3.3, 3.5, 3.6, 4.7, 4.7, 4.9, 4.7, 4.2, 3.7, 3.5, 2.3,
      2.0, 1.7, 1.7, 1.0, 0.5, -0.6, -1.1, -1.3, -1.5, -1.7, -0.9, -1.6, -2.3,
      -2.9, -3.5, -4.0, -3.4, -3.9, -4.2, -4.3, -4.3, -4.3, -3.5, -3.4, -3.1,
      -2.8, -3.0, -2.8, -2.0, -1.6, -1.3, -1.1, -0.8, -0.5, 0.1, 1.6, 1.9, 2.1,
      2.1, 2.0, 2.5, 2.4, 2.3, 1.6, 0.9, 0.5, 0.7, 0.0, -1.8, -2.5, -2.8, -3.0,
      -3.1, -3.4, -3.7, -3.6, -3.9, -4.3, -4.2, -4.5, -4.4, -4.3, -4.1, -4.0,
      -3.1, -2.8, -2.5, -2.6, -2.6, -2.6, -2.3, -2.7, -3.1, -3.5, -3.8, -4.2,
      -4.7, -5.0, -5.7, -6.3, -6.7, -7.2, -7.2, -8.1, -9.0, -9.8, -10.9, -12.2,
      -13.2, -13.9, -15.0, -16.0, -17.1, -18.5, -19.7, -20.7, -21.7, -22.5,
      -23.2, -23.7, -24.2, -24.4, -24.2, -23.6, -23.1, -22.6, -20.8, -21.2,
      -21.6, -22.0, -22.4, -22.8, -24.0, -24.2, -24.5, -24.7, -24.8, -24.9,
      -24.9, -24.9, -24.8, -24.7, -24.6, -24.7, -26.3, -26.1, -25.7, -25.0,
      -24.2, -23.5, -20.7, -21.0, -21.4, -21.7, -21.8, -21.9, -22.1, -22.0,
      -21.9
    ]
  }
}
```


В полученном результате видим JSON с температурой, разбитой по часам. Агрегируем информацию по дням, чтобы было удобнее ориентироваться.

```
import json
weather_json = json.loads(weather.text)
weather_day = {f"2023-01-0{i}": [] for i in range(1, 8)}
for i, timestamp in enumerate(weather_json["hourly"]["time"]):
    day = timestamp[0:10]
    weather_day[day].append(weather_json["hourly"]["temperature_2m"][i])
```

```
for this_day in weather_day:
    this_day_temp = sum(weather_day[this_day])/len(weather_day[this_day])
```

```
    print(f"{this_day}\t{this_day_temp:.5f}")
```

Финальный код:

```
import json
import requests
weather =
requests.get("https://api.open-meteo.com/v1/forecast?latitude=55.75&longi
tude=37.62&hourly=temperature_2m&timezone=Europe%2FMoscow&start_date=2023
-01-01&end_date=2023-01-07")
weather_json = json.loads(weather.text)
weather_day = {f"2023-01-0{i}": [] for i in range(1, 8)}
for i, timestamp in enumerate(weather_json["hourly"]["time"]):
    day = timestamp[0:10]
    weather_day[day].append(weather_json["hourly"]["temperature_2m"][i])
for this_day in weather_day:
    this_day_temp = sum(weather_day[this_day])/len(weather_day[this_day])
    print(f"{this_day}\t{this_day_temp:.5f}")
```

4. Практический пример работы с API

Возьмем в качестве примера API, который позволяет нам узнавать разные новости. Он является бесплатным, но для его использования нужно получить ключ. Ключ или токен – это чаще всего какая-то текстовая переменная, которая позволяет идентифицировать пользователя. Эти ключи строго индивидуальны. Делиться ключом не безопасно. Хорошая практика сохранять свой ключ в отдельный файл в Python. Значение ключа подтягиваем следующим образом:

```
from api_key import key
```

Запрос Everything

Рассмотрим методы в API.

Метод **everything**. Позволяет искать названия статей и их содержимое из 80000 источников и блогов. Обязательным параметром является ключ. Изучить подробнее можно в документации.

Попробуем поискать что-то, связанное с тюльпанами в заголовке или в тексте статьи. Ответ в API приходит в формате json.

```
from api_key import key
import json
import requests

base_url = “https://newsapi.org/v2”
path_everything = f”{base_url}/everything”
params_everything = {
      “apiKey”: key,
      “q”: “tulip”,
      “searchIn”: “title,content”
}
response = requests.get(path_everything, params=params_everything)
articles_dict = json.loads(response.text)
print(len(articles_dict))
```

Сами статьи, выведенные методом everything, являются списком, в котором много словарей, каждый из которых описывает свою статью. Посмотрим, что находится внутри этих статей. Для каждой статьи будем печатать ее заголовок.

```
from api_key import key
import json
import requests

base_url = “https://newsapi.org/v2”
path_everything = f”{base_url}/everything”
params_everything = {
      “apiKey”: key,
      “q”: “tulip”,
      “searchIn”: “title,content”
}
response = requests.get(path_everything, params=params_everything)
articles_dict = json.loads(response.text)
for article in articles_dict[“articles”]:
      print(article[“title”])
```

Теперь для каждой статьи выведем некоторую справку:

- название,
- ссылка на статью,
- количество символов в статье

Контент в API мы получаем не полностью. Выводится начало статьи, а потом в скобках указывается, сколько еще символов есть в этой статье. Воспользуемся регулярным выражением, чтобы вытащить число символов и кусок статьи (ее начало). В текстовом куске нужно посчитать количество символов. Для написания регулярного выражения зайдем на сайт regex101.

Составим регулярное выражение:

```
r"(.+)...\[\+(\d+)chars\]
```

Вернемся к нашему коду:

```
from api_key import key
import json
import re
import requests

base_url = "https://newsapi.org/v2"
path_everything = f"{base_url}/everything"
params_everything = {
      "apiKey": key,
      "q": "tulip",
      "searchIn": "title,content"
}
response = requests.get(path_everything, params=params_everything)
articles_dict = json.loads(response.text)
reg_article_desc = re.compile("(.+)...\[\+(\d+)chars\]")
for article in articles_dict["articles"]:
      title = article["title"]
      url = article["url"]
      content_raw = article["content"]
      content_parts = re.search(reg_article_desc, content_raw)
      if content_parts:
            desc = content_parts.group(1)
            char_count = content_parts.group(2)
            article_length = int(char_count) + len(desc)
      else:
            article_length = len(content_raw)
      print(title, url, article_length, sep="\n")
      print("===")
```

Стоит обратить внимание, что запрос мы отправили один раз. Дальше мы работали с его результатом. Это важно, потому что у некоторых API есть ограничение на количество запросов.

Запрос Top headlines

Рассмотрим запрос Top headlines. Он позволяет получить самые популярные заголовки для какой-то страны. Также содержит обязательный параметр apiKey. Более подробную информацию о запросе можно посмотреть в документации.

Возьмем список англоязычных стран, стран британского содружества, и при помощи запроса Top headlines посмотрим, какие источники предоставляют топовые заголовки.

```
from api_key import key
import json
import re
import requests

base_url = "https://newsapi.org/v2"
countries_sources = {cnt: set() for cnt in ["gb", "us", "au", "nz", "in",
"ca"]}
all_countries_sources = set()
path_headlines = f"{base_url}/top-headlines"
for cnt in countries_sources:
      params_headlines = {
            "apiKey": key,
            "country": cnt
      }
      response = requests.get(path_headlines, params=path_headlines)
      cnt_top = json.loads(response.text)
      cnt_sources = set()
      for article in cnt_top["articles"]:
            if article["source"]["name"] is not None:
                  cnt_sources.add(article["source"]["name"])
      all_countries_sources.update(cnt_sources)
      print(cnt)
      print(", ".join(sorted(cnt_sources)))
print("All countries resources: ", ",
".join(sorted(all_countries_sources)))
```

Мы снова делали запрос только один раз на каждую страну. То есть всего 6 запросов.

Дополнительные материалы для самостоятельного изучения

1. https://www.gutenberg.org/ebooks/4300
2. https://open-meteo.com/
3. https://mipt.ru/news/
4. https://newsapi.org/
5. https://newsapi.org/docs/endpoints/everything

6. https://regex101.com/
7. https://newsapi.org/docs/endpoints/top-headlines