**Методические указания к лабораторной работе № 1**

**Автор: Якунин Е.М.**

**Парсинг**

Данная лабораторная работа посвящена обучению сбора больших объемов данных при помощи парсинга.

**Цель работы:** узнать, что такое «Парсинг», и получить навыки его использования.

**Задачи:**

1. Получить теоретические знания по теме.
2. Написать простой парсер сайта новостей.
3. Применить полученный парсер.

**Парсинг**

**Парсинг** — это процесс автоматического сбора данных и их структурирование. Собранные данные чаще всего структурируют в виде таблицы. Парсинг работает на основе XPath-запросов, это язык, который обращается к определенному участку кода страницы и извлекает из него заданную критерием информацию.

**Парсер сайта** — это любая программа или сервис, которые осуществляют автоматический сбор информации с заданного ресурса. То есть парсер «ходит» по ссылкам указанного сайта и сканирует код каждой страницы, собирая необходимую информацию. Полученная информация со всех страниц сайта будет итогом парсинга сайта.

Пример: допустим, нужно собрать название, описание и цену ноутбука с веб-магазина электроники. Можно это сделать вручную, что займет у вас много времени, особенно, если на сайте представлены сотни моделей товара. Но процесс можно автоматизировать. Задать программе для парсинга условия выборки и через какое-то время получить готовую таблицу с данными.

**Зачем парсинг нужен и где его используют**

Если не углубляться в подробности, то парсинг можно разделить на 2 типа:

1. Технический парсинг сайта, который помогает находить различные проблемы сайта. Используя полученные данные, специалист составляет технические задания для устранения выявленных проблем. Например:
   * Поиск битых ссылок.
   * Сбор данных для обучения нейронных сетей.
   * Выявление дублей или других проблем с мета-тегами.
   * Обнаружение нежелательных страниц.
2. Парсинг сайта с целью развития бизнеса. Например:
   * Сбор информации об ассортименте конкурентов.
   * Парсинг названий товаров, цен и прочего для наполнения своего собственного интернет-магазина.
   * Анализ структуры сайтов-конкурентов.

**Плюсы парсинга**

Сравнивая парсинг с ручным трудом, можно выделить следующие плюсы:

* Скорость получения данных
* Множество параметров выборки
* Отсутствие ошибок в отчете (исключение человеческого фактора)
* Возможность настройки периодичности работы

В интернете предостаточно программ для парсинга. Они могут находиться в «облаке» или «коробке».

**Законно ли парсить сайты**

Парсинг данных не преследуется законом, если:

* Получаемая информация находится в открытом доступе и не несет коммерческую или государственную тайну.
* Не затрагиваются авторские права полученной информации.
* Парсинг не влияет на нормальную работу сайта (DDOS-атаки)

**Пишем парсер сайта новостей**

Для написания парсера сайта используем популярный язык программирования «Python» и среду разработки «PyCharm». Для этого не требуются глубокие знания языка, достаточно иметь базовые знания синтаксиса. Для выполнения задачи нам понадобятся библиотеки: BeautifulSoup, Requests.

**BeautifulSoup** − это библиотека Python для извлечения данных из файлов HTML и XML, анализа и преобразования их в синтаксические деревья. Она использует HTML и XML-парсера, такие как html5lib и Lxml, чтобы извлекать нужные данные.

**Requests** − это модуль Python, предназначенный для работы с HTTP-запросами.

В качестве примера будем парсить сайт новостей: «Городские заметки», раздел спорт (<https://gzt.ru/news/sport/>). Нам нужно будет спарсить заголовок, дату публикации, ссылку и текст на данную новость. Данный сайт выглядит следующим образом (рис.1):

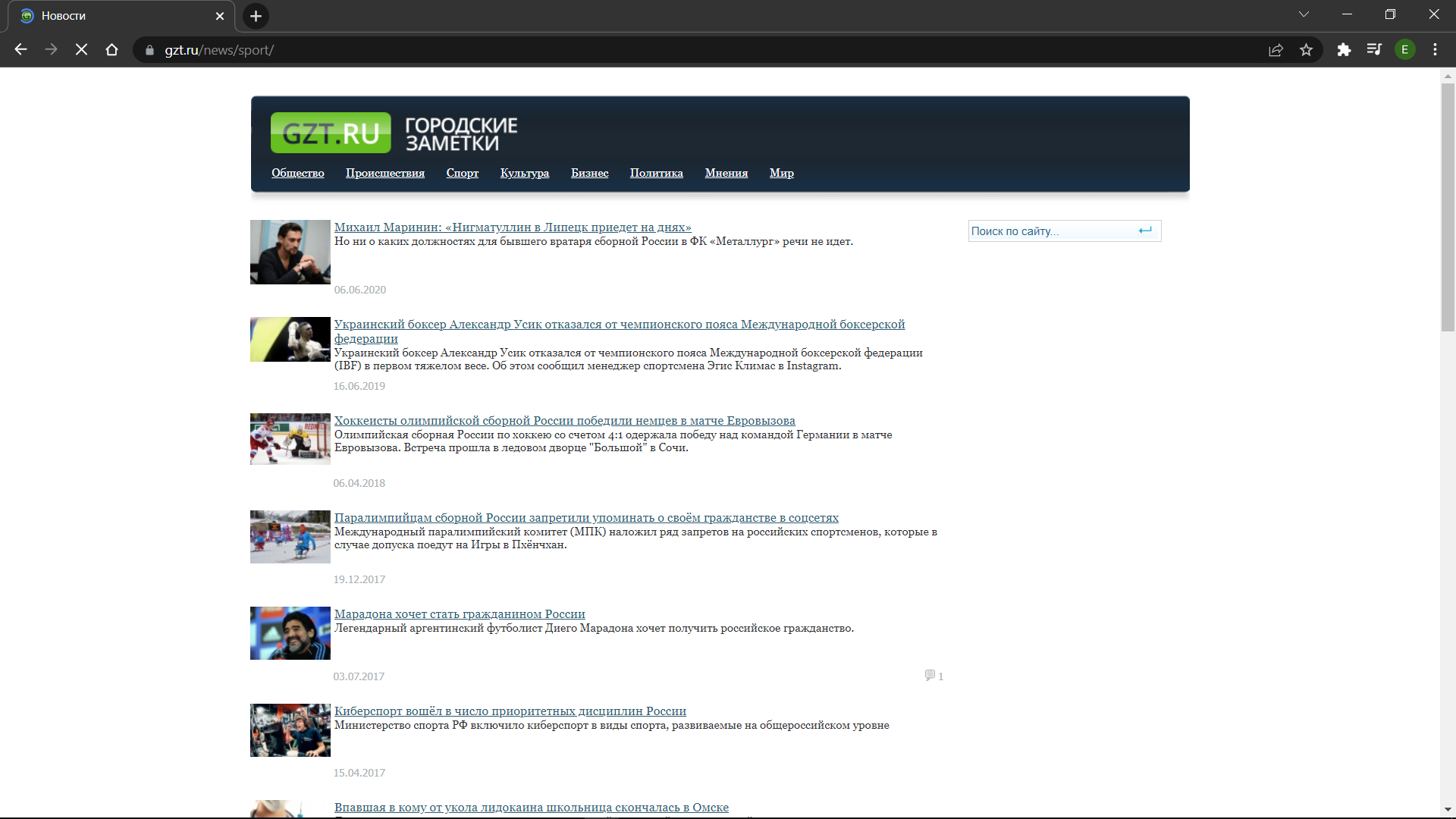


Рисунок 1 − Городские заметки, новости спорта

Когда определились, что мы будем парсить, перейдем к написанию нашего парсера. Для начала импортируем необходимые модули и зададим константы.

from bs4 import BeautifulSoup  
import requests as req  
import csv  
  
URL = "https://gzt.ru/news/sport/"  
HEADERS = {  
 'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36',  
 'accept': '\*/\*'}  
  
HOST = "https://gzt.ru"  
FILE = 'News.csv'

В константу «URL» заносим ссылку страницы, которую мы собираемся спарсить. «HEADERS» необходимо занести значения ключей user-agent и accept, чтобы сервер не посчитал, что мы боты, и не заблокировал нас. Для получения этих значений откройте сайт, нажмите правую кнопку мыши и выберите «посмотреть код» (вариант для google chrome) или просто нажмите «F12». В разделе «сеть», «заголовки запроса» можете найти данные ключи (рис. 2). «HOST» понадобится, чтобы парсить ссылку на новость. В константе «FILE» пишем название нашего csv файла, в который мы сохраним полученные данные.

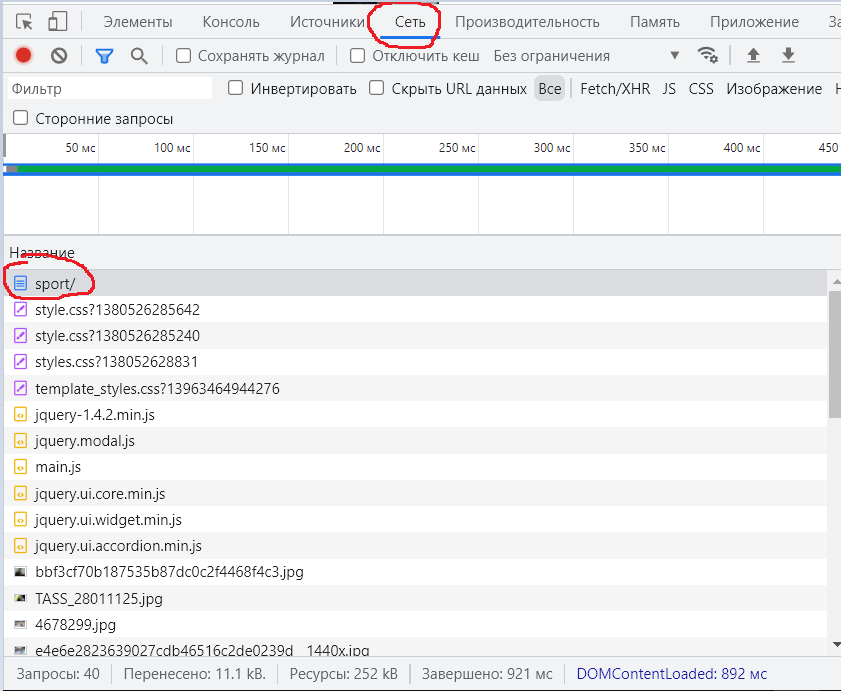
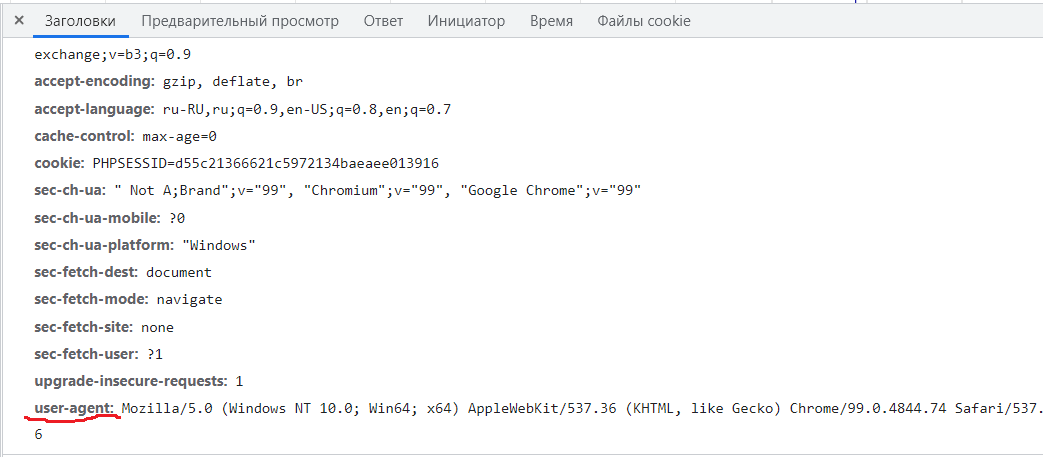
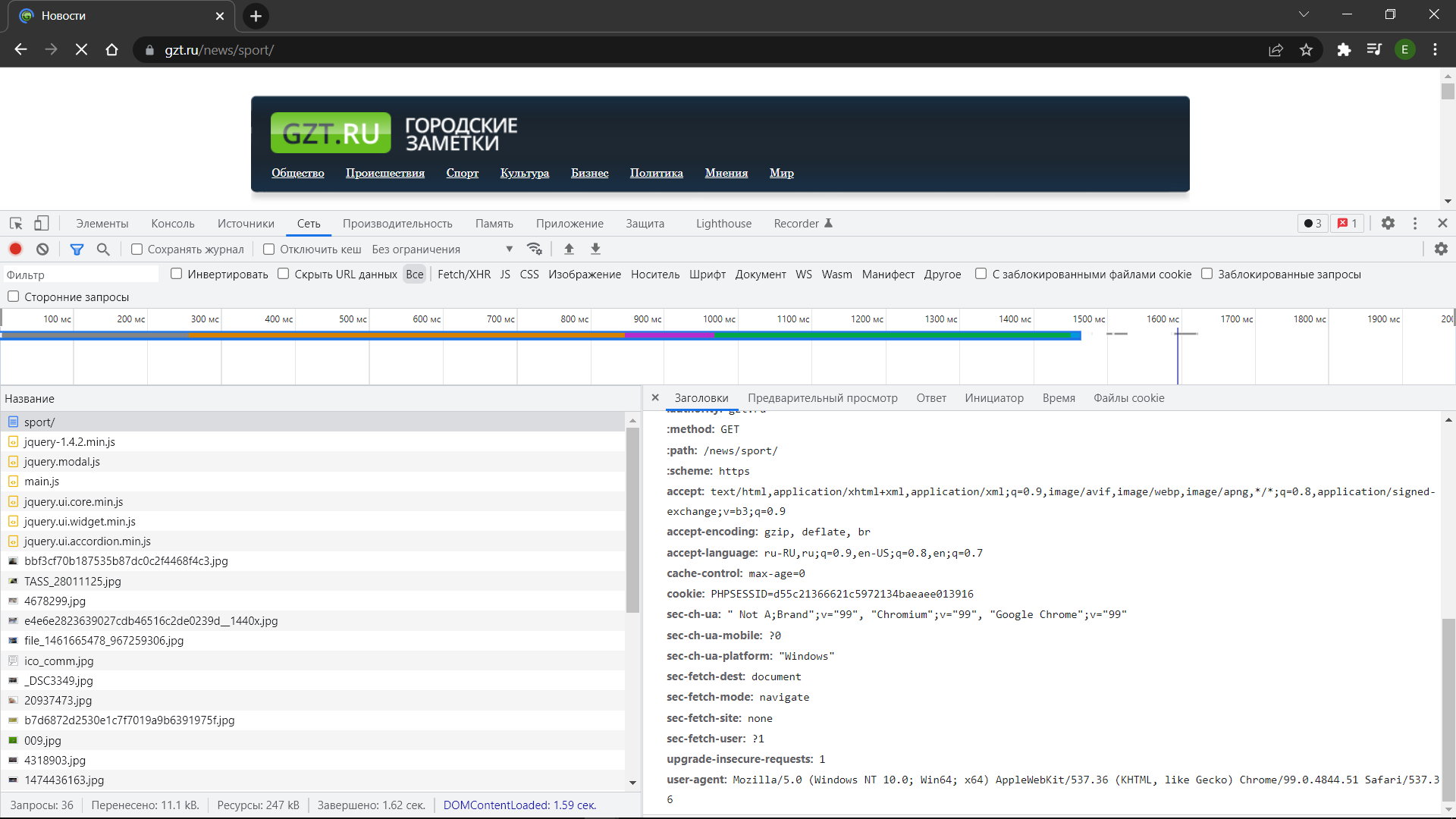


Рисунок 2 − Значения HEADERS

Для начала создадим функцию *get\_html()*, чтобы выполнять Get-запрос с помощью библиотеки requests. Функцию parse(), в которой будет выполняться вся программа.

def get\_html(url, params=None):  
 r = req.get(url, headers=HEADERS, params=params)  
 return r  
  
  
def parse():  
 pass  
  
  
parse()

Для непосредственного парсинга страницы создадим функцию *get\_content()*.

def get\_content(html):  
 soup = BeautifulSoup(html, 'html.parser')  
 items = soup.find\_all('div', class\_="news-item")  
  
 news = []  
 for item in items:  
 # создаем переменную и присваиваем ей ссылку новости  
 link\_news = HOST + item.find('a', class\_="title").get('href')  
  
 # получаем код страницы и парсим текст новости  
 link = get\_html(link\_news)  
 soup\_news = BeautifulSoup(link.text, 'html.parser')  
 # возможно 2 варианта тега div, поэтому проверяем через условие  
 text\_page = soup\_news.find('div', class\_='detail')  
 if text\_page:  
 text = text\_page.get\_text(strip=True)  
 else:  
 text = soup\_news.find('div', class\_='hidden').get\_text(strip=True)  
 #print(test1)  
  
 news.append({  
 'title': item.find('a', class\_="title").get\_text(strip=True),  
 'date': item.find('div', class\_='news-date-time').get\_text(),  
 'link': link\_news,  
 'text': text  
 })  
 return news

В качестве аргумента передаем html код страницы, которые нам поможет распарсить модуль BeautifulSoup. Полученный html код распарсенной страницы присваиваем переменной soup. Далее нам нужно найти все теги, в которых содержатся новости, из которой нам нужно будет вычленить интересующие нам данные (рис. 3). Используем метод find\_all(ищем тег ‘div’, с классом ‘news-item’) и заносим данные в переменную items. После чего создаем пустой список news, куда будем заносить наши спарсенные данные. В начале цикла получаем ссылку новости и присваиваем переменной. Далее уже знакомые функции для получения кода страницы и парсинга страницы, откуда получим текст новости. На страницах новостей возможно 2 тега div с разными классами (рис. 4), в которых содержится текст. Через условие проверяем наличие первого, если не находим, то берем текст из второго. Добавляем данные в наш список. В конце возвращаем полученные данные.

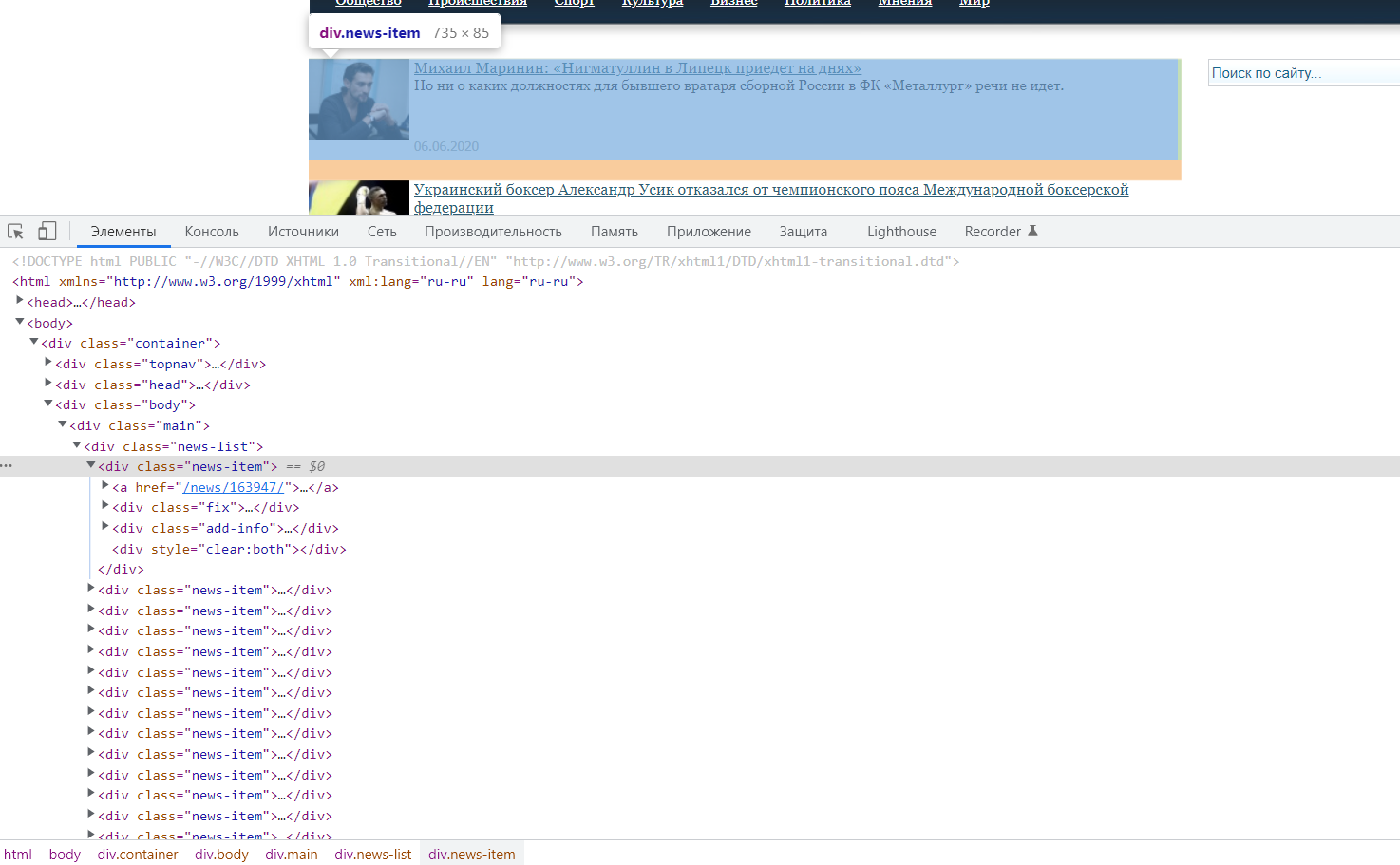


Рисунок 3 − Тег, содержащий новость

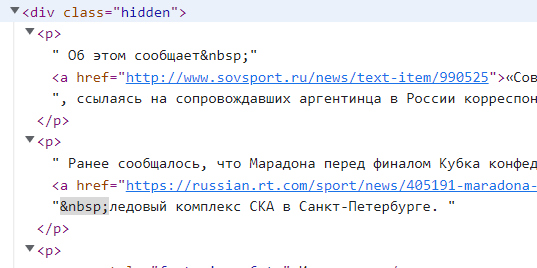
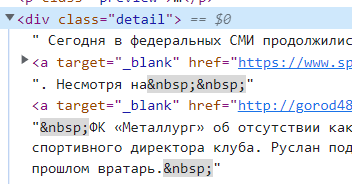


Рисунок 4 − Теги, содержащие текст новости

Теперь напишем функцию *get\_pages\_count()*, которая будет проверять количество страниц с новостями в данном разделе.

def get\_pages\_count(html):  
 soup = BeautifulSoup(html, 'html.parser')  
 page = soup.find\_all('font', class\_='text')  
 page1 = page[-1].find\_all('a')  
 pagination = page1[-2].get('href')  
 if pagination:  
 try:  
 return int(pagination[-2:])  
 except Exception:  
 return int(pagination[-1])  
 else:  
 return 1

Так как на сайте не очень удобно реализовано отображение страниц, то придется выбирать поэлементно данные. Для начала применяем уже знакомую библиотеку BeautifulSoup. Находим все теги font, в которых находится отображение страниц (рис. 5). Выбираем последний тег из списка и предпоследний элемент ‘a’ с получением ссылки. Далее мы проверяем условие наличия значений. В случае «True», с помощью try except выбираем 2 последний значения, если страниц ≥ 10, если страниц меньше 10, то в случае ошибки выбираем только последнее значение. Если условие не проходит, то просто возвращаем единицу.

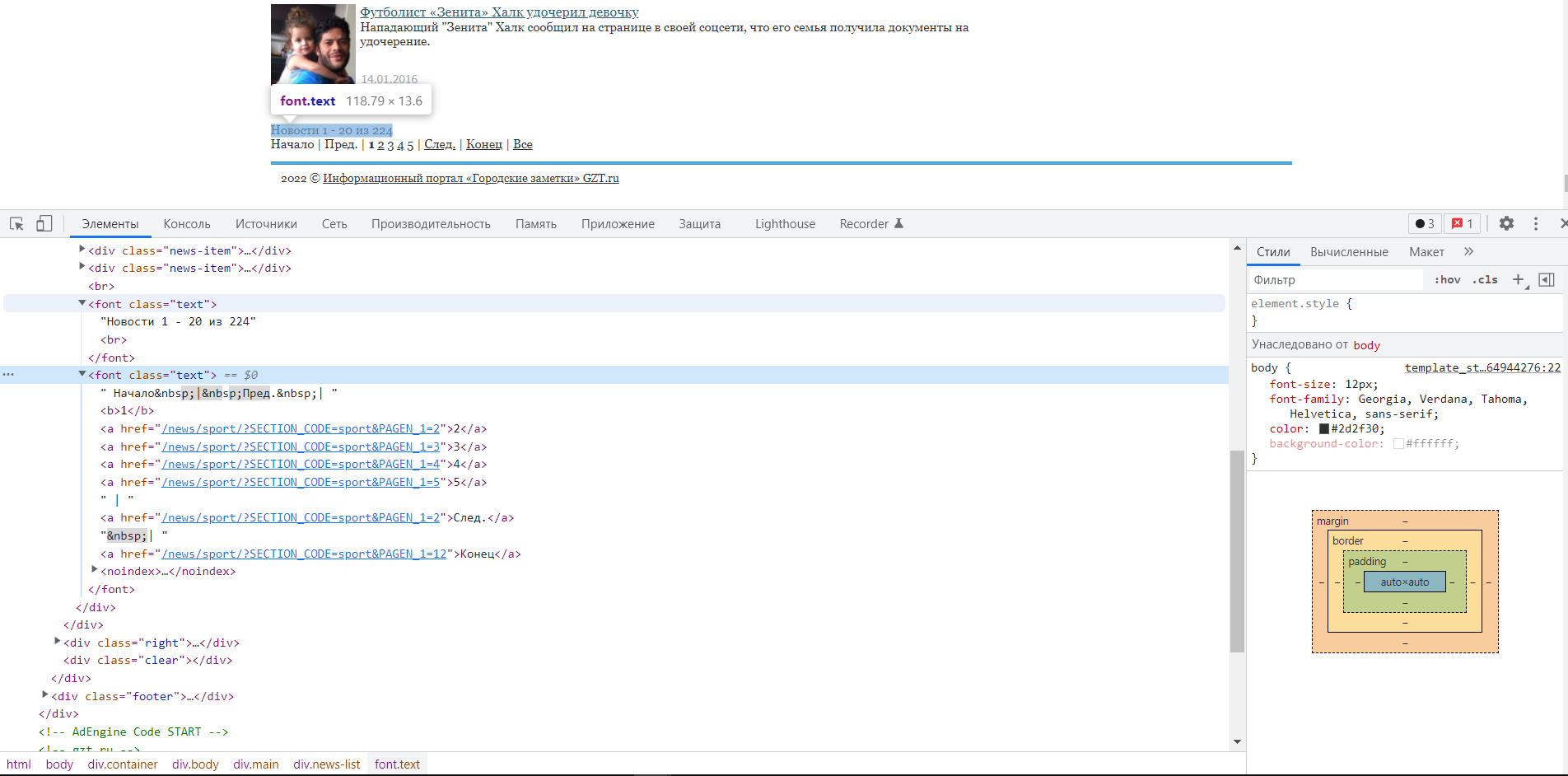


Рисунок 5 − Теги, содержащие информацию о страницах

Теперь создадим функцию *save\_file()*, которая будет сохранять наши данные в файл csv. На ней мы не будем подробно останавливаться, можете разобраться самостоятельно. Единственное уточнение, используем кодировку «utf-8-sig», так как excel файл может не читать определенные символы, что приведет к отображению нечитаемых символов.

def save\_file(items, path):  
 with open(path, 'w', newline='', encoding='utf-8-sig') as file:  
 writer = csv.writer(file, delimiter=';')  
 writer.writerow(['Заголовок', 'Дата', 'Ссылка', 'Текст'])  
 for item in items:  
 writer.writerow([item['title'], item['date'], item['link'], item['text']])

Осталось только прописать порядок действий для функции *parse()*, которую мы определили в самом начали, используя написанные функции до этого.

Итоговый код выглядит следующим образом:

from bs4 import BeautifulSoup  
import requests as req  
import csv  
  
URL = "https://gzt.ru/news/sport/"  
HEADERS = {  
 'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844.51 Safari/537.36',  
 'accept': '\*/\*'}  
  
HOST = "https://gzt.ru"  
FILE = 'News.csv'  
  
  
def get\_html(url, params=None):  
 r = req.get(url, headers=HEADERS, params=params)  
 return r  
  
  
def get\_pages\_count(html):  
 soup = BeautifulSoup(html, 'html.parser')  
 page = soup.find\_all('font', class\_='text')  
 page1 = page[-1].find\_all('a')  
 pagination = page1[-2].get('href')  
 if pagination:  
 try:  
 return int(pagination[-2:])  
 except Exception:  
 return int(pagination[-1])  
 else:  
 return 1  
  
  
def get\_content(html):  
 soup = BeautifulSoup(html, 'html.parser')  
 items = soup.find\_all('div', class\_="news-item")  
  
 news = []  
 for item in items:  
 # создаем переменную и присваиваем ей ссылку новости  
 link\_news = HOST + item.find('a', class\_="title").get('href')  
  
 # получаем код страницы и парсим текст новости  
 link = get\_html(link\_news)  
 soup\_news = BeautifulSoup(link.text, 'html.parser')  
 # возможно 2 варианта тега div, поэтому проверяем через условие  
 text\_page = soup\_news.find('div', class\_='detail')  
 if text\_page:  
 text = text\_page.get\_text(strip=True)  
 else:  
 text = soup\_news.find('div', class\_='hidden').get\_text(strip=True)  
 #print(test1)  
  
 news.append({  
 'title': item.find('a', class\_="title").get\_text(strip=True),  
 'date': item.find('div', class\_='news-date-time').get\_text(),  
 'link': link\_news,  
 'text': text  
 })  
 return news  
  
  
def save\_file(items, path):  
 with open(path, 'w', newline='', encoding='utf-8-sig') as file:  
 writer = csv.writer(file, delimiter=';')  
 writer.writerow(['Заголовок', 'Дата', 'Ссылка', 'Текст'])  
 for item in items:  
 writer.writerow([item['title'], item['date'], item['link'], item['text']])  
  
  
def parse():  
 html = get\_html(URL)  
 if html.status\_code == 200:  
 news = []  
 pages\_count = get\_pages\_count(html.text)  
 for page in range(1, pages\_count + 1):  
 print(f"парсим страницу {page} из {pages\_count}...")  
 html = get\_html(URL, params={'PAGEN\_1': page})  
 news.extend(get\_content(html.text))  
 save\_file(news, FILE)  
 print(f"Получено {len(news)} новостей")  
 else:  
 print("Сайт недоступен")  
  
  
parse()

Стоит уточнить про params для метода *get\_html()*. Передаем параметр в виде словаря, ключ и значение. Ключ − это параметр, используемый для 2, 3, 4 и т.д. страниц (рис. 6).

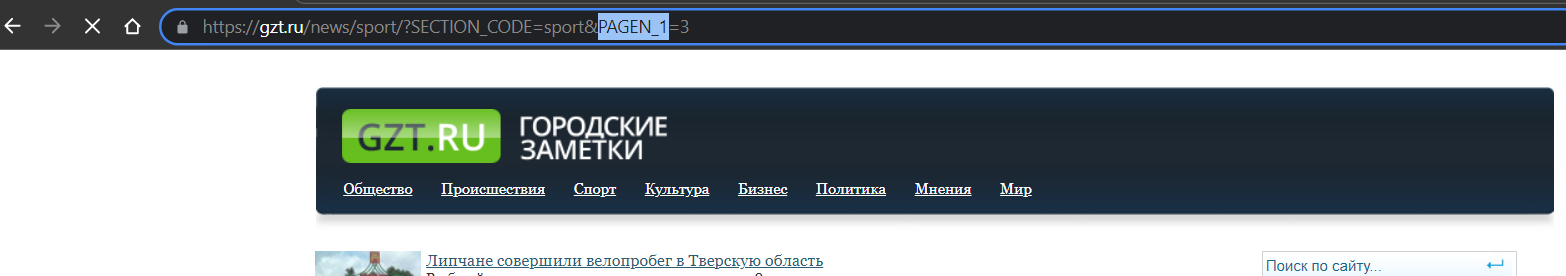


Рисунок 6 − Параметр страницы

В результате мы спарсили данные с сайта новостей и занесли их в csv файл (рис. 7, 8).

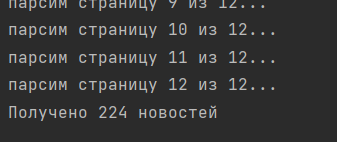


Рисунок 7 − Результат работы

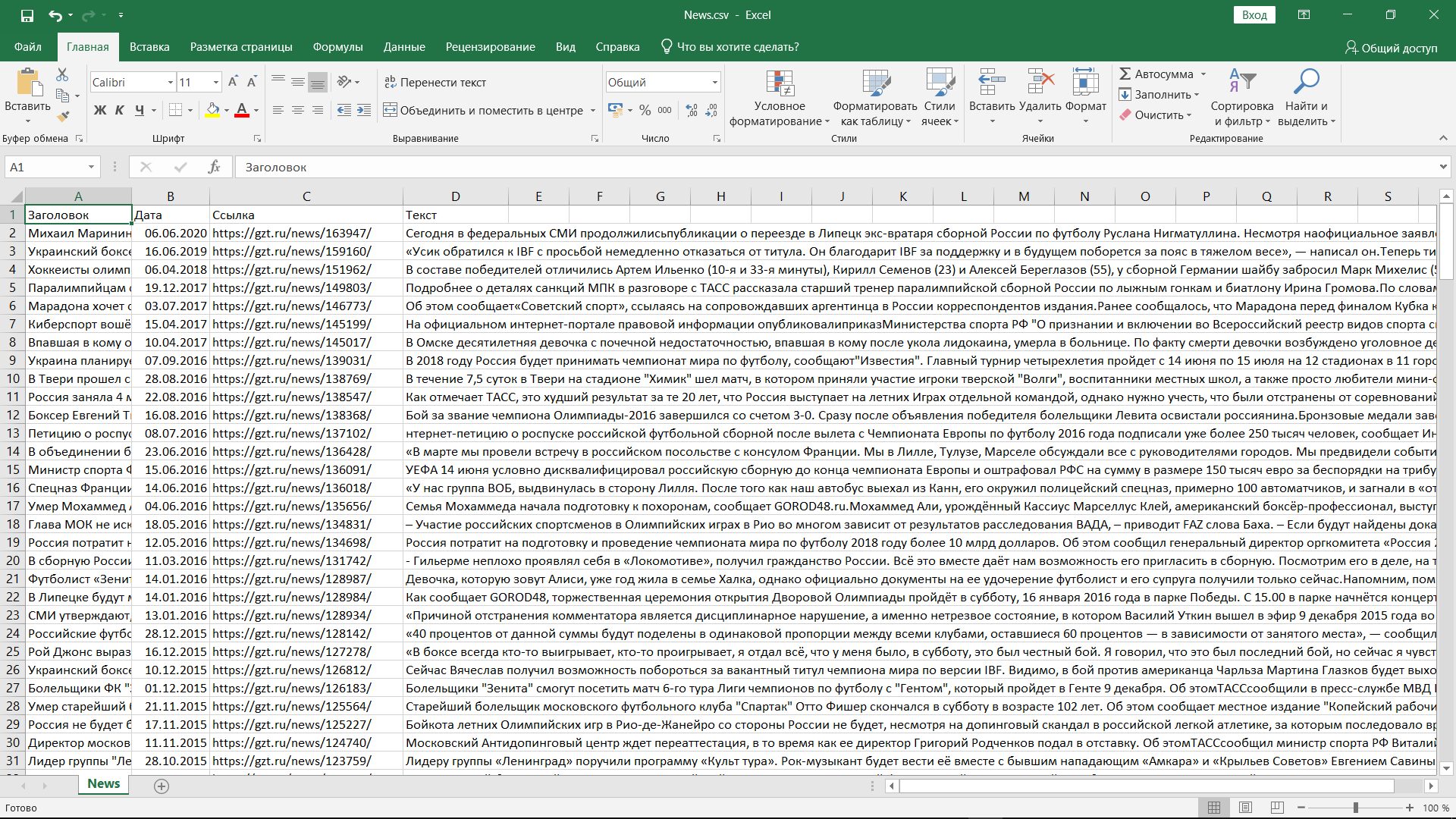


Рисунок 8 − News.csv

Вы можете немного доработать этот код, чтобы файл с данными открывался сразу после выполнения программы.

**Вывод**

В результате выполнения данной лабораторной работы познакомились с библиотеками Requests и BeautifulSoup, спарсили данные с сайта новостей и поместили их в csv файл.

**Критерии оценки**

…