ПРИЛОЖЕНИЕ №4

**Создание парсера новостных порталов и анализ полученных данных**

**Основы языка программирования Python**

В данной инструкции вы освоите основы программирования на Python, что позволит вам создавать специализированные парсеры для различных нужд, например, для автоматического сбора и анализа объявлений на Avito или для агрегирования новостей и статей из веб-ресурсов. Данный раздел предназначен для студентов факультета журналистики и познакомит с основами программирования, необходимыми для реализации подобных проектов.

**1.1 Переменные в Python**

Переменная в языке программирования Python представляет собой именованную область памяти, в которой хранится значение. Для работы с переменными необходимо понять несколько ключевых понятий: типы данных, циклы, библиотеки и взаимодействие с ними.

Переменные в Python создаются путем присваивания значения с помощью оператора =. Рассмотрим пример создания переменной:

cat = "Boris"

cat\_name = "Boris"

CatName = "Boris"

В данном примере мы создали три переменные, которые хранят одно и то же значение — строку "Boris". Название переменной всегда начинается с латинской буквы, после чего могут следовать буквы, цифры или символы подчеркивания. Важно помнить, что переменные чувствительны к регистру, поэтому cat\_name и Cat\_Name будут восприниматься как разные переменные.

Таким образом, переменная cat будет хранить значение "Boris" до тех пор, пока оно не будет изменено или программа не завершит свое выполнение.

**1.2 Циклы в Python**

Циклы — это конструкции, позволяющие выполнять набор инструкций несколько раз, до тех пор, пока выполняется заданное условие. В Python существуют два основных типа циклов: **while** и **for.**

Цикл **while** используется для выполнения инструкций до тех пор, пока условие истинно. Пример синтаксиса цикла **while**:

**while условие**:

последовательность инструкций

Важно соблюдать отступы: тело цикла должно начинаться с отступа в четыре пробела или один табулятор. Несоблюдение отступов приведет к ошибкам в коде, так как инструкции, не включенные в тело цикла, будут выполнены только один раз.

Цикл **for** используется для итерации по последовательности (например, списку или строке) и имеет следующий синтаксис:

**for элемент in последовательность**:

последовательность инструкций

Примеры использования циклов будут рассмотрены в процессе создания парсера.

**1.3 Типы данных в Python**

В Python существует несколько основных типов данных, которые необходимо знать для эффективного программирования. Важнейшие из них: **int, float и str.**

Тип int представляет целые числа и записывается без кавычек, например: 1, 4, 8, 50.

Тип float представляет числа с плавающей точкой, например: 1.2, 34.76, 382.9. Записывается также без кавычек, разделитель — точка.

Тип str представляет строки — последовательности символов, записываемые в одинарных или двойных кавычках, например: "Pasha".

Примеры присваивания значений переменным различных типов данных:

age = 21 # тип int

height = 1.92 # тип float

name = "Pasha" # тип str

В программировании символ # используется для обозначения комментариев внутри исходного кода. Все символы, следующие за символом # игнорируются интерпретатором или компилятором и служат исключительно для информирования разработчика.

Таким образом, переменные могут хранить различные типы данных, что позволяет гибко использовать их в программе.

**1.4 Структуры данных в Python**

Помимо простых типов данных, в Python также существуют более сложные структуры данных, такие как списки, кортежи и словари. Они позволяют хранить и управлять коллекциями данных.

Списки (list) представляют собой упорядоченные изменяемые коллекции элементов. Элементы списка могут иметь разные типы данных. Списки создаются с помощью квадратных скобок, например:

numbers = [1, 2, 3, 4, 5] # список, содержащий набор целых чисел int

names = ["Alice", "Bob", "Charlie"] # список, содержащий набор строк str

Кортежи (tuple) похожи на списки, но они неизменяемы. Это значит, что после создания кортежа его элементы нельзя изменить. Кортежи создаются с помощью круглых скобок, например:

coordinates = (10.0, 20.0) # кортеж, содержащий координаты в виде float

Словари (dict) представляют собой коллекции пар ключ-значение. Каждый ключ уникален и связан с определенным значением. Словари создаются с помощью фигурных скобок, например:

student = {"name": "Alice", "age": 21, "grades": [85, 92, 78]}

Эти структуры данных являются мощными инструментами для организации и обработки данных в ваших программах.

**1.5 Введение в библиотеки Python**

Одной из самых сильных сторон Python является богатая экосистема библиотек, которые расширяют возможности языка и упрощают решение различных задач. Библиотеки представляют собой наборы готовых функций и классов, которые можно использовать в своих программах.

Для использования библиотеки необходимо сначала ее импортировать с помощью ключевого слова import. Например, библиотека math предоставляет функции для математических операций:

import math

print(math.sqrt(16)) # нахождение корня, вывод: 4.0

Важные библиотеки для веб-парсинга включают в себя requests для работы с HTTP-запросами и BeautifulSoup для анализа HTML-документов. Эти библиотеки будут подробно рассмотрены в разделе о создании парсера.

**1.6 Работа с текстовыми файлами в Python**

Часто возникает необходимость считывать данные из файлов или записывать данные в файлы. Python предоставляет удобные средства для работы с файлами.

Для открытия файла используется функция open(), которая возвращает объект файла. Действия с файлом могут включать чтение, запись или добавление данных. После завершения работы с файлом его необходимо закрыть с помощью метода close().

Пример чтения данных из файла:

file = open("example.txt", "r")

content = file.read()

print(content)

file.close()

Пример записи данных в файл:

file = open("output.txt", "w")

file.write("Hello, World!")

file.close()

Работа с файлами важна для сохранения и обработки данных в различных форматах.

**1.7 Обработка исключений в Python**

При написании программ часто возникают ситуации, когда что-то идет не так: файл не найден, данные введены неверно или происходит деление на ноль. Чтобы программа не завершалась с ошибкой, необходимо обрабатывать исключения — особые условия, возникающие при выполнении программы.

Для обработки исключений в Python используется конструкция try-except. В блоке try размещается код, который может вызвать исключение (ошибку), а в блоке except — код, который будет выполнен в случае возникновения этого исключения (ошибки).

try:

result = 10 / 0

except ZeroDivisionError:

print("Деление на ноль невозможно!")

Можно также использовать несколько блоков except для обработки различных типов исключений:

try:

file = open("nonexistent.txt", "r")

except FileNotFoundError:

print("Файл не найден!")

except Exception as e:

print(f"Произошла неопределенная ошибка: {e}")

Обработка исключений помогает сделать программу более надежной и устойчивой к ошибкам.

**1.8 Функции в Python**

Функции — это блоки кода, которые можно вызывать по имени, передавать им аргументы и получать результат. Функции помогают структурировать код, делают его более читаемым и многократно используемым.

Функция определяется с помощью ключевого слова def, после которого следует имя функции, список параметров в круглых скобках и двоеточие. Тело функции должно быть с отступом. Пример функции:

def greet(name):

return f"Hello, {name}!"

message = greet("Alice")

print(message) # вывод: Hello, Alice!

Функции могут не только возвращать значения, но и выполнять действия, не возвращая результатов:

def print\_greeting(name):

print(f"Hello, {name}!")

print\_greeting("Bob") # вывод: Hello, Bob!

Использование функций делает код более организованным и удобным для повторного использования.

**1.9 Введение в объектно-ориентированное программирование (ООП)**

Объектно-ориентированное программирование (ООП) — это способ создания программ, при котором программа состоит из "объектов", каждый из которых представляет собой отдельный элемент (например, человек, автомобиль, книга), со своими свойствами и действиями, что позволяет писать программы, более похожие на реальный мир. Другими словами, объектно-ориентированное программирование — это парадигма программирования, основанная на концепции объектов. Объекты — это экземпляры классов, которые объединяют данные и методы для работы с этими данными.

Класс определяется с помощью ключевого слова class, после которого следует имя класса и двоеточие. Пример класса:

class Cat:

def \_\_init\_\_(self, name):

self.name = name

def meow(self):

return f"{self.name} says Meow!"

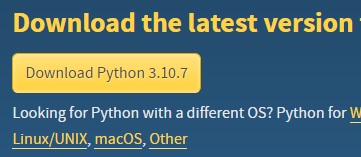
my\_cat = Cat("Boris")

print(my\_cat.meow()) # вывод: Boris says Meow!

Конструктор \_\_init\_\_ используется для инициализации объекта, а методы класса определяются как функции внутри класса. ООП позволяет создавать более сложные и структурированные программы, организуя код в виде объектов с четко определенными функциями и свойствами.

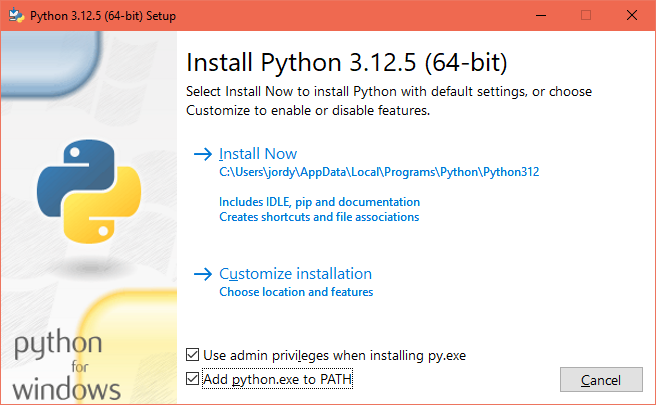
**2. Установка Python**

Для начала работы с языком программирования Python необходимо его установить. Посетите официальный сайт Python по адресу <https://www.python.org/downloads/>.



Нажмите на кнопку для скачивания, соответствующую вашей операционной системе.

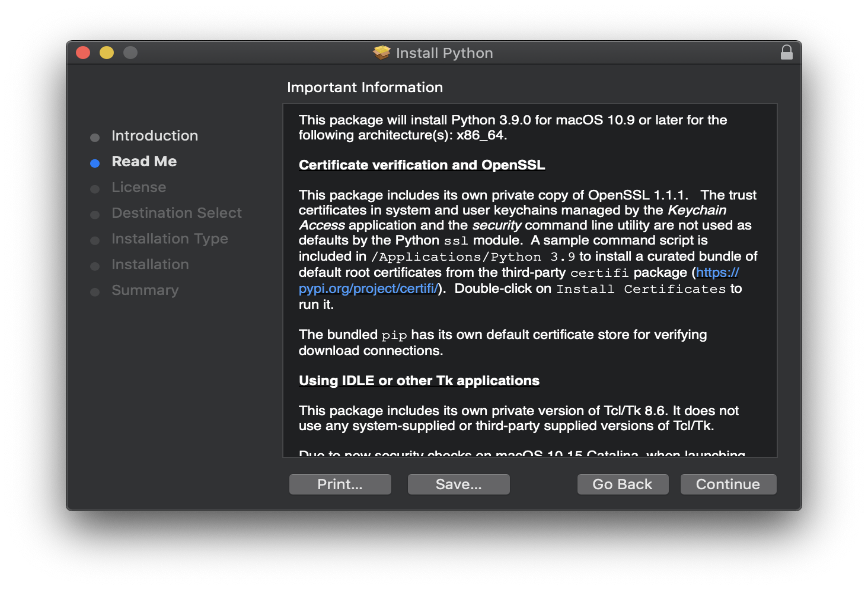
Установка Python на Windows:



Скачайте установочный файл и запустите его.

Важно: убедитесь, что выбрали внизу опцию "Add Python to PATH", прежде чем нажать "Install Now".

Установка Python на macOS:



Скачайте установочный файл и запустите его.

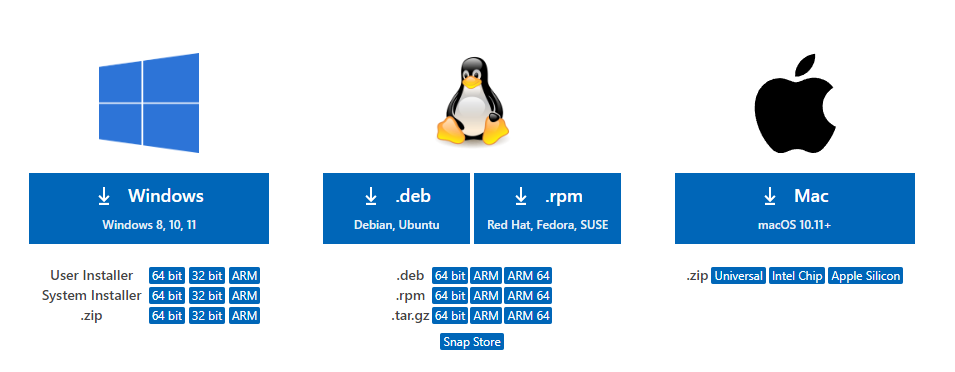
Следуйте инструкциям на экране, нажимая "Install Now" или "Continue" до завершения установки.

После успешной установки Python, вы готовы к настройке среды разработки.

**Настройка среды разработки**

Для написания и отладки кода необходима интегрированная среда разработки (IDE). Хотя в Windows с Python автоматически устанавливается IDLE, для более универсального подхода рекомендуется использовать Visual Studio Code (VSCode).

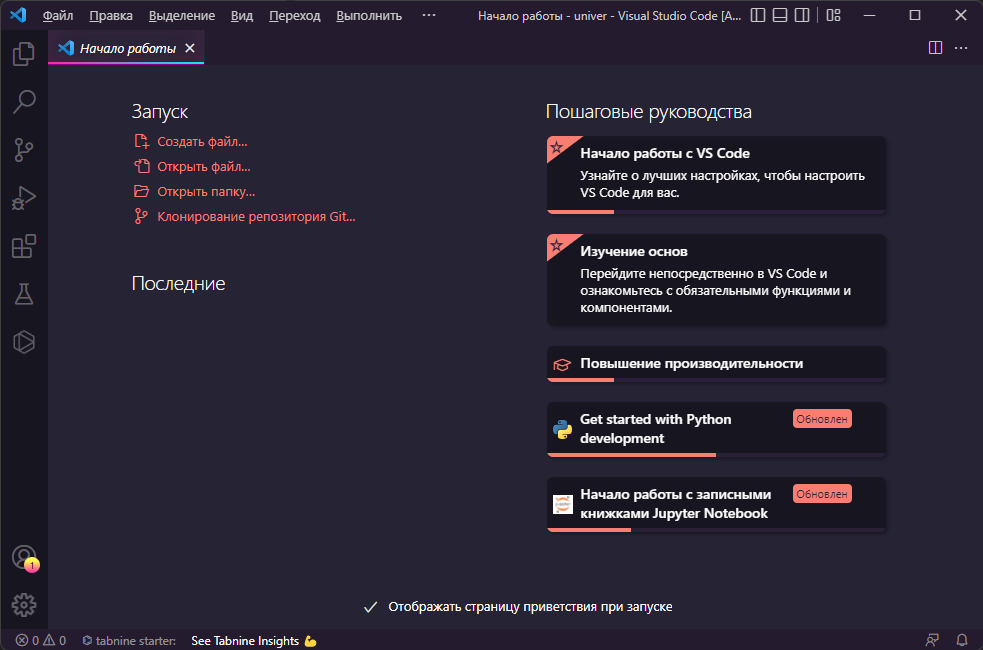
Перейдите на <https://code.visualstudio.com/download> и скачайте установочный файл, соответствующий вашей операционной системе.



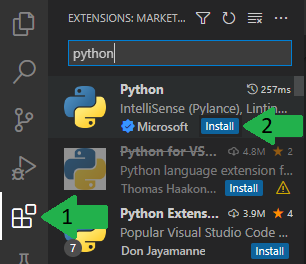
Установите VSCode, следуя стандартной процедуре установки программного обеспечения.

После установки VSCode выполните следующие шаги:

Запустите VSCode.

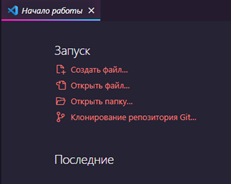


Перейдите к разделу "Extensions" (иконка кубиков слева) и введите в строку поиска "python". Установите первое найденное расширение для работы с Python.



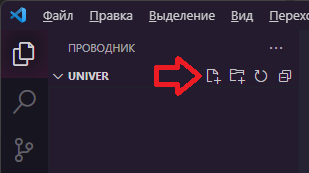
**Создание рабочего пространства**

В VSCode выберите "Открыть папку…" в разделе "Файл".



Создайте новую папку в удобном для вас месте, она будет содержать все файлы вашего проекта.

Откройте созданную папку.

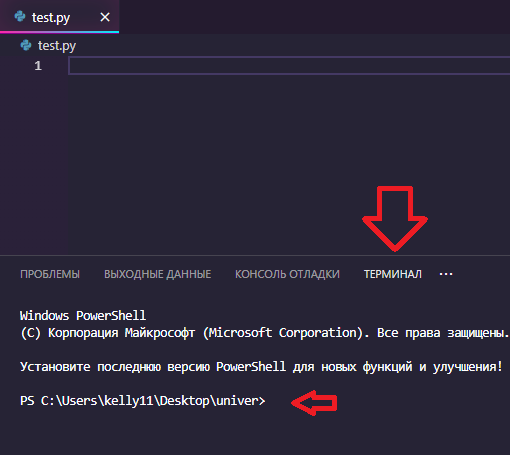


Теперь можно создать новый файл с расширением .py:

Нажмите на иконку нового файла, введите имя файла и добавьте расширение .py (например, parser.py английскими буквами).

Откроется редактор кода, где вы сможете вводить и редактировать текст программы.

**Установка необходимых библиотек**



В терминале VSCode (если он не открыт, выберите "Терминал" > "Создать терминал" в меню) выполните команды для установки нужных библиотек.

Проверьте наличие менеджера пакетов **pip**:

**pip3.12 --version**

Если появляется ошибка, попробуйте вашу версию или:

**pip --version**

Важно, чтобы одна из команд не выдавала ошибку. Если ошибка продолжает появляться, переустановите **Python** и внимательно проследите поставили ли вы галочку в самом низу до нажатия кнопки Установить. Затем перезапустите полностью редактор кода **VSCode**

Установите библиотеки **requests** и **BeautifulSoup:**

**pip install requests**

или

**pip3.12 install requests**

Далее:

**pip install bs4**

или

**pip3.12 install bs4**

После успешной установки библиотек вы увидите текст **Successfully installed без красных ошибок**, значит, вы готовы приступить к написанию парсера.

**Выбор источников данных**

Парсинг (от англ. parsing) - это процесс анализа и интерпретации структуры данных, например, текста или кода, для извлечения из него конкретной информации. Парсинг может быть применен к различным типам данных, включая текст, XML, JSON и другие.

Скрепинг (от англ. scraping) - это процесс автоматизированного извлечения данных из веб-страниц или других источников, часто с помощью парсинга HTML-кода страницы. Скрепинг обычно используется для сбора данных из веб-ресурсов, которые не предоставляют API или другие официальные способы доступа к данным.

Для скрейпинга и парсинга веб-сайтов важно выбрать подходящие источники данных. Из предложенных группой вариантов сайтов интересные все, но не все из них будут просты для начала обучения. Вообще скрейпинг и парсинг сайтов можно разделить на три типа: простой, средний и сложный. Простой это когда сайт при запросах возвращает данные в удобно читаемом **формате json,** которые разложить по полочкам можно за пару минут. Средний это когда приходится вытягивать нужную информацию из **html** элементов с помощью сторонних библиотек (например bs4). Сложный способ – сложные сплетения загрузки сайта путём использования **javascript,** без выдачи **json** ответов и с защитой от скрейпинга.

Мы разберём простой и средний варианты: оба случая мы рассмотрим на сайте <https://kinoart.ru/>, который содержит статьи, рецензии и новости о кино.

**Создание парсера**

Откройте сайт <https://kinoart.ru/> в браузере (Chrome, Firefox, Safari).

Проанализируйте структуру сайта, чтобы понять, какие данные вы хотите извлечь и как они представлены на веб-страницах.

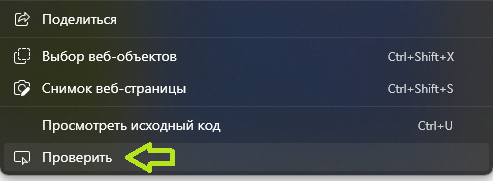
Пример простой задачи парсинга заключается в сборе всех статей и рецензий с главной страницы сайта. Важно помнить, что парсер может извлекать только те данные, которые видны на веб-странице и доступны через её интерфейс.

**4.1 Изучение запросов сайта**

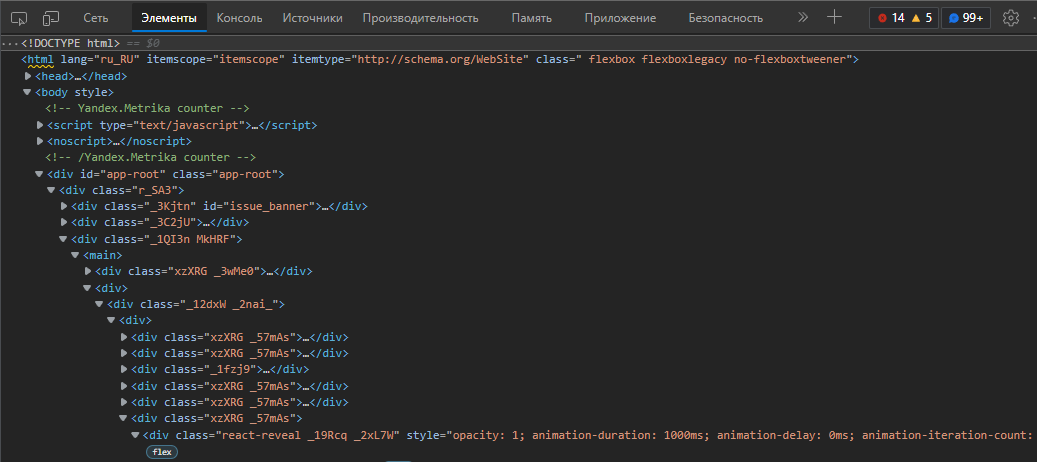
Начнем с освоения инструментов разработчика в браузере. Для этого на сайте <https://kinoart.ru/> откроем панель разработчика.

Открытие панели разработчика на **Windows:**

Кликните правой кнопкой мыши на странице сайта и выберите пункт **«Проверить»** или **«Исследовать элемент».** В разных браузерах этот пункт может называться по-разному, но обычно он находится в самом низу контекстного меню.



Вы увидите следующее окно:

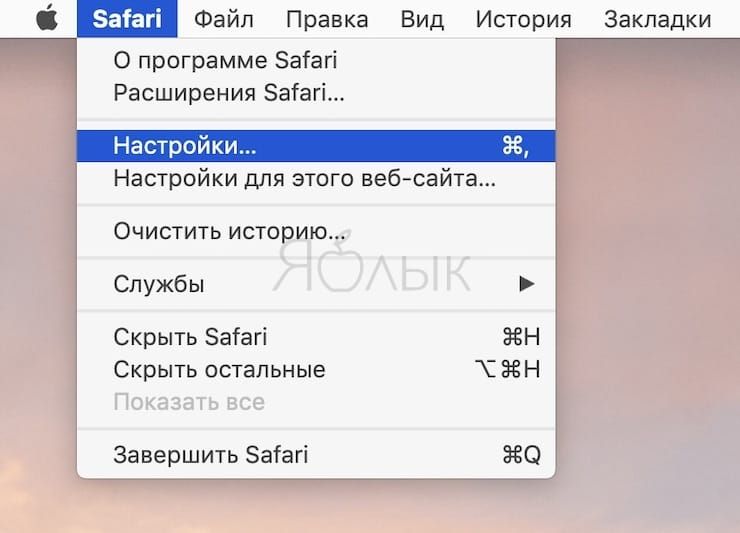


Это окно может располагаться снизу или справа. Расширьте его, потянув за край, чтобы увидеть вкладки «Элементы», «Консоль» и другие.

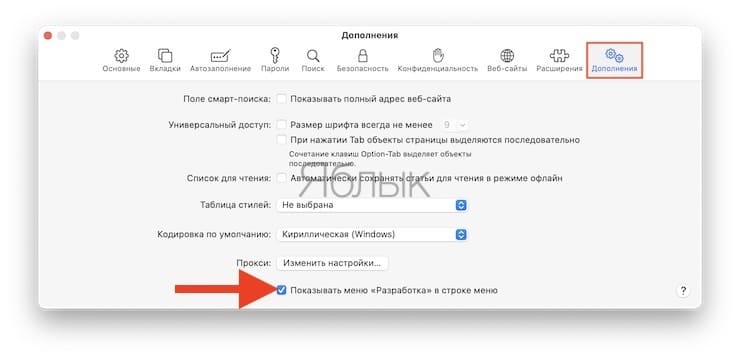
**Открытие панели разработчика на macOS:**

Если вы используете **Safari**, то сначала необходимо включить средства разработки:

Запустите **Safari** и выберите в меню **Safari → Настройки.**



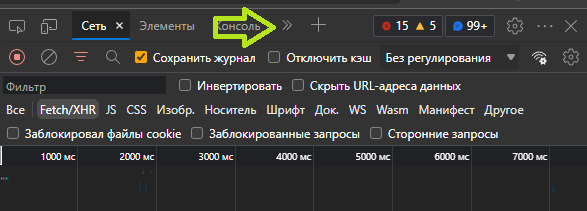
Перейдите на вкладку «Дополнения».



В самом низу поставьте галочку напротив пункта «Показывать меню “Разработчика” в строке меню».

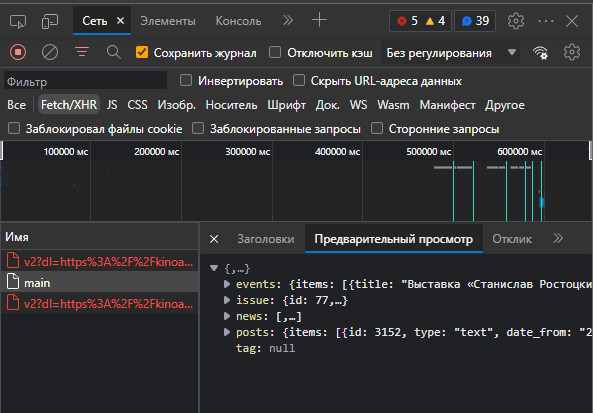
Теперь откройте панель разработчика через новый пункт в меню Safari.

Работа с вкладкой «Сеть»:



Откройте вкладку **«Сеть»** в панели разработчика. Если вы не видите этой вкладки, нажмите на две стрелочки **(>>)** и найдите её в выпадающем списке.

Убедитесь, что выбран пункт **«Fetch/XHR»** или просто **XHR.**



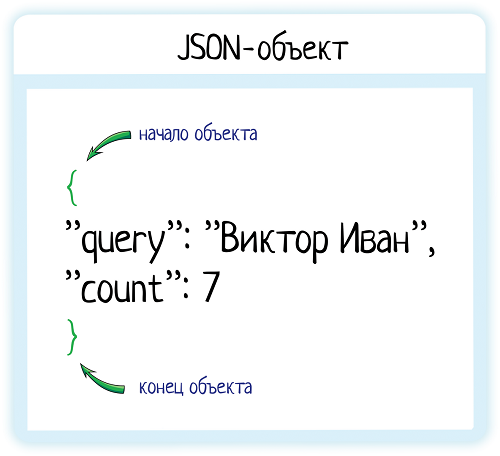
Теперь обновите страницу сайта (кнопкой **F5** или иконкой обновления).

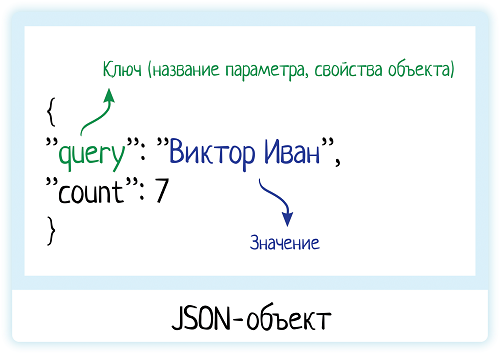
Если вы видите список запросов, значит вы на правильном пути! В списке слева будет строка с названием main (или похожим). Кликните на неё и выберите вкладку «Предварительный просмотр».

**4.2 Работа с JSON**

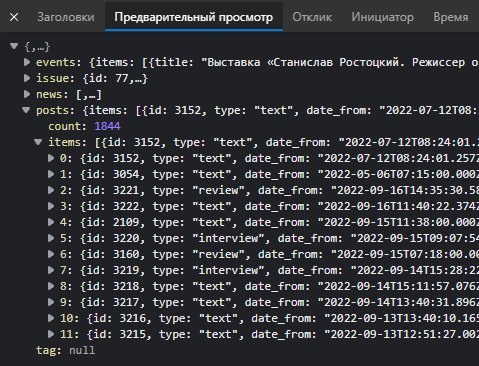
В правой части окна вы увидите данные в формате JSON.

JSON (JavaScript Object Notation) — это формат хранения и обмена данными, состоящий из пар ключ-значение, заключённых в фигурные скобки. Более подробное описание JSON вы можете найти на <https://habr.com/ru/post/554274/>.



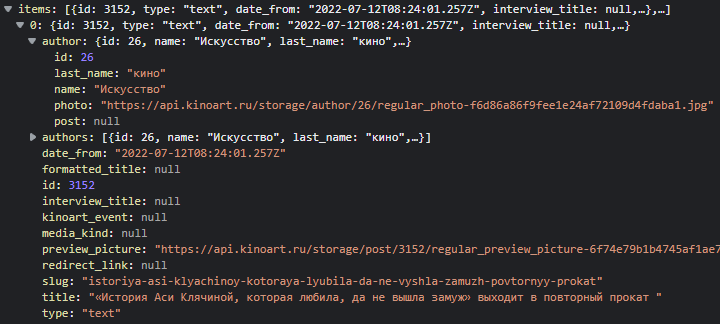


Рассмотрим данные подробнее:



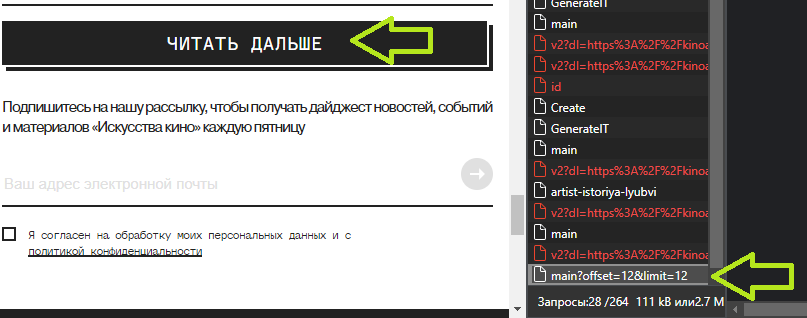
У нас есть ключи events, issue, news, posts, каждый из которых содержит вложенные объекты JSON. Ключ posts содержит новости и статьи. В нем вы найдете ключ count (общее количество новостей) и items (массив новостей с различными атрибутами).

Развернув первый элемент массива items, мы увидим такие поля как title, slug, preview\_picture, id, type и author.



**4.3 Извлечение данных**

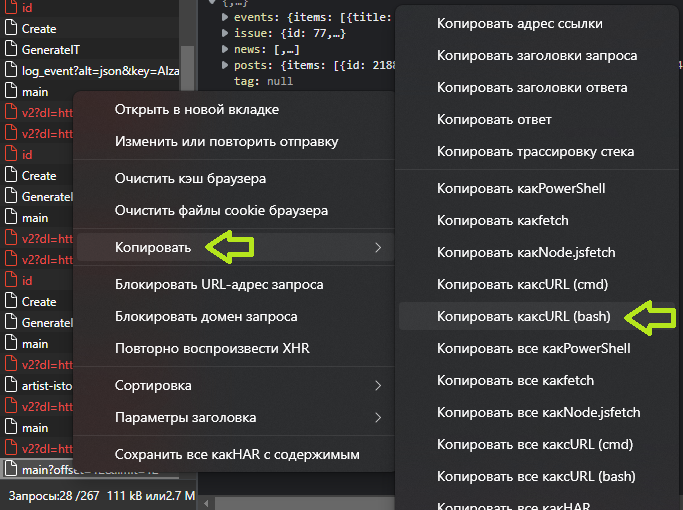
Чтобы извлечь данные, сначала получим запрос, который позволяет получить список новостей.



На сайте есть кнопка «Читать дальше» внизу страницы. Нажатие на неё отправляет запрос с параметрами оффсета и лимита новостей.

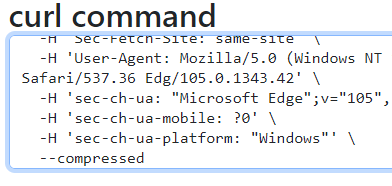
Нажмите на кнопку «Читать дальше» и в панели разработчика найдите новый запрос (может называться main или похоже).

Выделите этот запрос, нажмите правой кнопкой мыши и выберите «Копировать» → «Копировать как cURL (bash)».



Теперь преобразуем СURL запрос в код на Python:

Перейдите на сайт <https://curlconverter.com/> и вставьте скопированный запрос в поле «CURL COMMAND».



В поле ниже появится сгенерированный код на Python. Нажмите «Copy to clipboard» справа снизу, чтобы скопировать его.

Этот код послужит основой для вашего парсера новостей с сайта <https://kinoart.ru/>.

4.4 **Пишем тело парсера**

Вернёмся к **Visual Studio Code.**

Вставляем скопированный запрос в открытый **VSCode:**



Всё должно выглядеть как на скриншоте, за исключением некоторых слов в переменной **headers.** Общая картина должна быть такой.

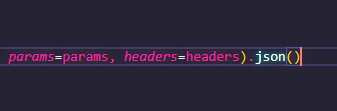
Обратите внимание, что **headers** важен в парсинге, так как позволяет программе «притвориться» настоящим человеком и обойти элементарные защиты.

Так как мы видели, что сайт использует формат данных JSON, нам нужно дописать код так, чтобы мы могли с ним работать:

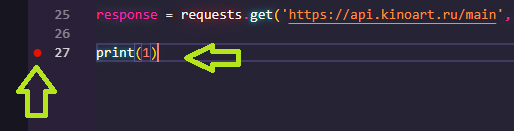
В конец переменной response дописываем

**.json()**

Получится так:



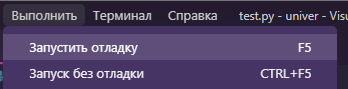
И далее, сделаем такую задачу, которая поможет нам понимать, что получила программа в себя и что не смогла: научимся пользоваться точками останова.



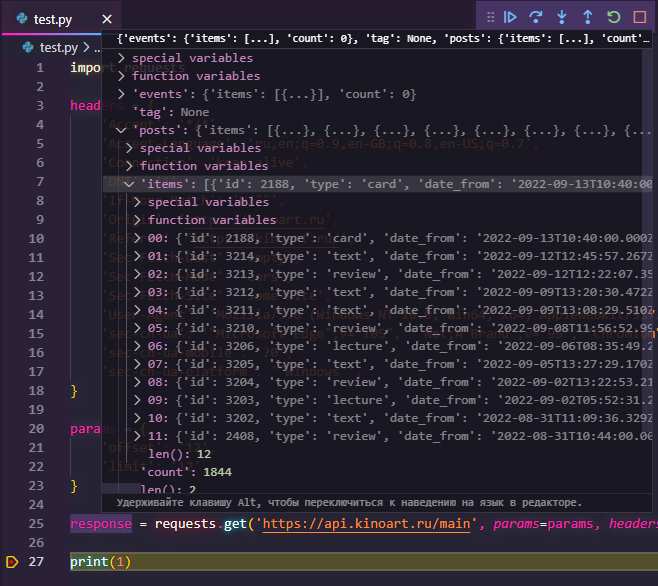
Под **response** напишем вывод цифры **1** в консоль и нажмём мышкой слева от номера строки так, что там появится красная точка. Если не ставится – попробуйте сохранить файл **(Ctrl+S или Ctrl+Shift+S**) и повторить попытку.

Эта точка означает, что программа остановится на выбранной нами строке и мы сможем увидеть всё что происходит внутри программы до неё.

После того как поставили красную точку пора запустить нашу программу – нажмите **F5,** либо в меню вверху выберите **Выполнить > Запустить отладку:**

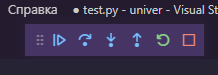


После того как программа остановится на этапе печати цифры 1 на строке 27 и выполнит обращение к сайту, мы сможем рассмотреть, что же содержит в себе переменная response, в которую парсер поместил ответ от сайта с содержимым JSON.



Если вы в таком режиме наведёте мышкой на переменную **response**, то вы увидите то же самое, что мы видели в окне разработчика в браузере. Это важный шаг, который значит, что парсер успешно получил список новостей с сайта.

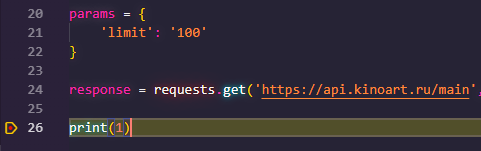
Остановить выполнение программы можно на иконку красного квадратика сверху:



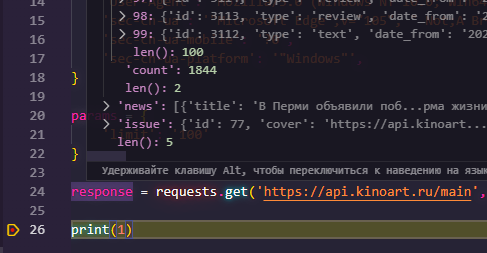
Первым делом давайте поэкспериментируем. Если вы обратите внимание на переменную **params** и то, что она в себе содержит, то увидите, что запрос к сайту имеет два параметра **offset** который нам сейчас не важен и **limit.** Вот лимит по умолчанию установлен на 12 новостей. Почему бы не собрать с сайта сразу все 1844 новости?

Пробуем так: удаляем **offset** вообще, а лимит ставим на 100. Бывают случаи, когда код сайта запрещает получать более чем определённое количество элементов за раз, но мы сейчас это выясним.

Сделаем запрос на 100 элементов:



Запускаем выполнение кода и наводимся на переменную **response:**

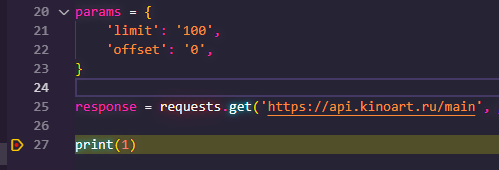


Развернув все ключи в **posts** и **items**, мы видим, что ключ **items** теперь содержит 100 элементов новостей. (*Мы только что сделали то, что не предусмотрено интерфейсом сайта*)

Давайте попросим запросить все 1844 новости, но, у вас это не получится, т.к. лимит сайта подразумевает выдачу не более 100 новостей за раз.

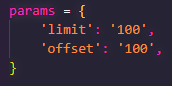
Давайте разберёмся как же тогда запросить все новости. Для этого программисты придумали **offset**, который позволяет делать смещение относительно последней нужной нам новости. Давайте разберем.

Возвращаем **offset** в **params:**



Такой запрос выдаст первые 100 новостей. Как же нам тогда получить все остальные начиная с 101? Offset!

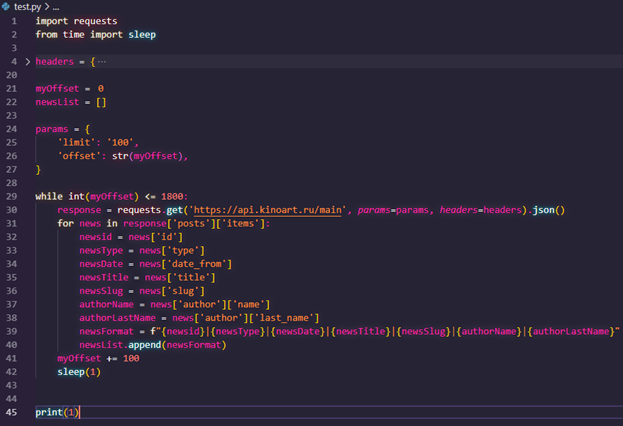
Выполняем следующие действия:



Таким образом, мы зададим команду сайту, что первые 100 новостей мы уже получили и хотим получить следующие 100, и сайт нам вернёт новости начиная с 101.

После чего снова увеличим offset на 100 и так до самого конца.

Теперь займёмся созданием функционала сохранения новостей.



Давайте допишем в самом верху импорт библиотек sleep:

**from time import sleep**

Это позволит «засыпать» коду в нужные моменты и не перегружать сильно интернет и данный сайт слишком сильно. Далее, давайте свернем переменную **headers,** что бы не отвлекала наше внимание. Для этого, нажмите на стрелочку слева.

Ниже делаем две переменные:

**myOffset = 0 # без кавычек**

**newsList = []**

Первая будет позволять нам увеличивать offset в теле цикла, вторая это пустой список, будем туда кидать новости про кино.

Далее создадим цикл while, который по условию будет работать до тех пор, пока offset не достигнет 1800 новостей.

Переместим внутрь тела цикла переменную response, которая будет каждый раз обращаться к сайту и запрашивать очередную пачку новостей.

Внутри цикла **while** создадим цикл **for**. Он нужен для того, что бы мы перебрали по очереди все новости из переменной response, которых там ровно 100 штук.

Пишется так:

**for news in response[‘posts’][‘items’]:**

Где **news** это будет конкретная порядковая новость из ключа **items** в переменной response. Как только цикл обработает эту новость хранящуюся в news, он в этот же news запишет следующую новость из **response items** и так пока они не кончатся.

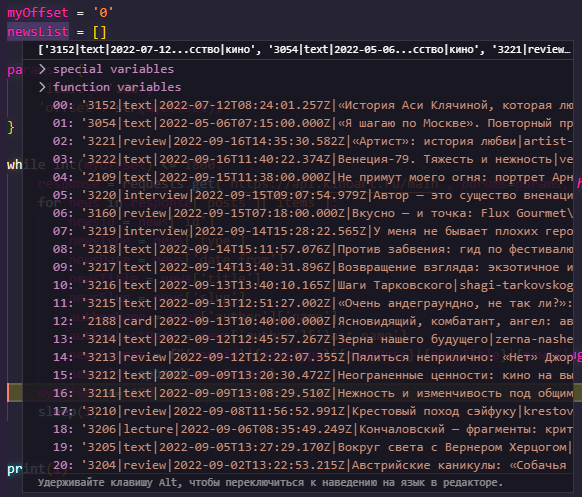
Дальше, для наглядности давайте создадим несколько говорящих за себя переменных и поместим в них данные из новости, например это **id** новости, название, тип новости, ссылка на неё.

Почти в конце, переменная **newsFormat,** призвана для простоты. Ей нужно собрать в себя все предыдущие переменные и превратить их в одну единую строку, где каждый элемент разделён символом ровной черты **|.** Переменные в такой ситуации пишутся внутри фигурных скобок **{}.**

В конце цикла, строка **newsList.append(newsFormat)** просто кидает в наш пустой список **newsFormat** только что созданную строку в цикле, со всей информацией по новости.

И в конце, мы просто увеличиваем смещение **offset** на 100, чтобы получить следующие 100 новостей, а не эти же самые, что мы уже собрали и кидаем программу в сон на 1 секунду.

В итоге если остановить выполнение на этапе увеличения смещения и посмотреть на список новостей **newsList** мы увидим, что наш список пополнился очередной сотней полученных новостей именно в том виде, в каком мы его сформировали в строке **newsFormat:**



Поздравляю! У вас только что получилось сделать очень большую работу и написать значительную часть парсера. По факту это его простая и далеко не законченная версия, которая не использует необходимые в таких ситуациях способы формирования и хранения данных. В идеале нужно создавать локальную базу данных и заносить все значения новостей уже туда. Она позволит работать и анализировать новости намного удобнее.

Дальше можно придумать что интересного сделать с полученными новостями. Например, дописать способ скрапинга ссылки на каждую отдельную новость прямо в программе (в теле цикла for), что бы мы смогли собрать дополнительно как минимум основной текст новости, а не только title. Затем отфильтровать новости по автору, по конкретной дате, найти новости, в которых обсуждается только фильм Груз 200, подключить нейросеть, которая будет определять правда ли существуют негативные рецензии на фильм Груз 200.

1. **Создание аналитической части (сложность – средне)**

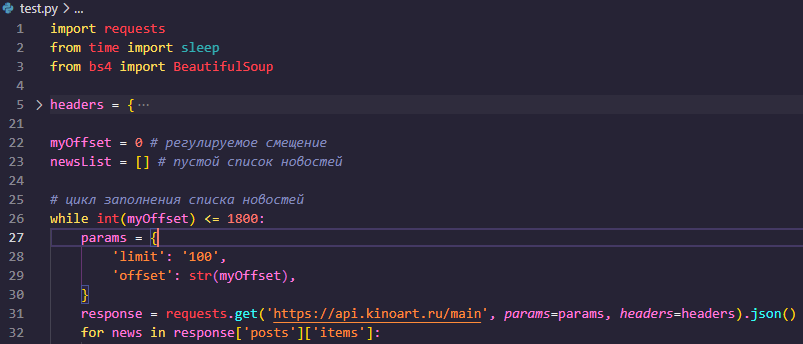
Для построения аналитической части программы вначале необходимо подключить библиотеку BeautifulSoup, которая поможет осуществлять парсинг веб-страниц. Это можно сделать следующим образом:

**from bs4 import BeautifulSoup**

Была исправлена ошибка, связанная с типом переменной **myOffset**, которая изначально была строкового типа **(string),** что приводило к ошибке при инкрементации. Для устранения этой проблемы переменной было присвоено значение **0 (без кавычек)**, что соответствует числовому типу данных.

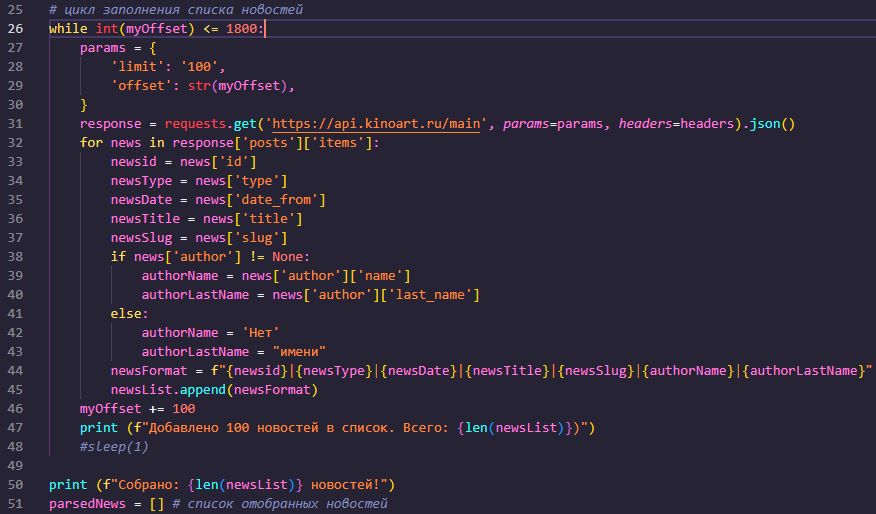
Также была скорректирована ошибка в коде, когда параметры запроса **(params)** были вынесены за пределы цикла. Это привело к тому, что значение смещения **(offset)** не обновлялось должным образом при каждом запросе. Внесённые изменения позволяют параметрам корректно обновляться при каждом выполнении цикла.

Код должен выглядеть так (**headers** всё ещё свёрнут):



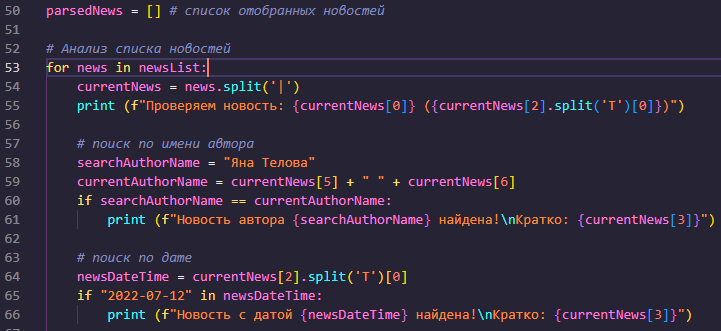
В программе могут встречаться символы решетки **#** с пояснительным текстом на русском языке. Это комментарии, которые поясняют, что содержится в тех или иных переменных или какие действия будут выполняться в дальнейшем. Комментарии не влияют на выполнение программы и игнорируются интерпретатором.

Анализ кода методом проб и ошибок выявил, что не все новости на сайте содержат указанного автора; в некоторых случаях значение автора равно **None**. Чтобы избежать ошибок, связанных с обращением к отсутствующему автору, была добавлена проверка на наличие автора в статье. Это изменение добавлено на 38-й строке кода, где осуществляется проверка, чтобы программа не завершалась с ошибкой при попытке доступа к отсутствующему значению ключа **["author"]["name"].**



Для отслеживания работы программы и её логики в разных местах кода были добавлены команды **print()**, которые выводят в консоль определённую информацию. Такой подход позволяет визуально понять, что происходит в программе, без необходимости использования точек останова для проверки текущего состояния выполнения кода.

На 46-й и 49-й строках выводится информация о количестве добавленных строк и общем количестве строк в списке. На 47-й строке используется команда sleep, которая приостанавливает выполнение программы на одну секунду. Команда была закомментирована для ускорения сбора новостей; при необходимости её можно вернуть, удалив символ решётки.



С 50-й строки начинается проектирование аналитической части, создаётся новый пустой список parsedNews, который будет использоваться для фильтрации отобранных по определённым параметрам новостей.

На 53-й строке начинается третий цикл, который проходит по списку newsList, собранному на предыдущем этапе парсинга. На строке 54 используется выражение:

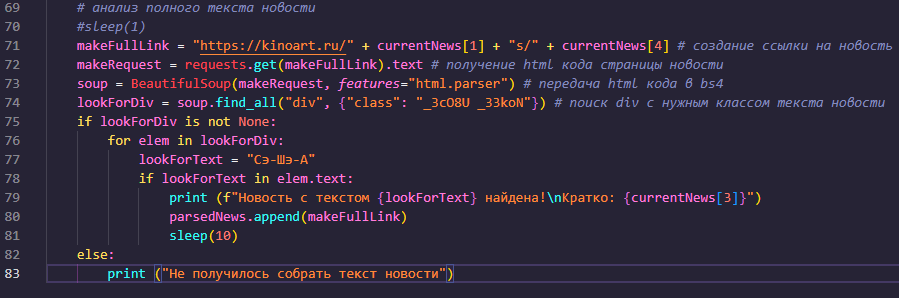
**currentNews = news.split(‘|’)**

Эта команда разделяет элементы собранных новостей. В первом цикле эта строка была собрана из идентификатора новости (ID), заголовка, времени публикации и других параметров, а теперь её «распаковывают» обратно в отдельные элементы.

На 58-й строке в переменной **searchAuthorName** указывается имя автора, статьи которого необходимо найти на сайте. В блоке кода с 58-й по 61-ю строку парсер проверяет каждую новость на наличие указанного имени автора; если такое имя найдено, программа уведомит об этом в консоли. На строке 59 создаётся новая переменная, в которую записывается комбинация имени и фамилии автора из списка отобранных статей.

С 64-й по 66-ю строки осуществляется поиск по указанной дате. Для этого используется метод **split** по символу **T**, который отделяет дату от времени в строке, полученной с сайта в формате 2022-07-12T08:24:01.257Z. Этот метод позволяет разделить строку на две части и обратиться к нулевому индексу для получения только даты публикации.

Таким образом, данный алгоритм обеспечивает корректную обработку данных, полученных в процессе парсинга новостного ресурса, с возможностью фильтрации и анализа информации на основании авторства и даты публикации.



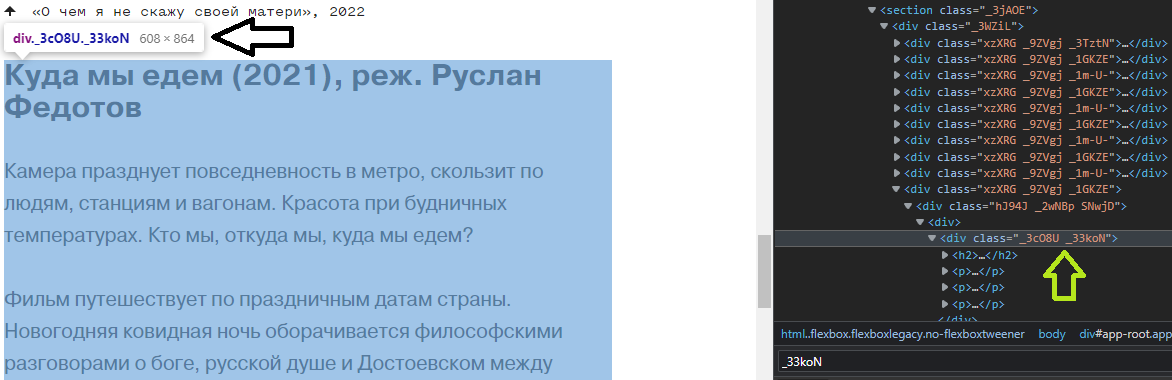
Разберём написанное: вернуть ли sleep – зависит от вашего желания.

**makeFullLink** собирает прямую ссылку на источник новости.

**makeRequest** отправляет get запрос на страницу новости и записывает ответ в переменную.

Soup переменная подгружает в библиотеку bs4 переменную **makeRequest**, которая содержит в себе html код страницы с новостью.

Переменная **lookForDiv** производит поиск div элементов в коде с классом \_3cO8U \_33koN. Со временем название класса может динамически меняться. Узнать название класса можно с помощью того же инструмента в браузере по кнопке Исследовать или Проверить.

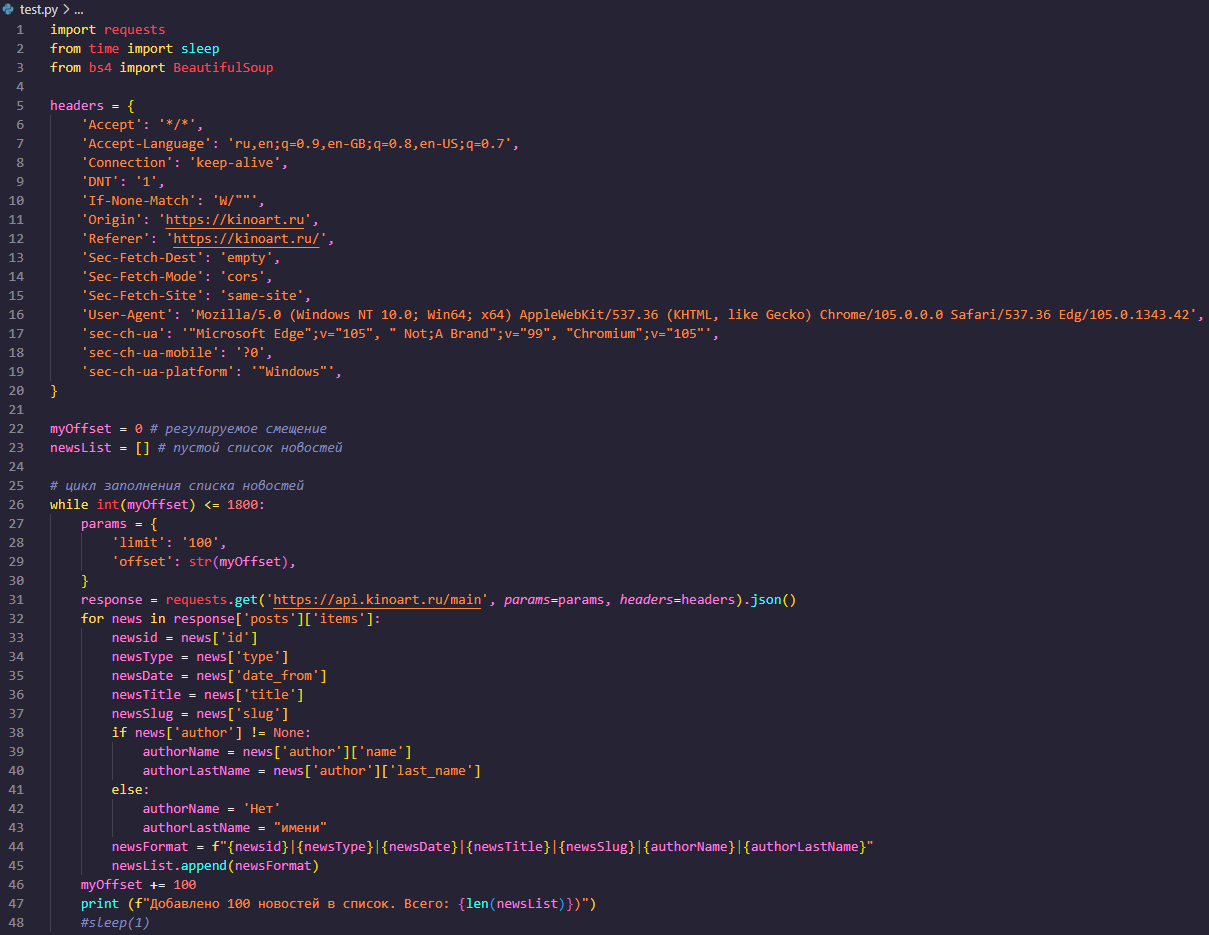


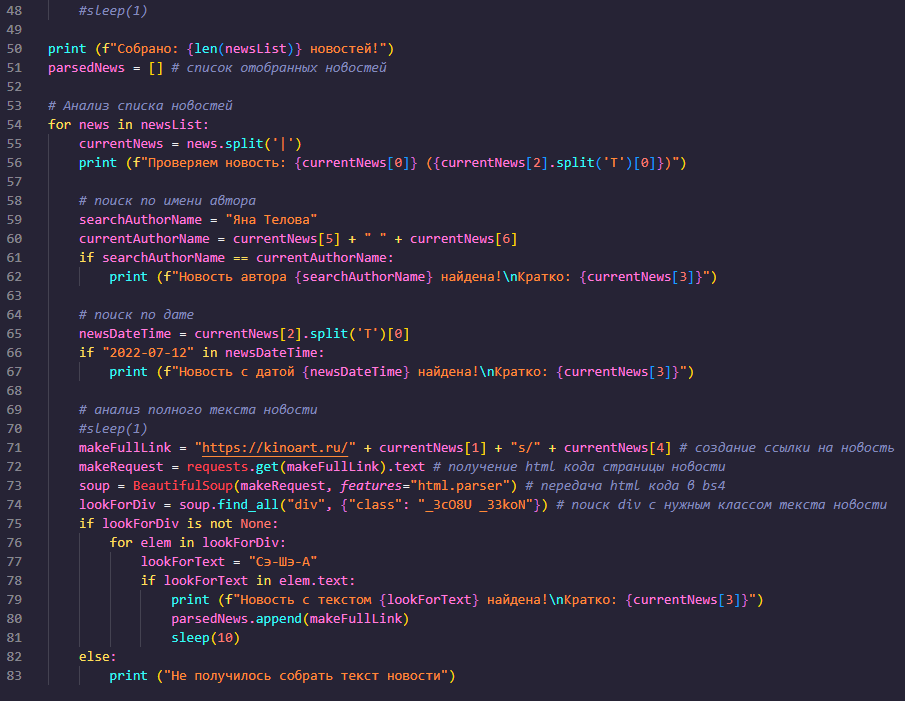
Чтобы определить, какой именно div содержит необходимый контент, следует воспользоваться функцией наведения курсора мыши на элементы кода страницы. При наведении на каждый элемент HTML справа в панели инструментов разработчика, соответствующий элемент будет подсвечен на самой веб-странице. Необходимо найти div, который включает исключительно нужный текст. После нахождения подходящего элемента следует скопировать его класс и использовать метод find\_all библиотеки BeautifulSoup (bs4) для извлечения текста со страницы.

На 75-й строке кода производится проверка: удалось ли библиотеке BeautifulSoup найти хотя бы один элемент div с необходимым классом. В процессе работы программы может возникнуть ситуация, когда в некоторых новостях отсутствует div с нужным классом, что может означать отсутствие текста в новости или то, что элемент div с текстом имеет другое название класса.

На 78-й строке выполняется проверка текста на соответствие заданному условию. Как только будет обнаружена новость, удовлетворяющая критериям, парсер уведомит пользователя о её нахождении

Полный код парсера:





Код полностью рабочий и протестирован. Поиск по имени автора и дате работает. Поиск по тексту находит совпадение через 10-30 новостей.