Веб-скрепінг з Python: Beautiful Soup та Selenium

Вступ: Що таке веб-скрепінг?

Веб-скрепінг — це процес автоматичного збору даних з веб-сайтів. Це як програма, що читає вміст веб-сторінки та витягує з неї потрібну інформацію. Для цього ми будемо використовувати дві потужні бібліотеки Python.

Бібліотека	Призначення	Ключова відмінність
Beautiful Soup	**Парсинг (розбір) HTML/XML).** Вона ідеально підходить для вилучення даних зі статичного вмісту, тобто з вмісту, який вже є на сторінці при її першому завантаженні.	Не може працювати з динамічним вмістом (JavaScript), клікати кнопки чи заповнювати форми.
Selenium	**Автоматизація дій браузера.** Вона може імітувати дії користувача (кліки, введення тексту, прокрутка) та працювати з динамічними сторінками.	Повільніша, оскільки запускає справжній браузер, але необхідна для складних завдань.

Частина 1: Beautiful Soup

1.1. Робота зі статичним контентом

Beautiful Soup — це бібліотека для зручного розбору HTML-документів. Вона перетворює складну структуру HTML на дерево об'єктів Python, з якими легко працювати. Це дозволяє нам легко шукати елементи за тегами, класами, ID та атрибутами.

Для початку встановіть необхідні бібліотеки:

```
pip install beautifulsoup4 requests
```

Requests — це бібліотека, яка допомагає завантажити вміст веб-сторінки перед тим, як Beautiful Soup почне його парсинг.

1.2. Приклад: Отримання даних з Вікіпедії

Ми використаємо сторінку української Вікіпедії про Python як приклад статичного сайту.

```
import requests
from bs4 import BeautifulSoup

# URL сторінки, яку ми будемо аналізувати
URL = 'https://uk.wikipedia.org/wiki/Python'

# 1. Завантажуємо HTML-код сторінки
try:
    response = requests.get(URL)
    response.raise_for_status()  # Перевіряємо на помилки HTTP
    html_content = response.text
except requests.exceptions.RequestException as e:
    print(f"Помилка завантаження сторінки: {e}")
    exit()

# 2. Створюємо об'єкт Beautiful Soup для парсингу
soup = BeautifulSoup(html_content, 'html.parser')

# 3. Вилучаємо дані за допомогою методів find() та find_all()
print("\nАналіз елементів...")

# Знаходимо заголовок сторінки (<title>)
title = soup.find('title')
print(f"Заголовок сторінки: {title.text}")

# Знаходимо перший заголовок H1
h1_tag = soup.find('h1')
if h1_tag:
    print(f"Перший заголовок H1: {h1_tag.text}")

# Знаходимо всі параграфи на сторінці
all_paragraphs = soup.find_all('p')
print(f"Знайдено {len(all_paragraphs)} параграфів.")

# Виводимо перші 100 символів тексту першого параграфа
if all_paragraphs:
```

```
    print(f"Перший параграф: {all_paragraphs[0].text[:100]}...")

# Знаходимо всі посилання на сторінці
all_links = soup.find_all('a')
print(f"Знайдено {len(all_links)} посилань.")

# Виводимо URL та текст перших п'яти посилань
for link in all_links[:5]:
    href = link.get('href')
    link_text = link.text.strip()
    print(f"Посилання: URL = {href}, Текст = {link_text}")
```

1.3. Методи пошуку елементів у Beautiful Soup

Beautiful Soup надає три основні та найбільш вживані методи для пошуку елементів: `find()`, `find_all()` та `select()`.

Метод	Опис	Повертає	Приклад
`.find()`	Знаходить **перший** елемент, що відповідає заданим критеріям.	Об'єкт тегу або `None`, якщо елемент не знайдено.	`soup.find('h1', class_='main-title')`
`.find_all()`	Знаходить **всі** елементи, що відповідають заданим критеріям.	Список об'єктів тегів.	`soup.find_all('p')`
`.select()`	Знаходить елементи за **CSS-селекторами**. Найбільш потужний та гнучкий метод.	Список об'єктів тегів.	`soup.select('div.movie-item a.showtime-item')`

🛠️ Required Tools

To start scraping, you'll need to install a few libraries. You can do this using `pip`, Python's package installer.

Requests: This library handles the HTTP requests to get the HTML content from a website.

pip install requests

Beautiful Soup 4: This is the core library for parsing the HTML.

pip install beautifulsoup4

A Parser: Beautiful Soup needs a parser to understand the HTML's structure. The `lxml` parser is the most popular and is known for being very fast.

pip install lxml

📝 The Scraping Process

The typical workflow for scraping a website with Beautiful Soup involves these four key steps:

1. **Fetch the HTML:** Use the `requests` library to download the raw HTML of the web page you want to scrape.
2. **Parse the HTML:** Pass the raw HTML to the `BeautifulSoup()` constructor. This converts the unstructured text into a structured, searchable object.
3. **Find the Elements:** Use Beautiful Soup's methods to locate the specific HTML tags that contain the data you're looking for. You can search by tag name (`h1`, `p`, `a`), or by attributes like `class` or `id`.
4. **Extract the Data:** Once you've found the right element, you can pull out the text content, a link URL, or any other attribute.

🎯 Key Methods

Here are the most important methods you'll use to find and extract data:

- **`find()`:** This method finds the **first** HTML tag that matches your criteria. It returns a single result.
    - **Example:** `soup.find('h1', class_='page-title')` would find the first `<h1>` tag with the class "page-title".
- **`find_all()`:** This method finds **all** HTML tags that match your criteria and returns

them as a list.
    - **Example:** `soup.find_all('a', href=True)` would find all `<a>` tags that have an `href` attribute.
- **`.text`:** This is a property used to extract the text content *inside* a tag.
    - **Example:** If `title_tag` is a Beautiful Soup object for `<h1>Hello World!</h1>`, then `title_tag.text` would return `"Hello World!"`.
- **`.get()`:** This method is used to get the value of a specific attribute of a tag.
    - **Example:** If `link_tag` is the tag `<a href="https://example.com">`, then `link_tag.get('href')` would return "https://example.com".

```
import requests
from bs4 import BeautifulSoup
url = "https://www.passiton.com/inspirational-quotes"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')
quotes = []
quote_boxes = soup.find_all('div', class_='col-6 col-lg-3 text-center
margin-30px-bottom sm-margin-30px-top')
for box in quote_boxes:
    quote_text = box.img['alt'].split(" #")
    quote = {
        'theme': box.h5.text.strip(),
        'image_url': box.img['src'],
        'lines': quote_text[0],
        'author': quote_text[1] if len(quote_text) > 1 else 'Unknown'
    }
    quotes.append(quote)
# Display extracted quotes
for q in quotes[:5]:  # print only first 5 for brevity
    print(q)
```

```
import csv

filename = "quotes.csv"
with open(filename, mode='w', newline='', encoding='utf-8') as file:
    writer = csv.DictWriter(file, fieldnames=['theme', 'image_url',
'lines', 'author'])
    writer.writeheader()
    for quote in quotes:
        writer.writerow(quote)
```

Частина 2: Selenium

2.1. Робота з динамічним контентом

Selenium дозволяє автоматизувати дії в браузері, що є необхідним, коли контент завантажується за допомогою JavaScript (наприклад, після кліку на кнопку або прокрутки). Selenium "бачить" сторінку так само, як і людина.

Вам потрібно встановити **Selenium** та **драйвер** для вашого браузера (наприклад, chromedriver для Google Chrome).

```
pip install selenium
```

2.2. Приклад: Автоматизація пошуку на Google

У цьому прикладі ми використаємо Selenium, щоб імітувати дії користувача: відкрити сторінку, знайти рядок пошуку, ввести текст і натиснути Enter.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

print("\n== Selenium: Автоматизація дій у браузері ==")

# 1. Налаштовуємо драйвер
driver = webdriver.Chrome()

# 2. Відкриваємо веб-сторінку
driver.get("https://www.google.com")
print(f"Відкрито сторінку: {driver.title}")

# 3. Знаходимо елементи на сторінці
try:
    # Знаходимо рядок пошуку за його атрибутом name="q"
    search_box = driver.find_element(By.NAME, "q")
```

```
    # 4. Взаємодіємо з елементом: вводимо текст
    search_box.send_keys("Selenium Python")

    # 5. Імітуємо натискання клавіші Enter
    search_box.send_keys(Keys.RETURN)

    # 6. Використовуємо явне очікування для надійності
    # Чекаємо до 10 секунд, поки елемент з ID 'search' не стане видимим
    wait = WebDriverWait(driver, 10)
    wait.until(EC.visibility_of_element_located((By.ID, "search")))

    # 7. Отримуємо заголовок сторінки з результатами
    print(f"Заголовок сторінки з результатами: {driver.title}")

except Exception as e:
    print(f"Виникла помилка: {e}")

finally:
    # 8. Закриваємо браузер
    driver.quit()
```

Приклад селектора	Що вибирає	Приклад використання у парсингу
`tag`	Усі елементи з тегом	`div`, `a`, `img`
`.class`	Усі елементи з класом	`.poster-title`
`#id`	Елемент з конкретним ID	`#main-header`
`tag.class`	Тег з конкретним класом	`div.poster-item`
`parent child`	Усі нащадки (будь-який рівень вкладеності)	`.poster-item .poster-title`

`parent > child`	Лише прямі діти	`ul > li`
`A + B`	Елемент B, який йде одразу після A	`h2 + p`
`A ~ B`	Усі елементи B після A (на одному рівні)	`h2 ~ p`
`[attr]`	Елементи, які мають атрибут	`a[href]`
`[attr="value"]`	Елементи з точним значенням атрибута	`a[href="/movies"]`
`[attr^="value"]`	Атрибут починається з `value`	`a[href^="/movie/"]`
`[attr$="value"]`	Атрибут закінчується на `value`	`img[src$=".jpg"]`
`[attr*="value"]`	Атрибут містить `value`	`a[href*="multiplex"]`
`:first-child`	Перший елемент у батьківському блоці	`ul li:first-child`
`:last-child`	Останній елемент у батьківському блоці	`ul li:last-child`
`:nth-child(n)`	Елемент з конкретним номером (або `odd` / `even`)	`tr:nth-child(2)`
`:not(selector)`	Елемент, який **не** відповідає селектору	`div:not(.active)`
`tag.class[attr*="val"]`	Комбінація тегу, класу та атрибуту	`a.btn[href*="/tickets"]`

Частина 3: Практика з Beautiful Soup та експорт у CSV (Розширений приклад)

На цьому етапі ми застосуємо отримані знання, щоб отримати інформацію про фільми з сайту **Multiplex** і зберегти її у файл CSV.

Чому Beautiful Soup?

Сайт multiplex.ua завантажує основний контент фільмів одразу, без потреби в прокрутці чи кліках. Тому **Beautiful Soup** і requests є ідеальним інструментом для цього завдання.

Що ми будемо витягувати?

- **Назва фільму**
- **Жанр**
- **Тривалість**
- **Віковий рейтинг**
- **URL постера**
- **URL сторінки фільму**
- **Часи сеансів**

Покрокова інструкція

1. **Імпортуємо** необхідні бібліотеки: requests, BeautifulSoup та csv.
2. **Завантажуємо** HTML-вміст сторінки.
3. **Парсимо** HTML, щоб знайти елементи, які містять інформацію про фільми.
4. **Створюємо** файл CSV та записуємо в нього заголовок.
5. **Перебираємо** знайдені елементи, витягуємо потрібні дані та **записуємо** їх у файл.

```
import requests
from bs4 import BeautifulSoup
import csv

# URL сторінки кінотеатру
URL = 'https://multiplex.ua/'

print("== Скрепінг Multiplex та експорт у CSV ==")

try:
    # 1. Завантажуємо HTML-код сторінки
```

```
    response = requests.get(URL)
    response.raise_for_status()
    soup = BeautifulSoup(response.text, 'html.parser')

    # 2. Знаходимо всі контейнери з фільмами.
    # Класи можуть змінюватися, тому важливо перевіряти структуру сайту
    movies_containers = soup.find_all('div', class_='movie-item')

    if not movies_containers:
        print("Не вдалося знайти контейнери з фільмами. Перевірте класи
елементів.")
        exit()

    # 3. Створюємо CSV-файл для запису
    with open('movies.csv', 'w', newline='', encoding='utf-8') as csvfile:
        fieldnames = ['Назва', 'Жанр', 'Тривалість', 'Віковий рейтинг', 'URL
постера', 'URL фільму', 'Часи сеансу']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        # Записуємо заголовок
        writer.writeheader()

        # 4. Перебираємо кожен контейнер і витягуємо дані
        for movie in movies_containers:
            # Знаходимо назву фільму
            title_tag = movie.find('div', class_='movie-title')
            movie_title = title_tag.text.strip() if title_tag else 'Назва
не знайдена'

            # Знаходимо URL сторінки фільму
            movie_link_tag = movie.find('a', class_='movie-item-link')
            movie_url = f"https://multiplex.ua{movie_link_tag.get('href')}"
if movie_link_tag and movie_link_tag.get('href') else 'URL не знайдено'

            # Знаходимо URL постера
            poster_tag = movie.find('img', class_='movie-item-poster')
            poster_url = poster_tag.get('src') if poster_tag and
poster_tag.get('src') else 'URL постера не знайдено'

            # Знаходимо інформацію про фільм (жанр, тривалість, рейтинг)
            movie_info_tags = movie.find_all('div',
class_='movie-info-item')
```

```
            genre = 'Жанр не знайдено'
            duration = 'Тривалість не знайдено'
            rating = 'Рейтинг не знайдено'

            for info_item in movie_info_tags:
                if 'жанр' in info_item.text.lower():
                    genre = info_item.text.replace('Жанр', '').strip()
                elif 'хв' in info_item.text.lower():
                    duration = info_item.text.strip()
                elif 'вік' in info_item.text.lower():
                    rating = info_item.text.strip()

            # Знаходимо всі сеанси
            showtime_tags = movie.find_all('a', class_='showtime-item')
            showtimes = [time.text.strip() for time in showtime_tags]
            showtimes_str = ', '.join(showtimes) if showtimes else 'Сеансів
немає'

            # 5. Записуємо дані в CSV
            writer.writerow({
                'Назва': movie_title,
                'Жанр': genre,
                'Тривалість': duration,
                'Віковий рейтинг': rating,
                'URL постера': poster_url,
                'URL фільму': movie_url,
                'Часи сеансу': showtimes_str
            })

    print("\nДані успішно збережено у файл movies.csv")

except requests.exceptions.RequestException as e:
    print(f"Помилка завантаження сторінки: {e}")
except Exception as e:
    print(f"Виникла помилка під час обробки: {e}")
```

Частина 4: Коли використовувати Selenium на Multiplex?

На головній сторінці Multiplex ми бачимо, що весь контент завантажується одразу. Але

що, якби, наприклад, нам потрібно було:

- Отримати відгуки про фільм, які з'являються тільки після кліку на нього.
- Змінити місто у випадаючому списку, щоб побачити розклад сеансів в іншому кінотеатрі.
- Увійти на сайт, щоб отримати доступ до особистих даних.

У таких випадках Beautiful Soup безсилий, і тут на допомогу приходить **Selenium**.

Підсумок:

- **Beautiful Soup** ідеально підходить для швидкого аналізу **статичного** HTML-коду, що дозволило нам витягти багато інформації з головної сторінки.
- **Selenium** необхідний для роботи з **динамічним** контентом та імітації взаємодії з користувачем, що дозволить нам отримати доступ до інформації, прихованої за кліками чи іншими діями.
- Їх поєднання дозволяє вирішувати найскладніші завдання веб-скрепінгу, поєднуючи переваги обох бібліотек.