DOI: https://doi.org/10.15276/ict.01.2024.08
УДК 004.896

Розробка та впровадження ефективних методів веб-скрапінгу для автоматизованого збору і обробки даних з використанням Python

Січкарюк Руслан Костянтинович[1)]
Магістр каф. Інформатики і комп'ютерних наук
ORCID: https://orcid.org/0000-0001-8353-7679; sichkaryk39@gmail.com. Scopus Author ID: 57195531548
Корніловська Наталя Володимирівна[1)]
Канд. техніч. наук, доцент каф. Інформатики і комп'ютерних наук
ORCID: https://orcid.org/0000-0002-8331-8027; pypsiknata2015@gmail.com. Scopus Author ID:57195531548
Лур'є Ірина Анатолієвна[1)]
Канд. техніч. наук, доцент каф. Інформатики і комп'ютерних наук
ORCID: https://orcid.org/0000-0001-8915-728X; lurieira@gmail.com. Scopus Author ID:57204941407
Вороненко Марія Олександрівна
Канд. техніч. наук, доцент каф. Інформатики і комп'ютерних наук
ORCID: https://orcid.org/ 0000−0002−5392−5125; mary_voronenko@i.ua. Scopus Author ID:57200175137
[1)] Херсонський національний технічний університетм, вул. Інститутська, 11. Хмельницький, 29016, Україна

АНОТАЦІЯ

З кожним роком процес цифровізації суспільства набирає обертів, що призводить до значного зростання попиту на оброблену та проаналізовану інформацію. У сучасному світі дані стали важливим ресурсом, а здатність швидко знаходити й аналізувати велику кількість інформації є важливою конкурентною перевагою для компаній, дослідників і аналітиків. У цьому контексті веб-скрапінг стає важливим інструментом, який дозволяє ефективно збирати дані з різних інтернет-джерел для подальшого аналізу та прийняття обґрунтованих рішень. У цій роботі розглядаються сучасні досягнення в області розробки та впровадження ефективних методів веб-скрапінгу для автоматичного збору та обробки даних за допомогою Python. Використання новітніх бібліотек Python, таких як BeautifulSoup, Selenium і Scrapy, дозволяє досягти високої швидкості і точності збору даних з різних веб-джерел, що охоплюють вторинний ринок. Запропоновані алгоритми знижують ризик блокування сайтів, забезпечують стабільність і надійність збору даних у різних ситуаціях. Крім того, в роботі велика увага приділяється автоматизації процесу збору даних, що досягається за рахунок розробки автоматизованих скриптів і впровадження програм планування роботи, таких як cron jobs. Це забезпечує постійне оновлення бази даних і збір нової інформації без необхідності ручного втручання. Особливий акцент робиться на обробці та очищенні зібраних даних, особливо на розробці методів фільтрації непотрібної інформації, дублювання та шуму, що сприяє покращенню якості даних. Ефективне використання зібраних даних показує їх цінність для аналізу ринку, оцінки потреб і прогнозування якості, підкреслюючи важливість використання розробленого методу. Дослідження містить приклади реальних випадків використання даних у різних сферах, таких як маркетинг, економіка та бізнес-аналіз. У цій роботі проведено порівняльний аналіз різних методів збору даних, що дає змогу оцінити ефективність і надійність запропонованих рішень.

Ключові слова: Веб-скрапінг; Python; автоматизація збору даних; вторинний ринок; обробка даних; аналіз ринку; завдання cron; BeautifulSoup; Selenium, Scrapy

Метою дослідження є розробка та впровадження ефективних методів веб-скрапінгу для автоматизованого збору і обробки даних з різноманітних онлайн-джерел з використанням Python [1]. Дослідження також спрямоване на оптимізацію процесу збору даних, забезпечення його стабільності та надійності, а також на підвищення якості зібраних даних через фільтрацію, очищення та структурування інформації. Отримані результати повинні мати практичне значення для ринкового аналізу, прогнозування тенденцій і прийняття обґрунтованих рішень у галузях маркетингу, економіки та бізнес-аналітики.

На даний момент для веб-скрапінгу розробники використовують багато різноманітних інструментів, кожен з яких відрізняється за мовою програмування, технологією, швидкістю обробки даних та рівнем складності [2]. Ключовим фактором при виборі інструмента для скрапінгу є специфічні потреби проєкту, які можуть включати обсяг даних, частоту збору інформації, ступінь інтерактивності веб-сторінок, а також рівень технічної підготовки

розробника [3]. Наприклад, для відносно простих завдань, таких як вилучення інформації з статичних веб-сторінок, доцільно використовувати бібліотеки, такі як BeautifulSoup [2]. Ця бібліотека забезпечує інтуїтивно зрозумілий інтерфейс для аналізу HTML і XML документів, що дозволяє швидко створити ефективний скрипт для парсингу.

Однак, у випадках, коли веб-сторінки містять динамічний контент, що генерується за допомогою JavaScript, прості підходи стають менш ефективними. Тут на допомогу приходять такі інструменти, як Selenium [4] або Puppeteer, які надають можливість автоматизувати взаємодію з веб-сторінками на рівні браузера. Вони дозволяють імітувати поведінку користувача, таку як кліки по кнопках або прокрутка сторінки, що робить їх ідеальними для скрапінгу складних, динамічних сайтів.

З іншого боку, у великих проєктах, які потребують одночасного збору даних з тисяч веб-ресурсів, найкращим вибором є потужні фреймворки, такі як Scrapy [3]. Цей інструмент забезпечує високу продуктивність завдяки можливості працювати з великим обсягом запитів одночасно, використовуючи асинхронну архітектуру. Scrapy дозволяє гнучко налаштовувати процес збору даних за допомогою користувацьких правил та модулів, що робить його незамінним для масштабних задач. Проте, кожен із цих інструментів має свої обмеження, і важливо правильно обирати підхід до збору даних з урахуванням специфіки завдання.

У рамках даного дослідження було розроблено комплексний підхід до веб-скрапінгу, що включає використання декількох інструментів одночасно для досягнення оптимальних результатів у різних умовах. Такий підхід дозволяє поєднувати переваги різних технологій, знижуючи їхні недоліки, та забезпечуючи високу швидкість, точність і стабільність процесу збору даних.

Давайте розглянемо бібліотеку BeautifulSoup більш детально. Цей інструмент є ідеальним вибором для простих веб-сторінок, де текстова інформація не залежить від роботи JavaScript. BeautifulSoup дозволяє розробникам швидко створити скрипт для вилучення даних, забезпечуючи простоту та ефективність у роботі з HTML і XML документами.

Нижче наведено простий приклад використання цієї бібліотеки для парсингу інформації.

```
import requests
from bs4 import BeautifulSoup

URL = 'https://www.olx.ua/uk/hobbi-otdyh-i-sport/muzykalnye-instrumenty/?page=1'
req = requests.get(URL)
soup = BeautifulSoup(req.text, "html.parser")

items = soup.find_all('div', class_='css-1sw7q4x')
for item in items:
   try:
      title = item.find('h6', class_='css-1wxaaza').text
      price = item.find('p', class_='css-13afqrm').text
      print(f'{price} ---- {title}')
   except AttributeError:
      print('немає даних')
```

Лістинг 1. Парсинг "www.olx.ua" за допомогою BeautifulSoup

Вище зазначений код демонструє простоту та легкість використання бібліотеки BeautifulSoup для веб-скрапінгу. На прикладі веб-ресурсу «www.olx.ua» показано, як можна отримати дані про музичні інструменти, використовуючи лише кілька рядків коду. У даному

випадку використано всього два методи: "find" та "find_all", які дозволяють ефективно здійснювати пошук необхідних елементів на веб-сторінці. Метод "find" шукає перший елемент, що відповідає вказаним критеріям, тоді як "find_all" дозволяє знайти всі елементи, що відповідають заданим умовам.

У коді зазначено, що ці методи використовуються для пошуку HTML-тегів та їхніх атрибутів, таких як "class", "id", "name" та інших. Наприклад, для отримання назв та цін музичних інструментів з веб-сторінки використано теги h6 та p, кожен з яких має свій унікальний клас. Використовуючи ці дані, бібліотека BeautifulSoup дозволяє швидко витягти та вивести необхідну інформацію.

Таким чином, BeautifulSoup є чудовим вибором для проєктів, де потрібно швидко і просто отримати дані з веб-сторінок, що мають статичний контент. Однак, коли мова йде про великомасштабні та складні проєкти, де парсинг є основною задачею, варто звернути увагу на більш потужні інструменти, такі як фреймворк Scrapy. Цей фреймворк зазвичай використовується для обробки великих обсягів даних та складних завдань парсингу.

Приклад виконання аналогічного завдання з використанням Scrapy наведено нижче.

```
import scrapy

class OlxInstrumentsSpider(scrapy.Spider):
    name = "olx_instruments"
    start_urls = [
        'https://www.olx.ua/uk/hobbi-otdyh-i-sport/muzykalnye-instrumenty/?page=1',
    ]

    def __init__(self, *args, **kwargs):
        super(OlxInstrumentsSpider, self).__init__(*args, **kwargs)
        self.start_time = time.time()

    def parse(self, response):
        items = response.css('div.css-1sw7q4x')
        for item in items:
            try:
                title = item.css('h6.css-1wxaaza::text').get()
                price = item.css('p.css-13afqrm::text').get()
                yield {
                    'title': title,
                    'price': price,
                }
            except AttributeError:
                yield {
                    'title': 'немає даних',
                    'price': 'немає даних',
                }
```

Лістинг 2. Парсинг "www.olx.ua" за допомогою Scrapy

Даний код також не є складним, що підтверджує схожість синтаксису та загальної структури між різними інструментами для веб-скрапінгу. В цьому випадку використовується фреймворк Scrapy, який дозволяє створювати скрейпери у вигляді класів. Це забезпечує

більш структурований підхід до парсингу, особливо при роботі з великими обсягами даних

У коді створено клас OlxInstrumentsSpider, який відповідає за обробку веб-сторінок. Вказавши URL-адресу сторінки, де розміщені дані, що нас цікавлять, ми можемо використовувати метод parse для пошуку та вилучення необхідної інформації. Scrapy дозволяє легко витягувати елементи за допомогою CSS-селекторів, що робить процес парсингу досить зручним.

Цей приклад демонструє, що читабельність і простота коду зберігаються незалежно від вибору інструменту. Далі давайте порівняємо результати роботи одного і того ж завдання, виконаного різними підходами до парсингу. Зазначимо, що всі тести проводилися на одній платформі, щоб уникнути розбіжностей через апаратні відмінності (Таблиця).

Для більш наглядної ілюстрації різниці в часі виконання ми створили діаграму на основі отриманих результатів. Завдяки цій діаграмі можна побачити, що BeautifulSoup майже вдвічі швидший, ніж Scrapy, у виконанні тих самих завдань. Це пояснюється тим, що BeautifulSoup краще підходить для простих, швидких парсингів з окремих сторінок, де немає необхідності у великомасштабному зборі даних.

Проте варто зазначити, що Scrapy створений для більш складних проєктів, де парсинг даних виконується одночасно з багатьох ресурсів. Крім того, Scrapy має вбудовані функції експорту даних у популярних форматах, що спрощує їх подальший аналіз. Також фреймворк надає можливість використання окремого файлу налаштувань, у якому можна вказати проксі та User-Agent для емуляції стандартного браузера, що робить його більш гнучким і потужним інструментом для складних завдань.

Таблиця. **Порівняння часу виконання однакових функцій рідними бібліотеками**

№ спроби	Час виконання BeautifulSoup	Час виконання Scrapy
1	3.2230255603790283 с.	4.373951196670532 с.
2	2.7165629863739014 с.	5.912944078445435 с.
3	3.2810637950897217 с.	5.7965922355651855 с.
4	2.810953378677368 с.	4.821292877197266 с.
5	3.222365617752075 с.	5.442413091659546 с.
6	2.9286129474639893 с.	5.51770544052124

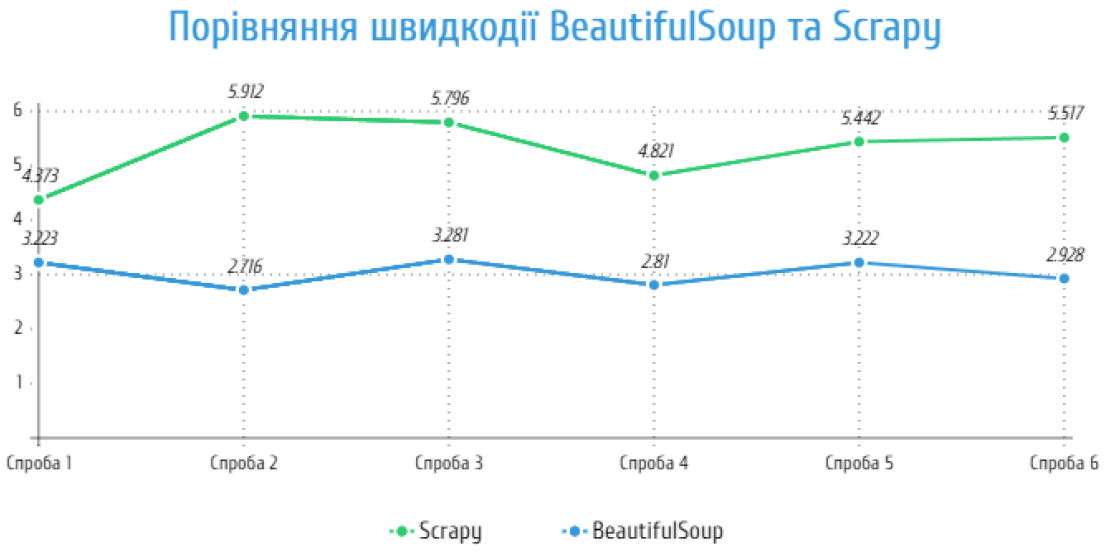

Рис 1. **Порівняння часу виконання однакових функцій рідними бібліотеками**

Ще одним потужним інструментом для веб-скрапінгу є Selenium. Він значно відрізняється від BeautifulSoup та Scrapy тим, що дозволяє автоматизувати взаємодію з веб-сторінками на рівні браузера. Це робить Selenium ідеальним інструментом для роботи з динамічними веб-сайтами, де контент генерується за допомогою JavaScript.

Selenium дозволяє імітувати дії користувача, такі як натискання кнопок, введення тексту, прокручування сторінок тощо. Це дуже корисно, коли потрібно отримати дані з веб-сторінок, де інформація завантажується асинхронно або з'являється лише після взаємодії з елементами сторінки.

На відміну від BeautifulSoup, який працює виключно з HTML-кодом сторінки, або Scrapy, який оптимізований для масового збору даних, Selenium забезпечує доступ до динамічного контенту і може працювати з різними браузерами. Це робить його незамінним для більш складних завдань парсингу, де важливо мати доступ до повного контенту сторінки, включаючи той, що генерується на стороні клієнта.

Нижче наведено приклад коду, який виконує завдання, схоже на попередні приклади, але за допомогою Selenium:

```
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.get('https://www.olx.ua/uk/hobbi-otdyh-i-sport/muzykalnye-instrumenty/?page=1')
items = driver.find_elements(By.CSS_SELECTOR, 'div.css-1sw7q4x')

for item in items:
    try:
        title = item.find_element(By.CSS_SELECTOR, 'h6.css-1vwxaza').text
        price = item.find_element(By.CSS_SELECTOR, 'p.css-13afqrm').text
        print(f'{price} ---- {title}')
    except Exception:
        print('немає даних')

driver.quit()
```

Лістинг 3. Парсинг "www.olx.ua" за допомогою Scrapy

Даний код демонструє, що синтаксис Selenium не дуже відрізняється від попередніх прикладів, але функціонал і призначення цієї бібліотеки кардинально інші. Selenium є повільнішою порівняно з іншими інструментами, проте його метод роботи стає незамінним у найскладніших ситуаціях парсингу окремих веб-ресурсів, таких як ті, де контент генерується за допомогою JavaScript. Порівнювати час виконання скриптів у цьому випадку немає сенсу, оскільки тривалість виконання може вимірюватися не в секундах, а в десятках секунд. Проте це не впливає на якість роботи цієї бібліотеки.

Після вибору інструмента для парсингу даних наступним кроком стає автоматизація процесу збору інформації, що є ключовим аспектом для багатьох проєктів, які потребують регулярного оновлення даних [5]. У цьому контексті важливим інструментом є cron — планувальник завдань, що використовується в Unix-подібних операційних системах [6].

Cron дозволяє налаштувати періодичний запуск скриптів для збору даних без необхідності ручного втручання. Наприклад, використовуючи cron, можна налаштувати запуск скрипта для парсингу даних щогодини, щодня або у будь-який інший зручний час. Це особливо корисно для проєктів, де необхідно постійно оновлювати бази даних або отримувати актуальну інформацію з веб-ресурсів.

Інтеграція cron із скриптами BeautifulSoup або Scrapy дозволяє автоматизувати процеси збору даних, що не лише знімає навантаження з розробників, але й забезпечує своєчасне оновлення інформації. Завдяки цьому підходу можна бути впевненим у тому, що зібрані дані завжди залишаються актуальними, що є критично важливим для аналізу ринку, прогнозування тенденцій та прийняття обґрунтованих рішень.

Таким чином, використання cron як інструмента для автоматизації процесів значно підвищує ефективність збору даних, особливо в умовах зростаючих обсягів інформації та необхідності у регулярних оновленнях.

Висновок. У даній статті було розглянуто різні підходи та інструменти для веб-скрапінгу, такі як BeautifulSoup, Scrapy та Selenium. Кожен інструмент має свої переваги та обмеження. BeautifulSoup виявився чудовим інструментом для швидкого і простого парсингу статичних веб-сторінок. Scrapy забезпечує високу продуктивність при роботі з великими обсягами даних. Selenium, з іншого боку, є незамінним інструментом для роботи з динамічними веб-сторінками, де контент генерується JavaScript.

Крім того, було підкреслено важливість автоматизації процесу збору даних за допомогою cron. Інтеграція cron із скриптами для веб-скрапінгу дозволяє забезпечити постійне оновлення даних та зменшити навантаження на розробників. Таким чином, використання комбінованого підходу, що включає різні інструменти для парсингу та автоматизації, дозволяє досягти високої ефективності у зборі та обробці даних.

Результати дослідження демонструють, що правильний вибір інструментів і методів веб-скрапінгу є ключовим фактором для досягнення оптимальних результатів у різних умовах. Це дослідження може бути корисним для розробників, аналітиків та дослідників, які працюють із великими обсягами даних і потребують надійних та ефективних методів їхнього збору та обробки.

СПИСОК ЛІТЕРАТУРИ

1. Martelli A. "Python Cookbook: Recipes for Mastering Python 3". *O'Reilly Media*. 2017.
2. Копей В. Б. «Мова програмування Python для інженерів і науковців». Навчальний посібник. *Івано-Франківськ, Україна. ІФНТУНГ*. 2019. С 56–62.
3. Sargent J., Nelson A. "Mastering Web Scraping with Python". *Packt Publishing*. 2015.
4. Aggarwal C. C. "Neural networks and deep learning". *A Textbook. Springer International Publishing AG*. 2018. DOI: https://doi.org/10.1007/978-3-319-94463-03.
5. VanderPlas J. "Python data science handbook: Essential tools for working with data". *O'Reilly Media.* 2016. p. 248–278.
6. Kumari S. "Linux Shell Scripting Essentials". 2015. p. 204–262.

DOI: https://doi.org/10.15276/ict.01.2024.08
UDC 004.896

Development and implementation of effective Web Scraping methods for automated data collection and processing using Python

Ruslan K. Sichkariuk[1)]
Master, Department of Informatics and Computer Science
ORCID: https://orcid.org/0000-0001-8353-7679; sichkaryk39@gmail.com. Scopus Author ID: 57195531548

Natalia V. Kornilovska[1)]
PhD, Associate Professor, Department of Informatics and Computer Science

ORCID: https://orcid.org/0000-0002-8331-8027; pypsiknata2015@gmail.com. Scopus Author ID: 57195531548
Iryna A. Lurie[1)]
PhD, Associate Professor, Department of Informatics and Computer Science
ORCID: https://orcid.org/0000-0001-8915-728X; lurieira@gmail.com. Scopus Author ID: 57204941407
Maria A. Voronenko[1)]
PhD, Associate Professor, Department of Informatics and Computer Science
ORCID: https://orcid.org/ 0000−0002−5392−5125; mary_voronenko@i.ua. Scopus Author ID: 57200175137
[1)] Kherson National Technical University, 11, Instytutska Street. Khmelnytskyi, 29016, Ukraine

ABSTRACT

With each passing year, the process of digitalization in society is accelerating, leading to a significant increase in demand for processed and analyzed information. In today's world, data has become a valuable resource, and the ability to quickly find and analyze large amounts of information is a key competitive advantage for companies, researchers, and analysts. In this context, web scraping has become an important tool, enabling the efficient collection of data from various online sources for further analysis and informed decision-making. This paper examines the latest advancements in the development and implementation of effective web scraping methods for automatic data collection and processing using Python. The use of the latest Python libraries, such as BeautifulSoup, Selenium, and Scrapy, allows for high-speed and accurate data collection from various web sources, particularly in secondary markets. The proposed algorithms reduce the risk of site blocking, ensuring the stability and reliability of data collection in various situations. Additionally, the paper places great emphasis on automating the data collection process through the development of automated scripts and the implementation of job scheduling programs, such as cron jobs. This ensures continuous database updates and the collection of new information without manual intervention. Special attention is given to the processing and cleaning of collected data, particularly in the development of methods for filtering out unnecessary information, duplicates, and noise, which enhances data quality. The efficient use of the collected data demonstrates its value for market analysis, demand assessment, and quality forecasting, highlighting the importance of the proposed method. The research includes examples of real-world data use cases in various fields such as marketing, economics, and business analysis. A comparative analysis of different data collection methods is also provided, allowing for the assessment of the effectiveness and reliability of the proposed solutions.

Keywords: Web scraping; Python; data collection automation; secondary market; data processing; market analysis; cron jobs; BeautifulSoup; Selenium; Scrapy.