Міністерство освіти і науки України
Львівський національний університет імені Івана Франка

Звіт
про виконання лабораторної роботи №7
на тему:

«Обробка CSV - файлу використовуючи бібліотеку async у Python»

Виконала:

Студентка групи ФеС-32

Філь Дарина

Львів - 2025

Мета роботи:

Навчитися парсити великі CSV - файли асинхронно.

Теоретичні відомості:

Синхронний підхід – це спосіб виконання програми, в якому всі інструкції виконуються послідовно, одна за одною. Кожна операція блокує виконання наступної, поки не завершиться.

Якщо одна операція займає багато часу (наприклад, очікування відповіді від сервера), вся програма зупиняється до завершення цієї операції.

Асинхронний підхід у Python:

Асинхронне програмування дозволяє виконувати кілька завдань одночасно без блокування головного потоку. Це особливо корисно для I/O-операцій (наприклад, робота з мережею, файлами, базами даних), де програма зазвичай чекає відповіді.

У Python асинхронний підхід реалізується через корутини та бібліотеку asyncio. Основна ідея – використовувати async і await, щоб дозволити іншим задачам виконуватись під час очікування результату.

Основні принципи роботи:

1. Корутини

○ Функції, що містять async def, є корутинами і виконуються асинхронно.

○ асинхронні функції, які можна призупиняти і відновлювати.

○ Їх можна викликати через await, що дозволяє іншим завданням виконуватися паралельно.

2. Події та цикл подій

○ asyncio використовує цикл подій (event loop), який керує виконанням корутин та інших асинхронних завдань.

○ Всі async функції виконуються всередині цього циклу.

3. Завдання (Task)

○ Корутини можна запускати як asyncio.Task, щоб вони виконувалися одночасно.

○ asyncio.gather() дозволяє запустити кілька завдань одночасно.

4. Асинхронні таймери та I/O

○ Асинхронний ввід/вивід, таймери (asyncio.sleep()), запити до мережі виконуються неблокуюче.

Припустимо, потрібно зробити кілька HTTP-запитів до сервера. Якщо використовувати синхронний код, то кожен запит чекатиме завершення попереднього, що займає багато часу. asyncio дозволяє запускати їх одночасно.Що таке реактивне програмування в Python?

Реактивне програмування (Reactive Programming) — це підхід до програмування, у якому обробка подій відбувається асинхронно та потоково. Воно базується на потоках даних (observables) і операторах для їх трансформації. У Python для реактивного програмування використовується бібліотека RxPY (rx), яка є адаптацією ReactiveX для Python.

Основні принципи реактивного програмування:

1. Observables (спостережувані потоки) — джерела даних, які можуть надсилати значення у часі.

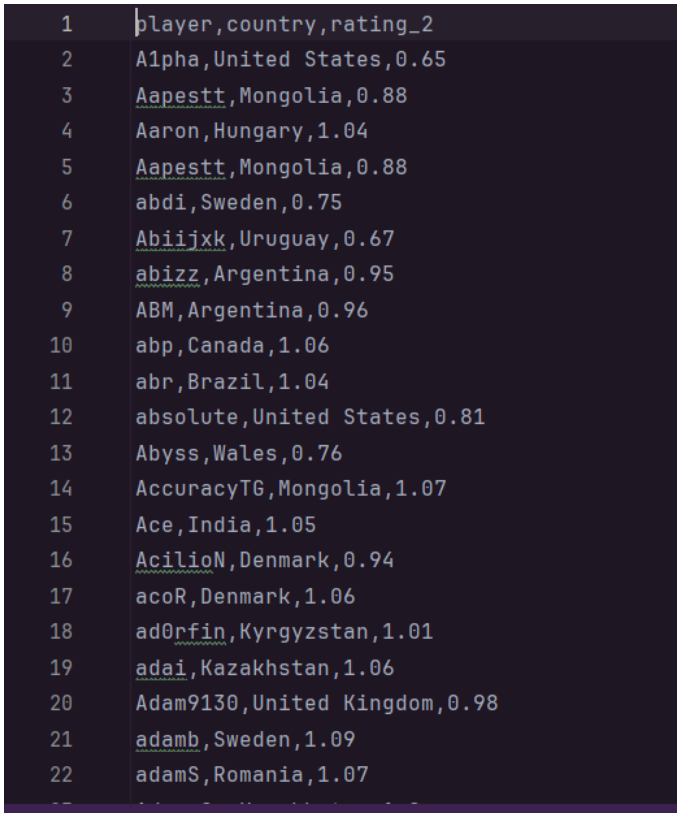
2. Operators (оператори трансформації) — функції, які змінюють потік даних (map(), filter(), flat_map()тощо).

3. Subscribers (споживачі) — код, який підписується (subscribe()) і обробляє значення потоку.

4. Schedulers (планувальники) — механізми керування виконанням (асинхронно, в окремих потоках тощо).

Хід роботи:

1. Отримати від викладача великий csv файл. Збільшити вміст файлу до 50 тисяч рядків, використовуючи копіювання



```
player,country,rating_2
A1pha,United States,0.65
Aapestt,Mongolia,0.88
Aaron,Hungary,1.04
Aapestt,Mongolia,0.88
abdi,Sweden,0.75
Abiijxk,Uruguay,0.67
abizz,Argentina,0.95
ABM,Argentina,0.96
abp,Canada,1.06
abr,Brazil,1.04
absolute,United States,0.81
Abyss,Wales,0.76
AccuracyTG,Mongolia,1.07
Ace,India,1.05
AcilioN,Denmark,0.94
acoR,Denmark,1.06
adOrfin,Kyrgyzstan,1.01
adai,Kazakhstan,1.06
Adam9130,United Kingdom,0.98
adamb,Sweden,1.09
adamS,Romania,1.07
```

2. Написати програму, яка читає даний файл і підраховує кількість рядків де country=Ukraine (без використання бібліотеки pandas).

```
def default_func():  1 usage
    with open(r'E:\СПП\Lab7\csv2.csv', 'r', encoding='utf8') as file:
        start = time.time()
        print('--- default ---')

        reader = csv.DictReader(file)
        count = sum(1 for row in reader if row['country'] == 'Ukraine')

        print(f'Ukraine count: {count}')
        print(f'Time: {round(time.time() - start, 4)} sec')
```

3. Використовуючи бібліотеки aiofiles та asyncio повторити п.2

```python
async def async_func():  1 usage
    async with aiofiles.open( file: r'E:\СПП\Lab7\csv2.csv',  mode: 'r', encoding='utf8') as file:
        start = time.time()
        print('--- async ---')

        lines = await file.readlines()
        header = lines[0].strip().split(',')
        country_index = header.index('country')

        count = sum(1 for line in lines[1:] if line.strip().split(',')[country_index] == 'Ukraine')

        print(f'Ukraine count: {count}')
        print(f'Time: {round(time.time() - start, 4)} sec')
```

4. Повторити п.2, використовуючи бібліотеку для реактивного програмування rx

```python
async def rx_func():  1 usage
    async with aiofiles.open( file: r'E:\СПП\Lab7\csv2.csv',  mode: 'r', encoding='utf8') as file:
        start = time.time()
        print('--- rx ---')

        lines = await file.readlines()
        header = lines[0].strip().split(',')
        country_index = header.index('country')
        data = lines[1:]

        rx.from_(data).pipe(
            ops.map(lambda line: line.strip().split(',')),
            ops.filter(lambda parts: len(parts) > country_index and parts[country_index] == 'Ukraine'),
            ops.count()
        ).subscribe(lambda count: print(f'Ukraine count: {count}'))

        print(f'Time: {round(time.time() - start, 4)} sec')
```

5. Використовуючи відповідні засоби, порівняти швидкодію виконання програми, дані навести в звіті та пояснити результати виконання програми

```
--- default ---
Ukraine count: 2393
Time: 0.061 sec

--- async ---
Ukraine count: 2393
Time: 0.0134 sec

--- rx ---
Ukraine count: 2393
Time: 0.0344 sec
```

Висновок:

Асинхронний підхід продемонстрував найкращий результат за швидкодією серед усіх трьох варіантів завдяки неблокуючому читанню файлу з використанням aiofiles, що суттєво пришвидшило обробку великого обсягу даних. Синхронне читання виявилося найповільнішим, оскільки операції введення-виведення виконуються послідовно та блокують основний потік програми. Реактивний підхід, реалізований через бібліотеку reactivex, показав кращий результат, ніж синхронний, проте все ж поступився звичайному асинхронному варіанту через додаткові витрати на побудову реактивного потоку й обробку операторів. Отже, для задач, де необхідна висока швидкість читання та проста обробка великих текстових файлів, я зробила висновок, що найефективнішим рішенням є асинхронний підхід без зайвих реактивних обгорток.

Додаток

Код програми:

```
import csv
import time
import asyncio
import aiofiles
import reactivex as rx
from reactivex import operators as ops


def default_func():
    with open(r'E:\СПП\Lab7\csv2.csv', 'r', encoding='utf8') as file:
        start = time.time()
        print('--- default ---')

        reader = csv.DictReader(file)
        count = sum(1 for row in reader if row['country'] == 'Ukraine')

        print(f'Ukraine count: {count}')
        print(f'Time: {round(time.time() - start, 4)} sec')


async def async_func():
    async with aiofiles.open(r'E:\СПП\Lab7\csv2.csv', 'r', encoding='utf8') as file:
```

```
        start = time.time()
        print('--- async ---')

        lines = await file.readlines()
        header = lines[0].strip().split(',')
        country_index = header.index('country')

        count = sum(1 for line in lines[1:] if line.strip().split(',')[country_index] ==
'Ukraine')

        print(f'Ukraine count: {count}')
        print(f'Time: {round(time.time() - start, 4)} sec')


async def rx_func():
    async with aiofiles.open(r'E:\СПП\Lab7\csv2.csv', 'r', encoding='utf8') as file:
        start = time.time()
        print('--- rx ---')

        lines = await file.readlines()
        header = lines[0].strip().split(',')
        country_index = header.index('country')
        data = lines[1:]

        rx.from_(data).pipe(
            ops.map(lambda line: line.strip().split(',')),
            ops.filter(lambda parts: len(parts) > country_index and
parts[country_index] == 'Ukraine'),
            ops.count()
        ).subscribe(lambda count: print(f'Ukraine count: {count}'))

        print(f'Time: {round(time.time() - start, 4)} sec')


# Запуск
default_func()
print()
asyncio.run(async_func())
print()
asyncio.run(rx_func())
```