Project 4: Multitasking

Due date: Midnight of Wednesday Mar 21, 2025

Project objective:

- Understand and practice multiprocessing and multithreading in Python.
- Be able to apply multitasking solutions based on problem profiles.
- Make programs run faster!

Project overview:

This is a mini-project closer to a homework assignment than a typical project. Therefore no report is needed, you can submit your answers by pasting them to this document in addition to optional .py files that you upload to the project_4 folder on Drive. Do not use Jupyter notebooks, they cause problems with multitasking.

Activity 1: Sum of Squares

You are given a function that computes the sum of squares for a range of numbers in a sequential manner. Your mission, should you choose to accept, it is to make it run faster without using unnecessary computational resources.

Here is the code:

```python
import time

def sum_of_squares(n):
    return sum([i * i for i in range(n)])

def main():
    N = 10**7  # Large number for heavy computation
    start_time = time.time()

    results = [sum_of_squares(N) for _ in range(4)]
    end_time = time.time()
    print("Sequential Execution Time:", end_time - start_time)

if __name__ == "__main__":
    main()
```

Question 1: Is this a CPU intensive task or an I/O intensive task?

Answer: This is a **CPU intensive** task. The function **sum_of_squares(n)** performs a large number of mathematical computations, which is a pure CPU-bound operation.

Question 2: Should you use Python multiprocessing or multithreading here? Which option would lead to faster execution without using unnecessary computational resources?

Answer: Since we're doing heavy CPU bound tasks, we should use multiprocessing

Question3: Keeping in mind that a solution that uses the wrong multitasking paradigm is considered incorrect, even if it runs faster than sequential code. Please write your multitasking Python solution for the problem here:

Answer: Added to the activity_1.py file

Activity 2: Data from URLs

You are given a function that fetches data from URLs sequentially using the *requests* library. Your mission, should you choose to accept, it is to make it run faster without using unnecessary computational resources.

Here is the code:

```python
import time
import requests

URLS = [
    "https://www.example.com",
    "https://www.example.org",
    "https://www.example.net",
    "https://www.example.edu",
]

def fetch_url(url):
    response = requests.get(url)
    return len(response.content)  # Return content size

def main():
    start_time = time.time()
    results = [fetch_url(url) for url in URLS]
    end_time = time.time()
    print("Sequential Execution Time:", end_time - start_time)
```

```
if __name__ == "__main__":
    main()
```

Question 1: Is this a CPU intensive task or an I/O intensive task?

Answer: This is an I/**O-intensive task.**

Question 2: Should you use Python multiprocessing or multithreading here? Which option would lead to faster execution without using unnecessary computational resources?

Answer: In this **I/O-intensive** task (network requests), you should use **Python multithreading**.

Question3: Keeping in mind that a solution that uses the wrong multitasking paradigm is considered incorrect, even if it runs faster than sequential code. Please write your multitasking Python solution for the problem here:

Answer: Added to the activity_2.py file