Week 5 task

* Download 5 URLs
* Use **threads** (parallel execution)
* Save each response into a file
* Measure **total execution time**

Here’s a Python example using threading and requests:

import threading

import requests

import time

# List of URLs to download

urls = [

"https://www.example.com",

"https://httpbin.org/get",

"https://jsonplaceholder.typicode.com/posts",

"https://www.wikipedia.org",

"https://www.python.org"

]

def download\_file(url, index):

try:

response = requests.get(url)

filename = f"output\_{index}.html"

with open(filename, "wb") as f:

f.write(response.content)

print(f"[✓] Saved {url} as {filename}")

except Exception as e:

print(f"[x] Failed {url}: {e}")

def main():

threads = []

start\_time = time.time()

# Create threads

for i, url in enumerate(urls):

t = threading.Thread(target=download\_file, args=(url, i))

threads.append(t)

t.start()

# Wait for all threads to finish

for t in threads:

t.join()

end\_time = time.time()

print(f"\nTotal time taken: {end\_time - start\_time:.2f} seconds")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**🔎 How it works:**

* Each URL is downloaded in its **own thread**.
* Files are saved as output\_0.html, output\_1.html, …
* time.time() is used to measure total duration.
* Because threads run in parallel, the total time will usually be much less than downloading sequentially

***processes***

from multiprocessing import Pool, cpu\_count

import time

def square(n):

return n \* n

def main():

# Create a large list

numbers = list(range(10\_000\_00)) # 1 million numbers

start = time.time()

# Create a Pool with number of CPU cores

with Pool(processes=cpu\_count()) as pool:

results = pool.map(square, numbers)

end = time.time()

print(f"Processed {len(numbers)} numbers")

print(f"Time taken: {end - start:.2f} seconds")

print("First 10 results:", results[:10])

if \_\_name\_\_ == "\_\_main\_\_":

main()

🔎 Explanation:

Pool(processes=cpu\_count()) → automatically uses all available CPU cores.

pool.map(square, numbers) → distributes work across processes.

Each process independently squares chunks of the list.

Much faster than a normal loop for very large lists.

Date time

1. **Take today’s date** (for example, today is 27 August 2025).
2. **Take your birthday** (for example, 15 August).
3. Compare today’s date with your birthday:
   * If your birthday this year has **not happened yet**, count the days from today until that date.
   * If your birthday has **already passed this year**, count the days until the **next year’s birthday**.

🔹 Example:

* Today = 27 August 2025
* Birthday = 15 August

Since 15 August has already passed in 2025, the next birthday will be **15 August 2026**.  
So, we count the days from **27 Aug 2025 → 15 Aug 2026 = 353 days**

In your Flask app, you can simply add a new **route** /about that will display "Hello About" on the page.

Here’s how:

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route('/')

def home():

return "Hello Home"

@app.route('/about')

def about():

return "Hello About"

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

### Explanation:

* @app.route('/about') → defines the /about URL route.
* The function about() returns the text "Hello About".
* When you run the app and visit http://127.0.0.1:5000/about, you’ll see **Hello About**.