🧵 Assignment: Multithreading in Python

🎯 **Objective:**
Develop a strong understanding of how multithreading works in Python, including the Global Interpreter Lock (GIL), and practice creating and managing threads using the `threading` module. Learn to handle concurrent execution, ensure thread safety with synchronization tools like locks, and apply multithreading to real-world scenarios such as I/O-bound operations, background processing, and shared resource access.

🟢 Level 1: Beginner – Basics of Multithreading

✅ Task 1: Introduction to Threads

- Write a Python program that prints "Hello from thread" using the `threading.Thread` class.

✅ Task 2: Multiple Threads

- Create two threads. One should print numbers from 1 to 5, and the other should print characters from 'a' to 'e'.

✅ Task 3: Thread with Function Arguments

- Write a function that accepts a name and prints "Hello `<name>` from a thread", then call it using `Thread(target=func, args=(...))`.

✅ Task 4: Identify Threads

- Print the current thread's name using `threading.current_thread()`.

🟡 Level 2: Intermediate – Synchronization & Management

✅ Task 5: Threading with Time Delays

- Use `time.sleep()` to delay output in a thread. Demonstrate how threads run in parallel.

✅ Task 6: Join Threads

- Modify your program to use `.join()` so the main thread waits for others to complete.

✅ Task 7: Thread Synchronization

- Use a `threading.Lock` to prevent race conditions in a shared counter variable accessed by multiple threads.

🧮 Assignment: Multithreading in Python

✅ Task 8: Daemon Threads

- Create a daemon thread that runs in the background (e.g., logging service) and stops when the main thread ends.

🔵 Level 3: Advanced – Real-world Use Cases

✅ Task 9: File Download Simulation

- Simulate downloading 5 files in parallel using multithreading. Use `time.sleep()` to mock download time.

✅ Task 10: Producer-Consumer Problem

- Implement a basic producer-consumer model using `queue.Queue()` and multiple threads.

✅ Task 11: Thread Pool with `concurrent.futures`

- Use `ThreadPoolExecutor` to run 10 tasks in parallel and print their results.

✅ Task 12: Thread-safe Logging

- Write a multithreaded program where each thread logs its activity to the same file safely

🔴 Level 4: Expert – Algorithmic & System Level

✅ Task 13: Web Scraper with Threads

- Use `requests` and `threading` to fetch HTML of 10 different web pages concurrently.

✅ Task 14: Thread-safe Singleton Class

- Implement a Singleton class that is safe to use with threads (use `__new__` and `Lock`).

✅ Task 15: Benchmarking with and without Threads

- Compare execution time for running CPU-bound tasks (e.g., factorial of large numbers) with and without threading.

🧵 Assignment: Multithreading in Python

✅ Task 16: Thread Deadlock Simulation

- Simulate a deadlock between two threads and resolve it using timeout or re-ordered locking.