🧮 Interview Questions : 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.

🟢 Beginner Level – Fundamentals

1. What is multithreading in Python? Why do we use it?
2. What is the difference between a process and a thread?
3. How do you create a thread in Python? Show with an example using `threading.Thread`.
4. What does `threading.current_thread()` return?
5. What happens if you start a thread more than once?
6. Explain the role of `start()` and `join()` in threading.
7. What is the default behavior of threads in Python – foreground or daemon?

🟡 Intermediate Level – Practical Applications

8. Explain the Global Interpreter Lock (GIL). How does it affect multithreading in Python?
9. In what types of programs does multithreading provide the most benefit in Python?
10. What is the difference between a daemon thread and a user thread? How do you create a daemon thread?
11. What are race conditions in multithreading? How can you avoid them?
12. How do you use `threading.Lock()` to prevent data corruption in shared resources?
13. What is the purpose of `threading.RLock()`? How is it different from a regular lock?
14. Demonstrate a producer-consumer scenario using `queue.Queue` and multiple threads.
15. What is the difference between `threading` and `multiprocessing` modules in Python?

🧮 Interview Questions : Multithreading in Python

🔵 Advanced Level – System and Concurrency Design

16. How does Python's `ThreadPoolExecutor` from `concurrent.futures` simplify multithreading?
17. Explain how to handle exceptions in threads. Do unhandled exceptions kill the main thread?
18. Write a program where multiple threads increment a shared counter. Make it thread-safe.
19. What is a deadlock? How can it occur in Python threading? Provide an example.
20. Explain the difference between `Lock`, `RLock`, `Semaphore`, and `Condition` in Python.
21. What is the role of `threading.Event`? Provide a use case.
22. How do you stop a thread gracefully in Python? Why is `Thread.stop()` not available?
23. Is it possible to run CPU-bound tasks effectively using multithreading in Python? Why or why not?

🔴 Expert Level – Concurrency Patterns & Pitfalls

24. Design a thread-safe Singleton class in Python.
25. Simulate a deadlock and demonstrate how to resolve it using lock ordering or timeout.
26. How would you design a multithreaded web scraper in Python?
27. Write a multithreaded logging system where multiple threads log messages to the same file.
28. Discuss how the GIL interacts with native extensions written in C or C++.
29. Can you use multithreading to speed up file I/O operations in Python? Give an example.
30. What are some best practices when working with threads in large Python applications?

 Interview Questions : Multithreading in Python