Multithreading in Python

Multithreading in Python allows different parts of a program to run concurrently, which can simplify the design and increase efficiency, especially for I/O-bound tasks. Python's `threading` module is used to create and manage threads. Here's a brief overview of key concepts:

- **Thread**: The smallest unit of execution within a process. Python threads are suitable for I/O-bound tasks but might not provide benefits for CPU-bound tasks due to the Global Interpreter Lock (GIL).

- **Starting a Thread**: Threads in Python can be started by creating an instance of `Thread` and calling its `start()` method.

- **Threading Module**: Provides various classes and functions to support multithreading, such as `Thread`, `Lock`, `Condition`, `Event`, and `Semaphore`.

MCQs on Python Multithreading

1. What is the correct way to import the threading module in Python?
   a) `import thread`
   b) `import threading`
   c) `import threads`
   d) `import multi-thread`
   Answer: b) `import threading`

2. Which method is used to start a new thread in Python?
   a) `run()`
   b) `start()`
   c) `execute()`
   d) `launch()`
   Answer: b) `start()`

3. What does the `join()` method do in the context of threads?
   a) Pauses the execution of the thread
   b) Stops the thread
   c) Ensures a thread does not start until another finishes
   d) Waits for the thread to complete
   Answer: d) Waits for the thread to complete

4. Which of the following is true about the Global Interpreter Lock (GIL) in Python?
   a) It allows multiple threads to execute Python bytecodes at once.
   b) It prevents multiple native threads from executing Python bytecodes at once.
   c) It speeds up multi-threaded programs.
   d) It applies to all programming languages.

Answer: b) It prevents multiple native threads from executing Python bytecodes at once.

5. Which threading method is used to run a function as a separate thread?
   a) `Thread(target=function_name)`
   b) `Thread(run=function_name)`
   c) `Thread(start=function_name)`
   d) `Thread(execute=function_name)`
   Answer: a) `Thread(target=function_name)`

6. What is the primary use of a `Lock` in multithreading?
   a) To pause all threads
   b) To manage the thread's execution state
   c) To prevent threads from executing simultaneously
   d) To terminate a thread
   Answer: c) To prevent threads from executing simultaneously

7. Which of the following is NOT a state of a thread in Python?
   a) Running
   b) Waiting
   c) Executing
   d) Stopped
   Answer: c) Executing

8. What is the default value of the `daemon` attribute for a Thread in Python?
   a) True
   b) False
   c) None
   d) Depends on the method used to create the thread
   Answer: b) False

9. Which function would you use to get the total number of active threads?
   a) `threading.active_count()`
   b) `threading.count()`
   c) `threading.total_active()`
   d) `threading.number_of_threads()`
   Answer: a) `threading.active_count()`

10. What is the effect of setting a thread as a daemon thread?
    a) The thread runs in the background and does not prevent the program from exiting.
    b) The thread has higher priority than non-daemon threads.
    c) The thread cannot access global variables.
    d) The thread runs only when the main thread is idle.
    Answer: a) The thread runs in the background and does not prevent the program from exiting.

Explanations for the Correct Answers

1. b) `import threading`
   - Explanation: The `threading` module is the correct module in Python for working with threads. The `import threading` statement is used to access the threading functionalities provided by Python. The other options either refer to non-existent modules or incorrect syntax.

2. b) `start()`
   - Explanation: The `start()` method of a `Thread` instance in Python is used to begin the execution of a thread. When `start()` is called, the thread is initiated, and its `run()` method is executed in a separate thread of control. The other methods listed do not exist for starting threads in Python's threading module.

3. d) Waits for the thread to complete
   - Explanation: The `join()` method is used to make the calling thread wait until the thread on which `join()` has been called is terminated. Using `join()` ensures that the main program waits for all threads to complete before it finishes.

4. b) It prevents multiple native threads from executing Python bytecodes at once.
   - Explanation: The Global Interpreter Lock (GIL) is a mutex that protects access to Python objects, preventing multiple threads from executing Python bytecodes simultaneously. This lock is necessary because Python's memory management is not thread-safe. The GIL can be a limitation in CPU-bound multi-threaded programs because it allows only one thread to execute at a time.

5. a) `Thread(target=function_name)`
   - Explanation: In Python's threading module, the `Thread` class is instantiated with the `target` argument to specify the function that will run in the thread. The `target` parameter takes the function name to be executed by the thread. The other options are incorrect syntax for creating a thread.

6. c) To prevent threads from executing simultaneously
   - Explanation: A `Lock` is a synchronization primitive that is used in multithreading to ensure that only one thread can enter a critical section of code at a time. This is crucial for preventing race conditions where multiple threads modify shared data.

7. c) Executing
   - Explanation: In Python threading, typical thread states include "Running", "Waiting", and "Stopped". "Executing" is not a defined state for threads in Python. The distinction between "Running" and "Executing" is important as "Running" refers to a thread that has been started and is eligible to run under the scheduler's control.

8. b) False
   - Explanation: By default, threads in Python are not daemon threads; their `daemon` attribute is set to `False`. This means that the program will wait for non-daemon threads to complete before it terminates. Daemon threads, on the other hand, are killed abruptly at program termination.

9. a) `threading.active_count()`
   - Explanation: The function `threading.active_count()` returns the number of Thread objects currently alive. It includes all threads regardless of their daemon status and whether they have been started or not. This function is useful for monitoring and debugging purposes.

10. a) The thread runs in the background and does not prevent the program from exiting.
    - Explanation: Daemon threads in Python are designed to run in the background and do not prevent the program from exiting. When the main program exits, all daemon threads are killed. Non-daemon threads, however, must complete their execution before the main program can terminate. This behavior is useful for threads that perform background tasks, such as monitoring or housekeeping functions.