Q1. What is multithreading in python? Why is it used? Name the module used to handle threads in python

""" Multithreading in Python refers to the concurrent execution of multiple threads within the same process. Used for various purposes :

- Concurrency
- Simplified Design
- Parallelism

Multithreading in Python is used to achieve concurrency, parallelism, and better program structure, and the threading module is employed to handle threads in Python. """

Q2.

""" The threading module in Python is used to create and manage threads.

threading.activeCount(): This function returns the number of Thread objects currently alive. It counts all Thread objects, including the main thread and any additional threads you've created.

threading.currentThread(): This function returns the currently executing Thread object.

threading.enumerate(): This function returns a list of all Thread objects currently alive. It provides a way to iterate over and access information about all the threads in your program

"""

Q3.

""" The run() method is not typically called directly by the programmer. Instead, it is the method that represents the code to be executed by a thread. You should subclass the Thread class and override the run() method in your own class.

The start() method is used to begin the execution of a thread. It initializes the thread and calls the run() method asynchronously.

The join() method is used to wait for a thread to complete its execution. When you call join() on a thread, the program will block until the thread has finished.

The isAlive() method is used to check whether a thread is currently running or has finished. It returns True if the thread is still running and False if it has completed. """

Q4.

In [3]:
```
import threading

def print_squares():
    for i in range(1, 6):
        print(f"Square of {i} is {i**2}")

def print_cubes():
    for i in range(1, 6):
        print(f"Cube of {i} is {i**3}")

thread1 = threading.Thread(target=print_squares)
thread2 = threading.Thread(target=print_cubes)

thread1.start()
thread2.start()

thread1.join()
thread2.join()

print("Both threads have finished.")
```

```
Square of 1 is 1
Square of 2 is 4
Square of 3 is 9
Square of 4 is 16
Square of 5 is 25
Cube of 1 is 1
Cube of 2 is 8
Cube of 3 is 27
Cube of 4 is 64
Cube of 5 is 125
Both threads have finished.
```

Q5.

""" Advantages :

- Concurrency
- Parallelism
- Responsiveness
- Modularity
- Resource Sharing

Disadvantages :

- GIL Limitations
- Concurrency bugs
- Complexity
- Portability
- Resource Management

"""

Q6.

""" A deadlock is a situation in which two or more threads or processes are unable to proceed because they are each waiting for the other to release a resource or take some action. Deadlocks occur when there's a circular dependency between two or more threads, where each thread is holding a resource that the other needs.

A race condition occurs when the behavior of a program depends on the relative timing of events, often in a way that leads to unexpected and incorrect results. In a multi-threaded or multi-process environment, race conditions happen when two or more threads or processes access shared resources concurrently, and the final outcome depends on the order in which they execute. """

In [ ]: