Q.1 What is multithreading in Python? Why is it used, and name the module used to handle threads in Python?

ANS-Multithreading in Python refers to the concurrent execution of multiple threads within a single Python process. Each thread represents an independent flow of control, allowing you to perform multiple tasks concurrently. Multithreading is used to achieve better performance and responsiveness in applications that perform I/O-bound or CPU-bound tasks.

The module used to handle threads in Python is called the "threading" module. It provides a high-level interface for creating and managing threads.

threading module is used for managing threads. It allows you to create, start, stop, and synchronize threads in a Python program.

Q.2 Why is the threading module used? Write the use of the following functions: activeCount, currentThread, enumerate.

ANS-1.activeCount(): This function from the threading module returns the number of Thread objects currently alive. It helps you keep track of the active threads in your program.

2.currentThread(): This function returns the current Thread object corresponding to the caller. It's useful to identify the thread in which the code is currently executing.

3.enumerate(): enumerate() returns a list of all Thread objects currently alive. It allows you to access and manipulate each active thread in your program.

Q.3 Explain the following functions: run, start, join, isAlive.

ANS- 1.run(): This method is meant to be overridden by a subclass of the Thread class. It contains the code that will be executed when the thread starts running. You should define the task or operation you want the thread to perform within this method.

2.start(): This method is used to start a thread's execution. When called, it initiates the run() method in a new thread of execution. You should call start() to launch a thread, not run() directly.

3.join(): The join() method is used to wait for a thread to complete its execution before proceeding further in the program. It blocks the calling thread until the target thread finishes. It's often used to synchronize threads.

4.isAlive(): This method checks if a thread is currently alive or active. It returns True if the thread is still running, and False otherwise.

Q.4 Write a Python program to create two threads. Thread one must print the list of squares, and thread two must print the list of cubes.

In [1]:
```python
import threading

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

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

# Create two threads
thread1 = threading.Thread(target=print_squares)
thread2 = threading.Thread(target=print_cubes)

# Start the threads
thread1.start()
thread2.start()

# Wait for both threads to finish
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.
```

Q.5 Advantages and Disadvantages of Multithreading:

ANS-Multithreading is a programming technique that involves running multiple threads within a single process. Each thread represents a separate flow of execution, and they can run concurrently. Here are the advantages and disadvantages of multithreading:

Advantages:

Improved Performance: Multithreading can lead to better performance on multi-core processors by allowing multiple threads to execute simultaneously, thus utilizing available CPU resources more efficiently.

1.sponsiveness: Multithreading can make applications more responsive because tasks that would otherwise block the main thread (e.g., file I/O or network communication) can be executed in separate threads, allowing the main thread to remain responsive.

2.rallelism: Multithreading enables parallel processing, which is essential for certain types of applications such as scientific simulations, multimedia processing, and data analysis.

3.esource Sharing: Threads within the same process share the same memory space, making it easier for them to communicate and share data.

4.implified Code: In some cases, multithreading can simplify code by allowing you to break complex tasks into smaller, more manageable threads.

Disadvantages:

Complexity: Multithreaded programming can be challenging and error-prone. Issues like race conditions and deadlocks can arise, making debugging difficult.

1.source Contention: Threads may compete for shared resources, leading to contention and potentially reduced performance.

2.nchronization Overhead: To prevent race conditions and ensure proper synchronization, you often need to use synchronization primitives like mutexes and semaphores. This introduces overhead and can lead to potential bottlenecks.

3.bugging Challenges: Debugging multithreaded programs can be more complicated than single-threaded programs. Issues may not be consistently reproducible and can be challenging to diagnose.

4.ncreased Memory Usage: Each thread has its own stack and thread-specific data, which can increase memory usage compared to single-threaded programs.

Q.6 adlocks and Race Conditions:

ANS- Deadlock: A deadlock is a situation in which two or more threads or processes are unable to proceed with their execution because they are each waiting for the other(s) to release a resource or take some action. A typical scenario involves the following conditions:

Mutual Exclusion: At least one resource must be held in a mutually exclusive manner, meaning only one thread can access it at a time.

Hold and Wait: A thread must hold at least one resource while waiting to acquire additional resources.

No Preemption: Resources cannot be forcibly taken away from a thread. A thread must release them voluntarily.

Circular Wait: A circular chain of threads exists, where each thread is waiting for a resource held by the next thread in the chain.

Deadlocks can result in a system freeze where none of the involved threads or processes can make progress, requiring intervention to resolve.

Race Condition: A race condition occurs when the behavior of a program depends on the relative timing of events, such as when multiple threads or processes access shared data concurrently without proper synchronization. Race conditions can lead to unpredictable and undesirable outcomes.

In [ ]: