Q1.what is multithreading in python? hy is it used? Name the module used to handle threads in python

Multithreading in Python is a way of running multiple threads (smaller units of execution) simultaneously within a single program. A thread is a lightweight process that can perform a specific task while sharing the same memory space as the other threads in the same program.

Multithreading is used in Python to improve the performance of programs that require concurrent or parallel execution of multiple tasks. For example, in a web application, multithreading can be used to handle multiple requests simultaneously, allowing the application to respond to user requests faster.

The threading module is used to handle threads in Python. This module provides a way to create and manage threads in Python programs. It allows the creation of new threads, starting and stopping threads, and synchronizing the execution of threads. The threading module also provides features such as locks, semaphores, and condition variables, which can be used to coordinate the execution of threads and prevent race conditions.

Q2. why threading module used? rite the use of the following functions 1.activeCount() 2.currentThread() 3.enumerate()

The threading module in Python is used to implement multi-threading, which allows concurrent execution of multiple threads within a single process. It can be used to improve the performance of applications that perform tasks that can be split into smaller, independent tasks that can be executed in parallel.

1.activeCount(): This function returns the number of Thread objects that are active and running in the current program. It can be used to keep track of the number of threads that are running at any given time.

2.currentThread(): This function returns a reference to the current Thread object. It can be used to get information about the current thread, such as its name, ID, and status.

3.enumerate(): This function returns a list of all Thread objects that are currently active in the program. It can be used to get information about all the threads that are currently running, such as their names, IDs, and statuses.

Q2.Explain the following functions 1.run() 2.start() 3.join() 4.isalive()

1.run(): This function is a method of the Thread class in Python's threading module. It defines the code that will be executed in the new thread. When a Thread object's start() method is called, the run() function of that thread is executed.

2.start(): This function is a method of the Thread class in Python's threading module. It is used to start the execution of a new thread. When a Thread object's

3.join(): This function is a method of the Thread class in Python's threading module. It is used to wait for a thread to complete its execution before continuing with the main

program. When a Thread object's join() method is called, the main program will wait for the thread to finish executing before continuing with the next line of code.

4.isAlive(): This function is a method of the Thread class in Python's threading module. It is used to check whether a thread is currently executing or not. When a

In [21]:
```
#Q4.write a python program to create two threads. Thread one must print the list of

import threading

# define a function to print the squares of numbers from 1 to 10
def print_squares():
    for i in range(1, 11):
        print(f"{i} squared is {i**2}")

# define a function to print the cubes of numbers from 1 to 10
def print_cubes():
    for i in range(1, 11):
        print(f"{i} cubed is {i**3}")

# create two Thread objects for the two functions
t1 = threading.Thread(target=print_squares)
t2 = threading.Thread(target=print_cubes)

# start the two threads
t1.start()
t2.start()

# wait for the two threads to complete
t1.join()
t2.join()

print("Both threads have completed")
```

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
6 squared is 36
7 squared is 49
8 squared is 64
9 squared is 81
10 squared is 100
1 cubed is 1
2 cubed is 8
3 cubed is 27
4 cubed is 64
5 cubed is 125
6 cubed is 216
7 cubed is 343
8 cubed is 512
9 cubed is 729
10 cubed is 1000
Both threads have completed
```

1. State advantages and disadvantages of multithreading

Multithreading is a technique that allows a program to perform multiple tasks simultaneously, using multiple threads of execution. Here are some advantages and disadvantages of multithreading:

Advantages:

1.Improved performance: Multithreading can improve the performance of a program by allowing it to execute multiple tasks simultaneously. This can help to reduce the overall execution time of the program.

2.Resource sharing: Multithreading allows multiple threads to share resources such as memory, files, and network connections, which can reduce the overhead of creating and managing multiple processes.

3.Improved user interface responsiveness: Multithreading can help to improve the responsiveness of user interfaces in applications such as video games and multimedia applications, where it is important to maintain a high frame rate.

4.Asynchronous I/O: Multithreading can be used to perform asynchronous I/O operations, such as network communication, which can improve the responsiveness of the program and avoid blocking the main thread.

Disadvantages:

1.Complexity: Multithreading can add complexity to a program, as it requires careful synchronization and coordination of the threads to avoid race conditions and other synchronization problems.

2.Debugging difficulties: Debugging multithreaded programs can be difficult, as it can be hard to reproduce and isolate bugs that only occur in certain thread interleavings.

3.Increased memory usage: Multithreading can increase the memory usage of a program, as each thread requires its own stack and other data structures.

4.Scalability issues: Multithreading can suffer from scalability issues if the program is heavily dependent on shared resources, as contention for these resources can limit the performance gains from multithreading.

1. Explain deadlocks and race conditions.

Deadlocks and race conditions are two common types of synchronization problems that can occur in multithreaded programs.

Deadlocks occur when two or more threads are blocked waiting for each other to release resources, resulting in a situation where none of the threads can proceed. This can happen when two threads acquire resources in different orders, and then try to acquire the other thread's resource while holding their own resource. This can lead to a situation where both threads are blocked waiting for the other thread to release its resource, resulting in a deadlock. Deadlocks can be difficult to detect and resolve, as they often require careful analysis of the program's execution flow and resource usage.

Race conditions occur when two or more threads access a shared resource simultaneously, resulting in unpredictable or incorrect behavior. This can happen when two threads access the same memory location without proper synchronization, or when one thread modifies a

shared resource while another thread is accessing it. Race conditions can lead to data corruption, inconsistent program behavior, and other synchronization problems. Race conditions can be prevented by using synchronization primitives such as locks or semaphores to ensure that only one thread can access a shared resource at a time.

Both deadlocks and race conditions are common synchronization problems that can occur in multithreaded programs. It is important to design and implement programs carefully to avoid these issues, by using proper synchronization techniques and avoiding circular dependencies on shared resources. Debugging and resolving these issues can be difficult, and often requires careful analysis of the program's execution flow and resource usage.

In [ ]: