**1)What is multithreading in python? Why is it used? Name the module used to handle threads in python**

Ans- Multithreading refers to concurrently executing multiple threads by rapidly switching the control of the CPU between threads (called context switching). The Python Global Interpreter Lock limits one thread to run at a time even if the machine contains multiple processors.​

The threading module is a high-level implementation of multithreading used to deploy an application in Python. To use multithreading, we need to import the threading module in Python Program. A start() method is used to initiate the activity of a thread.

**2) Why threading module used? Write the use of the following functions**

**1.activeCount()**

**2.currentThread()**

**3.enumerate()**

Ans- The newer threading module included with Python 2.4 provides much more powerful, high-level support for threads than the thread module discussed in the previous section. threading. activeCount() − Returns the number of thread objects that are active.

Active\_count() is an inbuilt method of the threading module in Python. It is used to return the number of Thread objects that are active at any instant. Parameter(s):

It is not necessary to retain an explicit handle to all of the daemon threads in order to ensure they have completed before exiting the main process.threading.enumerate() returns a list of all Thread objects currently alive.

The list includes daemonic threads, dummy thread objects created by current\_thread(), and the main thread. It excludes terminated threads and threads that have not yet been started.

**3) Explain the following functions run() start() join() isAlive().**

Ans- start() method is an inbuilt method of the Thread class of the threading module in Python. It is used to start a thread's activity. This method calls the run() method internally which then executes the target method. This method must be called at most one time for one thread.

The isAlive() method returns true if the thread upon which it is called is still running otherwise it returns false. But, join() method is used more commonly than isAlive().

**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.**

Ans- import threading

def print\_cube(num):

    # function to print cube of given num

    print("Cube: {}" .format(num \* num \* num))

def print\_square(num):

    # function to print square of given num

    print("Square: {}" .format(num \* num))

if \_\_name\_\_ =="\_\_main\_\_":

    # creating thread

    t1 = threading.Thread(target=print\_square, args=(10,))

    t2 = threading.Thread(target=print\_cube, args=(10,))

    # starting thread 1

    t1.start()

    # starting thread 2

    t2.start()

    # wait until thread 1 is completely executed

    t1.join()

    # wait until thread 2 is completely executed

    t2.join()

    # both threads completely executed

    print("Done!")

**5) State advantages and disadvantages of multithreading.**

Ans- Advantages of multithreading:

* Enhanced performance by decreased development time
* Simplified and streamlined program coding
* Improvised GUI responsiveness
* Simultaneous and parallelized occurrence of tasks
* Better use of cache storage by utilization of resources
* Decreased cost of maintenance
* Better use of CPU resource

Disdvantages of multithreading

* Complex debugging and testing processes
* Overhead switching of context
* Increased potential for deadlock occurrence
* Increased difficulty level in writing a program
* Unpredictable results

**6) Explain deadlocks and race conditions**.

Ans- A deadlock is when two (or more) threads are blocking each other. Usually this has something to do with threads trying to acquire shared resources. For example if threads T1 and T2 need to acquire both resources A and B in order to do their work. If T1 acquires resource A, then T2 acquires resource B, T1 could then be waiting for resource B while T2 was waiting for resource A. In this case, both threads will wait indefinitely for the resource held by the other thread. These threads are said to be deadlocked.

Race conditions occur when two threads interact in a negatve (buggy) way depending on the exact order that their different instructions are executed. If one thread sets a global variable, for example, then a second thread reads and modifies that global variable, and the first thread reads the variable, the first thread may experience a bug because the variable has changed unexpectedly.