In [1]:
```
#Q1
```

In [2]:
```
# A process of running multiple threads simultaneously within a single process.
# To improve the performance of a program by using multiple CPU Cores.The module na
# used to handle threads in python is Threading.
# In Python, the threading module provides a very simple and intuitive API for spaw
```

In [3]:
```
#Q2
```

In [4]:
```
# threading module is used for creating, controlling and managing threads in python

#Python threading allows you to have different parts of your program run concurrent

#Following functions are:-


#threading.activeCount() - Returns the number of thread objects that are active.
#threading.currentThread() - Returns the number of thread objects in the caller's t
#threading.enumerate() - Returns a list of all thread objects that are currently ac
```

In [5]:
```
#Q3
```

In [6]:
```
#To start a thread, we use the start() method of the Thread class.
# t1.start()

# In order to stop the execution of the current program until a thread is complete,
#t1.join()

#isAlive()  - The isAlive() method checks whether a thread is still executing.
#run()  - The run() method is the entry point for a thread.
```

In [7]:
```
#Q4
```

In [13]:
```
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
    thread1 = [threading.Thread(target=print_square, args=(i,))
    for i in range(0,10)]
    thread2 =[threading.Thread(target=print_cube, args=(i,))
    for i in range(0,10)]
```

```
    # starting thread 1
    for t in thread1:
        t.start()
    # starting thread 2
    for t in thread2:
        t.start()

    # wait until thread 1 is completely executed
    t.join()
    # wait until thread 2 is completely executed
    t.join()

    # both threads completely executed
    print("Done!")
```

```
Square: 0
Square: 1
Square: 4
Square: 9
Square: 16
Square: 25
Square: 36
Square: 49
Square: 64
Square: 81
Cube: 0
Cube: 1
Cube: 8
Cube: 27
Cube: 64
Cube: 125
Cube: 216
Cube: 343
Cube: 512
Cube: 729
Done!
```

In [14]:
```
#Q5
```

In [15]:
```
# Advantages of multithreading:

#1 Enhanced performance by decreased development time
# 2 Simplified and streamlined program coding
# 3 Improvised GUI responsiveness
# 4 Simultaneous and parallelized occurrence of tasks
# 5 Better use of cache storage by utilization of resources
# 6 Decreased cost of maintenance
# 7 Better use of CPU resource


# Disadvantages of multithreading:

# 1 Complex debugging and testing processes
# 2 Overhead switching of context
# 3 Increased potential for deadlock occurrence
```

```
# 4 Increased difficulty level in writing a program
# 5 Unpredictable results
```

In [16]:
```
#Q6
```

In [ ]:
```
# A race condition is a failure case where the behavior of the program is dependent

#A race condition occurs when two threads access a shared variable at the same time
#The first thread reads the variable, and the second thread reads the same value fr


#A deadlock is a concurrency failure mode where a thread or threads wait for a cond
#The result is that the deadlock threads are unable to progress and the program is
#This situation will stop both threads from processing or executing the functions.
```