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

In [2]:
```
# A Process of running multiple processes simultaneously within a single thread.
# Purpose of multiprocessing is to improve the perforamance of a program by using m
# Multiprocessing in Python is a built-in package that allows the system to run mul
# It will enable the breaking of applications into smaller threads that can run ind
```

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

In [4]:
```
# Multiprocessing:-In Multiprocessing, Many processes are executed simultaneously.M
# In Multiprocessing, Process creation is a time-consuming process.In Multiprocessi
# In Multiprocessing, CPUs are added for increasing computing power.


#Multithreading:-A process of running multiple threads simultaneously within a sing
# While in multithreading, many threads of a process are executed simultaneously.Wh
# While in Multithreading, a common address space is shared by all the threads.
```

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

In [21]:
```
import multiprocessing
```

In [26]:
```
def test():
    print("this is my multiprocessing")
if __name__=="__main__":
    m=multiprocessing.Process(target=test)
    print("this is my main program")
    m.start()
    m.join()
```

```
this is my main program
this is my multiprocessing
```

In [27]:
```
#Q4
```

In [28]:
```
#Python multiprocessing Pool can be used for parallel execution of a function acros
#Creating process pool is preferred over instantiating new processes for every task


# create a process pool
#pool = multiprocessing.pool.Pool(...)

# issues tasks for execution
#results = pool.map(task, items)
```

In [29]:
```
# Q5
```

In [34]:
```
#We can configure the number of worker processes in the multiprocessing.pool.Pool b
#We can set the “processes” argument to specify the number of child processes to cr
```

```
# EX
# create a process pool with 4 workers
# pool = multiprocessing.pool.Pool(processes=4)

#The “processes” argument is the first argument in the constructor and does not nee
#EX

# create a process pool with 4 workers
# pool = multiprocessing.pool.Pool(4)
```

In [35]:
```
#Q6
```

In [36]:
```
import multiprocessing

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 processes
    p1 = multiprocessing.Process(target=print_square, args=(10, ))
    p2 = multiprocessing.Process(target=print_cube, args=(10, ))

    # starting process 1
    p1.start()
    # starting process 2
    p2.start()

    # wait until process 1 is finished
    p1.join()
    # wait until process 2 is finished
    p2.join()

    # both processes finished
    print("Done!")
```

```
Square: 100
Cube: 1000
Done!
```

In [40]:
```
import multiprocessing

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))



def print_addition(num):
    #functon to add of a given number
    print("Additon:{}".format(num+num))

def print_subtraction(num):
    #Function to subtract of a given number
    print("Subtraction:{}".format(num-num))



if __name__ == "__main__":
    # creating processes
    p1 = multiprocessing.Process(target=print_square, args=(10, ))
    p2 = multiprocessing.Process(target=print_cube, args=(10, ))
    p3=multiprocessing.Process(target=print_addition, args=(10, ))
    p4=multiprocessing.Process(target=print_subtraction, args=(10, ))


    # starting process 1
    p1.start()
    # starting process 2
    p2.start()
    p3.start()
    p4.start()



    # wait until process 1 is finished
    p1.join()
    # wait until process 2 is finished
    p2.join()
    p3.join()
    p4.join()



    # both processes finished
    print("Done!")
```

```
Square: 100
Cube: 1000
Additon:20
Subtraction:0
Done!
```

In [ ]: