In [1]:
```
import threading
def print_one():
          for i in range(5):
                    print(1)
def print_two():
          for i in range(6):
                    print(2)

t1= threading.Thread(target=print_one)
t2= threading.Thread(target=print_two)

t1.start()

t2.start()

t1.join()

t2.join()

print("Done!")
```

```
1
1
1
1
1
2
2
2
2
2
2
Done!
```

In [2]:
```
import threading
def print_one(n):
          for i in range(n):print(1,'\n')

def print_two(m):
          for i in range(m):print(2,'\m')

#create threads
t1= threading.Thread(target=print_one, args=(5,))
t2= threading.Thread(target=print_two,args=(7,))

#start thread 1
t1.start()
#start thread 2
t2.start()

#wait umtil thread 1 is completely executed
t1.join()
#wait until thread 2 is completely executed
t2.join()
#both threads completely executed

print("Done!")
```

```
1

1

1

1

1

2 \m
2 \m
2 \m
2 \m
2 \m
2 \m
2 \m
Done!
```

FAQ's

Ans 1)

Process	Thread
Means any program is in execution	A segment of process.
takes more time to terminate	takes less time to terminate
takes more time for creation	takes less time for creation.
Process is less efficient in terms of communication	More efficient in terms of communication
Process is isolated	Threads store memory
Does not share data with each other	Threads share data with each other.

Devanshu Surana
1032210755
PC-12

MIT-WPU | MIT WORLD PEACE UNIVERSITY | PUNE

Python Programming
Lab Assignment - 10

Problem statement: Create 2 threads to display cube and square of 5 numbers from list. Threads can simultaneously execute a cube and square functions from program to access the shared list of 5 numbers.

Aim: Write a python program to implement multithreading scenarios.

Objectives: To learn and implement functions of threading library.

Theory:

1) A thread is a lightweight process that ensures the execution of the process separately on the system. In python3, when multiple processor are running on a program, each processor runs simultaneously to execute its task separately. Multithreading is a threading technique in python programming to run multiple threads concurrently by rapidly switching between threads with a CPU help (called context switching). It aims to perform multiple tasks simultaneously which increases performance, speed and improves the rendering of the application.

2) a) It ensures effective utilization of computer system resources.
b) Multithreaded applications are more responsive.
c) It shares resources and its state with subthreads which makes it more economical.
d) Makes the multiprocessor architecture more effective due to similarity.
e) It saves time by executing multiple threads at the same time.

f) The system does not require too much memory to store multiple threads.

3) a) In python, threading module is a built-in module which is known as threading and can be directly imported.
b) A thread is capable of holding data, stored in data structures like dictionaries, lists, sets, etc. and can be passed as a parameter to a function.
c) It can be imported as thread module.
d) Thread delay is a function which will take two parameters as input i.e name of thread and delay.
e) start new thread is the method used to add a new thread.
f) start() method, simply runs the thread
g) join() method, which means wait until all the thread execution is complete.

4) a) start() : starts a thread by calling the run method.
b) run() : run() method is the entry point for a thread
c) join() : waits for the thread to terminate
d) is alive() : The isalive() method checks whether a thread is still executing.
e) getname() : Returns the name of the thread.

Platform: Windows | Ubuntu - Python Editor (Jupyter Notebook)

Conclusion: Studied python threading library functions and its benefits.

```
Done!
```

In [5]:

```python
import threading
ar=[8,2,7]

def cube():
    for i in range(len(ar)):print(ar[i]*ar[i]*ar[i])

def square():
    for i in range(len(ar)):print(ar[i]*ar[i])

#create threads
t1= threading.Thread(target=cube)
t2= threading.Thread(target=square)

#start thread 1
t1.start()
#start thread 2
t2.start()

#wait umtil thread 1 is completely executed
t1.join()
#wait until thread 2 is completely executed
t2.join()
#both threads completely executed

print("Done!")
```

```
512
8
343
64
4
49
Done!
```