Multi-Threading:

- Multi-Tasking is the process of executing more than one thread simultaneously
- Multi-tasking is classified in to 2 types
- **Process Based multi-Tasking**
  - Process is nothing but a program which is under execution
  - Process based multi-tasking is the process of executing more than one program which is existed in the different location of the "RAM" memory
  - Process based multi-tasking is heavy weight .i.e. these are taking more resources and more process time
- **Thread based Multi-Tasking**
  - Thread is nothing but smallest part of the process
  - Thread Based Multi-Tasking is the process of executing more then on functionality of the process simultaneously
  - Thread Based Multi-Tasking is light weight , These are taking less resources and less processing time

Example :

```
class Sample:
    def method1(self):
        ...........................

    def method2(self):
        ......................

    def method3(self):
        ......................
```

```
S=Sample()
s.method1()
s.method2()
s.method3()
```

If we execute the above program then execution will be the sequential. In order to make all the functionalities simultaneously then we have re-engineer the above program using multi-Threading

If we develop the program using multi-Threading then complete program will be taken by thread scheduler for scheduling .

Scheduling is the process in which specific time is allocated for every functionality where the control remains in the particular functionality till the particular period of time. Once the time is lapsed then control will jumps to another functionality with different time slice, But we are not guarantee in execution order . The Order my change.

- **Multi-Threading can be done by using "threading" module**
- **Python will support only thread based multi-tasking**

Advantages Of Multi-Threading

- Making multiple functions to execute simultaneously
- Avoiding ideal state of the CPU
- Increasing the Efficiency

- Multi-Threading Application mostly used in online banking System | online reservation system | ATM Applications ….

Ways to create | defining Threads

- **Creating threads by taking already defined function**

Example:

```
import threading
def myGreetings():
   for i in range(1,11):
      print("MyGreetings ... :",i)

def myInfo():
   for i in range(20,31):
      print("MyInfo .... : ",i)

#calling
#syn: threading.Thread(target=nameoffunction,[args]) -> Thread
t1=threading.Thread(target=myGreetings)
t2=threading.Thread(target=myInfo)
t1.start()
t2.start()
```

Note: Once we define the thread we have to call " start( ) " of Thread class to invoke to thread execution

Thread(target,args) -> Thread object

Target represent name of the function | methods do you want execute as thread.

Args represet arguments which are required for the function or methods but args must be given in the form “tuple”

➢ Creating Threads by creating non sub class “Thread”

- There is a class with some methods, If you want execute methods of a class as threads then we have create thread by creating non sub class of “Thread”

Example:

```
import threading
class Sample:
    def method1(self):
        for i in range(1,11):
            print("Method-1 of Sample ... : ",i)
    def method2(self):
        for i in range(20,31):
            print("Method-2 of Sample .... : ",i)
#calling
s=Sample()
t1=threading.Thread(target=s.method1)
t2=threading.Thread(target=s.method2)
t1.start()
t2.start()
```

➢ Creating threads by creating subclass of “ Thread “

- Thread is the predefined class from threading module
- If you want execute any logic as thread then it is recommended to write required logics in the “run()” of thread class

- run() of thread should not be called explicitly rather "run()" of thread should be called by "start()" of thread class.
- If we call run() of thread class then it becomes sequential flow. If you want random flow then we have to call "start()" of thread

Example:

```
import threading
class Cat(threading.Thread):
    def run(self):  #overriding run() of Thread class
        for i in range(1,11):
            print("Cat ... : ",i)

class Rat(threading.Thread):
    def run(self):
        for i in range(20,31):
            print("Rat ... : ",i)
#default thread in Python is __main__ [Thread]
c=Cat()
r=Rat()
c.start()
r.start()
```

Note: default thread in Python is "main thread" and auto executed
Thread is also main thread

```
#main also acts as a thread
import threading

class Child(threading.Thread):
    def run(self):
        for i in range(1,10):
            print("Child Thread ... : ",i)

#__main__
c=Child()
c.start()
for i in range(20,31):
    print("Main Thread ... : ",i)
```

Note: **Whenever we defined any thread every thread is created by default with some name like [Thread-1, Thread-2 ,……]. If you want we set the names for thread or get the names of thread by using the following methods**

- setName(str) for setting the name for the thread
- getName() → for getting the name of the thread object

```
import threading
class Child(threading.Thread):
    def run(self):
        for i in range(1,11):
            print("child Thread ... : ",i)
#calling
c=Child()
name=c.getName()
```

```
print("Thread Name is : ",name)
c.setName("Child")
name=c.getName()
print("Thread Name is : ",name)
```

Note : in order to get name of current working thread then we have to use "current_thread()" it will return Current Thread object

- to get the name of thread "getName()" of thread class

```
import threading
ct=threading.current_thread()
cwtn=ct.getName()
print("Name of the CW Thread is: ",cwtn)

ct.setName("Parent_Thread")
tname=ct.getName()
print("Name of main Thread : ",tname)
```

Note: Based on the application requirements we can also delay the execution of thread till the specified time then we have to make use "sleep()" from "time" module

```
import time
import threading

def myGreetings():
    for i in range(1,11):
        print("MyGreetings : ",i)


t1=threading.Thread(target=myGreetings)
t1.start()
time.sleep(10)
for i in range(20,31):
    print("Main Thread ...",i)
```

Example :

```
import time
import threading
def task1():
    print("Add milk into a bowl")
    print("put on the flame")
    print("Boil it for 2 mis ")

def task2():
    print("Add Tea powder ")
    print("Boil it for 3 mins ")

def task3():
    print("Add sugar ")
    print("boil it for 2 mins")

t1=threading.Thread(target=task1)
t2=threading.Thread(target=task2)
t3=threading.Thread(target=task3)
```

```
t1.start()
time.sleep(2)
t2.start()
time.sleep(3)
t3.start()
time.sleep(1)
print("Tea is Ready For U Dear....")
```

Note: if you want delay the execution of thread till another thread execution is completed then we have to user "join()" from Thread class

```
import threading
import time

class child(threading.Thread):
    def run(self):
        for i in range(1,11):
            time.sleep(1)
            print("child ... : ",i)

#calling
c=child()
c.start()
c.join()
for i in range(20,31):
    print("Main Thread ....",i)
```

Note: in the above program c.join() will go to run() of child class and it will ensures that run() of the "child" is completed its execution process or not. Whenever "run()" of child is completed its execution process then control will be given next waiting thread i.e "main" Thread.

Example: Executing the functions

```
import threading
import time

def sq_numbers(x):
    for i in x:
        time.sleep(1)
        s=i*i
        print("SQ of ",i," is ",s)

def cu_numbers(x):
    for i in x:
        time.sleep(1)
        c=i*i*i
        print("CU of ",i," is ",c)

#calling
lst=[1,2,3,4,5,6,7,8,9,10]
st=time.time()  #will extract system time
```

```
sq_numbers(lst)
cu_numbers(lst)
et=time.time()
tt=et-st
print("total time taken is : ",tt)
```

Example:
Execute the above program using Threading Concept

```
import threading
import time

def sq_numbers(x):
    for i in x:
        time.sleep(1)
        s=i*i
        print("SQ of ",i," is ",s)

def cu_numbers(x):
    for i in x:
        time.sleep(1)
        c=i*i*i
        print("CU of ",i," is ",c)

#calling
lst=[1,2,3,4,5,6,7,8,9,10]
st=time.time()  #will extract system time
ts=threading.Thread(target=sq_numbers,args=(lst,) )
tc=threading.Thread(target=cu_numbers,args=(lst,) )
```

```
    ts.start()
    tc.start()
    ts.join()
    tc.join()
    et=time.time()
    tt=et-st
    print("total time taken is : ",tt)
```

- For Every Thread PVM Will Create a unique identity . to get that identity then we have to use "ident" attribute From the Thread class

Example:

```
import threading
'''default thread is main and main thread will start by PVM '''
t=threading.current_thread()
print("Name of CW_Thread : ",t.getName())
print("Identity of Thread ",t.ident)
```

- Whenever Thread is started it execution the then thread becomes running state when thread is completed its execution process then thread will become "dead" state. If you want know the thread is alive or not then we have to make use of "**is_alive()**" from Thread Class
- Is_alive() method returns True if the Thread is in "running" state whenever thread is completed its execution then it will return "False"

Example:

```
import threading
import time
```

```
def child():
    print(threading.current_thread().getName()," Thread Started Execution")
    time.sleep(5)
    print("Thread Execution is Done....!!! ")

#Creating Threads
t1=threading.Thread(target=child,name="Ramesh")
t1.start()
print("t1 is Alive ? : ",t1.is_alive())
time.sleep(10)
print("t1 is Alive ? : ",t1.is_alive())   # upto python 3.7 isAlive()
```

- Inorder to know how many threads are in running State or Threads which are in active then we have to use “active_count()” from threading Module.

```
import threading
import time

def myFun():
    print(threading.current_thread().getName()+" Thread  is started ...")
    time.sleep(10)

#creating
t1=threading.Thread(target=myFun,name='Child-1')
t2=threading.Thread(target=myFun,name='Child-2')
t3=threading.Thread(target=myFun,name='Child-3')
```

```
t1.start()
t2.start()
t3.start()

print("No.of.Active Thread : ",threading.active_count())

lst=threading.enumerate()   #returns list of active threads |names| ident
for t in lst:
    time.sleep(1)
    print("Name of Thread : ",t.getName())
    print("Ident of Thread : ",t.ident)
    print("-----------------------------------")
```

Note: In order to know all thread which are in execution then we have to use "enumerate()" from threading Module . it will return list of threads with their names and identities.

What is "' synchronization ? " How to achieve ?

- It is the process of restricting more than one thread to perform the operation on the same functionality simultaneously then we have to use synchronization mechanism Using LOCK Concept
- To Create an Object of Lock Class then we have to use lock() From Threading module
- Make the Lock class object is available for the thread which entered in the required function first by using "acquire() " from lock class. Then first entered thread will get Lock object and it will lock the that function till its process is done..
- Whenever the thread is completed its execution process make the thread to release lock object for next awaiting thread by using "release()" from lock class.

```
import threading
import time

class ATM:
    def __init__(self):
        self.l=threading.Lock()

    def wd(self,name):
        self.l.acquire()
        for i in range(1,11):
            time.sleep(1)
            print("Wd By Mr|Mrs ... ",name)
        self.l.release()

#Creating Threads
a=ATM()
t1=threading.Thread(target=a.wd,args=('Ramesh',))
t2=threading.Thread(target=a.wd,args=('Suresh',))
t1.start()
t2.start()
```

Advantages : Avoiding Data inconsistency
Disadvantages : It will lacking the Efficiency