Python Threading or Multithreading:
Threading allows multiple tasks run concurrently or indepedently. It is a separate flow of execution.
OR
Threads provide a way to improve application performance through parallelism.

Why Threading?
1.To implement Multimedia graphics
2 Parallel Computation
3 Standardization
4 Parallel I/O (Input/Output)
5 Asynchronous Events
6.To develop animations
7.To develop video games
8.To develop web and application servers..!

Methods of Thread Class
run(): It acts as the entry of the thread
start(): is used for starting the thread by calling the run()
isAlive(): is used to verify whether the still executing or not
getName(): is used for returning the name of a thread
setName(): is used to set the name of the thread

Example:
```
import threading
print("Current Executing
Thread:",threading.current_thread().getName())#mainthread
```

Example:
```
import time
import threading

def calc_square(numbers):
    print("calculate square numbers")
    for n in numbers:
        time.sleep(0.2)
        print('square:',n*n)

def calc_cube(numbers):
    print("calculate cube of numbers")
    for n in numbers:
        time.sleep(0.2)
        print('cube:',n*n*n)

arr = [2,3,8,9]
t = time.time()
t1= threading.Thread(target=calc_square, args=(arr,))
t2= threading.Thread(target=calc_cube, args=(arr,))

t1.start()
t2.start()

t1.join()
t2.join()
```

```
print("done in : ",time.time()-t)
print("Hah... I am done with all my work now!")
```

Example:

```
import threading
class Thread1(threading.Thread):
    def run(self):
        i=1
        while i<=5:
            print(i)
            i=i+1
class Thread2(threading.Thread):
    def run(self):
        i=1
        while i<=5:
            print(i)
            i=i+1
t1=Thread1()
t2=Thread2()
t1.start()
t2.start()
```

Setting and Getting Name of a Thread:
Every thread in python has name. It may be default name generated by Python or Customized Name provided by programmer. We can get and set name of thread by using the following Thread class methods.
t.getName() ==> Returns Name of Thread
t.setName(newName) ==>To set our own name

Note:
Every Thread has implicit variable "name" to represent name of Thread.

Example:

```
from threading import *
print(current_thread().getName())
current_thread().setName("PyThread")
print(current_thread().getName())
print(current_thread().name)
```

Thread Identification Number (ident):
For every thread internally a unique identification number is available. We can access this id by using implicit variable "ident"

Example:

```
from threading import *
def Test_Thread():
    print("MyThread",end=" ")
th=Thread(target=Test_Thread)
th.start()
print("Main Thread ID:",current_thread().ident)
print("Child Thread ID:",th.ident)
```

Daemon Threads:
The threads which are running in the background are called Daemon Threads.

Example:

Garbage Collector

Example:
```
from threading import *
print(current_thread().isDaemon()) #False
print(current_thread().daemon) #False
```

Default Nature:
By default Main Thread is always non-daemon.But for the remaining threads Daemon nature will be inherited from parent to child.

Example:
```
from threading import *
def ThTask():
    print("ChildThread")
th=Thread(target=ThTask)
print(th.isDaemon())
th.setDaemon(True)
print(th.isDaemon())
```

Synchronization:
If multiple threads are executing simultaneously then there may be a chance of data inconsistency problems.
Synchronization means at a time only one Thread The main application areas of synchronization are
1 Online Reservation system
2 Funds Transfer from joint accounts
3 Online Exams

In Python, we can implement synchronization by using the following
1.Lock 2.RLock 3.Semaphore

Locks are the most fundamental synchronization mechanism provided by threading module. The Lock object can be hold by only one thread at a time.
A Thread can acquire the lock by using acquire() method.
A Thread can release the lock by using release() method.

Example:
```
from threading import *
lck=Lock()
lck.acquire()
lck.release()
```

Example:
```
import time
import threading
from threading import *
lc=Lock()
def calc_square(numbers):
    lc.acquire()
    print("calculate square numbers")
    for n in numbers:
        time.sleep(0.3)
        print('square:',n*n)
    lc.release()
```

```
def calc_cube(numbers):
    lc.acquire()
    print("calculate cube of numbers")
    for n in numbers:
        time.sleep(0.3)
        print('cube:',n*n*n)
    lc.release()

arr = [2,3,8,9]
t = time.time()

t1= threading.Thread(target=calc_square, args=(arr,))
t2= threading.Thread(target=calc_cube, args=(arr,))

t1.start()
t2.start()

t1.join()
t2.join()

print("done in : ",time.time()-t)
```

RLock:
RLock keeps track of recursion level and hence for every acquire() call compulsory release() call should be available. i.e the number of acquire() calls and release() calls should be matched then only lock will be released.

Example:

```
import time
import threading
from threading import *
rc=RLock()
def calc_square(numbers):
    rc.acquire()
    print("calculate square numbers")
    for n in numbers:
        time.sleep(0.3)
        print('square:',n*n)
    rc.release()

def calc_cube(numbers):
    rc.acquire()
    print("calculate cube of numbers")
    for n in numbers:
        time.sleep(0.3)
        print('cube:',n*n*n)
    rc.release()

arr = [2,3,8,9]
t = time.time()
t1= threading.Thread(target=calc_square, args=(arr,))
t2= threading.Thread(target=calc_cube, args=(arr,))

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

```
t1.join()
t2.join()

print("done in : ",time.time()-t)
```

Difference between Lock and RLock:
Lock:
1.Lock object can be acquired by only one thread at a time.Even owner thread also cannot acquire multiple times.
2.Not suitable to execute recursive functions and nested access calls
3.In this case Lock object will takes care only Locked or unlocked and it never takes care about owner thread and recursion level.

RLock:
1.RLock object can be acquired by only one thread at a time, but owner thread can acquire same lock object multiple times.
2.Best suitable to execute recursive functions and nested access calls
3.In this case RLock object will takes care whether Locked or unlocked and owner thread

Synchronization by using Semaphore:
In the case of Lock and RLock,at a time only one thread is allowed to execute. Semaphore is advanced Synchronization Mechanism.

Syntax:
```
s=Semaphore(counter)
```

Example:
```
s=Semaphore(3)
```
In this case Semaphore object can be accessed by 3 threads at a time.The remaining threads have to wait until releasing the semaphore.

NOTE:
counter represents the maximum number of threads are allowed to access simultaneously. The default value of counter is 1.

Example:
```
import time
import threading
from threading import *
lc=Lock()
s=Semaphore(2)

def calc_square(numbers):
    s.acquire()
    print("Square of Numbers")
    for n in numbers:
        time.sleep(0.3)
        print('square:',n*n)
    s.release()

s=Semaphore(2)
def calc_cube(numbers):
    s.acquire()
    print("calculate cube of numbers")
    for n in numbers:
        time.sleep(0.3)
        print('cube:',n*n*n)
    s.release()

arr = [2,3,4,5,6,7]
t = time.time()

t1= threading.Thread(target=calc_square, args=(arr,))
t2= threading.Thread(target=calc_cube, args=(arr,))

t1.start()
t2.start()

t1.join()
t2.join()

print("done in : ",time.time()-t)
```

Difference between Multiprocessing and Multithreading?
Multithreading means in any single process, multiple threads is allowed and again, can run simultaneously. Multitasking is sharing of computing resources(CPU, memory, devices, etc.) among processes, while multithreading is sharing of computing resources among threads of a single proces.

Example:
```
import time
import multiprocessing

def calc_square(numbers):
    for n in numbers:
        time.sleep(0.3)
        print('Square ' ,n*n)

def calc_cube(numbers):
    for n in numbers:
        time.sleep(0.3)
        print('Cube ' , n*n*n)

arr = [2,3,8,9]
p1 = multiprocessing.Process(target=calc_square, args=(arr,))
p2 = multiprocessing.Process(target=calc_cube, args=(arr,))

p1.start()
p2.start()

p1.join()
p2.join()
print("Successfully Done!")
```

Example1:
```
import time
import threading
import math
from threading import *
lc=Lock()
def Multiplication(numbers):
    lc.acquire()
    print("Multiplication Table of: ", numbers)
    for n in numbers:
        time.sleep(0.3)
        for i in range(1, 11):
            print(i*n)
    lc.release()

def FactorialN(numbers):
    lc.acquire()
    print("calculate factorial of numbers")
    for n in numbers:
        print(math.factorial(n))
    lc.release()
arr = [2,3]
t = time.time()

t1= threading.Thread(target=Multiplication, args=(arr,))
t2= threading.Thread(target=FactorialN, args=(arr,))

t1.start()
t2.start()

t1.join()
t2.join()
print("done in : ",time.time()-t)
```

Example2:

```
import time
import threading
import math
from threading import *
rc=RLock()
def Multiplication(numbers):
    rc.acquire()
    print("Multiplication Table of: ", numbers)
    for n in numbers:
        time.sleep(0.3)
        for i in range(1, 11):
            print(i*n)
    rc.release()

def FactorialN(numbers):
    rc.acquire()
    print("calculate factorial of numbers: ", numbers)
    for n in numbers:
        time.sleep(0.3)
        print(math.factorial(n))
    rc.release()

arr = [2,3]
t = time.time()

t1= threading.Thread(target=Multiplication, args=(arr,))
t2= threading.Thread(target=FactorialN, args=(arr,))

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

```
t1.join()
t2.join()
print("done in : ",time.time()-t)
```

Example3:

```
import time
import threading
import math
from threading import *
sem=Semaphore(2)
def Multiplication(numbers):
    sem.acquire()
    print("Multiplication Table of: ", numbers)
    for n in numbers:
        time.sleep(0.3)
        for i in range(1, 11):
            print(i*n)
    sem.release()
def FactorialN(numbers):
    sem.acquire()
    print("calculate factorial of numbers: ", numbers)
    for n in numbers:
        time.sleep(0.3)
        print(math.factorial(n))
    sem.release()

arr = [2,3]
t = time.time()

t1= threading.Thread(target=Multiplication, args=(arr,))
t2= threading.Thread(target=FactorialN, args=(arr,))

t1.start()
t2.start()

t1.join()
t2.join()
print("done in : ",time.time()-t)
```