MULTITHREADING

Within a single application, there are multiple parts of execution, each part is called thread and all parts are called multithreading.

Multithreading is defined as the ability of a processor to execute multiple threads concurrently.

THREAD: It is a light weight process and is a small part of execution.
Python provides threading module for creating threads.

There are two types in Multithreading

1. Function based multithreading
2. Class based multithreading

Function based multithreading

Without multithreading:

```
def oddDisplay():
        for x in range(1,100,2):
                print(x)
def evenDisplay():
        for x in range(2,100,2):
                print(x)
oddDisplay()
evenDisplay()
```

In above program first all the odd numbers will get printed and then all even numbers will get printed.

With multithreading:

```
from threading import Thread
def oddDisplay():
        for x in range(1,100,2):
                print(x)
def evenDisplay():
        for x in range(2,100,2):
                print(x)
t1 = Thread(target = oddDisplay)
t2 = Thread(target = evenDisplay)
t1.start()
t2.start()
```

In above program, due to multithreading the output will come simultaneously beside each other. Here two threads are created using Thread class which will work simultaneously.

If we want to suspend (wait) execution of the current thread for a given number of seconds then use sleep function of time module. For e.g.

```
from threading import Thread
import time

def oddDisplay():
    for x in range(1,100,2):
        time.sleep(0.5)
        print(x)
def evenDisplay():
    for x in range(2,100,2):
        time.sleep(0.5)
        print(x)
t1 = Thread(target = oddDisplay)
t2 = Thread(target = evenDisplay)
t1.start()
time.sleep(0.5)
t2.start()
```

Single function with arguments:

```
from threading import Thread
import time

def display(start,end):
    for x in range(start,end,2):
        time.sleep(0.5)
        print(x)

t1 = Thread(target = display,args = (1,100))
t2 = Thread(target = display,args = (2,100))
t1.start()
time.sleep(0.5)
t2.start()
```

Print name of thread:

```
from threading import Thread,currentThread
import time
def display(start,end):
    for x in range(start,end,2):
        time.sleep(0.5)
        print(x,"--->",currentThread().getName())

t1 = Thread(target = display,args = (1,100),name = 'odd')
t2 = Thread(target = display,args = (2,100),name = 'even')
t1.start()
time.sleep(0.5)
t2.start()
```

Here use currentThread().getName() to get the name of current thread which is running.

In python 3.10.x onwards, currentThread() is deprecated. So import current_thread and use current_thread().name directly to get the name of running thread.

Class based multithreading

In this type of multithreading we have to do 2 things:

1. Override __init__() to add additional arguments.
2. Override run() method to implement what the thread should do when started.

```
from threading import Thread
import time

class NumberG(Thread):
   def __init__(self,st,end):
      super().__init__()
      self.st=st
      self.end=end
   def run(self):
      for i in range(self.st,self.end,2):
         time.sleep(0.2)
         print(i)

odd=NumberG(1,100)
even=NumberG(2,100)
odd.start()
even.start()
```

Join method:

On invoking the join() method, the calling thread gets blocked until the thread object (on which the thread is called) gets terminated.

Points to remember while joining threads using join():

- If the join() has the timeout argument then it should be a floating point number representing the time for the operation of thread in seconds.
- If the join() method has no argument, then until the thread terminates the operation is blocked.

```
from threading import Thread
def display():
   for x in range(50):
      print(x)

t1 = Thread(target = display) #Child thread
t1.start()
t1.join(0.1)

for i in range(30):
   print("---main---")  #Parent thread
```

In above program, child will start execution first and parent thread will wait for 0.1 sec & then it will start execution.

Multithreading with data share

All threads share same memory address.

```
from threading import Thread
import time
x=0
def display():
   global x
   x=20
   print("value--display-",x)
   for i in range(1,50):
      print(i)

t1=Thread(target=display)
t1.start()
t1.join()
print("value--main-",x)
```

MULTIPROCESSING

Multiprocessing is a system that has more than one or two processors. In Multiprocessing, CPUs are added for increasing computing speed of the system. Because of Multiprocessing, there are many processes executed simultaneously.

```
from multiprocessing import Process
import time

def oddDisplay():
   for x in range(1,100,2):
      time.sleep(0.5)
      print(x)
def evenDisplay():
   for x in range(2,100,2):
      time.sleep(0.5)
      print(x)
p1 = Process(target = oddDisplay)
p2 = Process(target = evenDisplay)

if __name__ == '__main__':
   p1.start()
   time.sleep(0.5)
   p2.start()
```

Note : Run multiprocessing programs in cmd prompt as there are bugs in IDLE to run it.

Multiprocessing with data share

Normal process sharing:

```
from multiprocessing import Process
import time

val = 10
def display():
   global val
   val = 20
   print("value---display",val)
   for i in range(10):
      print(i)

p1=Process(target=display)
if __name__=='__main__':
   p1.start()
   p1.join()
   print("value---main",val)
```

In above program the processes will share different memory address.

Sharing using Value class:

```
from multiprocessing import Process,Value
import time
def display(n,val):
        val.value=20
        print("value---display",val.value)
        for i in range(n):
                print(i)
val=Value('i',0)
p1=Process(target=display,args=(30,val))
if __name__=='__main__':
        p1.start()
        p1.join()
        print("value---main",val.value)
```

In above program the processes will share same data by using Value class.

Difference between Multiprocessing and Multithreading:

S.NO	MULTIPROCESSING	MULTITHREADING
1.	In Multiprocessing, CPUs are added for increasing computing power.	While In Multithreading, many threads are created of a single process for increasing computing power.
2.	In Multiprocessing, Many processes are executed simultaneously.	While in multithreading, many threads of a process are executed simultaneously.
3.	Multiprocessing are classified into Symmetric and Asymmetric.	While Multithreading is not classified in any categories.
4.	In Multiprocessing, Process creation is a time-consuming process.	While in Multithreading, process creation is according to economical.
5.	In Multiprocessing, every process owned a separate address space.	While in Multithreading, a common address space is shared by all the threads.