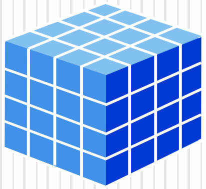
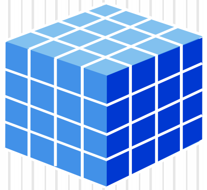
Scripting Language

UNIT-III





Multi Threading

Introduction to Thread

Advantages

Life cycle of a Thread

Threading Modules and its Methods

Multi Tasking

- In multi tasking environment, several tasks are given to processor at a time.
- Multi tasking can be done by scheduling algorithms.
- In this, most of the processor time is getting engaged and it is not sitting idle.
- **Multi-Processing**
- **Multi-Threading**

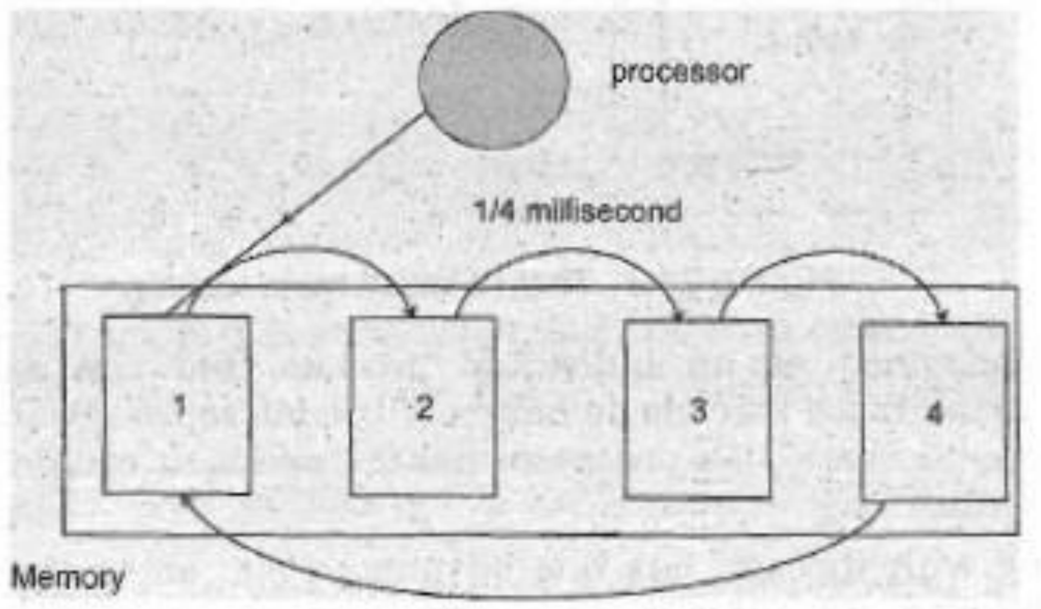
Uses:

- To use the processor time in a better way.
- To achieve good performance.
- To make processor not to sit idle for long time.

Multiprocessing

- Multiprocessing refers to the ability of a system to support more than one processor at the same time. Applications in a multiprocessing system are broken to smaller routines that run independently. The operating system allocates these threads to the processors improving performance of the system.
- Consider a computer system with a single processor. If it is assigned several processes at the same time, it will have to interrupt each task and switch briefly to another, to keep all of the processes going. Multiprocessing will dividing work between multiple processes.
- Multiprocessing is a feature that allows your computer to run two or more programs concurrently.
- **Example:- you can listen to music and at the same time chat with your friends on Facebook using browser.**

- Suppose we want to execute 4 jobs at a time. We load them into memory. Now the micro processor has to execute them all at a time. The memory is divided into 4 parts and the jobs are loaded there.
- Within the time slot, it will try to execute each job. So the processor will take small amount of time duration. This small part of the processor time is called " Time Slice". Within in this time slot the processor try to execute the jobs.

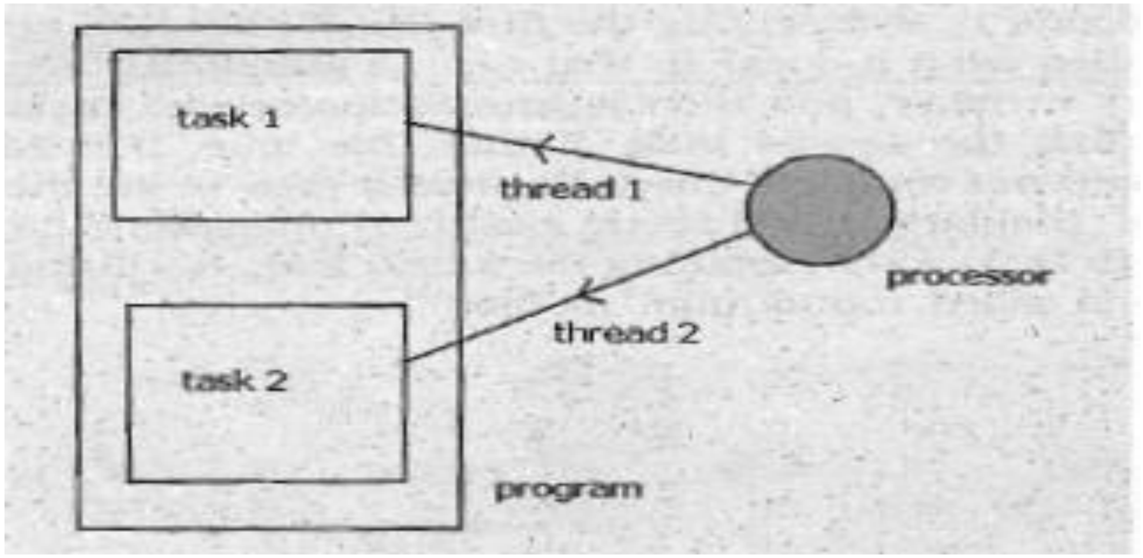


- In process based multi tasking, several programs are executed at a time, by the microprocessor.
- **Advantages of multiprocessing**
- The main advantage of multiprocessor system is to get more work done in a shorter period of time. These types of systems are used when very high speed is required to process a large volume of data.
- Multi processing systems can save money in comparison to single processor systems because the processors can share peripherals and power supplies.
- It also provides increased reliability in the sense that if one processor fails, the work does not halt, it only slows down. e.g. if we have 10 processors and 1 fails, then the work does not halt, rather the remaining 9 processors can share the work of the 10th processor. Thus the whole system runs only 10 percent slower, rather than failing altogether.

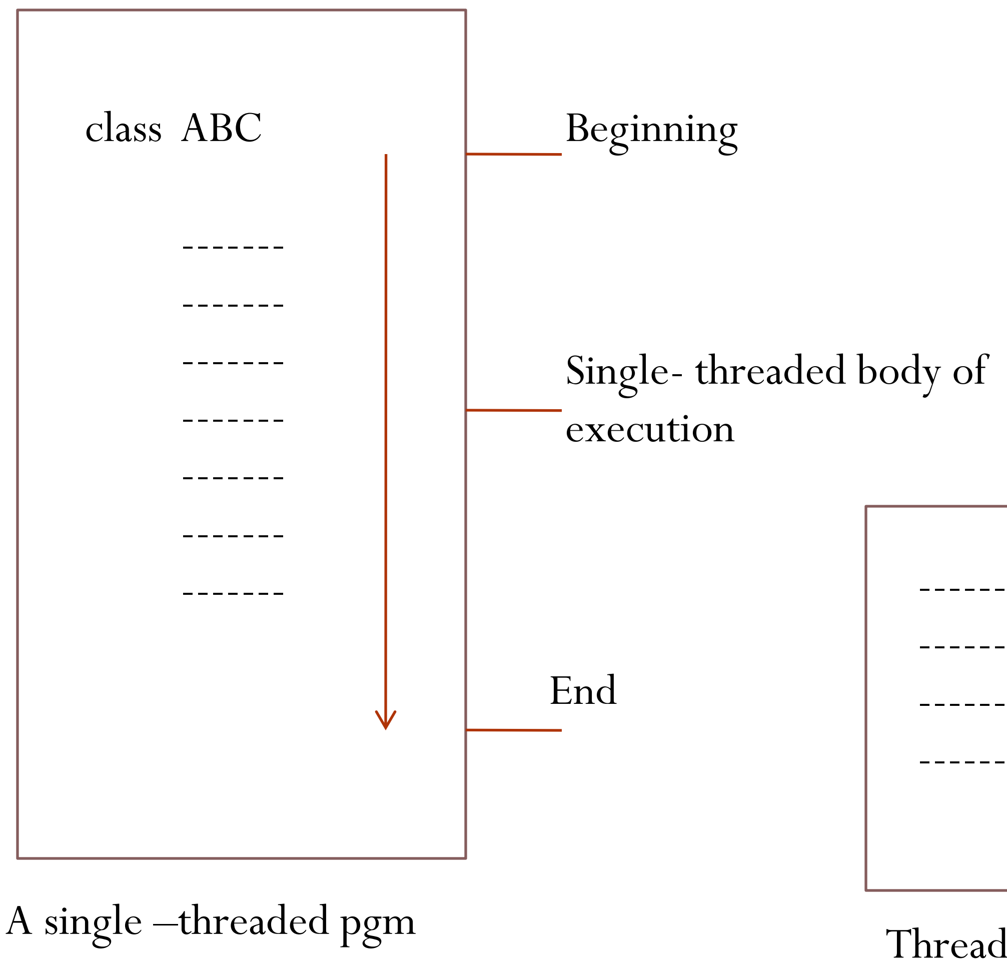
Multi Threading

- Multithreading is a conceptual programming concept where a program (process) is divided into two or more subprograms , that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.
- A **process** consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process.
- In Thread-based multitasking, thread is the smallest unit of code, which means a single program can perform two or more tasks simultaneously.
- **Example** :- a text editor can print and at the same time you can edit text file with provided text. That means those two tasks are perform by separate threads.

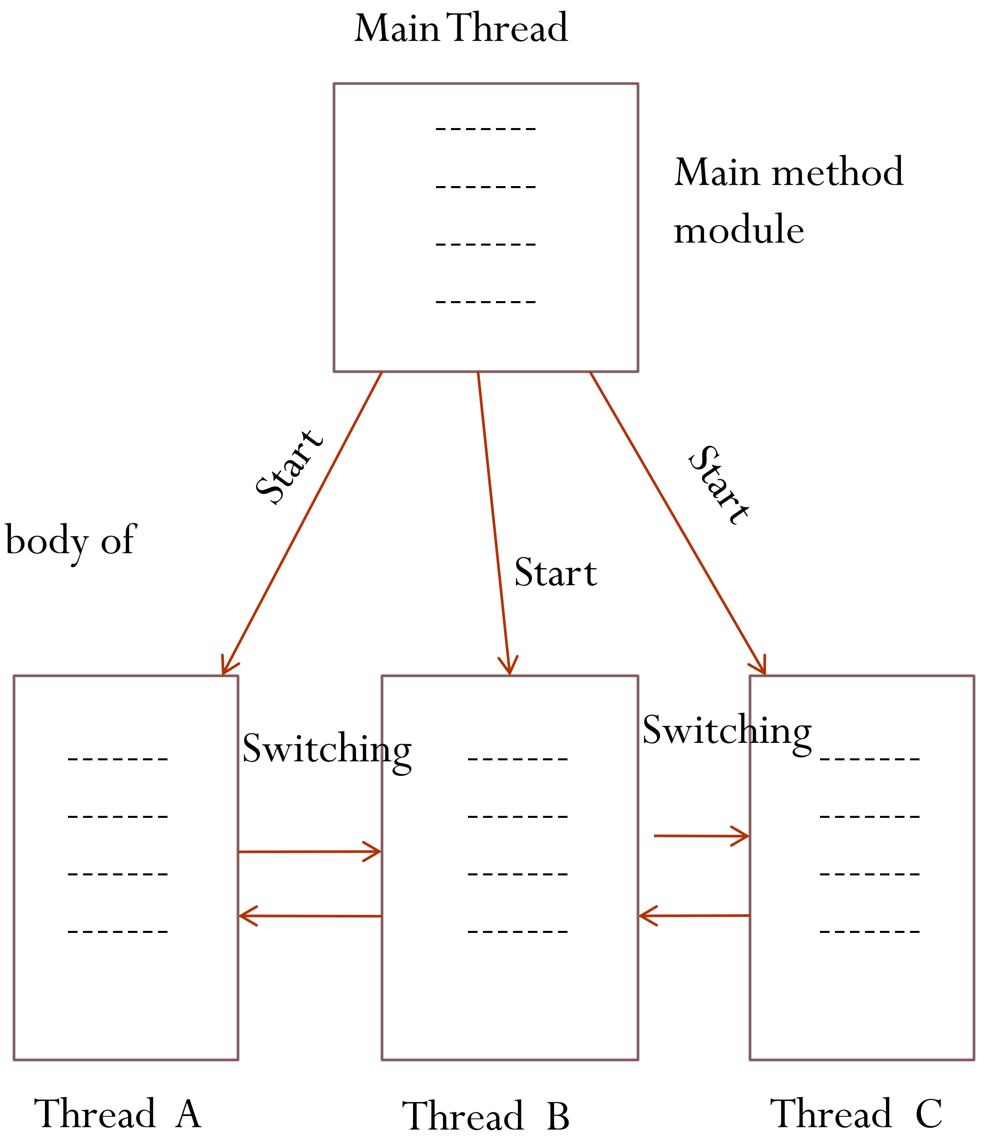
- In multi thread program several parts of the same program is executed at a time, by microprocessor.



- In the above process there are two programs. These parts may represent two separate blocks of code or two separate methods containing processes. Each part may perform a separate task. The processor should execute the two parts simultaneously. So the processor uses 2 separate threads to execute these two parts.
- Each thread can be imagined as an individual process that can execute separate statements.We can imagine these threads as hands of the microprocessor.



A single –threaded pgm



A Multi-threaded Program

Advantages of Multi-Threading

- Handling of threads is simpler than handling of processes for an operating system. This is why they are sometimes called lightweight process(LWP)
- Programmers can divide a long program into threads and execute them in parallel which eventually increases the speed of the program execution
- Improved performance and concurrency
- Enables programmers to do multiple things at one time
- Simultaneous access to multiple applications

Basis for Comparison	Multi Threading	Multi Processing
Execution	Multiple threads of a single process are executed concurrently.	Multiple processes are executed concurrently.
Creation	Creation of a thread is economical in both sense time and resource	Creation of a process is time consuming and resource intensive
Communication	With in the process threads are communicated	There is no communication between processors directly
Memory	Threads share the address space only	Processes require their own separate address space

All the above are meant for optimum utilization of CPU resources.

Python pgms that we have seen so far contain only a single sequential flow of control. The program begins, run through a sequence of executions & finally ends. At any given point of time, there is only one statement under execution

Introduction to Thread

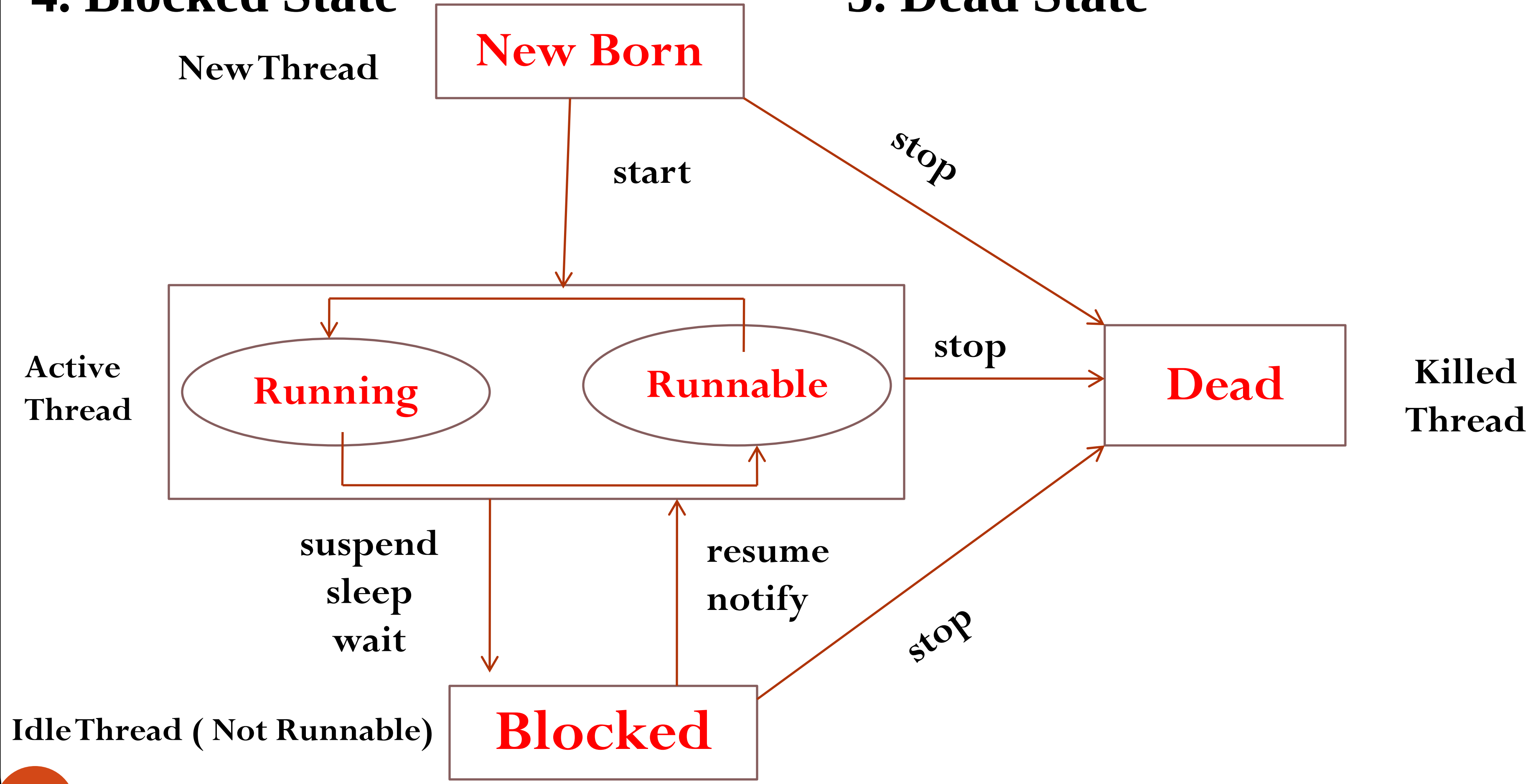
- A single sequential flow of control in a process (pgm under execution) is called a thread
- An independent path of execution in a program is a thread. It has a beginning, a body & an end
- Every program will have atleast one thread
- Based on the concept of threads, applications may be classified into two kinds
  1. Single-threaded application
  2. Multi-threaded application
- A single threaded appl. can perform only one job at-a-time, as they have only one flow of control
- An appl. is said to be multithreaded, if it has multiple flows of control. In each flow, we can perform one job.
- Whenever a single appl. has to perform multiple concurrent jobs, that appl. is a right on to be made as multithreaded.

Life Cycle of a Thread:

- During the life time of a thread, there are many states it can enter. The important states are,

1. **Newborn state** **2. Runnable State** **3. Running State**

4. Blocked State **5. Dead State**



A thread is always in any of these states & it can move from one state to another via a variety of ways

Newborn State:

when we create a thread object, the thread is born & is said to be in "Newborn State". The thread is not yet scheduled for running

At this state, we can do only one of two operations

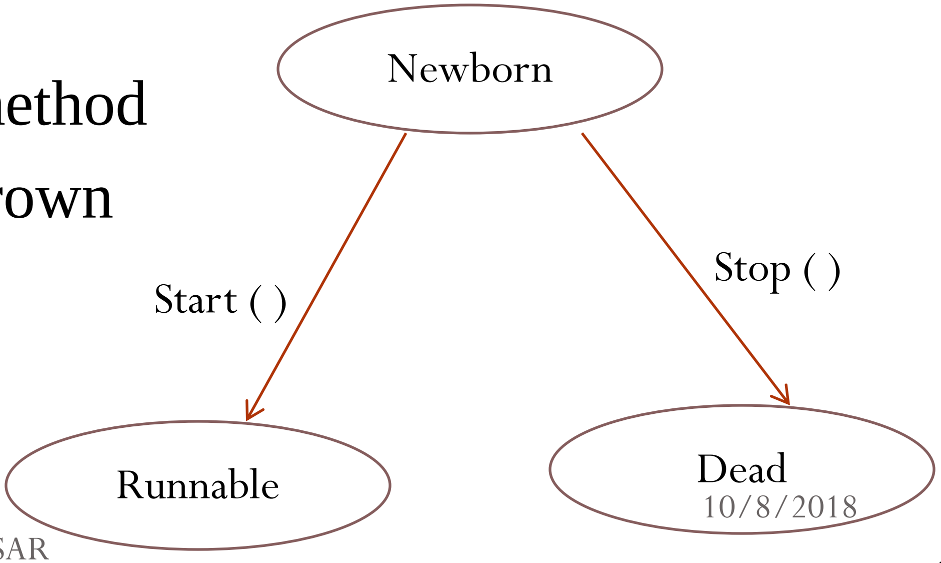
1. Schedule it for running using "start( ) method".

   Then it moves to the runnable state

2. Kill it using "stop( ) method".

Then it moves to the Dead state

If we attempt to use any other method at this state, an exception will be thrown



Runnable State:

- If a thread is in this state it means that the thread is ready for execution and waiting for the availability of the processor. If all threads in queue are of same priority then they are given time slots for execution

Running State:

- It means that the processor has given its time to the thread for execution. A thread keeps running until the following condition occurs
  - Thread give up its control when a thread is made to sleep for a specified period of time using **sleep(time)** method, where time in milliseconds.

Blocked State:

- A thread is said to be blocked, when it is prevented from entering into the Runnable state & subsequently the Running state.
- This happens when the thread is suspended, sleeping or waiting in order to satisfy certain requirements
- A blocked thread is considered "not runnable" but not dead & therefore fully qualified to run again

Dead State:

- A runnable thread enters the Dead or Terminated State when it completes its task or otherwise terminates
- A thread ends its life when it has completed executing its "run() method". It is a natural death
- We can kill the thread by sending the "stop() method" to it at any state, thus causing a premature death to it.

Starting a new THREAD using the _thread Module

- There are two modules which help in multi-threading namely **_thread** and **threading**
- To begin a new thread we need to call **start_new_thread( )** method of the **_thread module**
- **Syntax:**

  _thread .start_new_thread(function_name, args,[kwargs])

  Here,

  args is a tuple of arguments

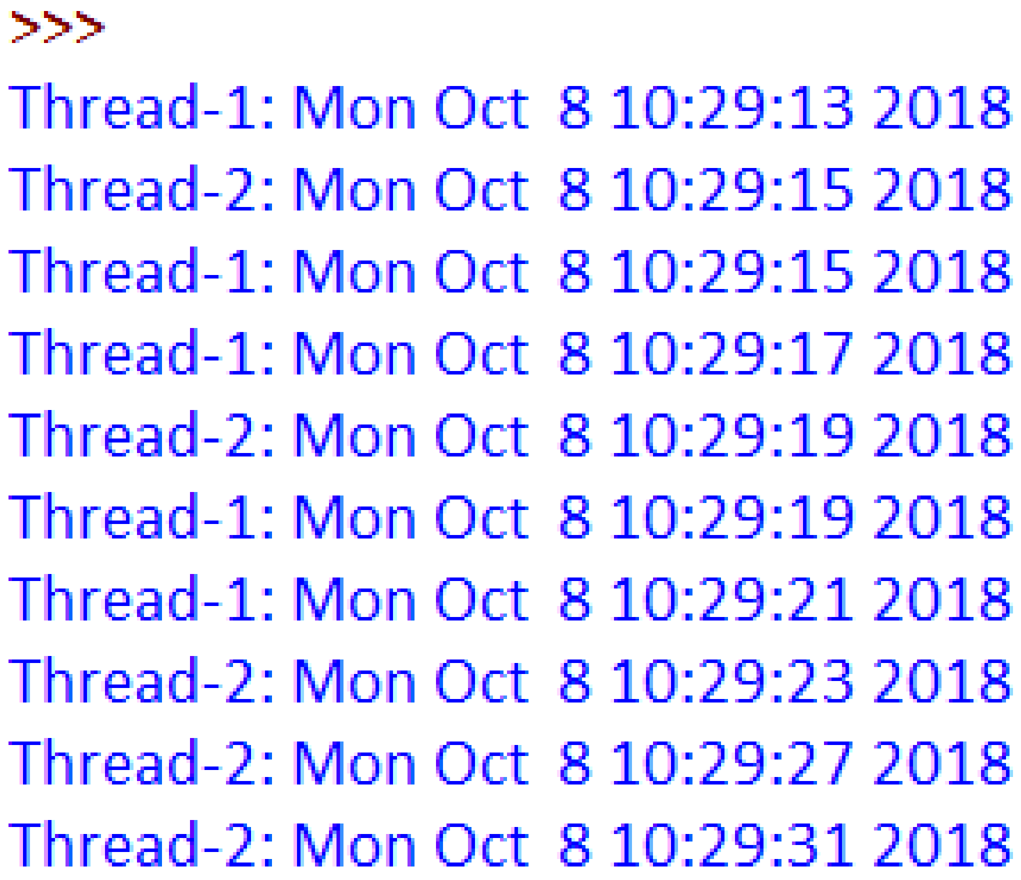
  kwargs (keyword arguments) is an optional

  The start_new_thread( ) It starts_new_thread( ) method returns immediately. It starts the child thread and calls function with the passed list of args. When function returns ,the thread terminates

- **Note:** In the start_new_thread() method use an empty tuple to call function without passing any arguments

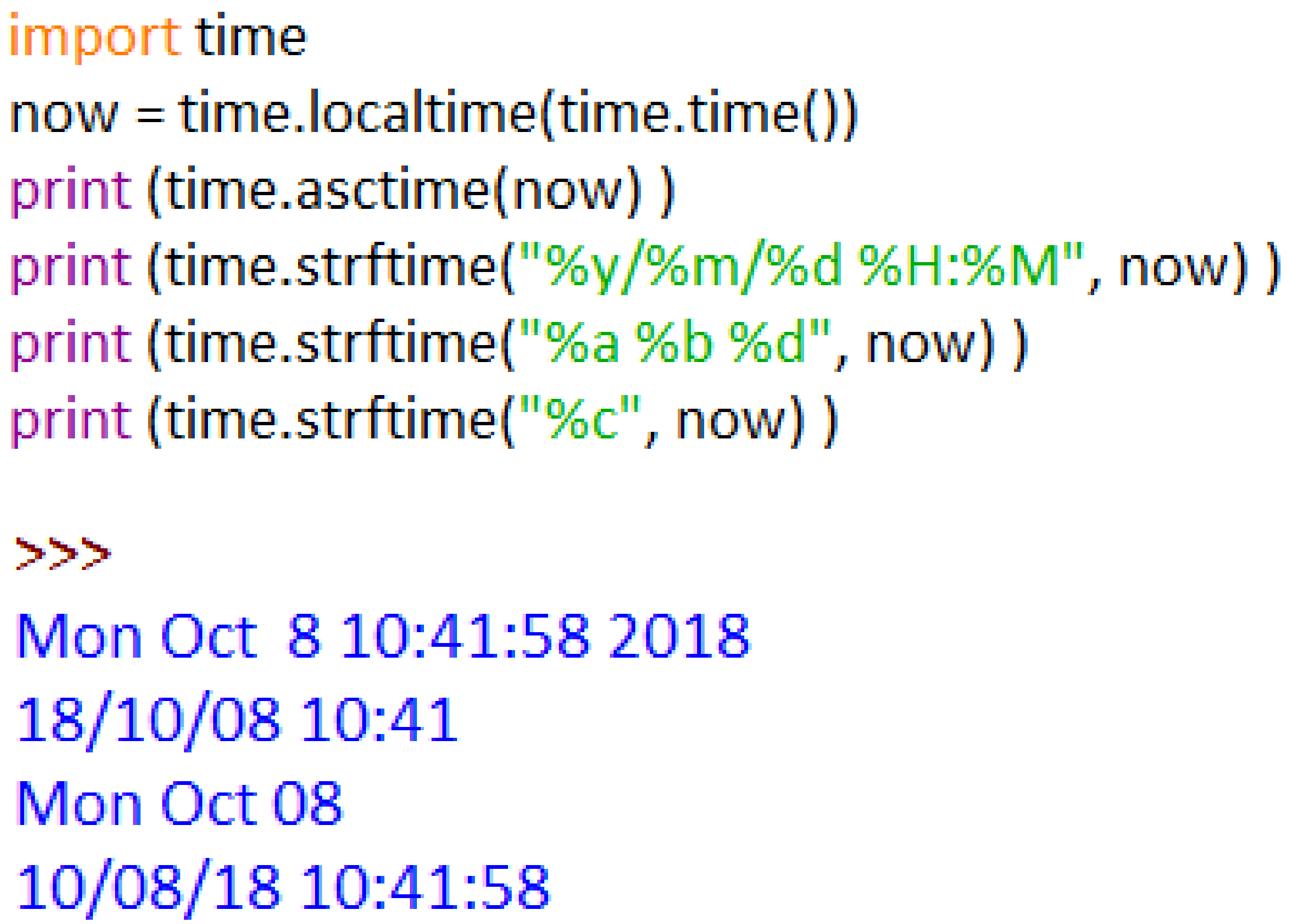
```
# MT_ex1: Program to implement multi-threading. The threads print the current time

import _thread
import time
#Define a function for the thread
def display_time(threadName, delay):
   count=0
   while count<5:
      time.sleep(delay)
      count +=1
      print("%s: %s" % (threadName, time.ctime(time.time())))
# Create two threads as follows
try:
   _thread.start_new_thread(display_time,("Thread-1", 2,))
   _thread.start_new_thread(display_time,("Thread-2", 4,))
except:
   print("Error : unable to start thread")
```

Result:



```
>>>
Thread-1: Mon Oct  8 10:29:13 2018
Thread-2: Mon Oct  8 10:29:15 2018
Thread-1: Mon Oct  8 10:29:15 2018
Thread-1: Mon Oct  8 10:29:17 2018
Thread-2: Mon Oct  8 10:29:19 2018
Thread-1: Mon Oct  8 10:29:19 2018
Thread-1: Mon Oct  8 10:29:21 2018
Thread-2: Mon Oct  8 10:29:23 2018
Thread-2: Mon Oct  8 10:29:27 2018
Thread-2: Mon Oct  8 10:29:31 2018
```

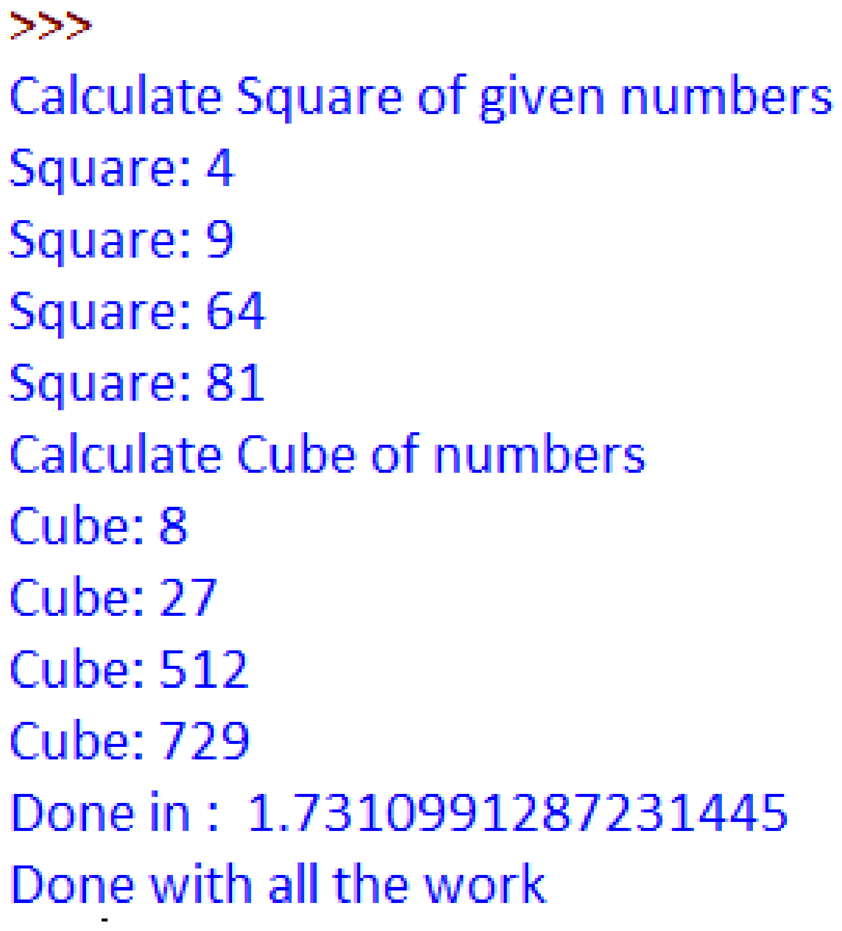
Time Module



```
import time
now = time.localtime(time.time())
print (time.asctime(now) )
print (time.strftime("%y/%m/%d %H:%M", now) )
print (time.strftime("%a %b %d", now) )
print (time.strftime("%c", now) )

>>>
Mon Oct  8 10:41:58 2018
18/10/08 10:41
Mon Oct 08
10/08/18 10:41:58
```

```
# Without using MultiThreading [ Improve the TimePeriod]
# For a given list of numbers print square and cube of every numbers
# Input= [2,3,8,9]
# Output: Square list - [4,9,64,81]
#        Cube lsit - [8,27,512,729]

import time # Time Module

def cal_square(numbers):    # First Function
    print("Calculate Square of given numbers")
    for n in numbers:
        time.sleep(0.2) # Time Delay, Wait for 0.2 sec Period, which mean ideal time where CPU is ideal
        print("Square:",n*n)   # Print Square

def cal_cube(numbers):      # Second Fucntion
    print("Calculate Cube of numbers")
    for n in numbers:
        time.sleep(0.2)   # Time Delay
        print("Cube:",n*n*n)    # # Time Delay, Wait for 0.2 sec Period, which mean ideal time where CPU is ideal

numbers=[2,3,8,9]

t=time.time() # to display the complete time taken to execute these two functions
cal_square(numbers)
cal_cube(numbers)
print("Done in : ",time.time()-t)
print("Done with all the work")
```

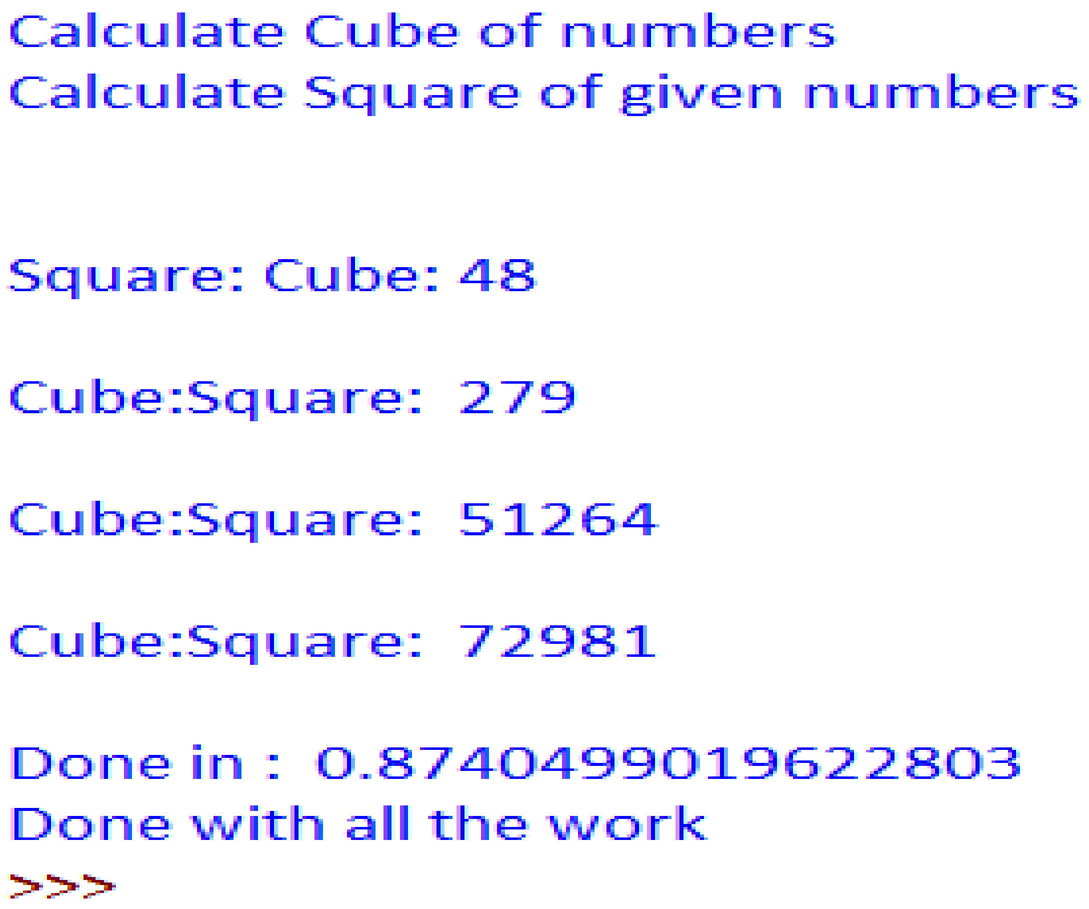
Result:



```
>>>
Calculate Square of given numbers
Square: 4
Square: 9
Square: 64
Square: 81
Calculate Cube of numbers
Cube: 8
Cube: 27
Cube: 512
Cube: 729
Done in :  1.7310991287231445
Done with all the work
```

```
# Using MultiThreading we can execute this program much faster
# For a given list of numbers print square and cube of every numbers
# Input= [2,3,8,9]
# Output: Square list - [4,9,64,81]   Cube lsit - [8,27,512,729]
import time # Time Module
import threading   # Multithreading Module
def cal_square(numbers):
    print("Calculate Square of given numbers\n")
    for n in numbers:
        time.sleep(0.2)
        print("Square:",n*n)
def cal_cube(numbers):
    print("Calculate Cube of numbers\n")
    for n in numbers:
        time.sleep(0.2)
        print("Cube:",n*n*n)
numbers=[2,3,8,9]
t=time.time()
# Creating threads 1 and 2
t1=threading.Thread(target=cal_square,args=(numbers,)) # Task1 you want to excute
t2=threading.Thread(target=cal_cube,args=(numbers,))    # Task2 you want to excute
# Start the threads
t1.start()
t2.start()
# Now two threads are executed in Parallel
t1.join()
t2.join()
print("Done in : ",time.time()-t)
print("Done with all the work")
```

Result:



```
Calculate Cube of numbers
Calculate Square of given numbers


Square: Cube: 48

Cube:Square:  279

Cube:Square:  51264

Cube:Square:  72981

Done in :  0.8740499019622803
Done with all the work
>>>
```

The Threading Module

- The threading module, which was first introduced in python 2.4, provides much more powerful, high-level support for threads than the thread module

- It defines some additional methods as follows:
- **threading.activeCount( )** : Returns the number of active thread objects
- **threading.currentThread( )** : Returns the count of thread objects in the caller's thread control
- **threading.enumerate( )** : Returns a list of all active thread objects

- Besides these methods, the threading module has the Thread class that implements threading.

- The methods providing by the Thread class include:

- **run( )** : This method is the entry point for a thread
- **start( )**: This method starts the execution of a thread by calling the run method
- **join([time])**: The join() method waits for threads to terminate
- **isAlive( )**: The isAlive() method checks whether a thread is still executing
- **getName( )**: As the name suggests, it returns the name of a thread
- **setName( )**: This method is used to set name of a thread

Creating a Thread using Threading Module

- Follow the steps to implement a new thread using the Threading module
  - Define a new subclass of the Thread class
  - Override the _init_() method
  - Override the run()method.
  - In the run() method specify the instruction that the thread should perform when started
  - Create a new Thread subclass and then use its object to start the thread by calling its start()method which in turn calls the run() method.

- # MT_ex2: Program to create a thread using the threading module

```
import threading
import time
class myThread(threading.Thread): #create a sub class
    def _init_(self,name,count):
        threading.Thread._init_(self)
        self.name =name
        self.count =count
    def run(self):
        print("\n Starting" +self.name)
        i=0
        while i<self.count:
            display(self.name,i)
            time.sleep(1)
            i+=1
        print("\n Exiting "+self.name)
def display(threadName,i):
    print("\n",threadName,i)
#create new threads
thread1 = myThread("ONE",5)
thread2 = myThread("TWO",5)
#Start new Threads
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print("\n Exiting Main Thread")
```

Output:

```
Starting ONE    Starting TWO
ONE   0
TWO  0
ONE    1
TWO  1
ONE   2
TWO  2
ONE   3
TWO  3
ONE   4
TWO  4
Exiting  ONE
Exiting   TWO
Exiting Main Thread
```

Synchronizing Threads

- When two or more threads need access to a shared resource, they need a way to ensure that the resource will be used only by one thread at a particular time. Else, may lead to serious problems
- The Threading module also includes a locking mechanism to synchronize threads
- It supports the following methods:
- **lock()** method when invoked returns the new lock
- **acquire([blocking])** method of the new lock object forces threads to run synchronously. The optional blocking parameter is used to control whether the thread waits to acquire the lock
  - If the value of blocking is 0, the thread returns 0 if the lock cannot be acquired and 1 if the lock was acquired
  - If blocking is set to 1, the thread blocks and waits for the lock to be released
- **release()** method of the new lock object releases the lock when it is no longer required
- **Note :** The output of the following program may vary on your PC subject to the processors speed and number of applications running currently

# Program to synchronize threads by locking mechanism

```
import threading
import time
class myThread(threading.Thread):
   # create a sub class
  stopFlag=0
  def __init__(self,name,msg):
    threading.Thread.__init__(self)
    self.name=name
    self.msg=msg
  def run(self):
    print("\n Starting" +self.name)
    self.display()
    time.sleep(1)
    print("\n Exiting "+self.name)
  def display(self):
      while self.stopFlag!=3:
        Lock.acquire()
        print("[",self.msg,"]")
        Lock.release()
        self.stopFlag+=1
Lock=threading.Lock()
#Create new threads
thread1=myThread("ONE","HELLO")
thread2=myThread("TWO","WORLD")
#Start new Threads
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print("\n Exiting Main Thread")
```

Output:

- StartingONE
- StartingTWO
- [HELLO]
- [HELLO]
- [HELLO]
- [WORLD]
- [WORLD]
- [WORLD]
- Exiting ONE
- Exiting TWO
- Exiting Main Thread

IIIT BASARA

Infosys | Campus Connect

For Details Contact Me @ :

9247448766

ravikanth27787@gmail.com

python

Programming Language