======================================

Decorators in Python

======================================

=>Decorator is one of the Function which will provides Additional Processing capability to the normal Function value and returns the modified value.

=>A Decorator Function is always takes Normal Function as parameter

--------------

Syntax: -

-------------

def functionname1(functionname ): # Decorator

def innerfunctionname(): # Inner Function name

val=functionname()

--------------------------

#Do the operation on ' val '

-------------------------

return resut # Inner Funtion must return modified value

return innerfunctionname # Decorator returns inner function name

=>here functionname1 is called Decorator function

=>here Functionname as a formal parameter. Every decorator function must take normal function as parameter.

#decex1.py

def getval(): # Normal Function defined by me and used by many programmers

return float (input ("Enter Any Numerical value:"))

def calsquare(gv): # Decorator

def operation ():

n=gv()

res=n\*\*2

return res

return operation

def calsquareroot(hyd):# Decorator

def op ():

val=hyd ()

res=val\*\*0.5

return res

return op

#main program

opresult=calsquare( getval )

res=opresult()

print("Square=",res)

print("----------------------------------------------")

opres=calsquareroot(getval)

r=opres()

print ("Square Root=",r)

#decex2.py

def calsquare(gv): # Decorator

def operation ():

n=gv ()

res=n\*\*2

return res

return operation

@calsquare

def getval(): # Normal Function defined by me and used by many programmers

return float (input ("Enter Any Numerical value:"))

#main program

res=getval()

print("Square =",res)

#decex3.py

def calsquareroot(gv): # Decorator

def operation():

n=gv()

res=n\*\*0.5

return res

return operation

@calsquareroot

def getval(): # Normal Function defined by me and used by many programmers

return float(input ("Enter Any Numerical value:"))

#main program

res=getval()

print ("Square Root =“, res)

#decex4.py

def calsquareroot(gv): # Decorator

def operation ():

n=gv ()

res=n\*\*0.5

return res

return operation

def calsquare(gv): # Decorator

def operation ():

n=gv()

res=n\*\*2

return res

return operation

@calsquare

def getval1(): # Normal Function defined by me and used by many programmers

return float(input("Enter Any Numerical value:"))

@calsquareroot

def getval2(): # Normal Function defined by me and used by many programmers

return float(input("Enter Any Numerical value:"))

#main program

res=getval1()

print("Square =",res)

res=getval2()

print("Square Root =",res)

#non-decex1.py

def getval(): # Normal Function defined by me and used by many programmers

return float (input("Enter Any Numerical value:"))

def calsquare(): # defined by Prog1

n=getval()

res=n\*\*2

print("Square=“, res)

def calsquareroot(): # defined by Prog2

n=getval ()

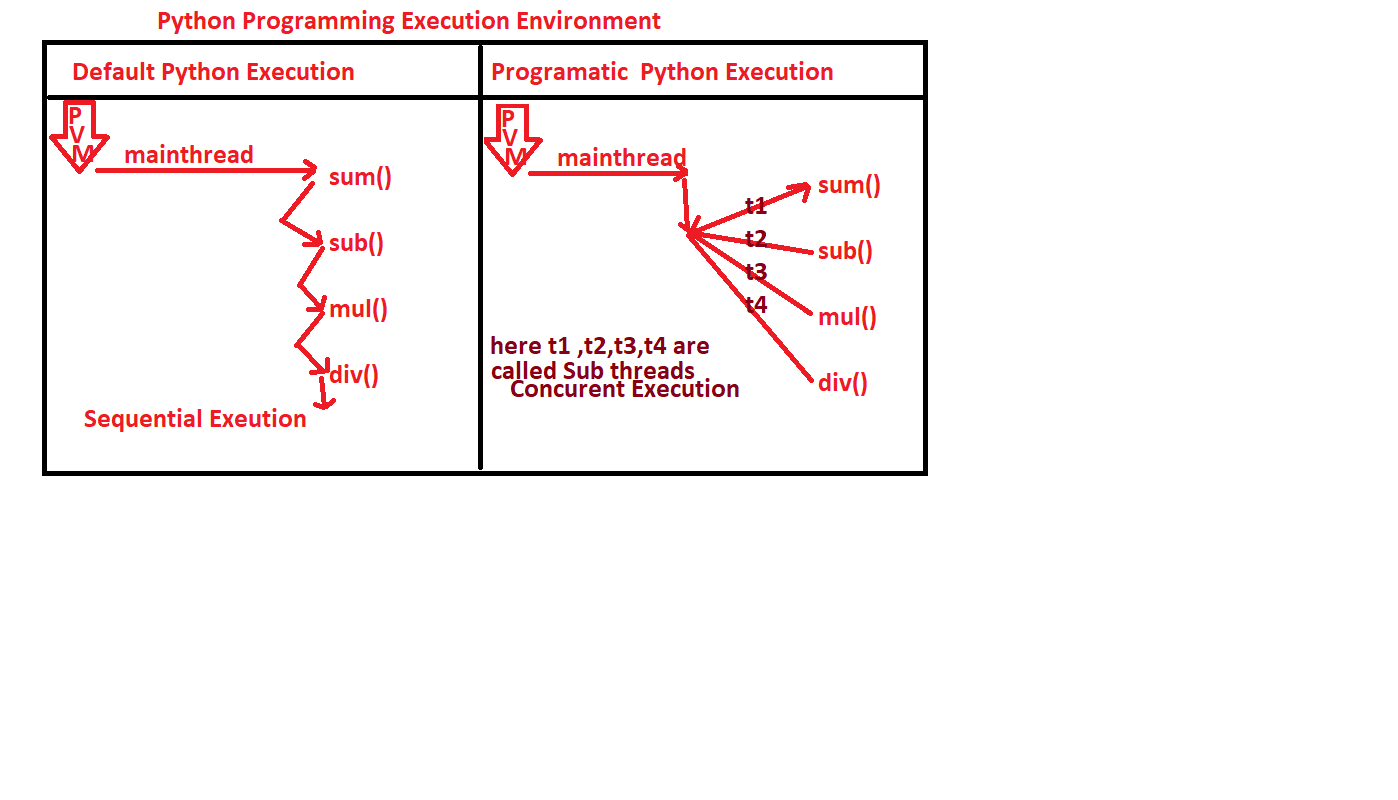
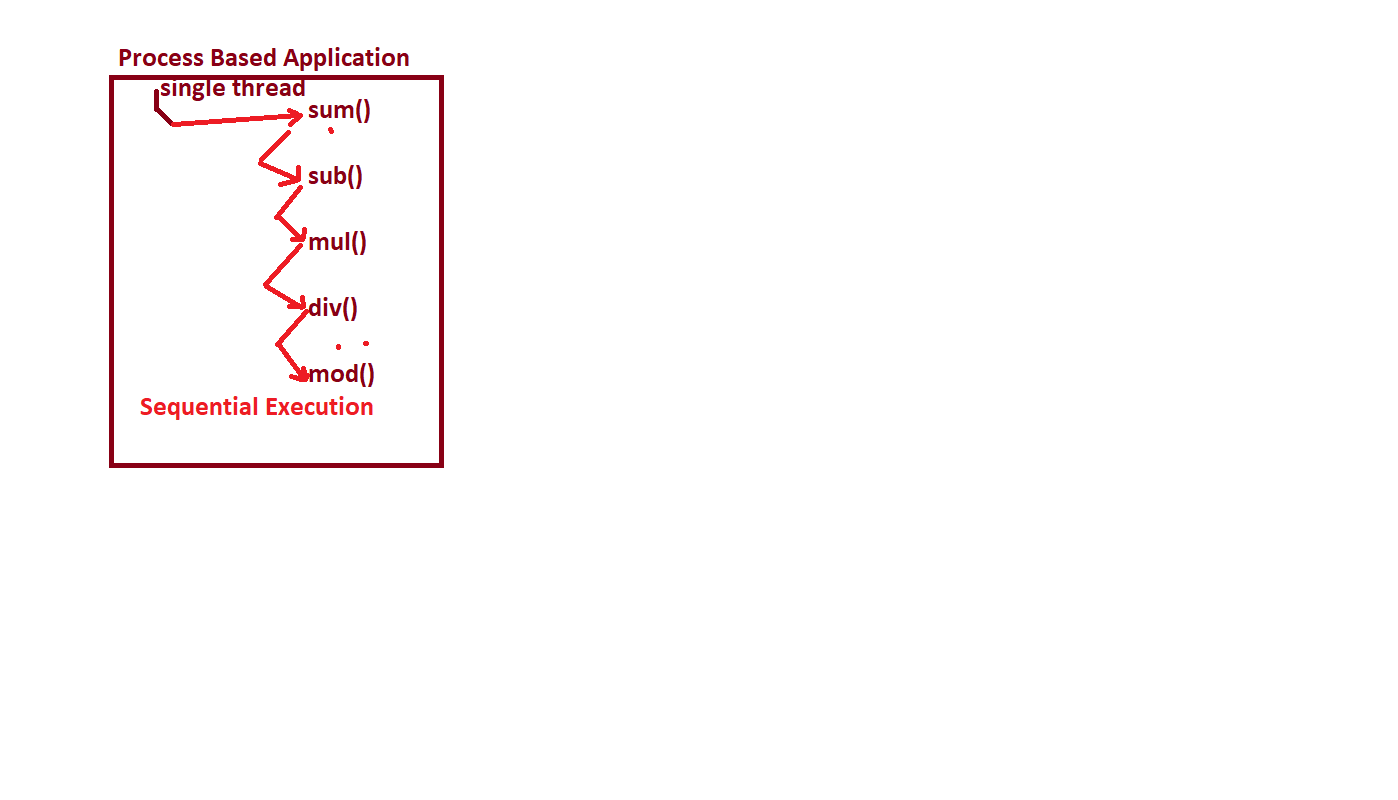
res=n\*\*0.5

print ("Square Root=",res)

#main program

calsquare()

calsquareroot()

#non-threadingEx1.py

import threading,time

def findsquares(lst):

for val in lst:

print ("5-->Therad Name: {}-->Square({})={}".format(threading.current\_thread().name,val,val\*\*2))

time.sleep(1)

def findcubes(lst):

for val in lst:

print ("10-->Therad Name: {} -->cubes({})={}".format(threading.current\_thread().name,val,val\*\*3))

time.sleep(1)

#main program

bt=time.time()

print("Line-15->Default Name Thread in main program=",threading.current\_thread().name)

lst=[12,5,6,10,23,-5,15]

findsquares(lst) # Function Call

print("\nMain Thread at Line: 18")

findcubes(lst)

et=time.time()

print ("Total Time Taken by this Program={}".format(et-bt))

#WiththreadingEx1.py

import threading,time

def findsquares(lst):

for val in lst:

print (" 5-->Therad Name:{}-->Square({})={}".format(threading.current\_thread().name,val,val\*\*2))

time.sleep(1)

def findcubes(lst):

for val in lst:

print("10-->Therad Name:{}-->cubes({})={}".format(threading.current\_thread().name,val,val\*\*3))

time.sleep(1)

#main program

bt=time.time()

print("Line-15->Default Name Thread in main program=",threading.current\_thread().name) # Main Thread

print("Initial Number of Threads=",threading.active\_count())

lst=[12,5,6,10,23,-5,15]

#create sub threads/ child threads

t1=threading.Thread(target=findsquares,args=(lst,) ) # Thread-1

t2=threading.Thread(target=findcubes,args=(lst,))# Thread-2

t1.name="ROssum"

t2.name="Travis"

t1.start()

t2.start()

print("Number of Threads=",threading.active\_count())

t1.join()

t2.join()

print("Number of Threads=",threading.active\_count())

et=time.time()

print("Total Time Taken by Threads Program={}".format(et-bt))

==============================================

Number of approaches for developing Thread Based Applications

==============================================

=>we can Develop thread-based applications in 2 ways. They are

1. By using Functional Programming

2. By using Object Oriented Programming

-----------------------------------------------------------------------

1. By using Functional Programming

-----------------------------------------------------------------------

=>Steps:

1. Import threading Module and other modules if required

2. Define Programmer-Defined Function which contains logic to perform the task.

3.create sub thread(s)

4. dispatch the sub thread(s) by using start ()

Examples:

--------------------

Refer NumberGenEx1.py

---------------------------------------------------------

1. By using Object Oriented Programming

--------------------------------------------------------

=>Steps:

1. Import threading Module and other modules if required

2. Define Programmer-defined class name

3. Define Programmer-defined Method Name

4. Create Sub Thread(s)

5. Dispatch the sub threads by using start ()

Examples: Refer NumberGenEx2.py

=========================x==============================

======================================================

Module Name used for Developing Thread Based Applications

======================================================

=>For Developing Thread Based Applications, we use a pre-defined module called "threading".

================================================================

MODULE NAME: threading

================================================================

Functions Names in "threading" module

-----------------------------------------------------------------------

1) current\_thread ():- This Function is used for obtaining Name of the thread.

Syntax: varname=threading.current\_thread()

-----------------

Examples: -

-----------------

tname=threading.current\_thread()

print ("Default Name of thread=“, tname.name)

2) active\_count(): This Function is used obtaining number of therads which are running / active.

Syntax: varname=threading.active\_count()

-----------------

Example

-----------------

noat=threading.active\_count()

print ("Number of active threads=",noat) # 1

-------------------

================================================================

Class Name in "threading" module: "Thread"

---------------------------------------------------------------------------------------------------------------

Methods in Thread Class

---------------------------------------------------------------------------------------------------------------

1) Thread(target,args)

-------------------------------------

Syntax: -

varname=threading.Thread(target=FunctionName,args=(list of values if any)

=>here varname is an object of Thread class and treated as sub thread / child thread.

=>Examples: t1=threading.Thread(target=findsqauares,args=(lst,) )

t2=threading.Thread(traget=welcome)

Here t1 and t2 are called Sub Threads OR Child Threads.

---------------------------------------------

2) is\_alive ()

---------------------------------------------

=>This Function is used for checking whether the Sub thread is under execution or not.

=>If the sub thread is under running Process, then it returns True otherwise it returns False.

=>Syntax: var name=threading.is\_alive()

Examples: Refer ThreadEx2.py

--------------------------------------------------------

3. start ()

--------------------------------------------------------

=>This Function is used for dispatching sub threads to the corresponding target

function.

=>Syntax: subthreadobj.start()

=>Examples Refer ThreadEx2.py

================================================================

4. setName(): It is used for setting the User-Freindly thread name for sub thread

syntax: subthreadobj.setName(" User-Freindly thread name")

----------

Here setName() deprecated on the name of "name" attribute

Syntax: subthreadobj.name=User-Freindly thread name

----------

Example: Refer ThreadEx3.py

---------------------------------------------

5. getName()

---------------------------------------------

=> It is used for getting the thread name

----------

syntax: varname=threadobj.getName(")

----------

Here getName() deprecated on the name of "name" attribute

-----------

Syntax: threadobj.name

----------

Example: Refer ThreadEx3.py

-----------------------------------------------------------------------------------------------------

6) join ():

---------------------------------------------

=>This Function is used for making sub threads to join with main thread.

=>When sub threads object calls join () then main thread will wait until until sub threads complete their execution.

=>Syntyax:- subthreadobj1.join()

subthreadobj2.join()

----------------------------

----------------------------

subthreadobj-n.join()

Examples: Refer NumberGenEx1.py

-----------------------------------------------------------------------------------------------------

#Program for finding default thread name

#ThreadEx1.py

import threading

tname=threading.current\_thread()

print("Default Name of thread=",tname.name)

#OR

dfname=threading.current\_thread().name

print("Default Name of thread=",dfname)

noat=threading.active\_count()

print("Number of active threads=",noat) # 1

#Program for creating sub threads

#ThreadEx2.py

import threading

def welcome ():

print ("\ni am from welcome () ")

print ("welcome () executed by:{}".format(threading.current\_thread().name))

# Main program

print("Default Thread Name:{}".format(threading.current\_thread().name))

t1=threading.Thread(target=welcome)#creating sub thread

es=t1.is\_alive()

print("Is sub thread under execution before start():{}".format(es))#False

t1.start()

print("Is sub thread under execution after start():{}".format(t1.is\_alive()))#True

#Program for creating sub threads and setting and getting the names

#ThreadEx3.py

import threading

def welcome():

print("\ni am from welcome() ")

print("welcome() executed by:{}".format(threading.current\_thread().name))

# main program

print ("Default Thread Name:{}".format(threading.current\_thread().name))

t1=threading.Thread(target=welcome) #creating sub thread

print("Default Name of Sub Thread=",t1.name) # Thread-1

#set programmer-defined name to sub thread by using setName()

t1.name="KVR"

print("Programmer-defined Name for Sub Thread=",t1.name) # KVR

#Write a python program which will generate 1 to n numbers after each and every second

#NumberGenEx1.py

import threading,time # Step-1

def numbergenerate(n): # Step-2

print ("Name of the thread in numbergenerate()=",threading.current\_thread().name)

if(n<=0):

print("{} is invalid input:".format(n))

else:

print("-"\*50)

print("Numbers within:{}".format(n))

print("-"\*50)

for i in range(1,n+1):

print("\t{}".format(i))

time.sleep(1)

print("-"\*50)

#main program

n=int(input("Enter How Many Numbers u want to generate:"))

t1=threading.Thread(target=numbergenerate,args=(n,)) # Sub thread -- # Step-3

t1.start() # Step-4

print("Line-21-->Number of active threads=",threading.active\_count())

t1.join()

print("Line-23-->Number of active threads after completion=",threading.active\_count())

#Write a python program which will generate 1 to n numbers after each and every second

#NumberGenEx2.py

import threading,time # Step-1

def numbergenerate(): # Step-2

n=int (input("Enter How Many Numbers u want to generate:"))

print ("Name of the thread in numbergenerate()=",threading.current\_thread().name)

if(n<=0):

print("{} is invalid input:".format(n))

else:

print("-"\*50)

print("Numbers within:{}".format(n))

print("-"\*50)

for i in range(1,n+1):

print("\t{}".format(i))

time.sleep(1)

print("-"\*50)

#main program

t1=threading.Thread(target=numbergenerate) # Sub thread -- # Step-3

t1.start() # Step-4

print("Line-21-->Number of active threads=",threading.active\_count())

t1.join()

print("Line-23-->Number of active threads after completion=",threading.active\_count())

#Write a python program which will generate 1 to n numbers after each and every second

#NumberGenEx3.py

import threading,time # Step-1

class Numbers: # Step-2

def generate(self,n): # Step-3

print ("Name of the thread in generate()=",threading.current\_thread().name)

if(n<=0):

print("{} is invalid input:".format(n))

else:

print("-"\*50)

print("Numbers within:{}".format(n))

print("-"\*50)

for i in range(n,0,-1):

print("\t{}".format(i))

time.sleep(1)

print("-"\*50)

#main program

no=Numbers()

t1=threading.Thread(target=no.generate,args=(int(input("Enter How Many Numbers u want to generate:")),)) # Step-4

t1.name="KVR"

t1.start() # Step-5

#Write a thread-based application which will generate even number and odd number separately by using multiple threads

#EvenOddThreadsEx1.py

import threading,time

def even(n):

for val in range(2,n+1,2):

print("{} --->{}".format(threading.current\_thread().name,val))

time.sleep(1)

def odd(n):

for val in range(1, n+1,2):

print("{} --->{}".format(threading.current\_thread().name,val))

time.sleep(1)

#main program

n=int(input("Enter How Many Even and Odd Number u want:"))

t1=threading.Thread(target=even, args=(n,))

t1.name="EvenThread"

t2=threading.Thread(target=odd, args=(n,))

t2.name="OddThread"

t1.start()

t2.start()

#write a thread based application which will generate even number and odd number separately by using multiple threads

#EvenOddThreadsEx2.py

import threading,time

class EvenNumbers:

def \_\_init\_\_(self,n):

self.n=n

def even(self):

for val in range(2,self.n+1,2):

print("{}--->{}".format(threading.current\_thread().name,val))

time.sleep(1)

class OddNumbers:

def \_\_init\_\_(self,n):

self.n=n

def odd(self):

for val in range(1,self.n+1,2):

print("{}--->{}".format(threading.current\_thread().name,val))

time.sleep(1)

#main program

n=int(input("Enter How Many Even and Odd Number u want:"))

eo=EvenNumbers(n) # Object creation and calling Parameterized Const

od=OddNumbers(n) # Object creation and calling Parameterized Const

t1=threading. Thread(target=eo.even)

t1.name="EvenThread"

t2=threading.Thread(target=od.odd)

t2.name="OddThread"

t1.start()

t2.start()

#Write a thread-based application which will generate even number and odd number separately by using multiple threads

#EvenOddThreadsEx3.py

import threading,time

class EvenNumbers:

def \_\_init\_\_(self,n):

self.n=n

def even(self):

for val in range (2, self. n+1,2):

print ("{} --->{}”. format(threading.current\_thread().name,val))

time.sleep(1)

class OddNumbers:

def \_\_init\_\_(self,n):

self.n=n

def odd(self):

for val in range (1, self. n+1,2):

print ("{} --->{}”. format(threading.current\_thread().name,val))

time.sleep(1)

#main program

n=int(input("Enter How Many Even and Odd Number u want:"))

t1=threading.Thread(target=EvenNumbers(n).even)

t1.name="EvenThread"

t2=threading.Thread(target=OddNumbers(n) .odd)

t2.name="OddThread"

t1.start()

t2.start()

#write a thread based application which will generate even number and odd number separately by using multiple threads

#EvenOddThreadsEx3.py

import threading,time

class EvenNumbers:

def \_\_init\_\_(self,n):

self.n=n

def even(self):

for val in range(2,self.n+1,2):

print("{}--->{}".format(threading.current\_thread().name,val))

time.sleep(1)

class OddNumbers:

def \_\_init\_\_(self,n):

self.n=n

def odd(self):

for val in range(1,self.n+1,2):

print("{}--->{}".format(threading.current\_thread().name,val))

time.sleep(1)

#main program

n=int(input("Enter How Many Even and Odd Number u want:"))

t1=threading.Thread(target=EvenNumbers(n).even)

t1.name="EvenThread"

t2=threading.Thread(target=OddNumbers(n) .odd)

t2.name="OddThread"

t1.start()

t2.start()

================================================

Synchronization in Multi-Threading

(OR)

Locking concept in Threading

================================================

=>When multiple threads are operating / working on the same resource (function / method) then by default we get dead lock result / race condition / wrong result / non-thread safety result.

=>To overcome this dead lock problems, we must apply the concept of Synchronization

=>The advantage of synchronization concept is that to avoid dead lock result and provides Thread Safety Result.

=>In Python Programming, we can obtain synchronization concept by using locking and un-locking concept.

--------------------------------------------------------------------------------------

=>Steps for implementing Synchronization Concept:

(OR)

Steps for avoiding dead lock

--------------------------------------------------------------------------------------

1) obtain / create an object of Lock class, which is present in threading module.

Syntax: -

-----------

lockobj=threading.Lock()

2) To obtain the lock on the sharable resource, we must use acquire ()

Syntax:

--------------

lockobj.acquire()

Once current object acquires the lock, other thread objects are made wait until curent thread object releases the lock.

3) To un-lock the sharable resource/current object, we must use release ()

Syntax:

-------------

lockobj.release()

Once current object releases the lock, other objects are permitted into shrable resource. This process of aquiring and releasing the lock will be continued until all the objects completed their execution.

#MulTablesFunNonSyncEx1.py

import threading,time

def multable(n):

if(n<=0):

print("{} is invalid Input".format(n))

else:

print("-"\*50)

print("\tMul Table for:{} by thread Name:{}".format(n,threading.current\_thread().name))

print("-"\*50)

for i in range(1,11):

print("\t{} x {}={}".format(n,i,n\*i))

time.sleep(1)

else:

print("-"\*50)

#main program

t1=threading.Thread(target=multable,args=(10,))

t2=threading.Thread(target=multable,args=(12,))

t3=threading.Thread(target=multable,args=(19,))

t4=threading.Thread(target=multable,args=(2,))

t1.start()

t2.start()

t3.start()

t4.start()

#MulTablesOopsNonSyncEx1.py

import threading,time

class Tables:

def multable(self,n):

if(n<=0):

print("{} is invalid Input".format(n))

else:

print("-"\*50)

print("\tMul Table for:{} by thread Name:{}".format(n,threading.current\_thread().name))

print("-"\*50)

for i in range(1,11):

print("\t{} x {}={}".format(n,i,n\*i))

time.sleep(1)

else:

print("-"\*50)

#main program

t1=threading.Thread(target=Tables().multable,args=(10,))

t2=threading.Thread(target=Tables().multable,args=(12,))

t3=threading.Thread(target=Tables().multable,args=(19,))

t4=threading.Thread(target=Tables().multable,args=(2,))

t1.start()

t2.start()

t3.start()

t4.start()

#MulTablesFunWithSyncEx1.py

import threading,time

def multable(n):

#Obtain the lock

L.acquire()

if(n<=0):

print("{} is invalid Input".format(n))

else:

print("-"\*50)

print("\tMul Table for:{} by thread Name:{}".format(n,threading.current\_thread().name))

print("-"\*50)

for i in range(1,11):

print("\t{} x {}={}".format(n,i,n\*i))

time.sleep(1)

else:

print("-"\*50)

#release the Lock

L.release()

#main program

L=threading.Lock() # Step-1--Creating an object of Lock class

#creating Multiple thrads with same target resurce

t1=threading.Thread(target=multable,args=(10,))

t2=threading.Thread(target=multable,args=(-12,))

t3=threading.Thread(target=multable,args=(19,))

t4=threading.Thread(target=multable,args=(2,))

t1.start()

t2.start()

t3.start()

t4.start()

#MulTablesOopsWithSyncEx1.py

import threading,time

class Tables:

def multable(self,n):

L.acquire()

if(n<=0):

print("{} is invalid Input".format(n))

else:

print("-"\*50)

print("\tMul Table for:{} by thread Name:{}".format(n,threading.current\_thread().name))

print("-"\*50)

for i in range(1,11):

print("\t{} x {}={}".format(n,i,n\*i))

time.sleep(1)

else:

print("-"\*50)

L.release()

#main program

L=threading.Lock()

t1=threading.Thread(target=Tables().multable,args=(10,))

t2=threading.Thread(target=Tables().multable,args=(12,))

t3=threading.Thread(target=Tables().multable,args=(19,))

t4=threading.Thread(target=Tables().multable,args=(2,))

t1.start()

t2.start()

t3.start()

t4.start()

#MulTablesOopsWithSyncEx2.py

import threading,time

class Tables:

L=threading.Lock() # class level Data Member

def multable(self,n):

Tables.L.acquire()

if(n<=0):

print("{} is invalid Input".format(n))

else:

print("-"\*50)

print("\tMul Table for:{} by thread Name:{}".format(n,threading.current\_thread().name))

print("-"\*50)

for i in range(1,11):

print("\t{} x {}={}".format(n,i,n\*i))

time.sleep(1)

else:

print("-"\*50)

Tables.L.release()

#main program

t1=threading.Thread(target=Tables().multable,args=(10,))

t2=threading.Thread(target=Tables().multable,args=(-12,))

t3=threading.Thread(target=Tables().multable,args=(19,))

t4=threading.Thread(target=Tables().multable,args=(-2,))

t1.start()

t2.start()

t3.start()

t4.start()

#MulTablesOopsWithSyncEx3.py

import threading,time

class Tables:

L=threading.Lock() # class level Data Member

def multable(self,n):

self.L.acquire()

if(n<=0):

print("{} is invalid Input".format(n))

else:

print("-"\*50)

print("\tMul Table for:{} by thread Name:{}".format(n,threading.current\_thread().name))

print("-"\*50)

for i in range(1,11):

print("\t{} x {}={}".format(n,i,n\*i))

time.sleep(1)

else:

print("-"\*50)

self.L.release()

#main program

t1=threading.Thread(target=Tables().multable,args=(10,))

t2=threading.Thread(target=Tables().multable,args=(-12,))

t3=threading.Thread(target=Tables().multable,args=(19,))

t4=threading.Thread(target=Tables().multable,args=(-2,))

t1.start()

t2.start()

t3.start()

t4.start()

#MulTablesOopsWithSyncEx4.py

import threading,time

class Tables:

@classmethod

def getLock(cls):

cls.lck=threading.Lock() # class level Data Member

def multable(self,n):

Tables.lck.acquire()

if(n<=0):

print("{} is invalid Input".format(n))

else:

print("-"\*50)

print("\tMul Table for:{} by thread Name:{}".format(n,threading.current\_thread().name))

print("-"\*50)

for i in range(1,11):

print("\t{} x {}={}".format(n,i,n\*i))

time.sleep(1)

else:

print("-"\*50)

Tables.lck.release()

#main program

Tables.getLock()

t1=threading.Thread(target=Tables().multable,args=(10,))

t2=threading.Thread(target=Tables().multable,args=(-12,))

t3=threading.Thread(target=Tables().multable,args=(19,))

t4=threading.Thread(target=Tables().multable,args=(-2,))

t1.start()

t2.start()

t3.start()

t4.start()

#MalyaOopsWithSyncEx4.py

import threading

class Malya:

def \_\_init\_\_(self):

self.amount=10000

def giveMoney(self,custamt):

L.acquire()

print ("\nCutsomer Name:{}".format(threading.current\_thread().name))

if(custamt<=self.amount):

self.amount=self.amount-custamt

print("{} got Money from Mayla".format(threading.current\_thread().name))

print("Remining amount in Malya Account:{}".format(self.amount))

else:

print("{},Check Bounced and contact Malya:".format(threading.current\_thread().name))

L.release()

#main program

L=threading.Lock()

m=Malya()

t1=threading.Thread(target=m.giveMoney,args=(10001,))

t2=threading.Thread(target=m.giveMoney,args=(9000,))

t3=threading.Thread(target=m.giveMoney,args=(10000,))

t4=threading.Thread(target=m.giveMoney,args=(1000,))

t1.start()

t2.start()

t3.start()

t4.start()

#Reservation.py

import threading,time

class Train:

def \_\_init\_\_(self,seats):

self.seats=seats

self.L=threading.Lock()

def reservation(self,pseats):

self.L.acquire()

if(pseats>self.seats):

print("Dear Passenger:{}, {} are not available:".format(threading.current\_thread().name,pseats))

else:

self.seats=self.seats-pseats

print("Dear Passenger:{}, {} are Reserved:".format(threading.current\_thread().name,pseats))

print("Reaming Seats in Train:{}".format(self.seats))

time.sleep(1)

self.L.release()

#main program

t=Train(10)

p1=threading.Thread(target=t.reservation,args=(14,))

p1.name="Ramesh"

p2=threading.Thread(target=t.reservation,args=(14,))

p2.name="Rajesh"

p3=threading.Thread(target=t.reservation,args=(5,))

p3.name="Rossum"

p4=threading.Thread(target=t.reservation,args=(2,))

p4.name="Sheela"

p1.start()

p2.start()

p3.start()

p4.start()