

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Chapter-24\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*24.Multi Threading \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

………………………………………………………………………………………………

\*\*\*\*\*\*Multi Tasking:\*\*\*\*:-

…………………………………………………………….

Executing several tasks simultaneously is the concept of multitasking.

There are 2 types of Multi Tasking

1. Process based Multi Tasking

2. Thread based Multi Tasking

…………………………………………………………………………

1. Process based Multi Tasking:

…………………………………………………

Executing several tasks simmultaneously where each task is a seperate independent process is called process based multi tasking.

……………………………………………………………

Eg: while typing python program in the editor we can listen mp3 audio songs from the same system. At the same time we can download a file from the internet. All these taks are executing simultaneously and independent of each other. Hence it is process based multi tasking.

……………………………………………….

This type of multi tasking is best suitable at operating system level.

………………………………………………………………………

2. Thread based MultiTasking:

………………………………………………..

Executing several tasks simultaneously where each task is a seperate independent part of the same program, is called Thread based multi tasking, and each independent part is called a Thread.

This type of multi tasking is best suitable at programmatic level.

…………………………………………………………….

Note: Whether it is process based or thread based, the main advantage of multi tasking is to improve performance of the system by reducing response time.

………………………………………………………..

The main important application areas of multi threading are: 1. To implement Multimedia graphics

2. To develop animations

3. To develop video games

4. To develop web and application servers etc...

………………………………………………………………………………

Note: Where ever a group of independent jobs are available, then it is highly recommended to execute simultaneously instead of executing one by one.For such type of cases we should go for Multi Threading.

…………………………………………

Python provides one inbuilt module "threading" to provide support for developing threads. Hence developing multi threaded Programs is very easy in python.

………………………………………………………….

Every Python Program by default contains one thread which is nothing but MainThread.

…………………………………………………………………………

Q.Program to print name of current executing thread:

1) import threading

2) print("Current Executing Thread:",threading.current\_thread().getName())

…………………………………………………

o/p: Current Executing Thread: MainThread

……………………………………………………………………………………..

Note: threading module contains function current\_thread() which returns the current executing Thread object. On this object if we call getName() method then we will get current executing thread name.

…………………………………………………………………………………

The ways of Creating Thread in Python:

We can create a thread in Python by using 3 ways

1. Creating a Thread without using any class

2. Creating a Thread by extending Thread class

3. Creating a Thread without extending Thread class

…………………………………………………………………

1. Creating a Thread without using any class:

1) from threading import \*

2) def display():

3) for i in range(1,11):

4) print("Child Thread")

5) t=Thread(target=display) #creating Thread object

6) t.start() #starting of Thread

7) for i in range(1,11):

8) print("Main Thread")

………………………………………………………………

If multiple threads present in our program, then we cannot expect execution order and hence we cannot expect exact output for the multi threaded programs. B'z of this we cannot provide exact output for the above program.It is varied from machine to machine and run to run.

…………………………………………………………………………..

Note: Thread is a pre defined class present in threading module which can be used to create our own Threads.

……………………………………………………………………………….

2. Creating a Thread by extending Thread class

We have to create child class for Thread class. In that child class we have to override run() method with our required job. Whenever we call start() method then automatically run() method will be executed and performs our job.

1) from threading import \*

2) class MyThread(Thread):

3) def run(self):

4) for i in range(10):

5) print("Child Thread-1")

6) t=MyThread()

7) t.start()

8) for i in range(10):

9) print("Main Thread-1")

…………………………………………………………………………………

3. Creating a Thread without extending Thread class:

1) from threading import \*

2) class Test:

3) def display(self):

4) for i in range(10):

5) print("Child Thread-2")

6) obj=Test()

7) t=Thread(target=obj.display)

8) t.start()

9) for i in range(10):

10) print("Main Thread-2")

…………………………………………………………………………………..

Without multi threading:

1) from threading import \*

2) import time

3) def doubles(numbers):

4) for n in numbers:

5) time.sleep(1)

6) print("Double:",2\*n)

7) def squares(numbers):

8) for n in numbers:

9) time.sleep(1)

10) print("Square:",n\*n)

11) numbers=[1,2,3,4,5,6]

12) begintime=time.time()

13) doubles(numbers)

14) squares(numbers)

15) print("The total time taken:",time.time()-begintime)

…………………………………………………………………………..

\*\*\*\*\*\*\*\*\*\*\*\*\*With multithreading:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

………………………………………………………………………………………

1) from threading import \*

2) import time

3) def doubles(numbers):

4) for n in numbers:

5) time.sleep(1)

6) print("Double:",2\*n)

7) def squares(numbers):

8) for n in numbers:

9) time.sleep(1)

10) print("Square:",n\*n)

11)

12) numbers=[1,2,3,4,5,6]

13) begintime=time.time()

14) t1=Thread(target=doubles,args=(numbers,))

15) t2=Thread(target=squares,args=(numbers,))

16) t1.start()

17) t2.start()

18) t1.join()

19) t2.join()

20) print("The total time taken:",time.time()-begintime)

………………………………………………………………………………………………\*\*\*\*\*\*\*\*\*\*\*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.

………………………………………………………………………

Eg:

1) from threading import \*

2) print(current\_thread().getName())

3) current\_thread().setName("Pawan Kalyan")

4) print(current\_thread().getName())

5) print(current\_thread().name)

…………………………………………………………………….

Output:

MainThread

Pawan Kalyan

Pawan Kalyan

………………………………………………………………………………………….

\*\*\*\*\*\*\*\*\*\*Thread Identification Number (ident):\*\*\*\*\*\*\*:-

…………………………………………………………………………………………..

For every thread internally a unique identification number is available. We can access this id by using implicit variable "ident"

1) from threading import \*

2) def test():

3) print("Child Thread")

4) t=Thread(target=test)

5) t.start()

6) print("Main Thread Identification Number:",current\_thread().ident)

7) print("Child Thread Identification Number:",t.ident)

…………………………………………………………

Output:

Child Thread

Main Thread Identification Number: 2492

Child Thread Identification Number: 2768

………………………………………………………………………

active\_count():-

……………………………..

This function returns the number of active threads currently running.

……………………………………………………………….

Eg:

1) from threading import \*

2) import time

3) def display():

4) print(current\_thread().getName(),"...started")

5) time.sleep(3)

6) print(current\_thread().getName(),"...ended")

7) print("The Number of active Threads:",active\_count())

8) t1=Thread(target=display,name="ChildThread1")

9) t2=Thread(target=display,name="ChildThread2")

10) t3=Thread(target=display,name="ChildThread3")

11) t1.start()

12) t2.start()

13) t3.start()

14) print("The Number of active Threads:",active\_count()) 15) time.sleep(5)

16) print("The Number of active Threads:",active\_count())

…………………………………………………………………………………..

Output:

D:\python\_classes>py test.py The Number of active Threads: 1 ChildThread1 ...started

ChildThread2 ...started

ChildThread3 ...started

The Number of active Threads: 4

ChildThread1 ...ended

ChildThread2 ...ended

ChildThread3 ...ended

The Number of active Threads: 1

………………………………………………………………………………..

\*\*\*\*\*\*\*\*\*\*\*\*\*enumerate() function:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

………………………………………………………………………………

This function returns a list of all active threads currently running.

……………………………………………………………………..

Eg:

1) from threading import \*

2) import time

3) def display():

4) print(current\_thread().getName(),"...started")

5) time.sleep(3)

6) print(current\_thread().getName(),"...ended")

7) t1=Thread(target=display,name="ChildThread1")

8) t2=Thread(target=display,name="ChildThread2")

9) t3=Thread(target=display,name="ChildThread3")

10) t1.start()

11) t2.start()

12) t3.start()

13) l=enumerate()

14) for t in l:

15) print("Thread Name:",t.name)

16) time.sleep(5)

17) l=enumerate()

18) for t in l:

19) print("Thread Name:",t.name)

…………………………………………………………

Output:

D:\python\_classes>py test.py

ChildThread1 ...started

ChildThread2 ...started

ChildThread3 ...started

Thread Name: MainThread

Thread Name: ChildThread1

Thread Name: ChildThread2

Thread Name: ChildThread3

ChildThread1 ...ended

ChildThread2 ...ended

ChildThread3 ...ended

Thread Name: MainThread

…………………………………………………………………………………………….

\*\*\*\*\*\*\*\*\*\*\*\*\*\*isAlive():\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

……………………………………………………………………………………………. isAlive() method checks whether a thread is still executing or not.

………………………………………………

Eg:

1) from threading import \*

2) import time

3) def display():

4) print(current\_thread().getName(),"...started")

5) time.sleep(3)

6) print(current\_thread().getName(),"...ended")

7) t1=Thread(target=display,name="ChildThread1")

8) t2=Thread(target=display,name="ChildThread2")

9) t1.start()

10) t2.start()

11) Note:-3.9.0 we should Use is\_alive() instead of isAlive()

12) print(t1.name,"is Alive :",t1.isAlive())

13) print(t2.name,"is Alive :",t2.isAlive())

14) time.sleep(5)

15) print(t1.name,"is Alive :",t1.isAlive())

16) print(t2.name,"is Alive :",t2.isAlive())

…………………………………..

Output:

D:\python\_classes>py test.py

ChildThread1 ...started

ChildThread2 ...started

ChildThread1 is Alive : True

ChildThread2 is Alive : True

ChildThread1 ...ended

ChildThread2 ...ended

ChildThread1 is Alive : False

ChildThread2 is Alive : False

……………………………………………………………………………………………..

\*\*\*\*\*\*\*\*\*\*\*\*\*\*join() method:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

…………………………………………………………………………………

If a thread wants to wait until completing some other thread then we should go for join() method.

Eg:

1) from threading import \*

2) import time

3) def display():

4) for i in range(10):

5) print("Seetha Thread")

6) time.sleep(2)

7)

8) t=Thread(target=display)

9) t.start()

10) t.join()#This Line executed by Main Thread

11) for i in range(10):

12) print("Rama Thread")

………………………………………………………………………………….

In the above example Main Thread waited until completing child thread. In this case output is:

Seetha Thread

Seetha Thread

Seetha Thread

Seetha Thread

Seetha Thread

Seetha Thread

Seetha Thread

Seetha Thread

Seetha Thread

Seetha Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

………………………………………………………………………………………

Note: We can call join() method with time period also.

………………………………………………….

t.join(seconds)

…………………………………………………………………..

In this case thread will wait only specified amount of time.

………………………………………………………………………………………..

Eg:

1) from threading import \*

2) import time

3) def display():

4) for i in range(10):

5) print("Seetha Thread")

6) time.sleep(2)

7)

8) t=Thread(target=display)

9) t.start()

10) t.join(5)#This Line executed by Main Thread

11) for i in range(10):

12) print("Rama Thread")

…………………………………………………………….

In this case Main Thread waited only 5 seconds.

…………………………………………………..

Output:

Seetha Thread

Seetha Thread

Seetha Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Rama Thread

Seetha Thread

Seetha Thread

Seetha Thread

Seetha Thread

Seetha Thread

Seetha Thread

Seetha Thread

……………………………………………………………………………….

Summary of all methods related to threading module and Thread

……………………………………………………………………..

\*\*\*\*\*\*\*\*\*\*\*Daemon Threads:\*\*\*\*\*\*\*\*\*\*\*\*:-

……………………………………………………………………………………………..

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

The main objective of Daemon Threads is to provide support for Non Daemon Threads( like main thread)

…………………………………………………………………………….

Eg: Garbage Collector

Whenever Main Thread runs with low memory, immediately PVM runs Garbage Collector to destroy useless objects and to provide free memory,so that Main Thread can continue its execution without having any memory problems.

We can check whether thread is Daemon or not by using t.isDaemon() method of Thread class or by using daemon property.

………………………………………………………………………………………..

Eg:

1) from threading import \*

2) print(current\_thread().isDaemon()) #False

3) print(current\_thread().daemon) #False

We can change Daemon nature by using setDaemon() method of Thread class. t.setDaemon(True) But we can use this method before starting of Thread.i.e once thread started,we cannot change its Daemon nature,otherwise we will get RuntimeException:cannot set daemon status of active thread

………………………………………………………………………………………..

Eg:

1) from threading import \*

2) print(current\_thread().isDaemon())

3) current\_thread().setDaemon(True)

………………………………………………………….

RuntimeError: cannot set daemon status of active thread

……………………………………………………………….

\*\*\*\*\*\*\*\*\*\*\*\*Default Nature:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

…………………………………………………………………………………………

By default Main Thread is always non-daemon.But for the remaining threads Daemon nature will be inherited from parent to child.i.e if the Parent Thread is Daemon then child thread is also Daemon and if the Parent Thread is Non Daemon then ChildThread is also Non Daemon.

…………………………………………………………………..

Eg:

1) from threading import \*

2) def job():

3) print("Child Thread")

4) t=Thread(target=job)

5) print(t.isDaemon())#False

6) t.setDaemon(True)

7) print(t.isDaemon()) #True

………………………………………………………………….

Note: Main Thread is always Non-Daemon and we cannot change its Daemon Nature b'z it is already started at the beginning only.

…………………………………………………………….

Whenever the last Non-Daemon Thread terminates automatically all Daemon Threads will be terminated.

…………………………………………………………………………

Eg:

1) from threading import \*

2) import time

3) def job():

4) for i in range(10):

5) print("Lazy Thread")

6) time.sleep(2)

7)

8) t=Thread(target=job)

9) #t.setDaemon(True)===>Line-1

10) t.start()

11) time.sleep(5)

12) print("End Of Main Thread")

………………………………………………………………..

In the above program if we comment Line-1 then both Main Thread and Child Threads are Non Daemon and hence both will be executed until their completion.

…………………………………………………………………………

In this case output is:

Lazy Thread

Lazy Thread

Lazy Thread

End Of Main Thread

Lazy Thread

Lazy Thread

Lazy Thread

Lazy Thread

Lazy Thread

Lazy Thread

Lazy Thread

……………………………………………………………………………………..

If we are not commenting Line-1 then Main Thread is Non-Daemon and Child Thread is Daemon. Hence whenever MainThread terminates automatically child thread will be terminated. In this case output is

Lazy Thread

Lazy Thread

Lazy Thread

End of Main Thread

………………………………………………………………………………………………

………………………………………………………………………………………………

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*………………………………………………………………………………………………