Thread in Python

Threads in python are an entity within a process that can be scheduled for execution. In simpler words, a thread is a computation process that is to be performed by a computer. It is a sequence of such instructions within a program that can be executed independently of other codes. In Python, there are two ways to create a new Thread. In this article, we will also be making use of the **threading module** in Python. Below is a detailed list of those processes:

1. **Creating python threads using class**

```
# import the threading module
  import threading

class thread(threading.Thread):
    def __init__(self, thread_name, thread_ID):
        threading.Thread.__init__(self)
        self.thread_name = thread_name
        self.thread_ID = thread_ID


        # helper function to execute the threads
    def run(self):
        print(str(self.thread_name) +" "+ str(self.thread_ID));

thread1 = thread("GFG", 1000)
thread2 = thread("GeeksforGeeks", 2000);

thread1.start()
thread2.start()

print("Exit")
```

Output:

```
GFG 1000

GeeksforGeeks 2000

Exit
```

Now let’s look into what we did up there in the code.

1. We created a sub-class of the thread class.
2. Then we override the __init__ function of the thread class.
3. Then we override the run method to define the behavior of the thread.
4. The start() method starts a Python thread.

Process

In Python, a process is **an instance of the Python interpreter that executes Python code**. In Python, the first process created when we run our program is called the 'MainProcess'. It is also a parent process and may be called the main parent process. The main process will create the first child process or processes

The Threading Module in Python

The newer threading module included with Python 2.4 provides much more powerful, high-level support for threads than the thread module discussed in the previous section.

The threading module exposes all the methods of the thread module and provides some additional methods –

- **threading.activeCount()** – Returns the number of thread objects that are active.
- **threading.currentThread()** – Returns the number of thread objects in the caller's thread control.
- **threading.enumerate()** – Returns a list of all thread objects that are currently active.

In addition to the methods, the threading module has the Thread class that implements threading. The methods provided by the Thread class are as follows –

- **run()** – The run() method is the entry point for a thread.
- **start()** – The start() method starts a thread by calling the run method.
- **join([time])** – The join() waits for threads to terminate.
- **isAlive()** – The isAlive() method checks whether a thread is still executing.
- **getName()** – The getName() method returns the name of a thread.
- **setName()** – The setName() method sets the name of a thread.

Creating Thread Using Threading Module

To implement a new thread using the threading module, you have to do the following –

- Define a new subclass of the Thread class.
- Override the __init__(self [,args]) method to add additional arguments.
- Then, override the run(self [,args]) method to implement what the thread should do when started.

Once you have created the new Thread subclass, you can create an instance of it and then start a new thread by invoking the start(), which in turn calls run() method.

Example

```
#!/usr/bin/python
import threading
import time
exitFlag = 0
class myThread (threading.Thread):
  def __init__(self, threadID, name, counter):
    threading.Thread.__init__(self)
    self.threadID = threadID
    self.name = name
    self.counter = counter
  def run(self):
        print "Starting " + self.name
        print_time(self.name, 5, self.counter)
        print "Exiting " + self.name
def print_time(threadName, counter, delay):
    while counter:
        if exitFlag:
            threadName.exit()
        time.sleep(delay)
        print "%s: %s" % (threadName, time.ctime(time.time()))
        counter -= 1
# Create new threads
thread1 = myThread(1, "Thread-1", 1)
thread2 = myThread(2, "Thread-2", 2)
# Start new Threads
thread1.start()
thread2.start()
print "Exiting Main Thread"
```

When the above code is executed, it produces the following result –

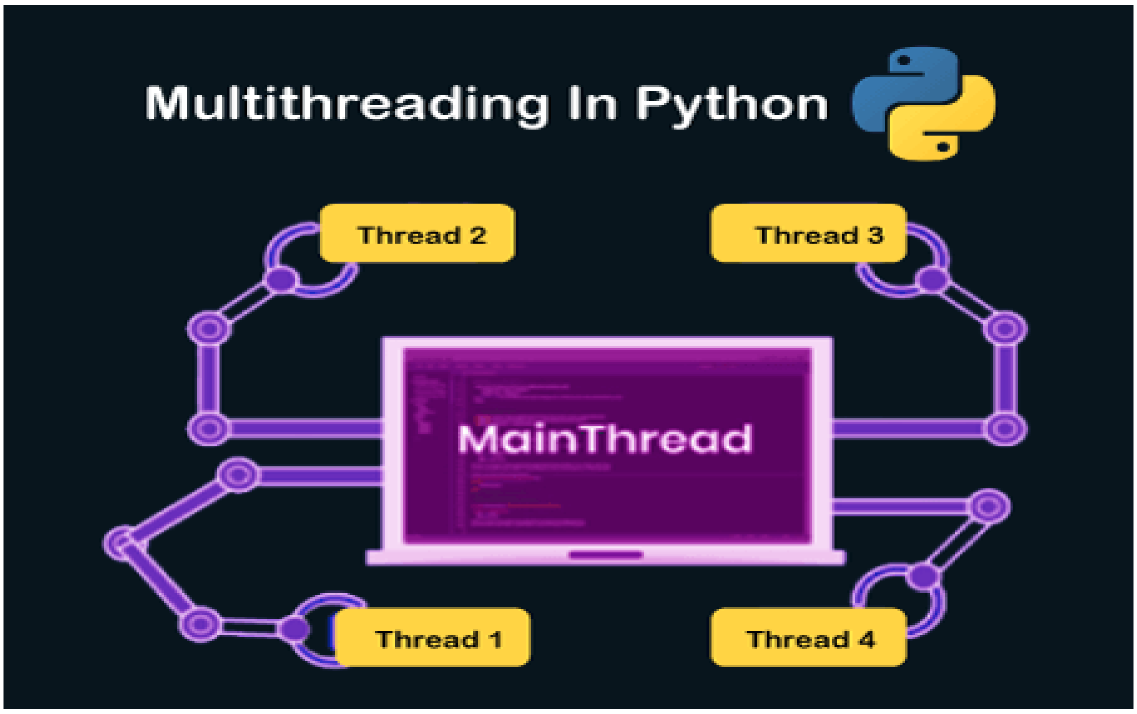
```
Starting Thread-1
Starting Thread-2
Exiting Main Thread
Thread-1: Thu Mar 21 09:10:03 2013
Thread-1: Thu Mar 21 09:10:04 2013
Thread-2: Thu Mar 21 09:10:04 2013
Thread-1: Thu Mar 21 09:10:05 2013
Thread-1: Thu Mar 21 09:10:06 2013
Thread-2: Thu Mar 21 09:10:06 2013
Thread-1: Thu Mar 21 09:10:07 2013
Exiting Thread-1
Thread-2: Thu Mar 21 09:10:08 2013
Thread-2: Thu Mar 21 09:10:10 2013
Thread-2: Thu Mar 21 09:10:12 2013
Exiting Thread-2
```

Multithreading in Python 3

A thread is the smallest unit of a program or process executed independently or scheduled by the Operating System. In the computer system, an Operating System achieves multitasking by dividing the process into threads. A thread is a lightweight process that ensures the execution of the process separately on the system. In Python 3, when multiple processors are running on a program, each processor runs simultaneously to execute its tasks separately.

Python Multithreading

Multithreading is a threading technique in Python programming to run multiple threads concurrently by rapidly switching between threads with a CPU help (called context switching). Besides, it allows sharing of its data space with the main threads inside a process that share information and communication with other threads easier than individual processes. Multithreading aims to perform multiple tasks simultaneously, which increases performance, speed and improves the rendering of the application.



Benefits of Multithreading in Python

Following are the benefits to create a multithreaded application in Python, as follows:

1. It ensures effective utilization of computer system resources.
2. Multithreaded applications are more responsive.
3. It shares resources and its state with sub-threads (child) which makes it more economical.
4. It makes the multiprocessor architecture more effective due to similarity.
5. It saves time by executing multiple threads at the same time.
6. The system does not require too much memory to store multiple threads.

When to use Multithreading in Python?

It is a very useful technique for time-saving and improving the performance of an application. Multithreading allows the programmer to divide application **tasks** into sub-tasks and simultaneously run them in a program. It allows threads to communicate and share resources such as files, data, and memory to the same processor. Furthermore, it increases the user's responsiveness to continue running a program even if a part of the application is the length or blocked.

How to achieve multithreading in Python?

There are two main modules of multithreading used to handle threads in Python.

1. The thread module
2. The threading module

Thread modules

It is started with Python 3, designated as obsolete, and can only be accessed with **_thread** that supports backward compatibility.

Syntax:

```
thread.start_new_thread ( function_name, args[, kwargs] )
```

To implement the thread module in Python, we need to import a **thread** module and then define a function that performs some action by setting the target with a variable.

Thread.py

```
import thread # import the thread module
import time # import time module

def cal_sqre(num): # define the cal_sqre function
    print(" Calculate the square root of the given number")
    for n in num:
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(' Square is : ', n * n)

def cal_cube(num): # define the cal_cube() function
    print(" Calculate the cube of  the given number")
    for n in num:
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(" Cube is : ", n * n *n)

arr = [4, 5, 6, 7, 2] # given array

t1 = time.time() # get total time to execute the functions
cal_sqre(arr) # call cal_sqre() function
cal_cube(arr) # call cal_cube() function

print(" Total time taken by threads is :", time.time() - t1) # print the total time
```

Output:

```
Calculate the square root of the given number
 Square is:  16
 Square is:  25
 Square is:  36
 Square is:  49
 Square is:  4
```

```
Calculate the cube of the given number
Cube is:  64
Cube is:  125
Cube is:  216
Cube is:  343
Cube is:  8
Total time taken by threads is: 3.005793809890747
```

Threading Modules

The threading module is a high-level implementation of multithreading used to deploy an application in Python. To use multithreading, we need to import the threading module in Python Program.

Thread Class Methods

Methods	Description
start()	A start() method is used to initiate the activity of a thread. And it calls only once for each thread so that the execution of the thread can begin.
run()	A run() method is used to define a thread's activity and can be overridden by a class that extends the threads class.
join()	A join() method is used to block the execution of another code until the thread terminates.

Follow the given below steps to implement the threading module in Python Multithreading:

1. Import the threading module

Create a new thread by importing the **threading** module, as shown.

Syntax:

```
1. import threading
```

A **threading** module is made up of a **Thread** class, which is instantiated to create a Python thread.

2. Declaration of the thread parameters: It contains the target function, argument, and **kwargs** as the parameter in the **Thread()** class.

- **Target**: It defines the function name that is executed by the thread.
- **Args**: It defines the arguments that are passed to the target function name.

For example:

```
1. import threading
2. def print_hello(n):
```

```
print("Hello, how old are you ", n)
t1 = threading.Thread( target = print_hello, args =(18, ))
```

In the above code, we invoked the **print_hello()** function as the target parameter. The **print_hello()** contains one parameter **n**, which passed to the **args** parameter.

3. Start a new thread: To start a thread in Python multithreading, call the thread class's object. The start() method can be called once for each thread object; otherwise, it throws an exception error.

Syntax:

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

4. Join method: It is a join() method used in the thread class to halt the main thread's execution and waits till the complete execution of the thread object. When the thread object is completed, it starts the execution of the main thread in Python.

Joinmethod.py

```
import threading
def print_hello(n):
    Print("Hello, how old are you? ", n)
T1 = threading.Thread( target = print_hello, args = (20, ))
T1.start()
T1.join()
Print("Thank you")
```

Output:

```
Hello, how old are you? 20
Thank you
```

When the above program is executed, the join() method halts the execution of the main thread and waits until the thread t1 is completely executed. Once the t1 is successfully executed, the main thread starts its execution.

Note: If we do not use the join() method, the interpreter can execute any print statement inside the Python program. Generally, it executes the first print statement because the interpreter executes the lines of codes from the program's start.

5. Synchronizing Threads in Python

It is a thread synchronization mechanism that ensures no two threads can simultaneously execute a particular segment inside the program to access the shared resources. The situation may be termed as critical sections. We use a race condition to avoid the critical section condition, in which two threads do not access resources at the same time.

Let's write a program to use the threading module in Python Multithreading.

Threading.py

```python
import time # import time module
import threading
from threading import *
def cal_sqre(num): # define a square calculating function
    print(" Calculate the square root of the given number")
    for n in num: # Use for loop
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(' Square is : ', n * n)

def cal_cube(num): # define a cube calculating function
    print(" Calculate the cube of  the given number")
    for n in num: # for loop
        time.sleep(0.3) # at each iteration it waits for 0.3 time
        print(" Cube is : ", n * n *n)

ar = [4, 5, 6, 7, 2] # given array

t = time.time() # get total time to execute the functions
#cal_cube(ar)
#cal_sqre(ar)
th1 = threading.Thread(target=cal_sqre, args=(ar, ))
th2 = threading.Thread(target=cal_cube, args=(ar, ))
th1.start()
th2.start()
th1.join()
th2.join()
print(" Total time taking by threads is :", time.time() - t) # print the total time
print(" Again executing the main thread")
print(" Thread 1 and Thread 2 have finished their execution.")
```

Output:

```
Calculate the square root of the given number
 Calculate the cube of the given number
 Square is:  16
 Cube is:  64
 Square is:  25
 Cube is:  125
 Square is:  36
 Cube is:  216
 Square is:  49
 Cube is:  343
 Square is:  4
 Cube is:  8
 Total time taken by threads is: 1.5140972137451172
 Again executing the main thread
 Thread 1 and Thread 2 have finished their execution.
```