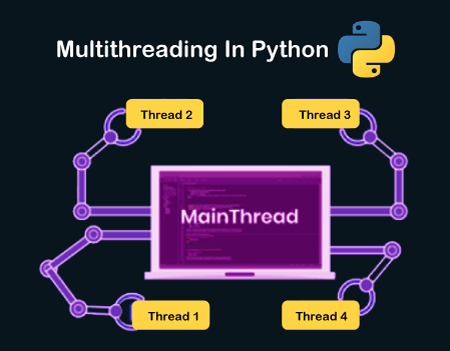
Multithreading in Python 3

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.

Note: The Python Global Interpreter Lock (GIL) allows running a single thread at a time, even the machine has multiple processors.

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](https://www.javatpoint.com/python-tutorial).

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:**

1. 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**

1. **import** thread # **import** the thread module
2. **import** time # **import** time module
4. def cal\_sqre(num): # define the cal\_sqre function
5. print(" Calculate the square root of the given number")
6. **for** n in num:
7. time.sleep(0.3) # at each iteration it waits **for** 0.3 time
8. print(' Square is : ', n \* n)
10. def cal\_cube(num): # define the cal\_cube() function
11. print(" Calculate the cube of  the given number")
12. **for** n in num:
13. time.sleep(0.3) # at each iteration it waits **for** 0.3 time
14. print(" Cube is : ", n \* n \*n)
16. arr = [4, 5, 6, 7, 2] # given array
18. t1 = time.time() # get total time to execute the functions
19. cal\_sqre(arr) # call cal\_sqre() function
20. cal\_cube(arr) # call cal\_cube() function
22. 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](https://www.javatpoint.com/python-applications). To use multithreading, we need to import the threading module in [Python Program](https://www.javatpoint.com/python-programs).

**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):
3. print("Hello, how old are you ", n)
4. 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:**

1. t1.start()
2. 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**

1. **import** threading
2. def print\_hello(n):
3. Print("Hello, how old are you? ", n)
4. T1 = threading.Thread( target = print\_hello, args = (20, ))
5. T1.start()
6. T1.join()
7. 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**

1. **import** time # **import** time module
2. **import** threading
3. from threading **import** \*
4. def cal\_sqre(num): # define a square calculating function
5. print(" Calculate the square root of the given number")
6. **for** n in num: # Use **for** loop
7. time.sleep(0.3) # at each iteration it waits **for** 0.3 time
8. print(' Square is : ', n \* n)
10. def cal\_cube(num): # define a cube calculating function
11. print(" Calculate the cube of  the given number")
12. **for** n in num: # **for** loop
13. time.sleep(0.3) # at each iteration it waits **for** 0.3 time
14. print(" Cube is : ", n \* n \*n)
16. ar = [4, 5, 6, 7, 2] # given array
18. t = time.time() # get total time to execute the functions
19. #cal\_cube(ar)
20. #cal\_sqre(ar)
21. th1 = threading.Thread(target=cal\_sqre, args=(ar, ))
22. th2 = threading.Thread(target=cal\_cube, args=(ar, ))
23. th1.start()
24. th2.start()
25. th1.join()
26. th2.join()
27. print(" Total time taking by threads is :", time.time() - t) # print the total time
28. print(" Again executing the main thread")
29. 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.