Python Multithreading – Python 3 threading module

Python Multithreading

Python Multithreading – Python's **threading** module/package allows you to create threads as objects.

In Python, or any programming language, a thread is used to execute a task where some waiting is expected. So that the main program does not wait for the task to complete, but the thread can take care of it simultaneously.

In this tutorial, we shall learn how to work with threads in detailed sections.

- Simple Example – Multiple threads
- Create a thread
- Start a thread
- Pass arguments to thread function
- Is Thread Alive
- Thread Name

Example 1 – Python Multithreading

We shall look into a simple example to threading module, and then go in detail of working with threads.

Note : The following examples are worked on environment with Python3 installed.

Following is a simple example to create multiple threads using **threading** module.

Python Program

```python
import threading

def f():
    print('Thread function\n')
    return

for i in range(3):
    t = threading.Thread(target=f)
    t.start()
```

1. Passing a method to Thread constructor.

```
def f():
    print('Thread function\n')
    return

t = threading.Thread(target=f)
```

Overriding run() method in a subclass of threading.Thread.

```
import threading

class CustomThread(threading.Thread):
    def run(self):
        print('Custom thread function.\n')

for i in range(3):
    t = CustomThread()
```

Start a Thread

A thread is started by applying start() method on the thread object.

Python Program

```
import threading
import time

def f():
    print('Thread running.\n')
    return

# start threads by passing function to Thread constructor
for i in range(3):
    t = threading.Thread(target=f)
    t.start()
```

Passing arguments to the function supplied to Thread

To pass arguments to the function supplied to Thread constructor, pass args in the Thread constructor as shown below :

Python Program

```
import threading
import time

def f(i):
    for p in range(3):
        time.sleep(i+1)
        print('Thread #',i,"\n")
        time.sleep(i)
    return

# start threads by passing function to Thread constructor
for i in range(3):
    t = threading.Thread(target=f, args=(i,))
    t.start()
```

```
Thread # 2
```

Is Thread Alive

threading.Thread.is_alive() could be used to check if the thread is alive or not.

Thread.is_alive() returns True if the thread is alive, False if not alive.

Python Program

```
import threading
import time

def f(i):
    time.sleep(i)
    return

# threads
t1 = threading.Thread(target=f, args=(1.2,), name="Thread#1")
t1.start()

t2 = threading.Thread(target=f, args=(2.2,), name="Thread#2")
t2.start()
```

```
import threading
import time

def f(i):
    time.sleep(i)
    return

# threads
t1 = threading.Thread(target=f, args=(1.2,), name="Thread#1")
t1.start()

t2 = threading.Thread(target=f, args=(2.2,), name="Thread#2")
t2.start()

for p in range(5):
    time.sleep(p*0.5)
    print('[',time.ctime(),']', t1.getName(), t1.is_alive())
    print('[',time.ctime(),']', t2.getName(), t2.is_alive())
```

Output

```
[ Tue Feb 27 17:58:54 2018 ] Thread#1 True
[ Tue Feb 27 17:58:54 2018 ] Thread#2 True
[ Tue Feb 27 17:58:55 2018 ] Thread#1 True
[ Tue Feb 27 17:58:55 2018 ] Thread#2 True
[ Tue Feb 27 17:58:56 2018 ] Thread#1 False
[ Tue Feb 27 17:58:56 2018 ] Thread#2 True
[ Tue Feb 27 17:58:57 2018 ] Thread#1 False
[ Tue Feb 27 17:58:57 2018 ] Thread#2 False
[ Tue Feb 27 17:58:59 2018 ] Thread#1 False
[ Tue Feb 27 17:58:59 2018 ] Thread#2 False
```

Thread Name

Thread name could be set or read using setName() and getName() methods.

In a function being called inside a thread, to get the current thread, use **threading.current_thread()**. In the following example we shall use this method to get current thread object.

Python Program

```
import threading
import time

def f(i):
```

following example we shall use this method to get current thread object.

Python Program

```
import threading
import time

def f(i):
    for p in range(3):
        time.sleep(i+1.5)
        print(threading.current_thread().getName())
    return

# start threads by passing function to Thread constructor
for i in range(3):
    t = threading.Thread(target=f, args=(i,))
    t.setName( 'Thread#'+str(i) )
    t.start()
```

Output

```
Thread#0
Thread#1
Thread#0
Thread#2
Thread#0
Thread#1
Thread#2
Thread#1
Thread#2
```

When does a Thread stops ?

A thread stops when :

- run() method terminates normally. [or]
- an unhandled exception causes run() method to terminate abruptly.
