How to Run a Function in a New Thread in Python

FEBRUARY 17, 2022 *by* JASON BROWNLEE *in* THREADING (HTTPS://SUPERFASTPYTHON.COM/CATEGORY/THREADING/)

You can run a function in a new thread via the "**target**" argument on the **threading.Thread** class (https://superfastpython.com/threading-in-python/).

In this tutorial you will discover how to run a function in a new thread.

Let's get started.

Table of Contents

Need to Run A Function in a New Thread

A thread is a thread of execution in a computer program.

Every Python program has at least one thread of execution referred to as the main thread. Both processes and threads are created and managed by the underlying operating system.

Sometimes we may need to create additional threads in our program in order to execute code concurrently.

Python provides the ability to create and manage new threads via the **threading.Thread** class.

One way of running a function in a new thread is via an argument on the **threading.Thread** class.

How can we run a function in a new thread using the **threading.Thread** class?

How to Run a Function In a Thread

To run a function in another thread:

1. Create an instance of the **threading.Thread** class.
2. Specify the name of the function via the "**target**" argument.
3. Call the **start()** function.

First, we must create a new instance of the **threading.Thread** class and specify the function we wish to execute in a new thread via the "**target**" argument.

```
...
# create a thread
thread = Thread(target=task)
```

The function executed in another thread may have arguments in which case they can be specified as a tuple and passed to the "**args**" argument of the **threading.Thread** class constructor or as a dictionary to the "**kwargs**" argument.

```
...
# create a thread
thread = Thread(target=task, args=(arg1, arg2))
```

We can then start executing the thread by calling the **start()** function.

The **start()** function will return immediately and the operating system will execute the function in a separate thread as soon as it is able.

```
...
# run the thread
thread.start()
```

And that's all there is to it.

We do not have control over when the thread will execute precisely or which CPU core will execute it. Both of these are low-level responsibilities that are handled by the underlying operating system.

Next, let's look at a worked example of executing a function in a new thread.

Confused by the threading module API?
Download my FREE PDF cheat sheet (https://marvelous-writer-6152.ck.page/088fc51f28)

Example of Running a Function in a Thread

First, we can define a custom function that will be executed in another thread.

We will define a simple function that blocks for a moment then prints a statement.

The function can have any name we like, in this case we'll name it "**task**".

```
# a custom function that blocks for a moment
def task():
    # block for a moment
    sleep(1)
    # display a message
    print('Th... thread')
```


Next, we can create an instance of the **threading.Thread** class and specify our function name as the "**target**" argument in the constructor.

```
...
# create a thread
thread = Thread(target=task)
```

Once created we can run the thread which will execute our custom function in a new native thread, as soon as the operating system can.

```
...
# run the thread
thread.start()
```

The **start()** function does not block, meaning it returns immediately.

We can explicitly wait for the new thread to finish executing by calling the **join()** function. This is not needed as the main thread will not exit until the new thread has completed but does make things clearer.

```
...
# wait for the thread to finish
print('Waiting for the thread...')
thread.join()
```

Tying this together, the complete example of executing a function in another thread is listed below.

```
# SuperFastPython.com
# example of running a function in another thread
from time import sleep
from threading import Thread

# a custom function that blocks for a moment
def task():
    # block for a moment
    sleep(1)
    # display a message
    print('This is from another thread')

# create a thread
thread = Thread(target=task)
# run the thread
thread.start()
# wait for the thread to finish
print('Waiting for the thread...')
thread.join()
```

Running the example first creates the **threading.Thread** then calls the **start()** function. This does not start the thread immediately, but instead allows the operating system to schedule the function to execute as soon as possible.

The main thread then prints a message waiting for the thread to complete, then calls the join() function to explicitly block and wait for the new thread to finish executing.

Once the custom function returns, the thread is closed. The **join()** function then returns and the main thread exits.

```
Waiting for the thread...
This is from another thread
```

Example of Running a Function in a Thread With Arguments

We can execute functions in another thread that take arguments.

This can be demonstrated by first updating our **task()** function from the previous section to take two arguments, one for the time in seconds to block and the second for a message to display.

```
# a custom function that blocks for a moment
def task(sleep_time, message):
    # block for a moment
    sleep(sleep_time)
    # display a message
    print(message)
```

Next, we can update the call to the **threading.Thread** constructor to specify the two arguments in the order that our **task()** function expects them as a tuple via the "**args**" argument.

```
...
# create a thread
thread = Thread(target=task, args=(1.5, 'New message from another thread'))
```

Tying this together, the complete example of executing a custom function that takes arguments in a separate thread is listed below.

```
# SuperFastPython.com
# example of running a function with arguments in another thread
from time import sleep
from threading import Thread

# a custom function that blocks for a moment
def task(sleep_time, message):
    # block for a moment
    sleep(sleep_time)
    # display a message
    print(message)

# create a thread
thread = Thread(target=task, args=(1.5, 'New message from another thread'))
# run the thread
thread.start()
# wait for the thread to finish
print('Waiting for the thread...')
thread.join()
```

Running the example creates the thread specifying the function name and the arguments to the function.

The thread is started and the function blocks for the parameterized number of seconds and prints the parameterized message.

```
Waiting for the thread...
New message from another thread
```


Further Reading

This section provides additional resources that you may find helpful.

APIs

- threading - Thread-based parallelism (https://docs.python.org/3/library/threading.html)

Guides

- Python Threading: The Complete Guide (https://superfastpython.com/threading-in-python/)

Books