How to Join a Thread in Python

MARCH 5, 2022 *by* JASON BROWNLEE *in* THREADING (HTTPS://SUPERFASTPYTHON.COM/CATEGORY/THREADING/)

You can **join a thread** by calling the **Thread.join()** function.

In this tutorial you will discover how to join threads in Python.

Let's get started.

Table of Contents

Need to Join a Thread

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

Every Python program has at least one thread of execution called 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 module and the **threading.Thread** class (https://docs.python.org/3/library/threading.html).

You can learn more about Python threads in the guude:

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

In concurrent programming, we may need to wait until another thread has finished running. This may be for many reasons, such as:

- The current thread needs a result from the target thread.
- A resource is shared between the current and target threads.
- The current thread has no other work to complete.

The **join()** method provides a way for one thread to block until another thread has finished.

How can we use the join() method to join a thread in Python?

Run your loops using all CPUs, download my FREE book (https://superfastpython.com/plip-incontent) to learn how.

How to Join a Thread

A thread can be joined in Python by calling the **Thread.join()** method (https://docs.python.org/3/library/threading.html#threading.Thread.join).

For example:

```
...
# join a thread
thread.join()
```

This has the effect of blocking the current thread until the target thread that has been joined has terminated.

The target thread that is being joined may terminate for a number of reasons, such as:

- Finishes executing it's target function.
- Finishes executing it's **run()** method if it extends the **Thread** class.
- Raised an error or exception.

Once the target thread has finished, the **join()** method will return and the current thread can continue to execute.

The **join()** method requires that you have a **threading.Thread** instance for the thread you wish to join.

This means that if you created the thread, you may need to keep a reference to the **threading.Thread** instance. Alternatively, you can use the **threading.enumerate()** function to enumerate through all active threads and locate the thread you wish to join by name.

The **join()** method also takes a "**timeout**" argument that specifies how long the current thread is willing to wait for the target thread to terminate, in seconds.

Once the timeout has expired and the target thread has not terminated, the **join()** thread will return.

```
...
# join the thread with a timeout
thread.join(timeout=10)
```

When using a timeout, it will not be clear whether the **join()** method returned because the target thread terminated or because of the timeout. Therefore, we can call the **is_alive()** function to confirm the target thread is no longer running.

For example:

```
...
# join the thread with a timeout
thread.join(timeout=10)
# check if the target thread is still running
if thread.is_alive():
    # timeout expired, thread is still running
else:
    # thread has terminated
```

Now that we know how to join a thread, let's look at some worked examples.

Example of Joining a Thread

We can explore how to join a target thread in Python.

In this example we will create a new **threading.Thread** instance and then join it from the main thread.

First, we can define a target task function to execute our new thread.

The function will first block for a moment by calling the **sleep()** function, then report a message. The complete function is listed below.

```
# target function
def task():
    # block for a moment
    sleep(1)
    # report a message
    print('All done in the new thread')
```

In the main thread, we will create a new **threading.Thread** instance configured to execute our new **task()** function, then execute the thread by calling the **start()** function.

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

Next, we wish for the main thread to wait until the new thread has terminated.

This can be achieved by calling the **join()** function on the **threading.Thread** instance for the new thread.

```
...
# wait for the new thread to finish
print('Main: Waiting for thread to terminate...')
thread.join()
```

This call will block, meaning that the main thread will not carry on executing until this function call returns, and this function call will only return once the thread that has been joined terminates.

Once the new thread terminates, the **join()** thread returns and the main thread is free to carry on executing.

```
...
# continue on
print('Main: Continuing on')
```

Tying this together, the complete example is listed below.

```
# SuperFastPython.com
# example of joining a thread
from time import sleep
from threading import Thread

# target function
def task():
    # block for a moment
    sleep(1)
    # report a message
    print('All done in the new thread')

# create a new thread
thread = Thread(target=task)
# start the new thread
thread.start()
# wait for the new thread to finish
print('Main: Waiting for thread to terminate...')
thread.join()
# continue on
print('Main: Continuing on')
```

Running the example first creates a new thread instance configured to execute our target **task()** function.

We then start the new thread from the main thread, and then wait for the new thread to finish by calling the **join()** function. The main thread does not progress, but instead blocks, waiting.

The new thread runs, blocks for a moment, reports a message, then terminates.

Once the new thread terminates, the **join()** function called in the main thread returns, and the main thread is free to continue on.

```
Main: Waiting for thread to terminate...
All done in the new thread
Main: Continuing on
```

Next, let's look at an example of joining a thread that terminates with an error.

Example of Joining a Thread That Has An Error

It just so happens that the target thread in the previous example terminated normally.

It is also possible for the target thread to terminate by raising an error or exception. Any error or exception raised in another thread will not reach the main thread, but will terminate the thread and allow the **join()** function to return and the current thread to continue on.

Let's demonstrate this with an example.

We can modify the above example so that the **task()** function raises an error instead of finishing gracefully.

```
# target function
def task():
    # block for a moment
    sleep(1)
    # report a message
    print('All done in the new thread')
    # terminate with an error
    raise RuntimeError('Something bad happened')
```

Tying this together, the complete example of the current thread joining a target thread that will terminate with an error is listed below.

```python
# SuperFastPython.com
# example of joining a thread that raises an error
from time import sleep
from threading import Thread

# target function
def task():
    # block for a moment
    sleep(1)
    # report a message
    print('All done in the new thread')
    # terminate with an error
    raise RuntimeError('Something bad happened')

# create a new thread
thread = Thread(target=task)
# start the new thread
thread.start()
# wait for the new thread to finish
print('Main: Waiting for thread to terminate...')
thread.join()
# continue on
print('Main: Continuing on')
```

Running the example first creates and starts the new thread as before.

The main thread joins the thread and waits for it to terminate.

The new thread blocks, reports a message then terminates with an error. The error is reported on standard error (stderr), the default behavior.

The new thread is terminated and the **join()** method in the main thread returns and the main thread continues on as before.

```
Main: Waiting for thread to terminate...
All done in the new thread
Exception in thread Thread-1:
Traceback (most recent call last):
  ...
    raise RuntimeError('Something bad happened')
RuntimeError: Something bad happened
Main: Continuing on
```

Next, let's look at an example of joining a thread with a timeout.

Example of Joining a Thread With a Timeout

We can join a thread and use a timeout.

A timeout allows the current thread to stop waiting for the target thread to timeout after a fixed number of seconds.

We can update the first example so that the target thread takes longer to execute, in this case five seconds. The updated **task()** function is listed below.

```
# target function
def task():
    # block for a moment
    sleep(5)
    # report a message
    print('All done in the new thread')
```

Next, we can join the new thread and set a timeout of two seconds.

```
...
# wait for the new thread to finish
print('Main: Waiting for thread to terminate...')
thread.join(timeout=2)
```

This will mean that the main thread will only block for two seconds waiting for the target thread to complete, it will not complete in time and return before the target thread has terminated.

The main thread will not know whether the **join()** function returned because the target thread terminated or because of the timeout.

Therefore, we can check if the target thread is still running by calling the **is_alive()** method on the target thread.

```
...
# check if the thread is still alive
if thread.is_alive():
    print('Main: The target thread is still running')
else:
    print('Main: The target thread has terminated')
```

Tying this together, the complete example is listed below.

```
# SuperFastPython.com
# example of joining a thread with a timeout
from time import sleep
from threading import Thread

# target function
def task():
    # block for a moment
    sleep(5)
    # report a message
    print('All done in the new thread')

# create a new thread
thread = Thread(target=task)
# start the new thread
thread.start()
# wait for the new thread to finish
print('Main: Waiting for thread to terminate...')
thread.join(timeout=2)
# check if the thread is still alive
if thread.is_alive():
    print('Main: The target thread is still running')
else:
    print('Main: The target thread has terminated')
```

Running the example creates and starts the new thread.

The main thread then joins the new thread and waits two seconds. The new thread continues running and fails to terminate in two seconds.

The **join()** function then returns after the timeout and then checks on the status of the new thread and finds that it is still running.

The main thread then terminates.

The new thread continues to execute and then reports its messages before terminating itself.

Note, the Python interpreter will only terminate when there are no non-daemon threads running, not when the main thread exits. The new thread we created is a non-daemon thread, as is the main thread.

```
Main: Waiting for thread to terminate...
Main: The target thread is still running
All done in the new thread
```

Try changing the timeout or the length of time the new thread sleeps in this example to see different messages printed (e.g. change the timeout to 10).

Next, let's look at an example of joining a thread that is not running.

Example of Joining a Thread That is Not Running

It is possible to join a thread that is not running.

The effect is that the **join()** function will not block and instead will return immediately.

We can demonstrate this with a worked example.

The main thread can block for a moment in a way that we know that the new thread has finished executing. In this case, it can sleep for two seconds, whereas we know the new thread will be finished after one second.

```
...
# block for a moment
print('Main: blocking for a moment...')
sleep(2)
```

The main thread can then attempt to join the new thread as before and wait for it to finish, even though we know it has already terminated at this point.

```
...
# wait for the new thread to finish
print('Main: Waiting for thread to terminate...')
thread.join()
# continue on
print('Main: Continuing on')
```

Tying this together, the complete example is listed below.

```
# SuperFastPython.com
# example of joining a thread that is not running
from time import sleep
from threading import Thread

# target function
def task():
    # block for a moment
    sleep(1)
    # report a message
    print('All done in the new thread')

# create a new thread
thread = Thread(target=task)
# start the new thread
thread.start()
# block for a moment
print('Main: blocking for a moment...')
sleep(2)
# wait for the new thread to finish
print('Main: Waiting for thread to terminate...')
thread.join()
# continue on
print('Main: Continuing on')
```

Running the example first creates and starts the new thread.

The main thread then blocks for two seconds with a call to **sleep()**.

The new thread finishes executing after one second and reports its message.

The main thread wakes up then attempts to join the new thread that has already terminated. The **join()** method returns immediately and the main thread carries on as per normal.

```
Main: blocking for a moment...
All done in the new thread
Main: Waiting for thread to terminate...
Main: Continuing on
```

This highlights that it is safe to call the **join()** method, even if the target thread may have already terminated.

Next, let's take a look at what happens if we try to join the current thread.

Example of Joining the Current Thread

It is possible for a thread to attempt to join itself.

That is, the main thread can call the **join()** function on a **threading.Thread** instance that represents the main thread.

The effect is that a **RuntimeError** is raised.

If an error was not raised, then the call would result in a deadlock as the thread would be blocking and waiting for itself to terminate, which of course, it would never terminate.

We can demonstrate this with an example.

First, we can get a **threading.Thread** instance for the current thread by calling the **threading.current_thread()** function.

```
...
# get the current thread
thread = current_thread()
```

We can then call the **join()** method on this thread instance.

```
...
# wait for the current thread to finish
print('Main: Waiting for thread to terminate...')
thread.join()
```

Tying this together, the complete example is listed below.

```
# SuperFastPython.com
# example of joining the current thread
from threading import current_thread
# get the current thread
thread = current_thread()
# wait for the current thread to finish
print('Main: Waiting for thread to terminate...')
thread.join()
# continue on
print('Main: Continuing on')
```

Running the example first gets a **threading.Thread** instance for the current thread.

The main thread then attempts to join the current thread (itself).

This fails and raises an error, terminating the main thread and the program as there are no other threads running.

```
Main: Waiting for thread to terminate...
Traceback (most recent call last):
...
RuntimeError: cannot join current 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