



Python Threading: 7-Day Crash Course



Super Fast Python · Follow

10 min read · Nov 12, 2023



18



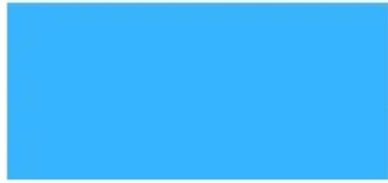
The **Python Threading** module allows you to create and manage new threads of execution in Python.

Although threads have been available since Python 2, they are not widely used. This is mainly because it is believed that Python threads cannot run in parallel because of the Global Interpreter Lock or GIL.

This is not (*entirely*) true, as the GIL is released in many cases, such as when reading/writing files, sockets over the internet, and working with libraries like NumPy. You can learn more about when to use threads here.

Ignoring threads in Python means you are leaving a lot of capability and performance on the table (e.g. fill NumPy arrays 3x faster with threads).

This crash course is designed to get you up to speed with Python threads, super fast!



Python Threading Crash Course

Python Threading: 7-Day Crash Course

Course Structure:

You have a lot of fun ahead, including:

- **Lesson 01:** How to run functions in new threads.
- **Lesson 02:** How to extend the Thread class.
- **Lesson 03:** How to be thread-safe with a mutex.
- **Lesson 04:** How to limit access with a semaphore.
- **Lesson 05:** How to return data from a thread.
- **Lesson 06:** How to use producer-consumer threads with a queue.
- **Lesson 07:** How to stop a thread.

I designed this course to be completed in one week (7 lessons in 7 days).

Take your time. Leave the page open in a browser tab and complete one lesson per day.

Download All Source Code:

You can download a zip of all the code used in this tutorial here:

- [Download all source code](#)

Email Version of This Course (+PDF Cheat Sheet)

If you would also like to receive this crash course via email, one lesson per day, you can sign up here:

- [Email version of this crash course \(+ free PDF cheat sheet\)](#).

Free Book-Length Guide to Python Threading:

Threading is a massive topic and we can't cover it all.

If you want to go deeper, I recommend my massive Python Threading guide:

- [Threading in Python: The Complete Guide](#)

Quick Question:

Your first lesson in this series is up next.

Before then, a quick question:

| *Why are you interested in Python threading?*

Let me know. Maybe I can point you in the right direction and save you a ton of time!

Lesson 01: Run A Function In A New Thread

We can easily run a function in a new thread.

First, we must create an instance of the Thread class and specify the name of the function to run via the target argument.

Next, we can start the thread by calling the `start()` method.

A new native thread will be created to execute our target function.

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.

This approach is great for running one-off tasks in a new thread.

The example below provides a complete working example of running a function in a new thread.

```
# 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()
```

Try running the example.

You can learn more about running functions in a new thread in the tutorial:

- [How to Run a Function in a New Thread in Python](#)

Lesson 02: Extend The Thread Class

We can extend the Thread class to run our code in a new child thread.

This can be achieved by first extending the Thread class, just like any other Python class.

Then the run() function of the Thread class must be overridden to contain the code that you wish to execute in another thread.

An instance of the class can then be created and the new thread started by calling the start() method.

And that's it.

Tying this together, the complete example of executing code in another thread by extending the Thread class is listed below.

```
# SuperFastPython.com
# example of extending the Thread class
from time import sleep
from threading import Thread

# custom thread class
class CustomThread(Thread):
    # override the run function
    def run(self):
        # block for a moment
        sleep(1)
        # display a message
        print('This is coming from another thread')

# create the thread
thread = CustomThread()
```

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

Try running the example.

You can learn more about how to extend the Thread class in the tutorial:

- [How to Extend the Thread Class in Python](#)

Lesson 03: Thread-Safe Code With A Mutex

Python threads can suffer thread race conditions, even with the GIL.

We can use mutual exclusion (mutex) lock for threads via the Lock class.

A mutual exclusion lock or mutex lock is a synchronization primitive intended to prevent a race condition.

An instance of the lock can be created and then acquired by threads before accessing a critical section, and released after the critical section.

The lock can be acquired via the `acquire()` method and released by calling the `release()` method.

We can achieve the same effect by using a lock object via the context manager interface. This is preferred as it ensures that the lock is always released, even if the block fails with an exception or returns.

A lock can be created and shared among multiple threads.

Tying this together, a complete example of sharing a lock among multiple threads is listed below.

```
# SuperFastPython.com
# example of a mutual exclusion (mutex) lock
from time import sleep
from random import random
from threading import Thread
from threading import Lock

# work function
def task(lock, identifier, value):
    # acquire the lock
    with lock:
        print(f'>thread {identifier} got the lock, sleeping for {value}')
        sleep(value)

# create a shared lock
lock = Lock()
# start a few threads that attempt to execute the same critical section
for i in range(10):
    # start a thread
    Thread(target=task, args=(lock, i, random())).start()
# wait for all threads to finish...
```

Try running the example.

You can learn more about how to use mutex locks with threads in the tutorial:

- [Threading Mutex Lock in Python](#)

Lesson 04: Semaphores With Threads

We can limit concurrent access by threads to a block of code using a semaphore.

A semaphore is a concurrency primitive that allows a limit on the number of threads that can acquire a lock protecting a critical section.

Python provides the Semaphore class that can be configured to let a fixed number of threads acquire it. Any additional threads that attempt to acquire it will have to wait until a position becomes available.

This is helpful in many situations such as limiting access to a file or server resource.

The number of positions is specified when creating the Semaphore object.

The semaphore can then be acquired by calling the `acquire()` method, and released via the `release()` method.

Alternatively, the context manager interface can be used, which is preferred to ensure that each acquisition is always released.

Tying this together, a complete example of limiting access to a block by threads using a semaphore is listed below.

```
# SuperFastPython.com
# example of using a semaphore
from time import sleep
from random import random
from threading import Thread
from threading import Semaphore

# target function
def task(semaphore, number):
    # attempt to acquire the semaphore
    with semaphore:
        # do work
        value = random()
        sleep(value)
        # report result
        print(f'Thread {number} got {value}')
```



```
# create a semaphore
semaphore = Semaphore(2)
# create a suite of threads
for i in range(10):
    worker = Thread(target=task, args=(semaphore, i))
    worker.start()
# wait for all workers to complete...
```

Try running the example.

You can learn more about how to use semaphores with threads in the tutorial:

- [Threading Semaphore in Python](#)

Lesson 05: Return Data From A Thread

Hi, we can run tasks in a new thread, but how can we return data?

For example, we may load a file or download some data in a new thread and then need to access the data back in the main thread.

We cannot return data directly from a Thread. For example, when we call the `start()` method, this method does not block and does not return a value.

Instead, we must return values from a thread indirectly.

We can return data from a thread by extending the Thread class to run the task, then store the data as an instance variable.

This may require initializing the instance variable in the constructor of the new class, then storing data in the instance variable in the overridden `run()` method.

The example below shows how to extend the Thread class and return data via an instance variable.

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

# custom thread
class CustomThread(Thread):
    # constructor
    def __init__(self):
        # execute the base constructor
        Thread.__init__(self)
        # set a default value
        self.value = None

    # function executed in a new thread
    def run(self):
        # block for a moment
        sleep(1)
        # store data in an instance variable
        self.value = 'Hello from a new thread'

# create a new thread
thread = CustomThread()
# start the thread
thread.start()
# wait for the thread to finish
thread.join()
# get the value returned from the thread
data = thread.value
print(data)
```

Try running the example.

You can learn more about how to return data from new threads in the tutorial:

- [Threading Return Values in Python](#)

Lesson 06: Producer-Consumer Threads With A Queue

Hi, we can share data between producer and consumer threads using a shared queue.

The queue.Queue class is thread-safe, meaning that we can add and remove data from the queue from multiple threads without fear of race conditions, data loss, or corruption.

A Queue object can be created and shared among multiple threads.

Producer threads can add data to the queue by calling the put() method.

Consumer threads can retrieve data from the queue by calling the get() method. If there are no items on the queue, the consumer will block them until some data is available.

The producer-consumer pattern is very common in concurrent programs, and the Queue class is a way we can implement it easily.

Tying this together, a complete example of producer-consumer threads with a Queue is listed below.

```
# SuperFastPython.com
# example of producer and consumer threads with a shared queue
from time import sleep
from random import random
from threading import Thread
from queue import Queue

# generate work
def producer(queue):
    print('Producer: Running')
    # generate work
    for i in range(10):
        # generate a value
        value = random()
```

```

        # block
        sleep(value)
        # add to the queue
        queue.put(value)
    # all done
    queue.put(None)
    print('Producer: Done')

# consume work
def consumer(queue):
    print('Consumer: Running')
    # consume work
    while True:
        # get a unit of work
        item = queue.get()
        # check for stop
        if item is None:
            break
        # report
        print(f'>got {item}')
    # all done
    print('Consumer: Done')

# create the shared queue
queue = Queue()
# start the consumer
consumer = Thread(target=consumer, args=(queue,))
consumer.start()
# start the producer
producer = Thread(target=producer, args=(queue,))
producer.start()
# wait for all threads to finish
producer.join()
consumer.join()

```

Try running the example.

You can learn more about how to use thread-safe queues in the tutorial:

- [Thread-Safe Queue in Python](#)

Lesson 07: Stop New Thread

Hi, we cannot kill a new thread.

The threading API does not provide a (direct) facility to terminate a new thread.

There are ways, such as killing the parent process, but it is generally not a good practice.

Instead, it is better to send a message to the thread and request that it stop as soon as possible.

This can be achieved using a threading Event.

This is a thread-safe boolean that can be created and shared among threads. Worker threads can check if the event is set and stop processes. The controlling thread can set the event when we need the thread to stop.

The example below shows how to stop a new thread using an Event.

```
# SuperFastPython.com
# example of stopping a new thread
from time import sleep
from threading import Thread
from threading import Event

# custom task function
def task(event):
    # execute a task in a loop
    for i in range(5):
        # block for a moment
        sleep(1)
        # check for stop
        if event.is_set():
            break
        # report a message
        print('Worker thread running...')
    print('Worker closing down')

# create the event
event = Event()

# create and configure a new thread
thread = Thread(target=task, args=(event,))
```

```
# start the new thread
thread.start()
# block for a while
sleep(3)
# stop the worker thread
print('Main stopping thread')
event.set()
# wait for the new thread to finish
thread.join()
```

Try running the example.

You can learn more about how to stop a thread in the tutorial:

- [How to Stop a Thread in Python](#)

Thank-You



Thank you kindly, from Jason Brownlee at SuperFastPython.com

Thank you for letting me help you learn more about threading in Python.

If you ever have any questions about this course or Python concurrency in general, please reach out.

- [Contact page](#), messages go directly to my email inbox.

Remember, you can download a zip of all the code used in this tutorial here:

- [Download all source code](#)

You can also receive this crash course via email, one lesson per day. Sign-up here:

- [Email version of this crash course \(+ free PDF cheat sheet\)](#)

Finally, if you want to go deeper into Python threading, I recommend my massive guide:

- [Threading in Python: The Complete Guide](#)

Did you enjoy this course? [Let me know](#).

Python

Threading

Multithreading

Concurrency

Python Thread



Written by Super Fast Python

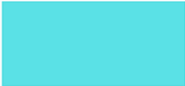
45 Followers

Follow



More from Super Fast Python

SuperFastPython
Crash Course


Python
ThreadPoolExecutor
Crash Course

 Super Fast Python


Python ThreadPoolExecutor: 7-Day
Crash Course

The Python ThreadPoolExecutor allows us to create and manage thread pools in Python.

10 min read · Dec 3, 2023

 6  1  

SuperFastPython
Crash Course

Python
ProcessPoolExecutor
Crash Course

 Super Fast Python

Python ProcessPoolExecutor: 7-
Day Crash Course

The Python ProcessPoolExecutor allows us to create and manage process pools in Python.

10 min read · Dec 10, 2023

 9  1  

Python Multiprocessing Crash Course

 Super Fast Python

Python Multiprocessing: 7-Day Crash Course

The Python multiprocessing module allows you to create and manage new child...

10 min read · Nov 26, 2023



4



Python Asyncio Crash Course

 Super Fast Python

Python Asyncio: 7-Day Crash-Course

Python Asyncio allows us to use asynchronous programming with coroutine...

13 min read · Nov 19, 2023



73



1

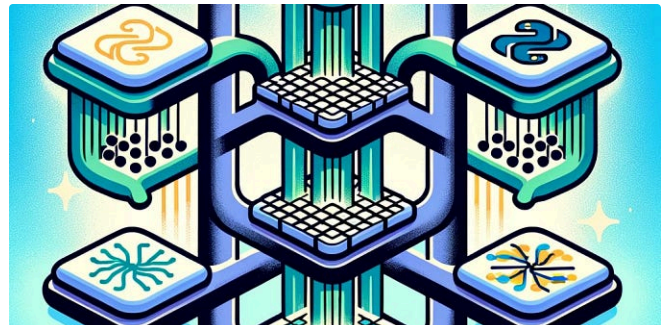


See all from Super Fast Python

Recommended from Medium

Python ThreadPoolExecutor Crash Course

 Super Fast Python



Utkarsh Singh

Python ThreadPoolExecutor: 7-Day Crash Course

The Python ThreadPoolExecutor allows us to create and manage thread pools in Python.

10 min read · Dec 3, 2023



6



1



Advanced Guide to Asyncio, Threading, and Multiprocessing i...

Python offers diverse paradigms for concurrent and parallel execution: Asyncio f...

3 min read · Dec 16, 2023



153



3

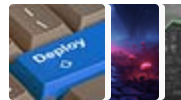


Lists



Coding & Development

11 stories · 634 saves



Predictive Modeling w/ Python

20 stories · 1239 saves



Practical Guides to Machine Learning

10 stories · 1494 saves



ChatGPT

21 stories · 655 saves



Moraneus

Mastering Python's Asyncio: A Practical Guide

When you dive into Python's world, one gem that truly shines for handling modern web an...

10 min read · Mar 7, 2024



205



2



How To Use ThreadPoolExecutor in Python 3



Gajanan Rajput

How To Use ThreadPoolExecutor in Python 3

Dive into the world of Python concurrency with our comprehensive guide on...

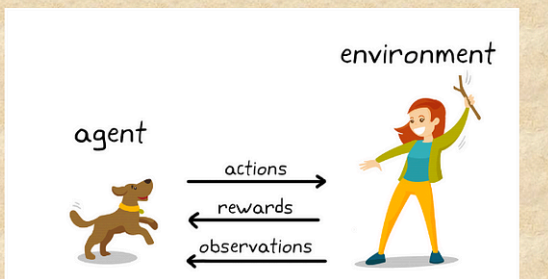
4 min read · Feb 27, 2024



155



The example of a dog



Shrutipitale

Introduction to Reinforcement Learning

Reinforcement Learning

8 min read · May 22, 2024



1



Gujarat story

Can Django Handle Multiple Requests?

Can Django Handle Multiple Requests?

4 min read · Apr 3, 2024



1



See more recommendations