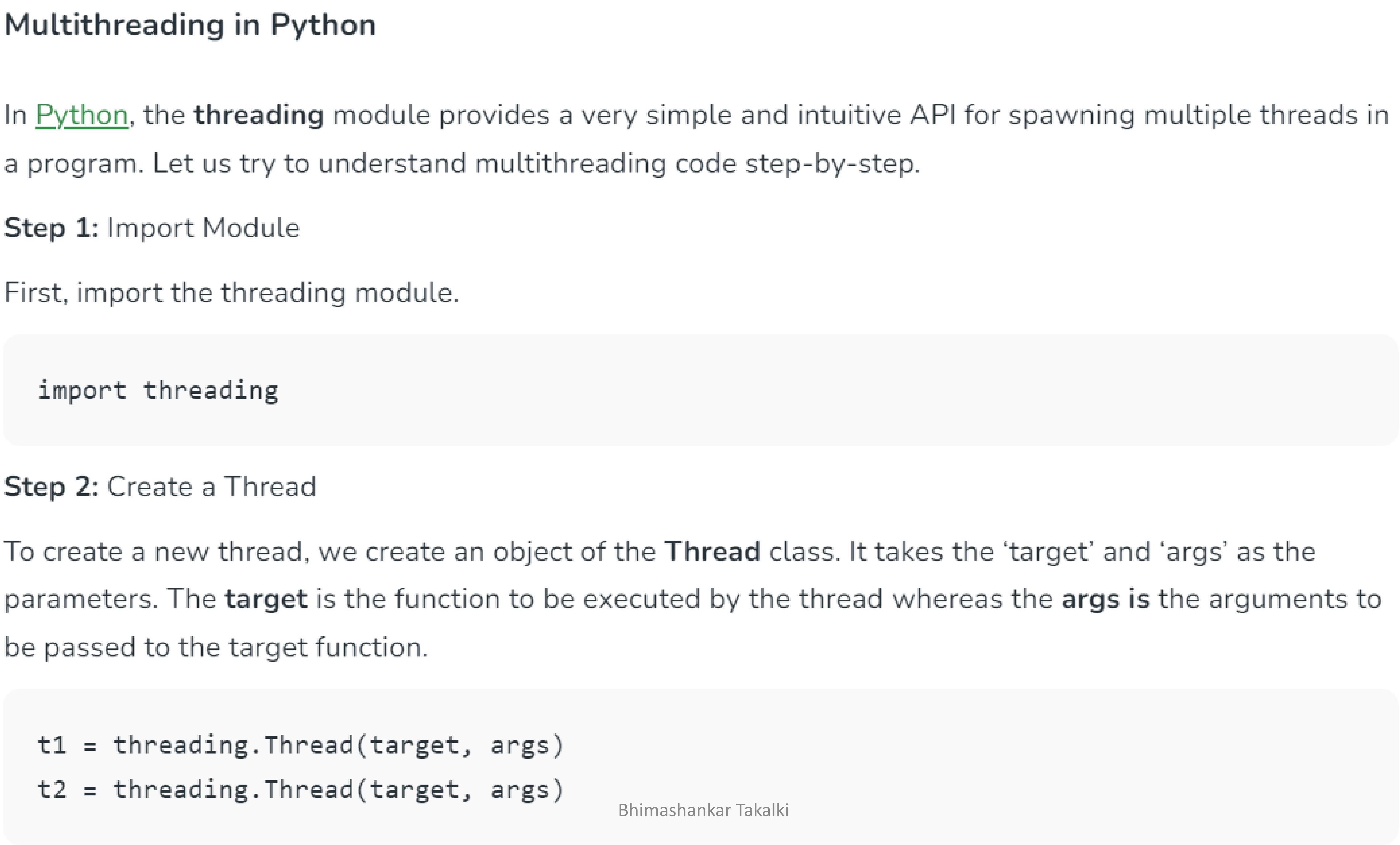
MultiThreading
In Python

Multithreading in Python allows you to execute multiple threads (lightweight processes) concurrently, enabling parallel execution of tasks and improving overall program performance, especially for I/O-bound operations. Python provides a built-in module called threading for working with threads.



Multithreading in Python

In Python, the **threading** module provides a very simple and intuitive API for spawning multiple threads in a program. Let us try to understand multithreading code step-by-step.

Step 1: Import Module

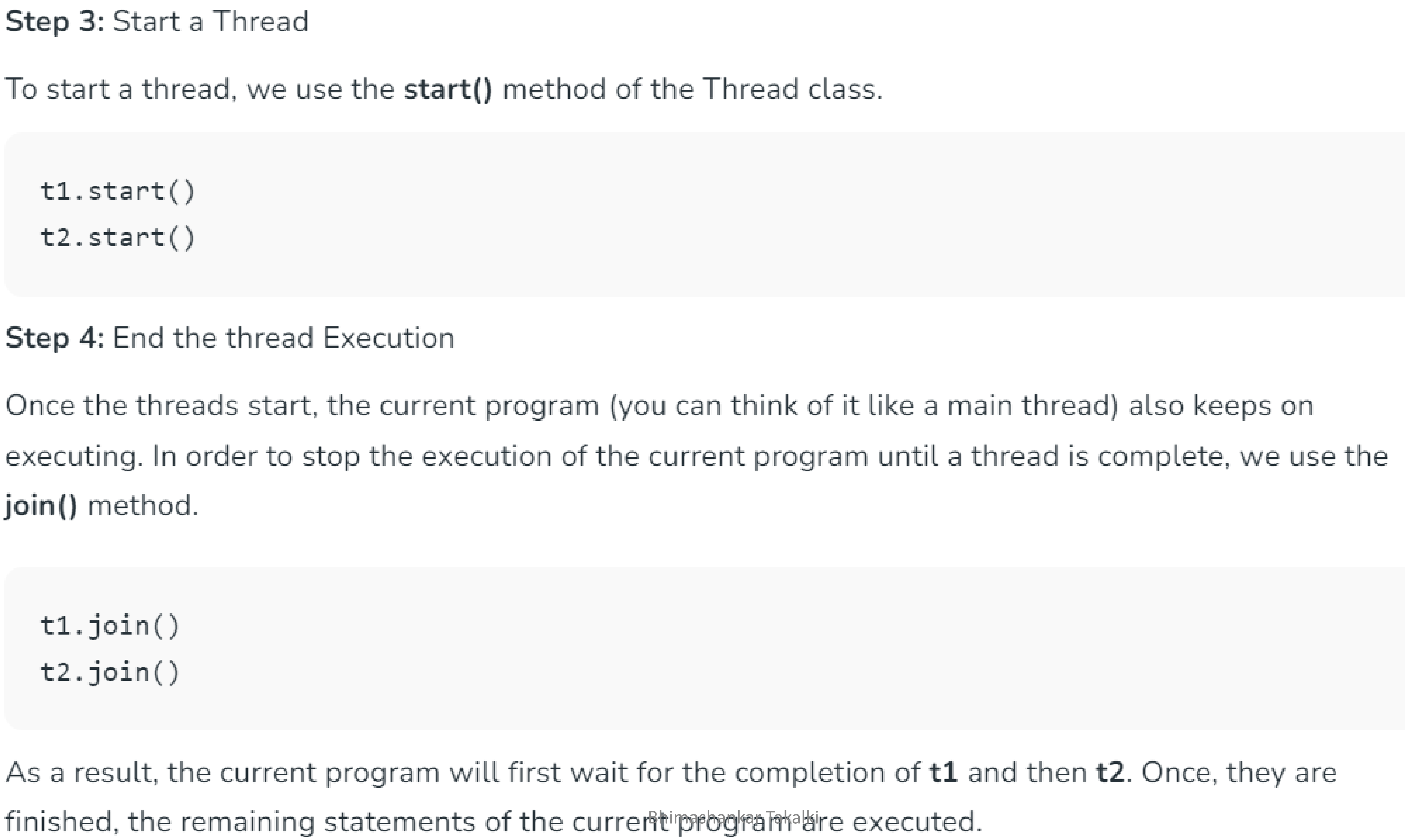
First, import the threading module.

```
import threading
```

Step 2: Create a Thread

To create a new thread, we create an object of the **Thread** class. It takes the 'target' and 'args' as the parameters. The **target** is the function to be executed by the thread whereas the **args is** the arguments to be passed to the target function.

```
t1 = threading.Thread(target, args)
t2 = threading.Thread(target, args)
```



Step 3: Start a Thread

To start a thread, we use the **start()** method of the Thread class.

```
t1.start()
t2.start()
```

Step 4: End the thread Execution

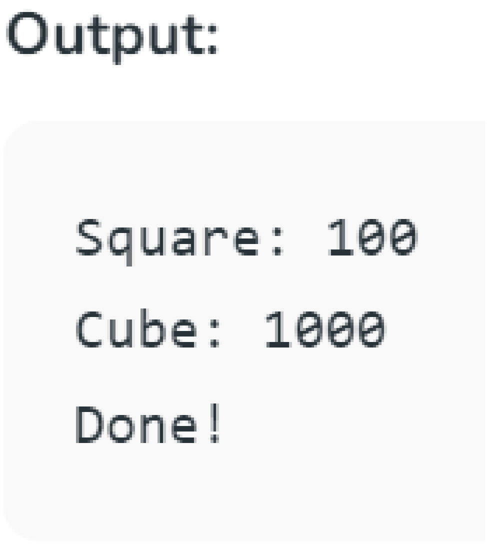
Once the threads start, the current program (you can think of it like a main thread) also keeps on executing. In order to stop the execution of the current program until a thread is complete, we use the **join()** method.

```
t1.join()
t2.join()
```

As a result, the current program will first wait for the completion of **t1** and then **t2**. Once, they are finished, the remaining statements of the current program are executed.

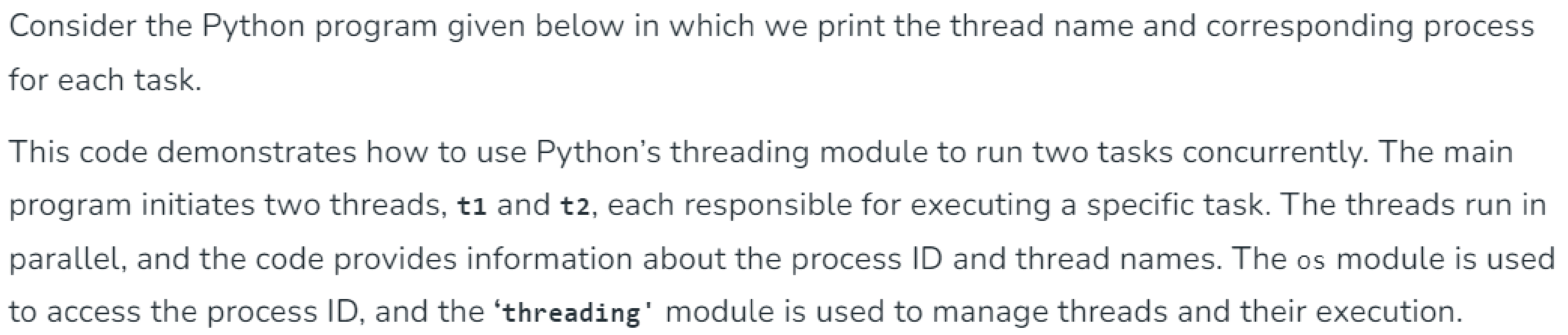
Example

```
import threading


def print_cube(num):
        print("Cube: {}" .format(num * num * num))


def print_square(num):
        print("Square: {}" .format(num * num))


if __name__ =="__main__":
        t1 = threading.Thread(target=print_square, args=(10,))
        t2 = threading.Thread(target=print_cube, args=(10,))

        t1.start()
        t2.start()

        t1.join()
        t2.join()

        print("Done!")
```



Output:

```
Square: 100
Cube: 1000
Done!
```



Consider the Python program given below in which we print the thread name and corresponding process for each task.

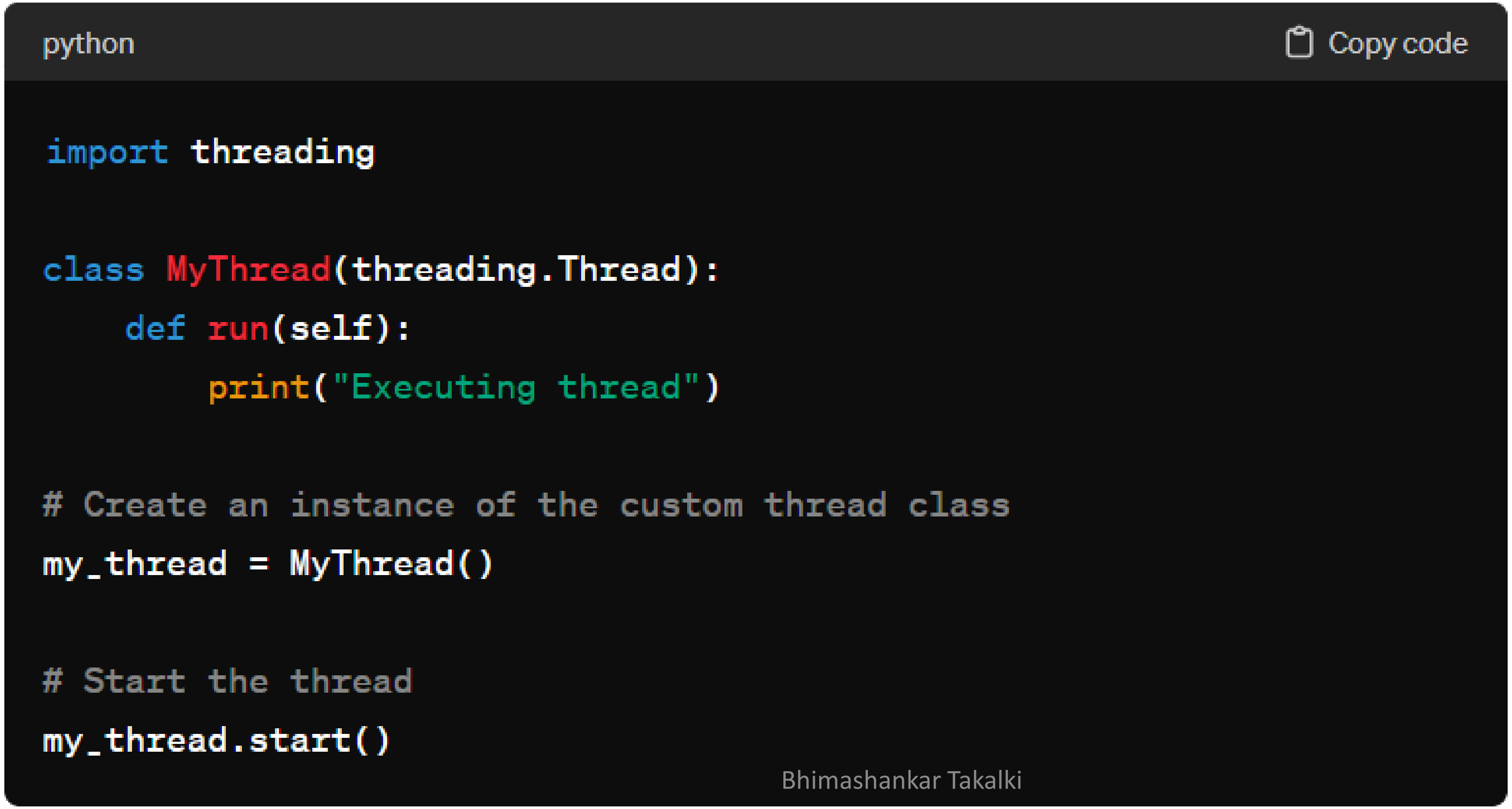
This code demonstrates how to use Python’s threading module to run two tasks concurrently. The main program initiates two threads, `t1` and `t2`, each responsible for executing a specific task. The threads run in parallel, and the code provides information about the process ID and thread names. The os module is used to access the process ID, and the ‘`threading`' module is used to manage threads and their execution.

```
import threading
import os

def task1():
        print("Task 1 assigned to thread:
{}".format(threading.current_thread().name))
        print("ID of process running task 1:
{}".format(os.getpid()))

def task2():
        print("Task 2 assigned to thread:
{}".format(threading.current_thread().name))
        print("ID of process running task 2:
{}".format(os.getpid()))
```

```
if __name__ == "__main__":

        print("ID of process running main program:
{}".format(os.getpid()))

        print("Main thread name:
{}".format(threading.current_thread().name))

        t1 = threading.Thread(target=task1, name='t1')
        t2 = threading.Thread(target=task2, name='t2')

        t1.start()
        t2.start()

        t1.join()
        t2.join()
```

Creating Threads:

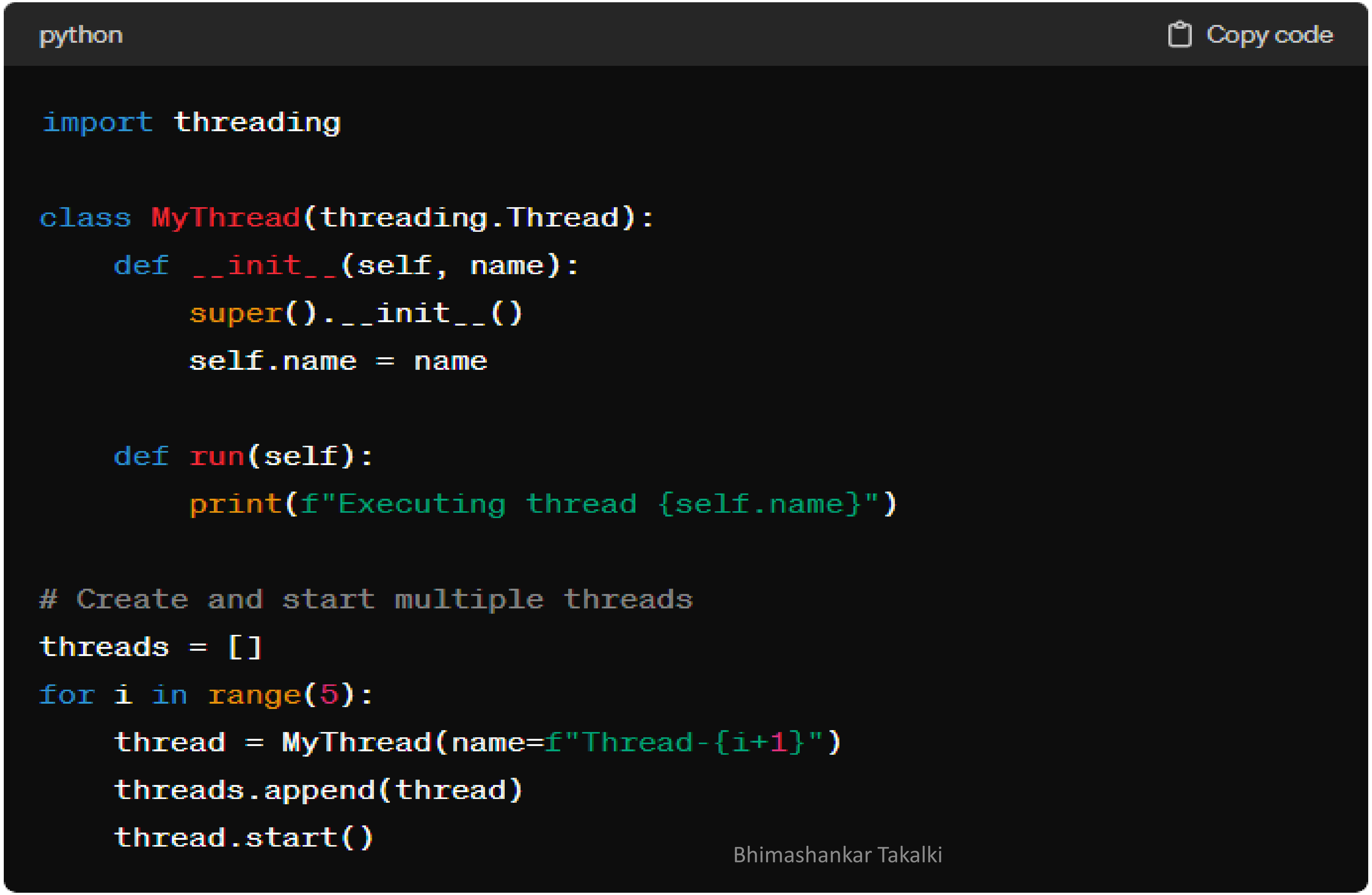
You can create a thread by subclassing the `Thread` class from the `threading` module and overriding the `run()` method with the code you want to execute in the thread.



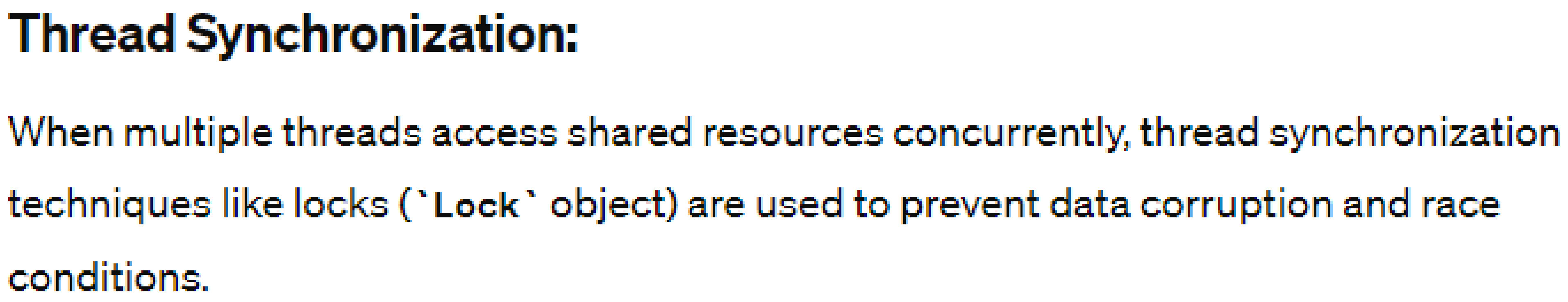
```python
import threading

class MyThread(threading.Thread):
    def run(self):
        print("Executing thread")

# Create an instance of the custom thread class
my_thread = MyThread()

# Start the thread
my_thread.start()
```

Running Multiple Threads:

You can create and start multiple threads to perform tasks concurrently.



```python
import threading

class MyThread(threading.Thread):
    def __init__(self, name):
        super().__init__()
        self.name = name

    def run(self):
        print(f"Executing thread {self.name}")

# Create and start multiple threads
threads = []
for i in range(5):
    thread = MyThread(name=f"Thread-{i+1}")
    threads.append(thread)
    thread.start()
```

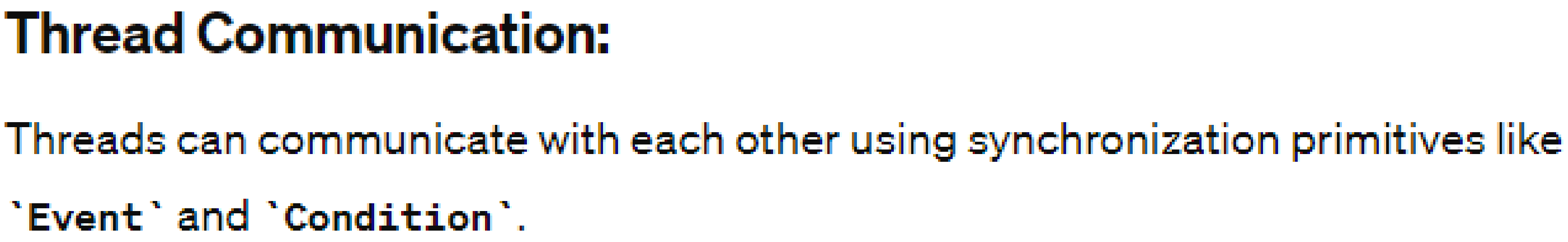


Thread Synchronization:

When multiple threads access shared resources concurrently, thread synchronization techniques like locks (`Lock` object) are used to prevent data corruption and race conditions.

```
import threading

shared_resource = 0
lock = threading.Lock()

def increment():
    global shared_resource
    for _ in range(100000):
        lock.acquire()
        shared_resource += 1
        lock.release()
```

```
# Create and start multiple threads
threads = []
for _ in range(5):
    thread = threading.Thread(target=increment)
    threads.append(thread)
    thread.start()

# Wait for all threads to complete
for thread in threads:
    thread.join()

print("Final value of shared resource:", shared_resource)
```

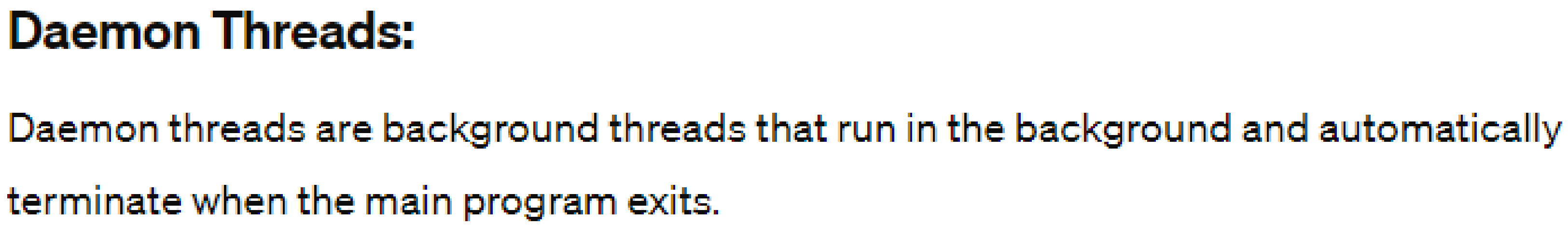


Thread Communication:

Threads can communicate with each other using synchronization primitives like `Event` and `Condition`.

```
import threading

event = threading.Event()

def worker():
    print("Waiting for event to be set")
    event.wait()
    print("Event is set, continuing execution")

# Create and start the worker thread
thread = threading.Thread(target=worker)
thread.start()
```

```
# Main thread waits for some time
threading.Event().wait(2)

# Set the event
print("Setting the event")
event.set()

# Wait for the worker thread to complete
thread.join()
```



Daemon Threads:

Daemon threads are background threads that run in the background and automatically terminate when the main program exits.

```
import threading
import time

def daemon_function():
    while True:
        print("Daemon thread is running")
        time.sleep(1)

# Create and start a daemon thread
daemon_thread = threading.Thread(target=daemon_function)
daemon_thread.daemon = True
daemon_thread.start()

# Main thread waits for some time
time.sleep(3)

print("Main thread is exiting")
```