Multithreading in Python

Multithreading is a technique in programming that allows multiple threads of execution to run concurrently within a single process. In Python, we can use the threading module to implement multithreading. In this tutorial, we will take a closer look at the threading module and its various functions and how they can be used in Python.

Importing Threading

We can use threading by importing the threading module.

```
import threading
```

Creating a thread

To create a thread, we need to create a Thread object and then call its start() method. The start() method runs the thread and then to stop the execution, we use the join() method. Here's how we can create a simple thread.

```
import threading
def my_func():
  print("Hello from thread", threading.current_thread().name)
  thread = threading.Thread(target=my_func)
  thread.start()
  thread.join()
```

Functions

The following are some of the most commonly used functions in the threading module:

- `threading.Thread(target, args)`: This function creates a new thread that runs the target function with the specified arguments.
- `threading.Lock()`: This function creates a lock that can be used to synchronize access to shared resources between threads.

Creating multiple threads

Creating multiple threads is a common approach to using multithreading in Python. The idea is to create a pool of worker threads and then assign tasks to them as needed. This allows you to take advantage of multiple CPU cores and process tasks in parallel.

```
import threading

def thread_task(task):
    # Do some work here
    print("Task processed:", task)
```

```
if __name__ == '__main__':
    tasks = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    threads = []
    for task in tasks:
        thread = threading.Thread(target=thread_task, args=(task,))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()
```

Using a lock to synchronize access to shared resources

When working with multithreading in python, locks can be used to synchronize access to shared resources among multiple threads. A lock is an object that acts as a semaphore, allowing only one thread at a time to execute a critical section of code. The lock is released when the thread finishes executing the critical section.

```
import threading

def increment(counter, lock):
    for i in range(10000):
        lock.acquire()
        counter += 1
        lock.release()

if __name__ == '__main__':
    counter = 0
    lock = threading.Lock()

    threads = []
    for i in range(2):
        thread = threading.Thread(target=increment, args=(counter, lock))
        threads.append(thread)
        thread.start()

    for thread in threads:
        thread.join()

    print("Counter value:", counter)
```

Conclusion

As you can see, the threading module provides a simple and efficient way to implement multithreading in Python. Whether you need to create a new thread, run a function across multiple input values, or synchronize access to shared resources, the threading module has you covered.

In conclusion, the threading module is a powerful tool for parallelizing code in Python. Whether you are a beginner or an experienced Python developer, the threading module is an essential tool to have in your toolbox. With multithreading, you can take advantage of multiple CPU cores and significantly improve the performance of your code.