Introductions to Parallel Processing: Takeaways

Syntax

- Creating a process:

```
import multiprocessing
process = multiprocessing.Process(target=my_function, args=my_arguments)
```

- Starting a process:

```
process.start()
```

- Waiting for a process to finish:

```
process.join()
```

- Using a shared value in memory to get the result:

```
def add(x, y, shared_value):
    shared_value.value = x + y
shared_value = multiprocessing.Value("f")
process = multiprocessing.Process(target=add, args=(3, 5, shared_value))
process.start()
process.join()
result = shared_value.value
```

Concepts

- The central processing unit (CPU) is the hardware component of a computer responsible for executing the instructions of a computer program.
- Multiprocessing provides a way to run a function as a separate Python program. This makes it possible to improve data processing algorithms by splitting the data in chunks and processing on several CPU in parallel.
- Processes run as distinct programs. For this reason, they don't share the memory with the script that started them.
- We can use special objects such as the `Value` object to have shared memory between processes.
- We need to ensure that a single process updates a memory location at a time. This can be achieve using locks. However, we should minimize the number of times each process acquired a lock, or otherwise, the processes won't be able to run in parallel.

Resources

- Parallel processing
- Multiprocessing in Python