Use ***threading*** if your program is **network-bound** (network intensive task)

Use ***multiprocessing*** if it is **CPU-bound** (CPU intensive task).

**Core**: The CPU’s processor. This term refers to the **hardware** component of your CPU. A core can work on a single task; multi-core processors can perform multiple tasks at once.

**Thread**: Refers to the **virtual component** that manages the tasks. Each CPU core can have up to two threads if your CPU has multi/hyper-threading enabled.

**Process**: An instance of a computer program that is being executed by one or many threads. Depending on the operating system, a process may be made up of multiple threads of execution that execute instructions concurrently.

**Multithreading**: The ability of a central processing unit (CPU) (or a single core in a multi-core processor) to provide multiple threads of execution concurrently, supported by the operating system.

**Multiprocessing**: The use of two or more CPUs within a single computer system. The term also refers to the ability of a system to support more than one processor or the ability to allocate tasks between them.

***Python Global Interpreter Lock (GIL) is******a type of process lock which is used by python whenever it deals with processes****. Generally, Python only uses one thread to execute the set of written statements. This means that* ***in******python only one thread will be executed at a time.***

 Using ThreadPoolExecutor with 8 threads, it will NOT speed up the process.

* The python interpreter creates a new process and spawns the threads
* Thread-1 starts running, acquiring the GIL
* Threads-2 to 8 wants to assist thread-1, but have to wait for thread-1 to release the GIL before any other threads can process it
* Since there are no I/O operations, thread-1 will continue processing the entire function operation

**Why is there a need for the GIL?**

In the above example, we are constantly updating the final variable with every iteration. If multiple threads are able to ***simultaneously*** access the execution of this code, the value of i which was retrieved and modified by each thread can change in between when other threads access it. What then, is the true value of i and consequently, final at any point in time? This is an unwanted state of a program known as a **race condition**.

**When then, should multithreading be used?**

A 100% computationally intensive task is unable to take advantage of multiple threads because only a single thread can be used at once (due to the GIL). If we perform a task that is largely I/O bound, we can then take advantage of multithreading.

***Summary***

1. Python is NOT a single-threaded language.
2. Python processes typically use a single thread because of the GIL.
3. Despite the GIL, libraries that perform computationally heavy tasks like numpy, scipy and pytorch utilise C-based implementations under the hood, allowing the use of multiple cores.
4. As a rule of thumb, processes that are largely I/O bound benefit from multithreading while computationally heavy tasks benefit from multiprocessing.