Multiprocessing and Multithreading

- **Python is slow** comparing to other programming languages;
- We can overcome the Python speed problem using multiprocessing or multithreading
- **Python script execution:**

  1. When you type python automation_script.py in your shell you instruct your processor **to create and to schedule a single process** which is the smallest unit of processing

  2. The allocated process will start to execute the script line by line.

  3. Once the script hit the EOF, the process will be terminated and its resources will be returned to the free pool to be used by other processes

- Inside a process there could be more running threads
- The problem with assigning a lot of threads to one process without special handling is what's called Race Condition
- *CPython uses **GIL(Global Interpreter Lock)***

Multithreading

- Threading in python is used to run multiple threads (tasks, function calls) at the same time inside the same process
- **Thereding is not suitable for CPU intensive application**
- Python threads are best used in cases where the execution of a task involves some waiting (ex: I/O operations)
- Python uses the threading module to start multiple threads

Multiprocessing

- **Multiprocessing achieves true parallelism in Python**
- Low risk of data-corruption when using multiprocessing
- Each spawned process will have their own allocated memory
- Each process has it’s owned GIL so there’s no resource conflict or **race condition** here
- Python uses the multiprocessing module to achieve parallel programming

Multiprocessing vs. Multithreading

Multiprocessing

Pros

- Separate memory space
- Code is usually straightforward
- Takes advantage of multiple CPUs & cores
- Avoids GIL limitations of CPython
- Child processes are interruptible/killable
- **A must with CPython for CPU-bound processing**

Cons

- Inter Process Communication (IPC) a little more complicated with more overhead
- Larger memory consumption

Multiprocessing vs. Multithreading

Multithreading

Pros

- Lightweight , low memory consumption
- Shared memory, makes access to state from another context easier
- Allows you to easily make responsive UIs
- **Great option for I/O - bound applications**

Cons

- CPython, subject to the GIL
- Threads are not interruptible/killable
- If not following a command queue/message model (using the Queue module), then manual use of synchronization becomes a necessity
- Code is usually harder to understand and to get right due to the potential of **race conditions** increases dramatically