The following scripts are written to demonstrate the use of the Python Multiprocessing API.

The Python multiprocessing package lets Python programs create new processes that will perform a computation and return a result to the parent. The parent and child processes can communicate using queues and pipes, synchronize their operations using locks and semaphores, and can share simple arrays of data.

The fundamental class is the ‘Process’, which is passed a callable object and a collection of arguments. The start()' method sets the callable running in a subprocess, after which you can call the is\_alive() method to check whether the subprocess is still running and the join() method to wait for the process to exit.

A Queue is used to communicate the result of the factorial. The Queue object is stored in a global variable. The child process will use the value of the variable when the child was created; because it’s a Queue, parent and child can use the object to communicate. (If the parent were to change the value of the global variable, the child’s value would be unaffected, and vice versa.)

Two other classes, ‘Pool’ and ‘Manager’, provide higher-level interfaces. Pool will create a fixed number of worker processes, and requests can then be distributed to the workers by calling apply() or apply\_async() to add a single request, and map() or map\_async() to add a number of requests.

The other high-level interface, the ‘Manager’ class, creates a separate server process that can hold master copies of Python data structures. Other processes can then access and modify these data structures using proxy objects.

Manager’s methods also include Lock(), RLock(), and Semaphore() to create shared locks.)

Compiled by Vakindu Philliam.