**Investigación sobre *Procesos* e *Hilos* en Python**

La concurrencia en Python existen diversos objetos los cuales pueden de manera simultánea, los cuales pueden tener diversos nombres como lo son los hilos, tareas y procesos pero a un alto nivel, todos ellos se refieren a una secuencia de instrucciones que correrán en orden aunque a la vista del humano parezca una ocurrencia simultánea.

Para poder entender lo que sucede a fondo al momento de hablar de procesos e hilos en python, es necesario entender los conceptos fundamentales, los cuales son:

- Procesos:
    - Son creados por los sistemas operativos para la ejecución de programas
    - Los procesos pueden tener múltiples hilos en ejecución
    - Dos procesos pueden ejecutar código simultáneamente en el mismo programa en python.
    - Los procesos tienen mayor gastos o consumo de recursos del sistema que los hilos, debido a que la abertura y cerradura toma tiempo.
    - Al compartir información entre procesos es demasiado lento a comparación del tiempo que toma en compartir información a los hilos, esto debido a que los procesos no comparten espacio en memoria. En Python, comparten información seleccionando estructuras de datos como matrices, las cuales requieren cierto tiempo I/O. El cual es el tiempo requerido para que suceda la operación I/O
- HIlos:
    - Los hilos son como mini-procesos que viven y son creados dentro de los procesos.
    - Comparten espacio de memoria y eficazmente leen y escriben en las mismas variables.
    - Dos hilos no pueden ejecutar código de manera simultánea en el mismo programa en Python.

Otro tema importante es conocer la diferencia entre **CPU** y núcleo o core. El CPU, o el procesador, maneja el trabajo computacional de una computadora. La CPU tiene uno o más cores, permitiendo que la CPU ejecute código de manera simultánea

Existe un problema conocido con la implementación estándar de python, el cual se llama **GIL (**Global Interpreter Lock**)**, los cuales previenen que dos hilos se ejecuten de manera simultánea en el mismo programa. Para muchos programadores esto es considerado algo malo, mientras que otros lo defienden vehementemente.

Una pregunta importante que nos debemos hacer es cuando debemos usar los hilos o los procesos. Los *procesos* aceleran las operaciones en el programa de Python que son intensas para la CPU, gracias a que se beneficia de los múltiples núcleos,

evitando asi el **GIL.** Mientras que los *hilos* son la mejor opción al tratar con tareas tipo I/O o tareas que envuelven a sistemas externos, dado que los hilos pueden combinar mejor el trabajo de manera eficiente. (Nota, los hilos no proporcionan ningún beneficio en el programa en Python para las tareas intensas para la CPU debido a GIL)

**Creación de procesos en Python.**
En python la librería multiprocessing nos permite crear procesos de forma simple.

```
import multiprocessing

def helloWorld(word):
    print(word)

p1 = multiprocessing.Process(target=helloWorld, args=("holaMundo", ))
p1.start()
p1.join()
print("Terminado")
```

```
holaMundo
Terminado
```

La clase *multiprocessing.Process(target, args)* genera un nuevo proceso, el cual ejecutará la función *target*, usando los parámetros *args.*

El método *Process.start()* inicializa el proceso.

De manera similar a C usando el método *Process.join()* nuestro proceso padre esperará a que el proceso hijo termine su ejecución.

**Creación de hilos en Python.**
La librería threading nos permite crear hilos de forma simple.

```
import threading

def helloWorld(word):
    print(word)
    return

t1 = threading.Thread(target=helloWorld, args=("holaMundo",))
t1.start()
t1.join()
print("Terminado")
```

```
holaMundo
Terminado
```

La clase *threading.Thread(target, args)* genera un nuevo hilo, el cual ejecutará la función *target*, usando los parámetros *args.*

El método *Thread.start()* inicializa el hilo.

Usando el método *Thread.join()* nuestro thread principal esperará que se termine la ejecución del hilo.

**Bibliografía**

Multiprocessing Python Documentation.
https://docs.python.org/2/library/multiprocessing.html.
Consultado el 18 de Octubre del 2020

Threading Python Documentation.
https://docs.python.org/3/library/threading.html
Consultado el 18 de Octubre del 2020.

Intro to Threads and Processes in Python.
https://medium.com/@bfortuner/python-multithreading-vs-multiprocessing-73072ce5600b Consultado el 18 de Octubre del 2020.