# Thread

## Iniciarlo

import threading

def metodoALlamar2(parametro1,parametro2):

    time.sleep(5)

    print("se ejecuto 2")

Vt2=threading.Thread(name="nombre\_hilo\_2"

                     ,target=metodoALlamar2

                     ,args=(VvalorParametro1,VvalorParametro2,)

                     )

### Clases personalizadas

Tienen que heredar de threading.Thread, en su constructor debe de llamar al de este

#### Llamar método interno de clase

Si lo que se va a llamar internamente es un método definido en la clase se puede simplemente sobrescribir al método run y luego decirle en la llamada al constructor de super que el método de ejecución es el run

class HiloPersonalizado(threading.Thread):

    def \_\_init\_\_(self, nombre\_hilo, parametro1):

        super().\_\_init\_\_(name=nombre\_hilo,target=HiloPersonalizado.run)

     self.parametro1=parametro1

    def run(self):

        logging.info("se ejecuto 3"+ self.parametro1)

#### llamar método externo

si el método tienen argumentos hay que especificarlos en el constructor del super (tener en cuenta que si es un método interno con parámetros, el primer argumento a pasar es el self)

class HiloPersonalizado2(threading.Thread):

    def \_\_init\_\_(self, nombre\_hilo,metodoQueSeVaAEjecutar, parametro1,parametro2):

        super().\_\_init\_\_(name=nombre\_hilo,target=metodoQueSeVaAEjecutar,args=(parametro1,parametro2))

## Llamarlo

Vt2.start()

## Unirlo al hilo principal

Para que nuestro programa principal espere por la terminación de la ejecución de este hilo antes de continuar, generalmente se pone después de haber llamado a todos los start de todos los hilos, para poder dar un mensaje de resultado general de la ejecución de todos los hilos

Vt2.join()

# logging

una mejor maner de mostrar los mensajes es usando logging, provee configuraciones y variables que se pueden usar directamente en los mensajes, con datos como el tiempo actual o si el mensaje corresponde a un debug, error o info, se le pude pasar un formato que utilice estas variables y configure la forma que se verán todos los mensajes

## configurar

logging.basicConfig(level=logging.INFO,#.DEBUG

                    format='[%(levelname)s] (%(threadName)-s) %(message)s',

                    # datefmt='%a, %d %b %Y %H:%M:%S',

                    # filename='my\_log.txt',

                    # filemode='w'

                    )

La variable %(levelname) corresponde al dato de si es INFO, DEBUG …

## Imprimir mensaje

logging.info("mensaje")

logging.debug("mensaje")

# Pool

1ro se crea el conjunto, especificando la cantidad de hilos totales a manejar

from concurrent.futures import ThreadPoolExecutor

VpoolEjecutora=ThreadPoolExecutor(max\_workers=5)

2do por cada método o ejecución necesaria se manda a llamar, y el automáticamente reutilizar los hilos disponibles

VpoolEjecutora.submit(metodoALlamar1,VvalorParametro1,VvalorParametro2)

VpoolEjecutora.submit(metodoALlamar2,VvalorParametro1,VvalorParametro2)

Esto los llama y ejecuta inmediatamente

# Synchronizer

from threading import Thread,Lock

-1ro se crea el objeto lock que es el que se va a encargar de bloquear el paso de otros hilos

Vlk=Lock()

Esta variable debe de encontrarse fuera de los hilos secundarios, por lo que debería ser declarada como global o algo

## Usando métodos

-2do antes del código protegido (área critica) se bloquea el paso y luego se desbloquea al terminar la ejecución

Vlk.acquire()

// código (área critica)

Vlk.release()

Estos dos métodos deben de asegurarse que siempre que se llamen sea en un subproceso

## Usando with

# Enter critical section

**with** self.lock:

# Do critical work

# Exit critical section