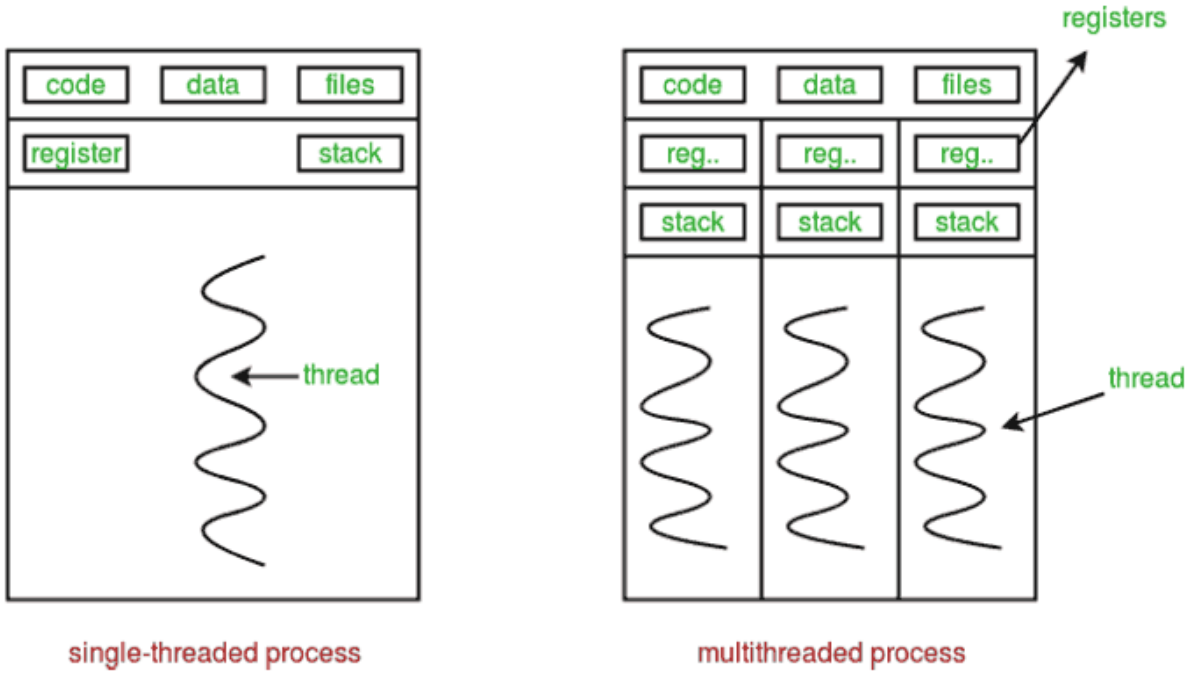
Programación Concurrente

13. Multithreading en Python

Un **proceso** es una instancia de un programa informático que se está ejecutando

Un **hilo** (o hebra; *thread* en Inglés) es una entidad dentro de un proceso que se puede programar para su ejecución. Además, es la unidad de procesamiento más pequeña que se puede realizar en un SO (sistema operativo). En palabras simples, un hilo es una secuencia de instrucciones dentro de un programa que se puede ejecutar independientemente de otro código.



Cada hilo contiene su propio conjunto de registros y variables locales. Todos los hilos de un proceso comparten variables globales y el código del programa.

Un procesador puede ejecutar varios subprocesos simultáneamente. En una CPU simple de un solo núcleo, esto se logra mediante cambios frecuentes entre subprocesos.

En Python, esto se puede realizar mediante la librería *threading*.

1. Importo la Librería

```
import threading
```

1. Crea Hilos

Para crear un nuevo hilo, creamos un objeto de la clase **Thread**. Este toma '*target*' y '*args*' como parámetros. El *target* es la función que ejecutará el hilo, mientras que los *args* son los argumentos que se pasarán a la función *target*.

```
t1 = threading.Thread(target, args)
t2 = threading.Thread(target, args)
```

1. Iniciar un hilo

Para iniciar un hilo, utilizamos el método start() de la clase Thread.

```
t1.start()
t2.start()
```

1. Finalizar la ejecución del hilo

Mientras no finalice un hilo, este continúa ejecutándose. Para detenerlo, utilizamos el método .join()

```
t1.join()
t2.join()
```

Ejemplo:

Vamos a ejecutar un Programa básico: la suma y la resta de dos valores numéricos. Puesto que cada función es independiente de la otra, no debería haber problema entre ellas. Es más, vamos a realizarlo de manera Concurrente, mediante multi-hilos.

```
def add(a, b):
    print("La suma es: ", a + b)


def sub(a, b):
   print("La resta es: ", a - b)

t1 = threading.Thread(target=add, args=(10,4))
t2 = threading.Thread(target=sub, args=(10,4))


t1.start()
t2.start()

t1.join(0.90)
t2.join(0.0000000003)

La suma es: La resta es:  6
 14
```

Ejercicio (se carga en Blackboard):

1. Define dos variables, a y b. Dales el valor numérico que tú quieras.
2. Crea tres funciones en Python.

- La primera va a calcular el logaritmo natural de a+b y va a imprimir el resultado.
- La segunda va a calcular el valor absoluto a-b y va a imprimir el resultado.
- La tercera va a calcular a elevado a la b potencia y va a imprimir el resultado.

1. Corre las tres funciones de manera concurrente, usando multi-hilos.

2. Ejecuta nuevamente las tres funciones de manera concurrente. Esta vez, juega con el parámetro *timeout* del método `.join`. ¿Observas algo raro? Describe qué viste y a qué crees que se deba.

```python
import math

a = 10
b = 4

def lng_task():
    print("El ln de a + b es: ", math.log(a + b))

def valor_task():
    print("El valor absoluto de a - b es: ", abs(a - b))

def potencia_task():
    print("El valor de a elevado a b es: ", math.pow(a, b))

t1 = threading.Thread(target=lng_task)
t2 = threading.Thread(target=valor_task)
t3 = threading.Thread(target=potencia_task)


t1.start()
t2.start()
t3.start()

t1.join(0.90)
t2.join(0.000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000003)
t3.join(0.0000000003)
```

```
El ln de a + b es: El valor absoluto de a - b es:  6
 2.6390573296152584
El valor de a elevado a b es:  10000.0
```

En la librería threading podemos trabajar con concurrencia porque podemos controlar el inicio y el fin de varios hilos. Cuando usamos un timeout, le estamos diciendo a un hilo que espere solo cierto tiempo. Eso implica que puede soltar el paso, o sea, deja que otro hilo se ejecute mientras tanto.

Esto cambia mucho las cosas porque en programas concurrentes el orden importa. Si un hilo tenía que hacer algo primero, pero se quedó esperando, otro hilo puede adelantarse. Eso puede

romper la lógica, hacer que veas datos incompletos, que accedas a una variable antes de que esté lista, o que se activen cosas en momentos en los que no deberían activarse.