Concurrencia

HILOS

Máster Programación Python 20/21

CONCURRENCIA

Hasta ahora hemos estado programando de manera ***secuencial***.
Si la representáramos en esta línea temporal, la llamada de la *función 2* no podría darse hasta que hubiese finalizado el tiempo de ejecución de la *función 1*. La ***concurrencia*** nos permite llamar a un proceso sin que haya terminado otro.

NO ES PARALELISMO
NI MULTIPROCESAMIENTO.

Supongamos que tenemos una función, que se encarga de hacer un request.get a la API dónde podemos encontrar una lista con todos los paises del mundo con sus datos correspondientes. Supongamos también que este proceso tiene una duración de 1 minuto. (voy a exagerar mucho los tiempos para que se entienda mejor, ya que 1 segundo, a nosotros ahora mismo no nos supone nada, pero es importante saber que siempre que podamos ahorrar tiempo debemos hacerlo, para hacelo mas eficiente)

```
def get_data(pais):
    response = request.get(f'https://restcountries.eu/rest/v2/region/{pais}').json()
    return response
```

Tiempo de ejecución: 60 segundos

Supongamos que, también, tenemos una función que hace lo mismo que la anterior pero sin return, es decir, no devuelve nada pero si que guarda en csv la nueva búsqueda e imprime por pantalla el resultado.

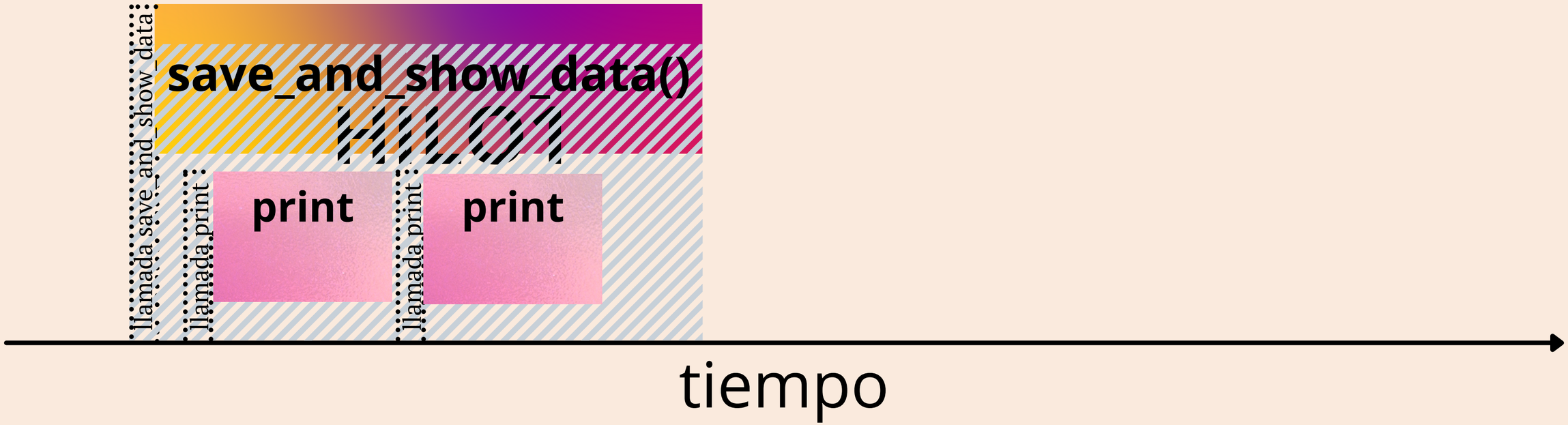
```
def save_and_show_data(pais):
    response = request.get(f'https://restcountries.eu/rest/v2/region/{pais}').json()
    print(f"Name: {response[0]['name']} \nCapital: {response[0]['capital']} \nRegion: {response[0]['region']}
    data_to_save = [response[0]['name'], response[0]['capital'], response[0]['region']]
    add_to_csvFile( data_to_save )  <---Esta función es la encargada de guardar en csv
                                        No nos interesa como está escrita ni nada ahora mismo.
                                        Simplemente veríamos el print, y en el csv resultado, podríamos
                                        ver la escritura.
```

Tiempo de ejecución: 75 segundos

import threading

ROUND 1

Primero, llamamos a la función **save_and_show_data()** para ver la información del país que queremos consultar. Para no tener que esperar 75 segundos sin poder realizar nada mas, lo que hacemos es meter esta función en un hilo para que permita, en cierta manera, que quede ejecutandose en segundo plano, mientras el programa sigue con la siguiente línea de código. De esta manera, podemos, por ejemplo, lanzar un par de mensajes asíncronos, advirtiéndole al usuario que la información que está pidiiento, está en proceso.



```
import threading

hilo1 = threading.Thread( target= save_and_show_data, args= ['Spain'] )
hilo1.start()
print('Los datos del país que desea consultar se están procesando.')
print('Puede tardar unos segundos...')
```

De esta manera, el usuario ha recibido los mensajes mientras la información que estaba solicitando aún estaba siendo procesada.

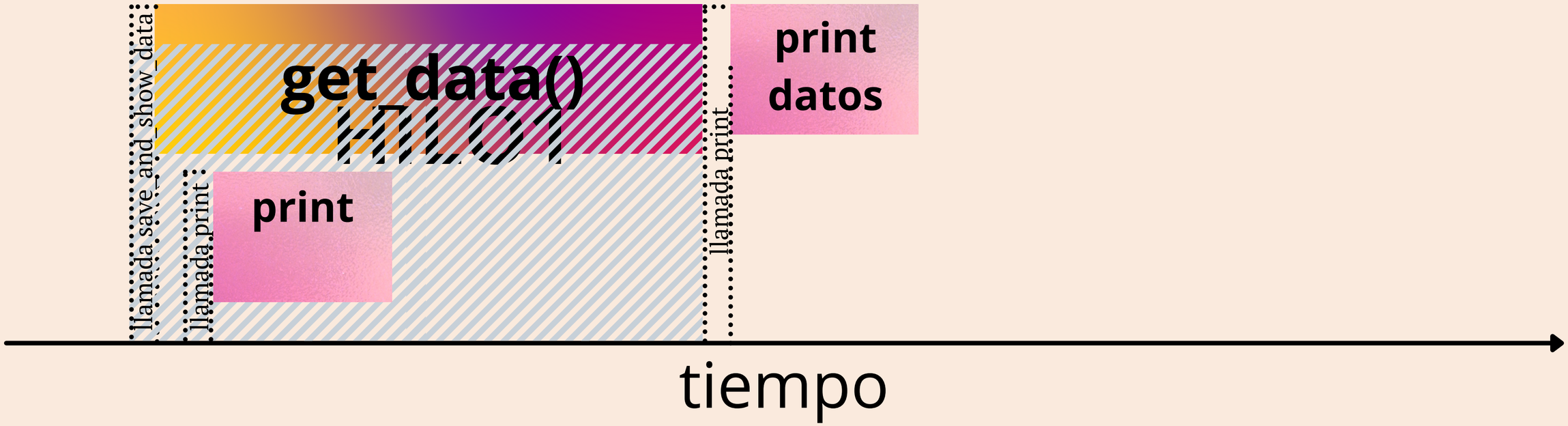
De la misma manera, si tienes funciones independientes (que ninguna depende del resultado de la otra), y las metes en hilos, se van a ir ejecutando, digamos, en cascada. Con lo que seguramente tu script tardará menos en ejecutarse.

from concurrent.futures

import ThreadPoolExecutor

ROUND 2

Ahora, llamamos a la función **get_data()** para obtener los datos del país que queremos consultar y guardarlos en una variable (objeto concurrent.futures._base.Future). De manera asíncrona, vamos a imprimir un mensaje para que el usuario sepa que la información que requiere está siendo procesada. Finalmente, imprimiremos los datos que se han quedado guardados en nuestra variable tras la ejecución y finalización de nuestra función get_data().



```python
from concurrent.futures import ThreadPoolExecutor

with ThreadPoolExecutor() as executor:
    future = executor.submit(get_data, pais='Spain')
    print('\n\nObteniendo datos resultado.....')
print(f"Name: {future.result()[0]['name']}\nCapital: {future.result()[0]['capital']}\nRegion: {future.result()[0]['region']}")
```