# Ejemplos prácticos de técnicas de concurrencia y paralelismo en Python

En este documento, exploraremos varias técnicas modernas de concurrencia y paralelismo en Python, utilizando módulos como concurrent.futures, asyncio, y threading.

## 1. Uso de concurrent.futures

El módulo concurrent.futures proporciona una forma de ejecutar tareas en paralelo usando hilos o procesos. Aquí mostramos un ejemplo usando ThreadPoolExecutor.

### Ejemplo: Uso de ThreadPoolExecutor

```
from concurrent.futures import ThreadPoolExecutor
import time

def tarea(n):
    print(f'Tarea {n} iniciada')
    time.sleep(2)
    print(f'Tarea {n} finalizada')
    return n

def main():
    with ThreadPoolExecutor(max_workers=3) as executor:
        resultados = list(executor.map(tarea, range(5)))
        print('Resultados:', resultados)

if __name__ == '__main__':
    main()
```

# 2. Uso de asyncio

El módulo asyncio permite escribir código concurrente usando la sintaxis async/await. Es ideal para operaciones de entrada/salida no bloqueantes.

## Ejemplo: Uso de asyncio con async/await

```
import asyncio

async def tarea(n):
    print(f'Tarea {n} iniciada')
    await asyncio.sleep(2)
    print(f'Tarea {n} finalizada')
    return n

async def main():
    tareas = [tarea(i) for i in range(5)]
    resultados = await asyncio.gather(*tareas)
    print('Resultados:', resultados)

if __name__ == '__main__':
    asyncio.run(main())
```

# 3. Uso de threading

El módulo threading permite la creación de hilos en Python. A continuación, mostramos un ejemplo básico de su uso.

## Ejemplo: Uso de threading

```
import threading
import time

def tarea(n):
    print(f'Tarea {n} iniciada')
    time.sleep(2)
    print(f'Tarea {n} finalizada')
```

```
def main():
    hilos = []
    for i in range(5):
        hilo = threading.Thread(target=tarea, args=(i,))
        hilos.append(hilo)
        hilo.start()
    for hilo in hilos:
        hilo.join()

if __name__ == '__main__':
    main()
```

# 4. Comparación de técnicas

Cada técnica tiene sus propios casos de uso. Aquí hay una breve comparación:

- **concurrent.futures**: Ideal para tareas de CPU-bound y IO-bound, proporciona una API simple para trabajar con hilos y procesos.
- **asyncio**: Mejor para tareas IO-bound, especialmente cuando se manejan operaciones asíncronas o de red.
- **threading**: Útil para tareas IO-bound pero con limitaciones en el rendimiento debido al Global Interpreter Lock (GIL) en Python.