Conceptos clave

¿Cómo se ejecutan las tareas normalmente?

En un programa secuencial tradicional, las instrucciones se ejecutan una después de otra, en un solo flujo de control (main thread).

Ejemplo:

```
print("Paso 1")
time.sleep(2)  # Espera simulando tarea
print("Paso 2")
time.sleep(2)
print("Paso 3")
```

Resultado:

- El programa espera 2 segundos entre cada paso.
- No hace nada más mientras espera.
- Si tuviera que descargar archivos, mostrar una barra de carga o responder a clics, se congelaría.

Concurrencia

Es la capacidad de un sistema para manejar múltiples tareas *al mismo tiempo* en un sentido lógico, aunque no necesariamente se ejecuten simultáneamente.

En Python, esto se puede lograr con:

- threading
- asyncio (para I/O)

Ejemplo:

- Mientras esperas una respuesta del servidor (tarea 1), puedes seguir mostrando una animación de carga (tarea 2).
- Aunque las tareas no se ejecutan al mismo tiempo, **se alternan** tan rápido que parece que sí.

Paralelismo

Es la capacidad de ejecutar múltiples tareas **literalmente al mismo tiempo** usando múltiples núcleos de CPU.

Ejemplo real:

- CPU con 4 núcleos:
    - Núcleo 1 calcula primos del 1 al 100000
    - Núcleo 2 procesa una imagen
    - Núcleo 3 responde a clics del usuario
    - Núcleo 4 descarga archivos

Clave:

- Requiere hardware de **múltiples núcleos**.
- En Python, solo se logra con multiprocessing, ya que el GIL impide paralelismo real en threading.
- **Concurrencia ≠ paralelismo**, aunque pueden parecer iguales externamente.

Diferencia entre tareas I/O-bound y CPU-bound

Tipo de tarea	Características	Ejemplos típicos
I/O-bound	Pasa mucho tiempo esperando recursos externos	Leer archivos, peticiones web, sleep
CPU-bound	Usa el procesador intensamente para cálculos	Algoritmos matemáticos, compresión

Comparación

Concepto	¿Simultáneo?	¿Usa varios núcleos?	¿Bloquea otras tareas?
Secuencial	No	No	Sí
Concurrencia	Sí (lógica)	No	No (puede alternar)
Paralelismo	Sí (real)	Sí	No

Analogía rápida:

- **Secuencial**: una sola persona cocinando un platillo a la vez.
- **Concurrencia**: una sola persona que va alternando entre cortar verduras, hervir agua y servir, sin hacer todo al mismo tiempo, pero de forma eficiente.
- **Paralelismo**: varias personas en la cocina, cada una con su propia tarea, trabajando al mismo tiempo

Librerias para conseguir concurrencia y paralelismo en Python

Threading en Python

Threading es una librería de Python que permite ejecutar varias tareas (funciones) de forma concurrente usando hilos (threads). Su uso ideal es para tareas que sean I/O-bound.

¿Cómo funciona?

- Todos los hilos comparten el mismo espacio de memoria.
- A nivel del sistema operativo, Python intercambia entre hilos, pero…
- **Limitación:** en Python existe el Global Interpreter Lock (GIL):
    - Solo un hilo puede ejecutar bytecode Python a la vez.
    - Impide el paralelismo real en tareas CPU-bound.

¿Cuándo usar threading?

- Operaciones de red (peticiones HTTP).
- Lectura/escritura de archivos.
- Programas que dependen de esperas (sleep, input, etc.).
- Interfaces gráficas (mantener responsividad).

Ejemplo:

```
import threading


def tarea():
    print("Tarea en hilo")


hilo = threading.Thread(target=tarea)
hilo.start()
hilo.join()
```

Multiprocessing en Python

Multiprocessing es una librería de Python que ejecuta varias tareas en procesos separados, cada uno con su propio intérprete y memoria, es ideal para tareas CPU-bound.

¿Cómo funciona?

- No hay GIL en este caso: cada proceso se ejecuta en su propio núcleo (si está disponible).
- Aporta paralelismo real.
- Requiere más memoria y tiempo de creación, pero se justifica para tareas pesadas.

¿Cuándo usar multiprocessing?

- Cálculos matemáticos intensivos.
- Análisis de datos en grandes volúmenes.
- Procesamiento de imágenes, video, etc.

Ejemplo:

```
from multiprocessing import Process


def tarea():
    print("Tarea en proceso")


proceso = Process(target=tarea)
proceso.start()
proceso.join()
```

Comparación: threading vs multiprocessing

Característica	**threading**	**multiprocessing**
Paralelismo real	❌ (por el GIL)	✅ (usa múltiples núcleos)
Tipo de tarea ideal	I/O-bound	CPU-bound
Consumo de memoria	Bajo	Alto
Creación	Rápida	Lenta
Comunicación	Fácil (comparten memoria)	Compleja (usa colas/pipes)
Seguridad de datos	Riesgo de race conditions	Aislados (más seguros)

Recomendaciones prácticas

Se debe utilizar threading cuando:

- Hay muchas tareas que esperan (red, disco, input).
- Quieres mejorar la responsividad sin gastar CPU.

Se debe utilizar multiprocessing cuando:

- Tu aplicación realiza muchos cálculos pesados.
- Quieres distribuir carga entre varios núcleos.

Actividad práctica

Cálculo de números primos
Comparar el tiempo de ejecución:

- De forma secuencial.
- Usando threading.
- Usando multiprocessing.