Concurrencia y paralelismo en lenguajes interpretados

En lenguajes interpretados como **Python** y **JavaScript**, la concurrencia y el paralelismo se manejan de manera diferente debido a sus arquitecturas internas. La **concurrencia** permite que varias tareas progresen "al mismo tiempo" intercalando su ejecución, mientras que el **paralelismo** implica que estas tareas se ejecutan simultáneamente en múltiples núcleos de CPU. Sin embargo, estos lenguajes presentan algunas limitaciones intrínsecas: Python está afectado por el **Global Interpreter Lock (GIL)**, mientras que JavaScript se ejecuta de manera **monohilo** con un bucle de eventos (*event loop*).

Python proporciona varias herramientas para implementar concurrencia y paralelismo, aunque su rendimiento puede estar limitado por el **GIL**, un mecanismo que permite que solo un hilo de Python ejecute bytecode a la vez, incluso en sistemas multinúcleo. Sin embargo, es posible lograr paralelismo real utilizando procesos en lugar de hilos.

El módulo `threading` permite ejecutar múltiples tareas concurrentemente dentro del mismo proceso. Sin embargo, debido al GIL, no se obtiene paralelismo efectivo para operaciones pesadas en CPU.

```
import threading
def tarea():
	print("Ejecutando una tarea en un hilo")
hilo = threading.Thread(target=tarea)
hilo.start()
hilo.join()
```

El módulo `multiprocessing` permite crear procesos independientes, cada uno con su propio intérprete de Python, lo que evita la limitación del GIL y permite paralelismo real.

```
from multiprocessing import Pool
def cuadrado(n):
	return n * n
with Pool(processes=4) as pool:
	resultados = pool.map(cuadrado, [1, 2, 3, 4])
print(resultados)  # [1, 4, 9, 16]
```

El módulo `asyncio` facilita la ejecución concurrente de tareas I/O-bound mediante la programación asíncrona basada en corutinas. Es ideal para operaciones como peticiones web y lectura/escritura de archivos.

JavaScript tiene un enfoque diferente basado en su modelo **monohilo** y el uso de un **bucle de eventos** (*event loop*). A pesar de que solo un hilo de JavaScript puede ejecutarse a la vez, este modelo permite gestionar múltiples tareas concurrentes de forma eficiente a través de **callbacks**, **promesas**, y **async/await**.

El **bucle de eventos** maneja operaciones asíncronas (como solicitudes HTTP o temporizadores) en segundo plano y ejecuta su callback cuando están listas, sin bloquear el hilo principal.

`async` y `await` facilitan la escritura de código asíncrono más legible, permitiendo manejar operaciones que devuelven promesas sin usar callbacks anidados.

En JavaScript, el paralelismo real se logra mediante **Web Workers** en entornos del navegador. Los workers permiten ejecutar scripts en hilos separados para tareas intensivas en CPU sin bloquear el hilo principal.

Tanto Python como JavaScript ofrecen mecanismos efectivos para manejar concurrencia, pero con diferencias en su enfoque. Python permite concurrencia mediante hilos y paralelismo mediante procesos, aunque su GIL impone ciertas limitaciones. JavaScript, en cambio, se enfoca en la programación asíncrona y eventos, siendo ideal para aplicaciones web y en tiempo real. Cada lenguaje aprovecha su modelo de ejecución para maximizar el rendimiento, y la elección entre uno u otro dependerá del tipo de aplicación y del entorno en el que se ejecute.