async / await Fundamentals

Why async / await exists

Thread-based concurrency becomes expensive at scale. async / await enables cooperative multitasking, allowing thousands of I/O-bound tasks to be handled efficiently within a single thread.

Target Usage

```
async def fetch(i):
    await asyncio.sleep(1)
    return i
```

Coding Problem

Rewrite blocking I/O code so that tasks pause cooperatively instead of blocking the thread, allowing other tasks to run.

Baseline Solution

```
import asyncio

async def fetch(i):
    await asyncio.sleep(1)
    return i

async def main():
    results = await asyncio.gather(*(fetch(i) for i in range(5)))
    print(results)

asyncio.run(main())
```

Observed Effect

All tasks start together and yield control explicitly. Total runtime approaches the longest individual wait instead of the sum of waits.

Key Insight

async / await enables concurrency by cooperative waiting, not parallel execution. CPU-bound work must still be offloaded.