Async I/O and Event Loops

Why event loops exist

Once code can pause with await, a coordinator is needed to decide which task runs next. The event loop schedules, suspends, and resumes coroutines based on I/O readiness.

Target Usage

```
async def task(name):
    await asyncio.sleep(1)
    print(name)
```

Coding Problem

Run many asynchronous tasks concurrently without threads and ensure that blocking operations do not freeze the entire program.

Baseline Solution

```
import asyncio

async def task(name):
    await asyncio.sleep(1)
    print(name)

async def main():
    await asyncio.gather(task("A"), task("B"), task("C"))

asyncio.run(main())
```

Observed Effect

Tasks are interleaved cooperatively by the event loop. Blocking the loop prevents all progress and must be avoided.

Key Insight

The event loop is the heart of async Python. Blocking it blocks everything.