High-Level Concurrency APIs in Python

Why high-level concurrency APIs exist

Low-level concurrency primitives are powerful but error-prone. High-level APIs encode common orchestration patterns such as fan-out/fan-in, structured task lifetimes, and predictable error propagation.

Target Usage

```
import asyncio

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

async def main():
    results = await asyncio.gather(fetch(1), fetch(2), fetch(3))
    print(results)

asyncio.run(main())
```

Coding Problem

Launch multiple concurrent tasks, wait for all of them to complete, collect results, and propagate errors cleanly without manual bookkeeping.

Baseline Solution

```
await asyncio.gather(task1(), task2(), task3())
```

Integration with Blocking Code

```
loop = asyncio.get_running_loop()
result = await loop.run_in_executor(None, blocking_func, arg)
```

Key Insight

High-level concurrency APIs encourage structured concurrency, reduce boilerplate, and prevent common orchestration bugs.