Futures and Executors in Python

Why Futures exist

Concurrent programs need a clean way to represent work that finishes later. Futures act as placeholders for results, separating task execution from result handling and error propagation.

Target Usage

```
from concurrent.futures import ThreadPoolExecutor

def work(x):
    return x * x

with ThreadPoolExecutor() as executor:
    future = executor.submit(work, 10)
    result = future.result()
```

Coding Problem

Run tasks in the background, retrieve results later, and handle exceptions without callbacks or shared state.

Baseline Solution

```
from concurrent.futures import ThreadPoolExecutor

def work(x):
    return x * x

with ThreadPoolExecutor() as executor:
    future = executor.submit(work, 10)
    print(future.result())
```

Observed Effect

Futures capture return values and exceptions, making concurrent code easier to compose and debug.

Key Insight

Futures decouple how work runs from how results are consumed. Executors manage resources while Futures manage outcomes.