Object Memory Layout in Python — __slots__

Why this concept exists

Python objects are flexible but heavy. Each instance normally carries a per-object __dict__, which becomes costly when millions of objects are created. __slots__ allows trading flexibility for memory efficiency.

Target Usage

```
class Point:
    __slots__ = ("x", "y")

    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Coding Problem

You need to represent millions of small, fixed-shape objects efficiently. Dynamic attribute addition is not required, but memory footprint matters.

Baseline Solution

```
class Point:
    __slots__ = ("x", "y")

    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Observed Effect

Using __slots__ removes the per-instance __dict__, reduces memory usage, improves cache locality, and often improves attribute access speed.

Key Insight

__slots__ is an explicit opt-in to efficiency. It fixes attribute layout at class creation time and sacrifices dynamic flexibility for predictable memory usage.