When Object Overhead Dominates Computation

Why this concept exists

In Python, performance bottlenecks often arise not from complex algorithms but from object creation, memory access, and garbage collection overhead. Understanding this helps prevent misdirected optimizations.

Target Usage

```
items = [{"x": i, "y": i * i} for i in range(1_000_000)]
total = sum(item["y"] for item in items)
```

Coding Problem

Explain why a program performing simple arithmetic becomes slow when object count and memory pressure increase, even though algorithmic complexity remains unchanged.

Baseline Observation

```
for n in [10_000, 100_000, 1_000_000]:
    process(n)
```

Observed Effect

As object count increases, cache misses and garbage collection dominate runtime, causing performance to degrade sharply despite simple logic.

Key Insight

In Python, reducing object count and simplifying data representation often yields larger performance gains than improving algorithmic complexity.