Design Trade-offs in Python

Why this topic exists

Python offers multiple mechanisms—duck typing, ABCs, Protocols, and concrete types—because no single approach balances flexibility, safety, and performance for all systems. Design requires conscious trade-offs.

Target Usage

```
# Rapid prototyping
def log(writer):
    writer.write("hello")
```

Coding Problem

Choose the right abstraction for extension points so that systems remain flexible for developers but safe and maintainable in production.

Baseline Comparison

```
# Duck typing
class FileLogger:
    def write(self, msg):
        print(msg)

# ABC
from abc import ABC, abstractmethod
class Writer(ABC):
    @abstractmethod
    def write(self, msg): ...
```

Observed Trade-offs

Duck typing maximizes flexibility but defers errors. ABCs provide runtime safety. Protocols improve developer ergonomics with static guarantees.

Key Insight

Good Python design is about choosing the right mechanism for the right context, not blindly following a single style.