Protocols vs Abstract Base Classes

Why this distinction exists

Modern Python supports both static typing and dynamic execution. Protocols and ABCs exist to solve different problems at different times: developer-time safety versus runtime safety.

Target Usage (Protocols)

```
from typing import Protocol

class PaymentGateway(Protocol):
    def pay(self, amount: int) -> None:
        ...

class UpiGateway:
    def pay(self, amount: int) -> None:
        print(amount)
```

Target Usage (ABCs)

```
from abc import ABC, abstractmethod

class PaymentGateway(ABC):
    @abstractmethod
    def pay(self, amount: int) -> None:
        pass
```

Coding Problem

Design an extension point that is pleasant for developers using IDEs, but also safe when running untyped or user-provided code in production.

Baseline Insight

Protocols provide zero-runtime-cost structural typing for static analysis. ABCs provide runtime validation and defensive guarantees. Real systems often combine both.

Key Insight

Use Protocols for developer ergonomics and static guarantees. Use ABCs for runtime contracts and safety. They are complementary, not competing tools.