Lecture 2.11

Abstraction - abstract class (Continued)

Agenda

- Recap of Abstract Classes
- Abstract Properties in Python
- Abstract Classes vs Interfaces
- Multiple Inheritance with Abstract Classes
- Best Practices and Common Pitfalls

Recap: The abc Module

- Python provides the ‘abc‘ module to implement abstract classes.
- An abstract class inherits from ‘ABC‘ (Abstract Base Class).
- Methods are declared abstract using the ‘@abstractmethod‘ decorator.
- Abstract classes cannot be instantiated directly.
- Subclasses must override all abstract methods to become concrete classes.

Abstract Properties

- Beyond methods, Python allows defining abstract properties.
- An abstract property forces subclasses to define a specific attribute.
- Use ‘@property‘ and ‘@abstractmethod‘ together.
- The order of decorators matters: ‘@property‘ must be the outermost decorator.
- Ensures consistent state representation across all subclasses.

Example: Defining Abstract Properties

- from abc import ABC, abstractmethod
- class Vehicle(ABC):
- @property
- @abstractmethod
- def wheels(self):
- pass
- class Car(Vehicle):
- @property
- def wheels(self):
- return 4

Abstract Classes vs Interfaces

- Python does not have a strict ‘interface‘ keyword like Java.
- An abstract class with ONLY abstract methods acts as an interface.
- Abstract Class: Can have both abstract (unimplemented) and concrete (implemented) methods.
- Interface: Typically has only abstract methods.
- Abstract classes allow code reusability; interfaces define a strict contract.

Structural Diagram: Interfaces vs Abstract Classes

- Abstract Class -¿ Contains ConcreteMethod() and AbstractMethod()
- Interface -¿ Contains only AbstractMethod1(), AbstractMethod2()
- Subclass inherits and implements the required methods.

Multiple Inheritance with ABCs

- Python supports multiple inheritance.
- A subclass can inherit from multiple abstract base classes.
- Useful for composing behaviors (e.g., Mixins).
- Subclass must implement all abstract methods from all parent ABCs.
- Method Resolution Order (MRO) applies to concrete methods inherited.

Example: Multiple Abstract Classes

- class Walkable(ABC):
- @abstractmethod
- def walk(self): pass
- class Swimmable(ABC):
- @abstractmethod
- def swim(self): pass
- class Amphibian(Walkable, Swimmable):
- def walk(self): print('Walking')
- def swim(self): print('Swimming')

Best Practices for Abstraction

- Keep abstract classes focused on a single responsibility.
- Use ABCs to define clear, strict APIs for plugins or large modules.
- Avoid adding too many concrete methods to an ABC, which can lead to tight coupling.
- Document expected behavior of abstract methods.
- Use abstract properties for state that subclasses must maintain.

Summary

- Abstract properties can be defined using ‘@property‘ and ‘@abstractmethod‘.
- Interfaces in Python are just ABCs with exclusively abstract methods.
- Multiple inheritance allows classes to conform to multiple ABCs.
- ABCs provide a strong foundational architecture for complex Python applications.

Next Lecture Preview

- Introduction to Stacks and Queues
- Unit 3: Stack and Queues
- Understanding LIFO and FIFO principles
- Implementing Stacks using Python Lists