# Lecture 2.9

Polymorphism & Abstraction in Python

# Agenda

- What is Polymorphism?
- Polymorphism through Inheritance (Method Overriding)
- Dynamic Binding in Python
- What is Abstraction?
- Abstract Base Classes (ABCs)
- The abc Module and @abstractmethod
- Summary and Q&A

# Understanding Polymorphism

- The term 'Polymorphism' is derived from Greek words meaning 'many forms'.
- In Object-Oriented Programming, it refers to the ability of different objects to respond to the same method call in a way that is specific to their class.
- It allows a single function, method, or operator to behave differently based on the object it is acting upon.
- Python naturally supports polymorphism due to its dynamic typing system.

# Polymorphism through Inheritance

- Inheritance sets the stage for polymorphism.
- A child class inherits methods from its parent class.
- Polymorphism occurs when a child class provides a specific implementation of a method that is already defined in its parent class.
- This mechanism is known as 'Method Overriding'.

# Method Overriding in Action

- class Shape:
- def area(self):
- pass
- class Circle(Shape):
- def area(self):
- return 3.14 * (self.radius ** 2)
- class Square(Shape):
- def area(self):
- return self.side * self.side

# Dynamic Method Resolution

- When an overridden method is called, Python determines at runtime which method to execute.
- If the method is called on a Circle object, Circle.area() is executed.
- If called on a Square object, Square.area() is executed.
- This dynamic resolution allows for highly decoupled and extensible code.

# Using super() with Polymorphism

- Sometimes a subclass wants to extend, rather than completely replace, the parent's behavior.
- class Employee:
- def get_salary(self): return 50000
- class Manager(Employee):
- def get_salary(self):
- base_salary = super().get_salary()
- return base_salary + 20000 # Bonus

# Introduction to Abstraction

- Abstraction is the concept of hiding complex implementation details and showing only the essential features of an object.
- It helps in reducing programming complexity and effort.
- By defining a clear interface, we ensure that specific subclasses implement required methods.
- In Python, pure abstraction is not built-in by default but is enforced using the 'abc' module.

# Abstract Base Classes (ABC)

- An Abstract Base Class is a class that cannot be instantiated.
- It serves solely as a blueprint for other classes.
- An ABC can have both concrete methods (with implementation) and abstract methods (without implementation).
- Subclasses must implement all abstract methods of the ABC to be instantiated.

# The 'abc' Module

- Python provides the 'abc' (Abstract Base Classes) module to create abstract classes.
- To make a class abstract, it must inherit from 'ABC'.
- To define an abstract method, we use the '@abstractmethod' decorator.
- from abc import ABC, abstractmethod

# Implementing an Abstract Class

- from abc import ABC, abstractmethod
- class DatabaseOperation(ABC):
- @abstractmethod
- def connect(self):
- pass
- def log(self, message):
- print(f'Log: {message}')

# Enforcing the Blueprint

- class MySQLDatabase(DatabaseOperation):
- def connect(self):
- print('Connecting to MySQL Database...')
- 
- # This works fine:
- db = MySQLDatabase()
- db.connect()
- db.log('Connection successful')

# What Happens If We Don't Implement?

- class OracleDatabase(DatabaseOperation):
- def query(self):
- print('Running query...')
- 
- # This raises an error at instantiation:
- db2 = OracleDatabase()
- # TypeError: Can't instantiate abstract class
- # OracleDatabase with abstract methods connect

# Lecture Summary

- Polymorphism allows different objects to share the same method names but act differently via Method Overriding.
- It leads to dynamic, flexible, and reusable code.
- Abstraction hides implementation details, focusing on defining a rigorous interface.
- Python uses the 'abc' module to enforce abstract base classes and abstract methods.
- Combined, polymorphism and abstraction enable powerful design patterns.

# Next Lecture Preview

- Next time, we will dive into Unit 3: Stack and Queues.
- We will explore the concepts of LIFO and FIFO.
- We will implement these data structures using Python Lists.
- Prepare by reviewing Python List operations (append, pop).