VaultofCodes

Assignment 2

By
Thatha Keerthi

Table Of Contents

Introduction

OOPs in Python

Object-Oriented Programming (OOP)

- Definition: A programming paradigm based on concept of “objects”, which contains data and methods.
- Key concepts: Encapsulation, Inheritance, Polymorphism, and Abstraction
- Advantages: Code reusability, Scalability, easy to maintenance

Classes and objects

- Classes
  - Blueprint for creating objects
  - Define properties and behaviors
- Objects
  - Instances of classes
  - Contain specific data and can use class methods

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

my_dog = Dog("Buddy", 3)
```

Encapsulation

- Definition: Bundling of data and methods that operate on data within one unit , e.g., a class.
- Private Attributes: use underscores to indicate that a variable is intended to be private.

```
class Car:
    def __init__(self, model, year):
        self.__model = model
        self.__year = year

    def get_model(self):
        return self.__model
```

Inheritance

- Definition: A mechanism where one class (child class) inherits the attributes and methods of another class (parent class)

```
class Animal:
    def __init__(self, species):
        self.species = species

class Dog(Animal):
    def __init__(self, name, age):
        super().__init__('Dog')
        self.name = name
        self.age = age
```

- Benifits: Code reuse and the creation of a hierarchical relationship between classes.

Polymorphism

- Definition: The ability to present the same interface for different underlying data types.
- Example: Methods with the same name in different classes.

```
class Cat:
    def speak(self):
        return "Meow"

class Dog:
    def speak(self):
        return "Bark"

def make_animal_speak(animal):
    return animal.speak()

dog = Dog()
cat = Cat()
print(make_animal_speak(dog))  # Outputs: Bark
print(make_animal_speak(cat))  # Outputs: Meow
```

Abstraction

- Definition: Hiding complex implementation details and showing only the necessary features of an object.
- Abstract classes: Cannot be instantiated and often include one or more abstract methods.

```
from abc import ABC, abstractmethod

class Animal(ABC):
    @abstractmethod
    def sound(self):
        pass

class Dog(Animal):
    def sound(self):
        return "Bark"
```

Conclusion

- Clear Structure: Provides a modular and organized code structure.
- Code Reuse: Inheritance promotes reusability, reducing redundancy.
- Scalability: Easier to add new features with minimal changes.
- Maintainability: Encapsulation and abstraction enhance readability and debugging.
- Real-world Modeling: Intuitive modeling of real-world entities.

Final Thought: OOP enhances Python’s flexibility and efficiency, making it essential for modern software development.

THANK YOU!!!