Subject: OOP Practical
Department: Computer Network System

Object-Oriented Programming

Hamsa M Ahmed

Name: Hamsa M Ahmed
Subject: OOP Practical
Department: Computer Network System

Lecture 1: Introduction to Object-Oriented Programming (OOP)

1. Introduction

Object-Oriented Programming (OOP) is a programming paradigm centered around the concept of "objects," which represent real-world entities. These objects contain data, called attributes, and functions, called methods, that operate on the data. OOP helps in designing software that is modular, reusable, and easier to maintain.

In contrast to procedural programming, which focuses on procedures or functions, OOP organizes code based on the objects and their interactions. This approach mirrors how humans naturally perceive the world, making it more intuitive and effective for complex software systems.

2. Core Principles of OOP

OOP is built on four fundamental pillars:

2.1 Encapsulation

Encapsulation means bundling the data (attributes) and methods (functions) that manipulate the data into a single unit, the class. It restricts direct access to some of an object's components, which helps protect the integrity of the data.

2.2 Inheritance

Inheritance allows a new class (child class) to inherit properties and behavior (methods) from an existing class (parent class). This enables code reuse and the creation of hierarchical relationships among classes.

2.3 Polymorphism

Polymorphism allows methods to have different implementations based on the object calling them. It enables a single interface to represent different underlying forms (data types).

2.4 Abstraction

Abstraction hides complex details and shows only the essential features of the object. This simplifies interaction and protects the internal state of objects from outside interference.

Name: Hamsa M Ahmed
Subject: OOP Practical
Department: Computer Network System

3. Classes and Objects in Python

A **class** is a blueprint for creating objects. It defines a set of attributes and methods that the created objects will have.

An **object** is an instance of a class. When a class is defined, no memory is allocated until an object is instantiated from the class.

Example:

```
class Car:
    def __init__(self, brand, model):
        self.brand = brand
        self.model = model

    def display_info(self):
        print(f"Brand: {self.brand}, Model: {self.model}")

my_car = Car("Toyota", "Corolla")
my_car.display_info()
```

Output:

```
Brand: Toyota, Model: Corolla
```

4. Detailed Explanation of Code

- `class Car:` defines a new class named Car.
- `__init__` is a special method called a constructor; it initializes the object's attributes when an object is created.
- `self` refers to the current object instance.
- `display_info()` is a method that prints the car's details.
- `my_car = Car("Toyota", "Corolla")` creates an instance (object) of Car.
- `my_car.display_info()` calls the method to display details.

---

5. More Examples

5.1 Encapsulation Example

```
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance  # private attribute

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount

    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
        else:
            print("Invalid withdrawal amount")

    def get_balance(self):
        return self.__balance

account = BankAccount(1000)
account.deposit(500)
account.withdraw(200)
print(account.get_balance())  # Output: 1300
```

5.2 Inheritance Example

```
class Vehicle:
    def __init__(self, brand):
        self.brand = brand

    def drive(self):
        print(f"{self.brand} is driving.")

class Car(Vehicle):
    def __init__(self, brand, model):
        super().__init__(brand)
        self.model = model

    def display(self):
        print(f"Car brand: {self.brand}, model: {self.model}")

my_car = Car("Toyota", "Corolla")
my_car.drive()
my_car.display()
```

Output:

```
Toyota is driving.
Car brand: Toyota, model: Corolla
```

Name: Hamsa M Ahmed
Subject: OOP Practical
Department: Computer Network System

6. Exercises

1. Write a `Student` class with attributes `name`, `age`, and `grade`. Add a method to check if the student passed (grade >= 50). Create at least two student objects and display their information and results.
2. Create a `Rectangle` class with width and height attributes and methods to compute area and perimeter. Instantiate rectangles and print their area and perimeter.
3. Design an `Employee` class with name, salary, and department. Include a method to increase salary by a given percentage. Create employees and test the salary raise method.

7. Tips and Best Practices

- Use meaningful class and variable names starting with uppercase letters for classes and lowercase for variables.
- Keep methods focused and small.
- Use encapsulation to protect data.
- Utilize inheritance to reduce code duplication.
- Use comments to explain complex logic.

8. Summary

In this lecture, you learned the basics of Object-Oriented Programming and its core principles: encapsulation, inheritance, polymorphism, and abstraction. You saw how to define classes and objects in Python and practiced with practical code examples.

9. References

1. Python Official Documentation - Classes
2. Real Python - Object-Oriented Programming
3. GeeksforGeeks - Python OOP Concepts