Unit:3; Class:1

Introduction to Object-Oriented Programming (OOP)

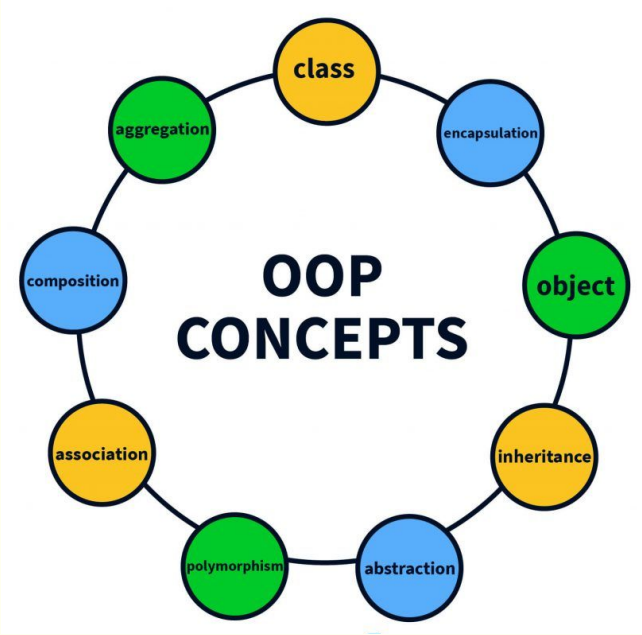
Instructor: Musfique Ahmed

What is OOP?

- **Definition**: Object-Oriented Programming (OOP) is a programming paradigm based on the concept of "objects."

- **Key Idea**: Objects represent real-world entities with attributes (data) and behaviors (methods).

- **Why Learn OOP?**
  - Makes coding easier to understand and reuse.
  - Allows you to build more complex and structured programs.



Procedural vs. OOP

Procedural Programming

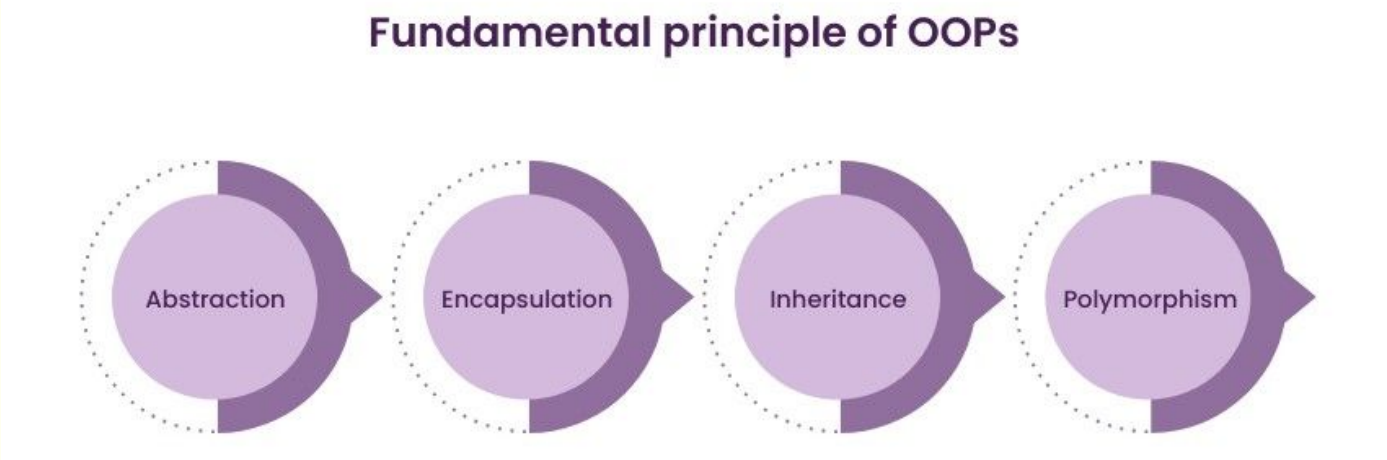
- Focuses on functions and sequences of steps to solve a problem.
- Example: Writing steps to calculate the area of a rectangle.

Object-Oriented Programming

- Focuses on objects that interact with each other.
- Example: Create a "Rectangle" object that knows how to calculate its own area.

Key Principles of OOP

1. **Encapsulation**: Bundling data and methods into a single unit (class).
2. **Inheritance**: Reusing code by creating new classes from existing ones.
3. **Polymorphism**: Using a single interface to represent different types of objects.
4. **Abstraction**: Hiding complex details and showing only the necessary parts.



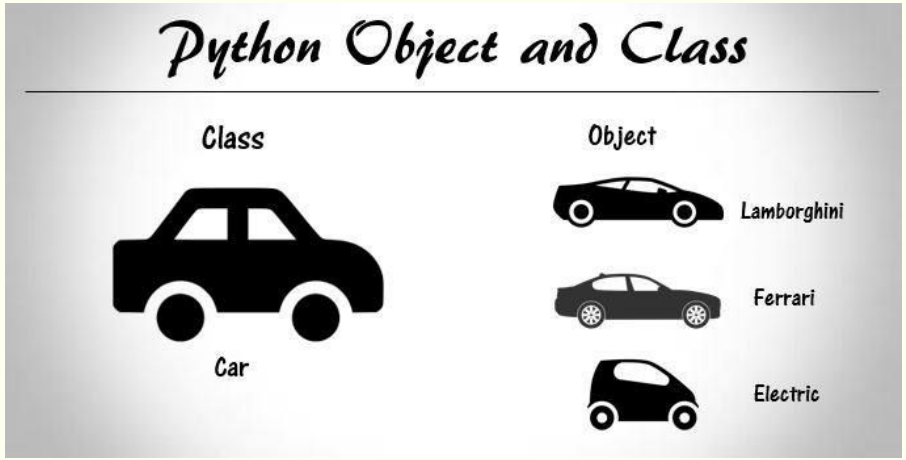
Classes and Objects

- **Class**: A blueprint for creating objects (e.g., a "Car" blueprint).
- **Object**: An instance of a class (e.g., a specific car like a red Tesla).

```python
class Animal:
    def __init__(self, name, sound):
        self.name = name
        self.sound = sound

    def make_sound(self):
        print(f"{self.name} says {self.sound}")

# Creating an object
dog = Animal("Dog", "Woof")
dog.make_sound()  # Output: Dog says Woof
```



The `__init__` Method

- **What is `__init__`?**
  - A special method called a constructor.
  - Automatically executed when an object is created.
- **Purpose**: Initialize the attributes of the object.

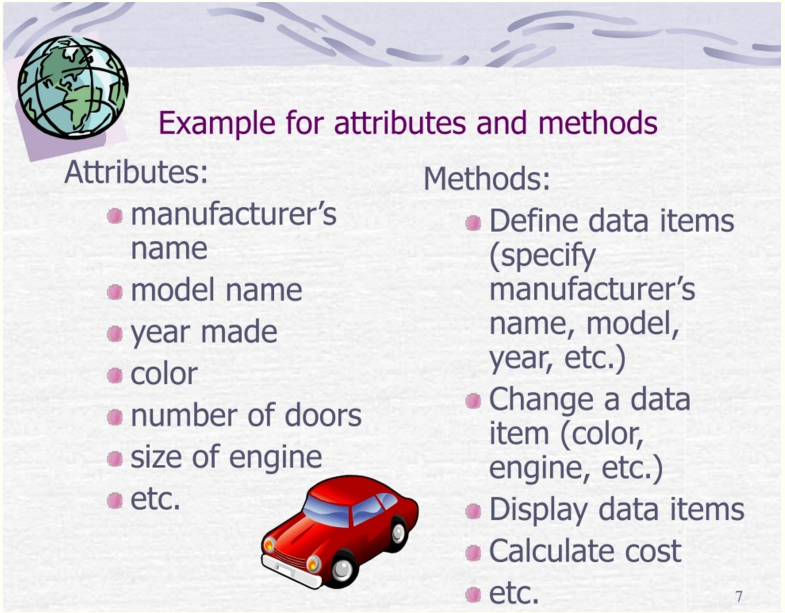
```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def introduce(self):
        print(f"My name is {self.name} and I am {self.age} years old.")

student = Student("Alice", 10)
student.introduce()  # Output: My name is Alice and I am 10 years old.
```

Attributes and Methods

- **Attributes**: Variables that store data for the object.
  1. Example: `name`, `color`.
- **Methods**: Functions that define the behavior of an object.
  1. Example: `make_sound()`.
- **Types of Methods**:
  1. Instance Methods (operate on the object).
  2. Class Methods (operate on the class).
  3. Static Methods (do not depend on object or class).



Activity - Create Your Own Class

- **Task:** Create a class named `Car` with the following:
  - Attributes: `brand`, `model`, `color`.
  - Method: `display_info()` to print details of the car.

- **Example Output:**
  My car is a red Tesla Model S.

- Try it on your own and share your code with the class!

Summary of Today's Class

- **OOP Basics**: Objects and classes.
- **Why OOP?**: Helps organize and simplify code.
- **Key Concepts**: Encapsulation, Inheritance, Polymorphism, Abstraction.
- **Hands-On**: Created a simple class with attributes and methods.

ASSIGNMENT

- Create a class named `Pet`:
  - Attributes: `name`, `type` (e.g., dog, cat).
  - Method: `describe()` to print details about the pet.

- Submit your code to the google classroom for review!

Fun Fact About Python OOP

- Python was designed with simplicity in mind, and its OOP features make it one of the easiest languages to learn and apply!
- Famous OOP-Based Libraries:
  - **pygame** for game development.
  - **turtle** for drawing fun shapes.
  - **Streamlit** For creating data-driven web apps.
  - **Tkinter** For building graphical user interfaces (GUIs).

```python
class Employee:
    def __init__(self,name):
        self.name = name
    def display (self):
        print('The name of employee is:',self.name)

first = Employee('Rushabh')
second = Employee('Dhaval')

second.display()
first.display()
```

Thank You

Do the Quiz Please, you have 10 minutes to do that!