Object-Oriented Programming (OOP) – Python

---

1. Introduction to OOP

Object-Oriented Programming (OOP) is a programming approach where a program is designed using **objects** and **classes**.

Advantages of OOP

- Code reusability
- Easy maintenance
- Better organization
- Real-world modeling

---

2. Class and Object

Class

A **class** is a blueprint or template used to create objects.

Object

An **object** is an instance of a class. It represents real-world entities.

Syntax

```
class ClassName:
    statements
```

Example

```
class Student:
    def display(self):
        print("This is a student")


s1 = Student()   # object creation
s1.display()
```

---

3. Constructor and __init__() Method

Constructor

A **constructor** is a special method used to initialize (assign values to) variables when an object is created.

__init__() Method

- Automatically called when an object is created
- Used to initialize instance variables

Syntax

```
def __init__(self, parameters):
    statements
```

Example

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

    def display(self):
        print("Name:", self.name)

s1 = Student("Anu")
s1.display()
```

---

4. Instance Variables vs Class Variables

Instance Variables

- Belong to an object
- Different for each object

Class Variables

- Shared by all objects
- Defined inside class but outside methods

Example

```
class Student:
    school = "ABC School"    # class variable

    def __init__(self, name):
        self.name = name    # instance variable

s1 = Student("Anu")
s2 = Student("Akhil")

print(s1.name, s1.school)
print(s2.name, s2.school)
```

---

5. Inheritance

Definition

Inheritance allows a child class to use properties and methods of a parent class.

Syntax

```
class ChildClass(ParentClass):
    statements
```

Example

```
class Animal:
    def sound(self):
        print("Animals make sound")
```

```
class Dog(Animal):
    pass

d = Dog()
d.sound()
```

---

6. Method Overriding

Definition

Method overriding occurs when a child class provides its own implementation of a method already defined in the parent class.

Example

```
class Animal:
    def sound(self):
        print("Animal sound")

class Dog(Animal):
    def sound(self):
        print("Dog barks")

d = Dog()
d.sound()
```

---

7. Encapsulation

Definition

Encapsulation means binding data and methods together and hiding data from outside access.

- Achieved using **private variables** (__variable)

Example

```
class Bank:
    def __init__(self):
        self.__balance = 1000

    def show(self):
        print("Balance:", self.__balance)

b = Bank()
b.show()
```

---

8. Polymorphism

Definition

Polymorphism means **one method name, different behavior**.

Example

```
class Cat:
    def sound(self):
        print("Meow")

class Dog:
    def sound(self):
        print("Bark")

for animal in (Cat(), Dog()):
    animal.sound()
```