Core OOP concept

Session Objectives

- Concept
- Core OOP Methodologies in Software Development
- OOP Characteristics
- Class and Object
- Constructors & Destructor
- Inheritance & its Types
- Polymorphism(Function Overloading, Overriding & Operator Overloading)
- Summary

- **Concept**

  - Python is a multi-paradigm programming language. It supports different programming approaches.
  - One of the popular approaches to solve a programming problem is **by creating objects**. This is known as **Object-Oriented Programming (OOP).**
  - An object has two characteristics:
    - attributes
    - behavior(methods)
  - Example:A Boy is an object, as it has the following properties:
  - name, age, color ,hight,weight as attributes ,singing, dancing as behavior
  - The concept of OOP in Python focuses on creating **reusable code**. This concept is also known as DRY (Don't Repeat Yourself).

- # OOP Methedalogy
  - Object Oriented Methodology (OOM) is a system development approach encouraging and facilitating **re-use of software components.**
  - Computer system can be developed on a component basis which enables the effective re-use of existing components and facilitates the sharing of its components by other systems.
  - OOSD is a practical method of **developing a software system** which focuses on the objects of a problem throughout development.
  - OOSD's **focus on objects** early in the development, with attention to generating a useful model, creates a picture of the system that is **modifiable, reusable, reliable, and understandable.**

- OOP characteristics
  - An object-oriented paradigm is to design the program using **classes and objects.**
  - The object is related to real-word entities such as book, house, pencil, etc.
  - The oops concept focuses on writing the **reusable code**. It is a widespread technique to solve the problem by creating objects.Major **principles** of OOPs are
    - Class
    - Object
    - Method
    - Inheritance
    - Polymorphism
    - Data Abstraction
    - Encapsulation

Class and Object

- The class can be defined as a **collection of objects**. It is a logical entity that has some specific attributes and methods.
- A class is a **user-defined blueprint** or prototype from which objects are created.
- Classes provide a means of **bundling data and functionality** together.
- Creating a new class creates a new type of object, allowing new instances of that type to be made.
- Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state

Class Definition Syntax:

```
class ClassName:
    # Statement-1
    .
    .
    .
      # Statement-N
```

- Object
  - An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with *actual values*
  - Example
    - **class** MyClass:

      x = 5

  - **Creating Object**
    - Example

```
p1 = MyClass()          #p1 is object of MyClass
print(p1.x)             #5
```

- __init__() method
  - All classes have a function called __init__(), which is always executed when the **class is being initiated i. e.** class is being used to create a new object.
  - Use the __init__() function to **assign values to object properties**, or other operations that are necessary to do when the object is being created:

```
class myclass:
    def __init__(self, x):
        self.x = x


p1 = myclass( 36)
print(p1.x)          #36
```

- Methods in class

```
class myclass:
        def __init__(self, x):
                self.x = x
        def display(self):
                print("value of x:",self.x)


p1 = myclass( 36)
p1.display()                    #value of x:36
p2 = myclass(22)
p2.display()                    #value of x:22
p2.x=40                         #Modifing object property value
```

- About “self”
  - It is a parameter passed to methods of class.
  - It is used to access variables belongs to the class.
  - It does not have to be named ‘self’ ,you can call it whatever you like.
  - But it has to be a first parameter of any method of class.
  - Example:

```
def __init__(myobj , x):
    x=myobj.x
```

- Constructor and Destructor
  - Constructor:
    - Constructors are generally used for instantiating an object i.e. **initialize(assign values) to the data members** of the class .
    - In Python the **__init__() method** is called the constructor and is always called when an object is created.
    - Types of constructor
      - Default constructor
      - Parameterized constructor

- Destructor
  - Destructors are called when an object gets destroyed.
  - Python has a garbage collector that handles memory management automatically.
  - **Syntax** :

```
def __del__(self):
    # body of destructor
```

```
class Person:
“””
    def __init__(self, ):     #Default Constructor
        self.name = “Virat”
“””
    def __init__(self, name):     #Parametrized Constructor
        self.name = name

    def display(self):
                print("hello my name is:"+self.name)

    def __del__(self):                #Destructor
        print("Bye bye "+self.name)


p1=Person("Raj")
p1.display()
print("Program ends")
```

Output: hello my name is Raj
Program ends
Bye bye Raj

- Inheritance
  - Inheritance allows us to define a class that inherits all the methods and properties from another class.
  - Two basic concept
    - Parent class (base class)
    - Child class (derived class)
  - It provides reusability of a code.
  - It is transitive in nature. (A→ B, B→ C, so that A→ C)

- **Syntax**
  - **subclass_name(Baseclass_name):**
- Example:

```
class X:                          #Base class
    def __init__(self,x):
        self.x=x
class Y(X):                       #Derived class
    def __init__(self,x,y):
        super().__init__(x)       #X.__init__(self,x)
        self.y=y

obj1=Y(10,20)
print(obj1.x,obj1.y)              #Output:10 20
```

- **Types of Inheritance**
  - Single Inheritance
    - (1 Base class and 1 Derived class)

    Syntax:

    class sub_class(base_class):
  - Multiple Inheritance
    - (Multiple Base class and 1 Derived class)

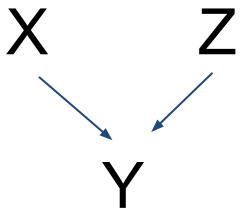
    Syntax:

    class sub_class(base_class1,Base_class2):

```
class X:                        #Base class 1
    def __init__(self,x):
        self.x=x
class Z:                        #Base class 2
    def __init__(self,z):
        self.z=z

class Y(X,Z):                   #Derived class
    def __init__(self,x,z,y):
        X.__init__(self,x)
        Z.__init__(self,z)
        self.y=y


obj1=Y(10,20,30)
print(obj1.x,obj1.z,obj1.y)             #Output:10 20 30
```



- Polymorphism
  - Function Overloading
    - Same name function represents different forms.
      - print(len("Python")) #length of string→ 6
      - print(len([12,23,45])) #how many elements in list→ 3
  - Function Overriding
    - Methods in the child class that have the same name as the methods in the parent class.
    - This process of **re-implementing a method in the child** class is known as Method Overriding.
      - def __init__() present in both base class and child class.

Function Overloading

```
class maths:
  def __init__(self):
      pass
  def cal(self,arg,op):
     if isinstance(arg,int):
        if op==2:
          print("Square:",arg*arg)
        if op==3:
          print("Triple:",arg*arg*arg)
     else:
          print("Length of string: ",len(arg))

m1=maths()
m1.cal(10,2)                #Square:100
m1.cal("Python",0)          #Length of string: 6
```

Function Overriding

```
class colours:
    def __init__(self):
        print("I am in colour class")
    def myfav(self):
        print("all are my favorite colour")

class Black(colours):
    def __init__(self):
        print("I am in Black class")
        super().__init__()              #way to handle overriding
    def myfav(self):
        print("my favorite colour is black")

b1=Black()
b1.myfav()
```

Output:

```
I am in Black class
I am in colour class
my favorite colour is black
```

Summary

- Concept
- Core OOP Methodologies in Software Development
- OOP Characteristics
- Class and Object
- Constructors & Destructor
- Inheritance & its Types
- Polymorphism(Function Overloading, Overriding)

Constructor Demo

Inheritance Demo

Polymorphism Demo1 Demo2