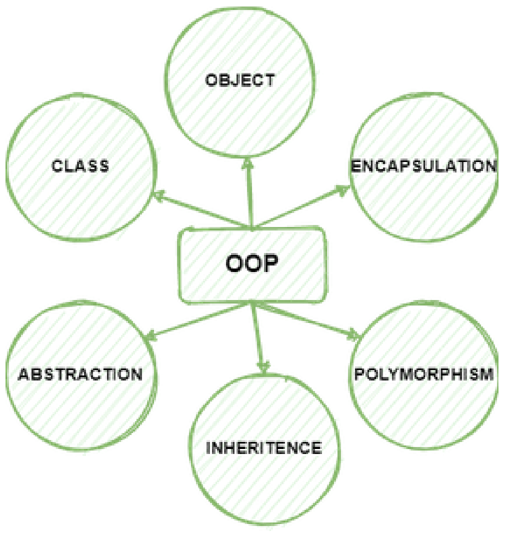
UNIT-V

5.1 PRINCIPLES OF OBJECT-ORIENTED PROGRAMMING (OOPS)

In Python, object-oriented Programming (OOPs) is a programming paradigm that uses objects and classes in programming. It aims to implement real-world entities like inheritance, polymorphisms, encapsulation, etc. in the programming. The main concept of OOPs is to bind the data and the functions that work on that together as a single unit so that no other part of the code can access this data.

- Class
- Objects
- Polymorphism
- Encapsulation
- Inheritance
- Data Abstraction



CLASS

A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.

Some points on Python class:

- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator.

  Eg.: Myclass.Myattribute

Syntax:

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

Example: Creating an empty Class in Python

```
# Python3 program to
# demonstrate defining
# a class

class Dog:
    pass
```

In the above example, we have created a class named dog using the class keyword.

OBJECTS

The object is an entity that has a state and behaviour associated with it. It may be any real-world object like a mouse, keyboard, chair, table, pen, etc. Integers, strings, floating-point numbers, even arrays, and dictionaries, are all objects. More specifically, any single integer or any single string is an object. The number 12 is an object, the string “Hello, world” is an object, a list is an object that can hold other objects, and so on. You’ve been using objects all along and may not even realize it.

An object consists of:

- **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
- **Behaviour:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

To understand the state, behaviour, and identity let us take the example of the class dog (explained above).

- The identity can be considered as the name of the dog.
- State or Attributes can be considered as the breed, age, or color of the dog.
- The behaviour can be considered as to whether the dog is eating or sleeping.

Example: Creating an object

```
obj = Dog()
```

Example 1: Creating a class and object with class and instance attributes

```
class Dog:
     # class attribute
    attr1 = "mammal"
     # Instance attribute
    def __init__(self, name):
        self.name = name
 # Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")
 # Accessing class attributes
print("Rodger is a {}".format(Rodger.__class__.attr1))
print("Tommy is also a {}".format(Tommy.__class__.attr1))
 # Accessing instance attributes
print("My name is {}".format(Rodger.name))
print("My name is {}".format(Tommy.name))
```

Output

```
Rodger is a mammal
Tommy is also a mammal
My name is Rodger
My name is Tommy
```

Example 2: Creating Class and objects with methods

```
class Dog:
     # class attribute
    attr1 = "mammal"
     # Instance attribute
    def __init__(self, name):
        self.name = name
             def speak(self):
        print("My name is {}".format(self.name))
 # Driver code
# Object instantiation
Rodger = Dog("Rodger")
Tommy = Dog("Tommy")
 # Accessing class methods
Rodger.speak()
Tommy.speak()
```

Output

```
My name is Rodger
My name is Tommy
```

INHERITANCE

Inheritance is the capability of one class to derive or inherit the properties from another class. The class that derives properties is called the derived class or child class and the class from which the properties are being derived is called the base class or parent class. The benefits of inheritance are:

- It represents real-world relationships well.
- It provides the reusability of a code. We don't have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Types of Inheritance –

Single Inheritance:

Single-level inheritance enables a derived class to inherit characteristics from a single-parent class.

Multilevel Inheritance:

Multi-level inheritance enables a derived class to inherit properties from an immediate parent class which in turn inherits properties from his parent class.

Hierarchical Inheritance:

Hierarchical level inheritance enables more than one derived class to inherit properties from a parent class.

Multiple Inheritance:

Multiple level inheritance enables one derived class to inherit properties from more than one base class.

Example: Inheritance in Python

```
# Python code to demonstrate how parent constructors
# are called.
 # parent class
class Person(object):
     # __init__ is known as the constructor
    def __init__(self, name, idnumber):
        self.name = name
        self.idnumber = idnumber
       def display(self):
        print(self.name)
        print(self.idnumber)
       def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))
     # child class
class Employee(Person):
    def __init__(self, name, idnumber, salary, post):
        self.salary = salary
        self.post = post
         # invoking the __init__ of the parent class
        Person.__init__(self, name, idnumber)
         def details(self):
        print("My name is {}".format(self.name))
        print("IdNumber: {}".format(self.idnumber))
        print("Post: {}".format(self.post))
# creation of an object variable or an instance
a = Employee('Rahul', 886012, 200000, "Intern")
 # calling a function of the class Person using
# its instance
a.display()
a.details()
```

Output

```
Rahul
```

886012

My name is Rahul

IdNumber: 886012

Post: Intern

POLYMORPHISM

Polymorphism simply means having many forms. For example, we need to determine if the given species of birds fly or not, using polymorphism we can do this using a single function.

Example: Polymorphism in Python

```
class Bird:
       def intro(self):
        print("There are many types of birds.")
     def flight(self):
        print("Most of the birds can fly but some cannot.")
 class sparrow(Bird):
       def flight(self):
        print("Sparrows can fly.")
 class ostrich(Bird):
     def flight(self):
        print("Ostriches cannot fly.")
 obj_bird = Bird()
obj_spr = sparrow()
obj_ost = ostrich()
 obj_bird.intro()
obj_bird.flight()
 obj_spr.intro()
obj_spr.flight()
 obj_ost.intro()
obj_ost.flight()
```

Output

```
There are many types of birds.
Most of the birds can fly but some cannot.
There are many types of birds.
```

Sparrows can fly.

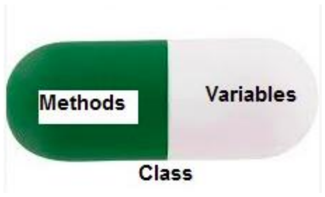
There are many types of birds.

Ostriches cannot fly.

ENCAPSULATION

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as private variables.

A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc.



Example: Encapsulation in Python

```
# Python program to
# demonstrate private members
# Creating a Base class
class Base:
    def __init__(self):
        self.a = "GeeksforGeeks"
        self.__c = "GeeksforGeeks"
 # Creating a derived class
class Derived(Base):
    def __init__(self):
 # Calling constructor of
# Base class
        Base.__init__(self)
        print("Calling private member of base class: ")
        print(self.__c)
 # Driver code
obj1 = Base()
```

```
print(obj1.a)
 # Uncommenting print(obj1.c) will
# raise an AttributeError
 # Uncommenting obj2 = Derived() will
# also raise an AtrributeError as
# private member of base class
# is called inside derived class
```

Output

```
GeeksforGeeks
```

DATA ABSTRACTION

It hides the unnecessary code details from the user. Also, when we do not want to give out sensitive parts of our code implementation and this is where data abstraction came.

Example:

```
# A Python program to demonstrate that hidden members can be accessed outside a class
class MyClass:
# Hidden member of MyClass
    hiddenVariable = 10

# Driver code
myObject = MyClass()
print(myObject._MyClass__hiddenVariable)
```

Output:

```
10
```

5.2 PYTHON CLASSES/OBJECTS

Python is an object oriented programming language. Almost everything in Python is an object, with its properties and methods. A Class is like an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the keyword class:

Example

Create a class named MyClass, with a property named x:

```
class MyClass:
  x = 5
<class '__main__.MyClass'>
```

Create Object

Now we can use the class named MyClass to create objects:

Example

Create an object named p1, and print the value of x:

```
p1 = MyClass()
print(p1.x)
```

Output

```
5
```

5.3 INSTANCE METHOD IN PYTHON

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 its class) for modifying its state.

Example:

```
# Python program to demonstrate
# classes
   class Person:
        # init method or constructor
   def __init__(self, name):
       self.name = name

   # Sample Method
   def say_hi(self):
       print('Hello, my name is', self.name)
```

```
p = Person('Nikhil')
p.say_hi()
```

Output:

```
Hello, my name is Nikhil
```

INSTANCE METHOD

Instance attributes are those attributes that are not shared by objects. Every object has its own copy of the instance attribute.

For example, consider a class shapes that have many objects like circle, square, triangle, etc. having its own attributes and methods. An instance attribute refers to the properties of that particular object like edge of the triangle being 3, while the edge of the square can be 4.

An instance method can access and even modify the value of attributes of an instance. It has one default parameter:-

- self **–** It is a keyword which points to the current passed instance. But it need not be passed every time while calling an instance method.

Example:

```
# Python program to demonstrate
# instance methods
  class shape:


    # Calling Constructor
    def __init__(self, edge, color):
        self.edge = edge
        self.color = color


    # Instance Method
    def finEdges(self):
        return self.edge


    # Instance Method
    def modifyEdges(self, newedge):
        self.edge = newedge

# Driver Code
circle = shape(0, 'red')
square = shape(4, 'blue')

# Calling Instance Method
print("No. of edges for circle: "+ str(circle.finEdges()))

# Calling Instance Method
square.modifyEdges(6)


print("No. of edges for square: "+ str(square.finEdges()))
```

Output

```
No. of edges for circle: 0

No. of edges for square: 6
```

5.4 TYPE IDENTIFICATION:

Python type() is a built-in function that returns the type of the objects/data elements stored in any data type or returns a new type object depending on the arguments passed to the function. The Python type() function prints what type of data structures are used to store the data elements in a program

Syntax For Type () Function

type(object)

type(name, bases, dict)

where

- name = name of the class, which becomes __name__ attribute
- bases = tuple from which current class is derived, becomes   __bases__ attribute
- dict   = a dictionary which specifies the namespaces for the class, later becomes __dict__ attribute

The first method accepts a single parameter in the function. While the second method accepts 3 parameters in it.

If you pass a single argument to the type(object) function, it will return the type of the object. When you pass 3 arguments to the type(name, bases, dict) function, it creates a class dynamically and returns a new type object.

Type () with single parameter:

Python Type() with Single Parameter

We have already seen the syntax for passing a single argument to the type() function. Let us look at a few examples to understand this type() form.

Example 1:

```
a = 10
b = 10.5
c = True
d = 1 + 5j
 print(type(a))
print(type(b))
print(type(c))
print(type(d))
```

Output:

```
<class 'int'>
<class 'float'>
<class 'bool'>
<class 'complex'>
```

5.5 PYTHON SPECIAL CHARACTERS:

In Python strings, the backslash "\" is a special character, also called the "escape" character. To escape a single character or symbol. Only the character immediately following the backslash is escaped It is used in representing certain whitespace characters: "\t" is a tab, "\n" is a newline, and "\r" is a carriage return.

To insert characters that are illegal in a string, use an escape character.

An escape character is a backslash \ followed by the character you

Some Of The Special Characters In Python :

\'	Single Quote
\\	Backslash
\n	New Line
\r	Carriage Return

\t	Tab
\b	Backspace
\f	Form Feed
\ooo	Octal value
\xhh	Hex value

Single Quote Escape Character

single quote in the string.

Python Program

```
 A   = 'hello\'world'
 C  = 'hello\\world'
 D = 'hello\nworld'
 e  = 'hello\rworld'
 F  = 'hello\tworld'
G  = 'hello\bworld'
H  = 'hello\fworld'
I   = '\101\102\103'
J   = '\101\102\103'
Print (A)
Print  (C)
Print  (D)
Print  (e)
Print  (F)
Print  (G)
Print  (H)
Print  (I)
Print (J)
```

Output :

 hello ‘world

- hello \world
- hello

world

- world
- hello world

- hellworld
- hello

world

- ABC
- ABC

```````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````
```````````````````````````````````````````````````````````````````````````````````````````````````````````````````````````