**DATE:31/08/2022**

**ASSIGNMENT NO:06.**

**TOPIC: OOP’S CONCEPTS IN PYTHON.**

Python is also an object-oriented language since its beginning. It allows us to develop applications using an Object-Oriented approach. In [Python](https://www.javatpoint.com/python-tutorial), we can easily create and use classes and objects.

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 object-oriented programming system are given below.

* Class
* Object
* Method
* Inheritance
* Polymorphism
* Data Abstraction
* Encapsulation

**CLASS:**

The class can be defined as a collection of objects. It is a logical entity that has some specific attributes and methods. For example: if you have an employee class, then it should contain an attribute and method, i.e. an email id, name, age, salary, etc

**Syntax:**

**class** ClassName:

        <statement-1>

        .

        .

         <statement-N>

**OBJECT:**

The object is an entity that has state and behavior. It may be any real-world object like the mouse, keyboard, chair, table, pen, etc.

Everything in Python is an object, and almost everything has attributes and methods. All functions have a built-in attribute \_\_doc\_\_, which returns the docstring defined in the function source code.

When we define a class, it needs to create an object to allocate the memory. Consider the following example.

**class** car:

**def** \_\_init\_\_(self,modelname, year):

        self.modelname = modelname

        self.year = year

**def** display(self):

**print**(self.modelname,self.year)

c1 = car("Toyota", 2016)

c1.display()

**INHERITENCE:**

Inheritance is the most important aspect of object-oriented programming, which simulates the real-world concept of inheritance. It specifies that the child object acquires all the properties and behaviors of the parent object.

By using inheritance, we can create a class which uses all the properties and behavior of another class. The new class is known as a derived class or child class, and the one whose properties are acquired is known as a base class or parent class.

It provides the re-usability of the code.

**SYNTAX:**

**class** derive-**class**(<base **class** 1>, <base **class** 2>, ..... <base **class** n>):

    <**class** - suite>

**EXAMPLE:**

class Animal:

def speak(self):

print("Animal Speaking")

class Dog(Animal):

def bark(self):

print("dog barking")

d = Dog()

d.bark()

d.speak()

**OUTPUT:**

dog barking

Animal Speaking

**ENCAPSULATION:**

Encapsulation is one of the most fundamental concepts in object-oriented programming (OOP). This is the concept of wrapping data and methods that work with data in one unit. This prevents data modification accidentally by limiting access to variables and methods. An object's method can change a variable's value to prevent accidental changes. These variables are called private variables.

Encapsulation is demonstrated by a class which encapsulates all data, such as member functions, variables, and so forth.

**PROTECTED MEMBERS:**

The protected variable can be accessed from the class and in the derived classes (it can also be modified in the derived classes), but it is customary to not access it out of the class body.

The **\_\_init\_\_** method, which is a constructor, runs when an object of a type is instantiated.

**Example:**

class Base1:

def \_\_init\_\_(self):

self.\_p = 78

class Derived1(Base):

def \_\_init\_\_(self):

Base1.\_\_init\_\_(self)

print ("We will call the protected member of base class: ",

self.\_p)

self.\_p = 433

print ("we will call the modified protected member outside the class: ",

self.\_p)

obj\_1 = Derived1()

obj\_2 = Base1()

print ("Access the protected member of obj\_1: ", obj\_1.\_p)

print ("Access the protected member of obj\_2: ", obj\_2.\_p)

**OUTPUT:**

We will call the protected member of base class: 78

we will call the modified protected member outside the class: 433

Access the protected member of obj\_1: 433

Access the protected member of obj\_2: 78

**PRIVATE MEMBERS:**

Private members are the same as protected members. The difference is that class members who have been declared private should not be accessed by anyone outside the class or any base classes. Python does not have Private instance variable variables that can be accessed outside of a class.

However, to define a private member, prefix the member's name with a double underscore **"\_\_"**.

Python's private and secured members can be accessed from outside the class using Python name mangling

**Example:**

class Base1:

def \_\_init\_\_(self):

self.p = "Javatpoint"

self.\_\_q = "Javatpoint"

class Derived1(Base1):

def \_\_init\_\_(self):

 Base1.\_\_init\_\_(self)

 print("We will call the private member of base class: ")

 print(self.\_\_q)

obj\_1 = Base1()

print(obj\_1.p)

**OUTPUT:**

Javatpoint

**POLYMORPHISM:**

Polymorphism refers to having multiple forms. Polymorphism is a programming term that refers to the use of the same function name, but with different signatures, for multiple types.

**POLYMORPHISM WITH CLASS AND METHODS:**

**EXAMPLE:**

class xyz():

def websites(self):

print("Javatpoint is a website out of many availabe on net.")

def topic(self):

print("Python is out of many topics about technology on Javatpoint.")

def type(self):

print("Javatpoint is an developed website.")

class PQR():

def websites(self):

print("Pinkvilla is a website out of many availabe on net. .")

def topic(self):

print("Celebrities is out of many topics.")

def type(self):

print("pinkvilla is a developing website.")

obj\_jtp = xyz()

obj\_pvl = PQR()

for domain in (obj\_jtp, obj\_pvl):

domain.websites()

domain.topic()

domain.type()

**OUTPUT:**

Javatpoint is a website out of many availabe on net.

Python is out of many topics about technology on Javatpoint.

Javatpoint is an developed website.

Pinkvilla is a website out of many availabe on net.

Celebrities is out of many topics.

pinkvilla is a developing website

**POLYMORPHISM WITH INHERITENCE:**

**EXAMPLE:**

class Birds:

def intro1(self):

print("There are multiple types of birds in the world.")

def flight1(self):

print("Many of these birds can fly but some cannot.")

class sparrow1(Birds):

def flight1(self):

print("Sparrows are the bird which can fly.")

class ostrich1(Birds):

def flight1(self):

print("Ostriches are the birds which cannot fly.")

obj\_birds = Birds()

obj\_spr1 = sparrow1()

obj\_ost1 = ostrich1()

obj\_birds.intro1()

obj\_birds.flight1()

obj\_spr1.intro1()

obj\_spr1.flight1()

obj\_ost1.intro1()

obj\_ost1.flight1()

**OUTPUT:**

There are multiple types of birds in the world.

Many of these birds can fly but some cannot.

There are multiple types of birds in the world.

Sparrows are the bird which can fly.

There are multiple types of birds in the world.

Ostriches are the birds which cannot fly.