Python OOPS Concepts

OOPs Concepts in Python

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

Python 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.

- 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

Python Objects

The object is an entity that has a **state and behavior 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

obj = Dog()

The Python self

1. Class methods must have an **extra first parameter** in the **method definition. We do not give a value for this parameter** when we call the method, Python provides it.
2. If we have a method that takes no arguments, then we still have to have one argument.
3. This is similar **to this pointer** in C++ and this reference in Java.

The Python __init__ Method

The __init__ method is similar to constructors in C++ and Java. It is run as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

Python Inheritance

Inheritance is the c**apability 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.

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.

Python 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.

Python 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.**

Data Abstraction

It **hides 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.

Data Abstraction in Python can be achieved **by creating abstract classes**

Python Class and Object

```
class Parrot:

    # class attribute
    name = ""
    age = 0

# create parrot1 object
parrot1 = Parrot()
parrot1.name = "Blu"
parrot1.age = 10

# create another object parrot2
parrot2 = Parrot()
parrot2.name = "Woo"
parrot2.age = 15

# access attributes
print(f"{parrot1.name} is {parrot1.age} years old")
print(f"{parrot2.name} is {parrot2.age} years old")
```

Python Inheritance

Example 2: Use of Inheritance in Python

```
# base class
class Animal:

    def eat(self):
        print( "I can eat!")

    def sleep(self):
        print("I can sleep!")

# derived class
class Dog(Animal):

    def bark(self):
        print("I can bark! Woof woof!!")

# Create object of the Dog class
dog1 = Dog()

# Calling members of the base class
dog1.eat()
dog1.sleep()

# Calling member of the derived class
dog1.bark();
```

Python Encapsulation

```
# 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
```

Access Modifiers in Python encapsulation

They are as follows :

1. Public Members
2. Private Members
3. Protected Members

Class member access specifier	Access from own class	Accessible from derived class	Accessible from object
Private member	Yes	No	No
Protected member	Yes	Yes	No
Public member	Yes	Yes	Yes

SCALER Topics

```
# illustrating public members & public access modifier
class pub_mod:
    # constructor
    def __init__(self, name, age):
        self.name = name;
        self.age = age;

    def Age(self):
        # accessing public data member
        print("Age: ", self.age)
# creating object
obj = pub_mod("Jason", 35);
# accessing public data member
print("Name: ", obj.name)
# calling public member function of the class
obj.Age()
```

```
# illustrating private members & private access modifier
class Rectangle:
  __length = 0 #private variable
  __breadth = 0#private variable
  def __init__(self):
    #constructor
    self.__length = 5
    self.__breadth = 3
    #printing values of the private variable within the class
    print(self.__length)
    print(self.__breadth)

rect = Rectangle() #object created
#printing values of the private variable outside the class
print(rect.length)
print(rect.breadth)
```

```python
# illustrating protected members & protected access modifier

class details:
    _name="Jason"
    _age=35
    _job="Developer"

class pro_mod(details):
    def __init__(self):
        print(self._name)
        print(self._age)
        print(self._job)

# creating object of the class
obj = pro_mod()

# direct access of protected member
print("Name:",obj._name)
print("Age:",obj._age)
```

Polymorphism

```
class Polygon:
    # method to render a shape
    def render(self):
        print("Rendering Polygon...")

class Square(Polygon):
    # renders Square
    def render(self):
        print("Rendering Square...")

class Circle(Polygon):
    # renders circle
    def render(self):
        print("Rendering Circle...")

# create an object of Square
s1 = Square()
s1.render()

# create an object of Circle
c1 = Circle()
c1.render()
```

```
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()
```