Python Programming

Practical Questions - Database Management

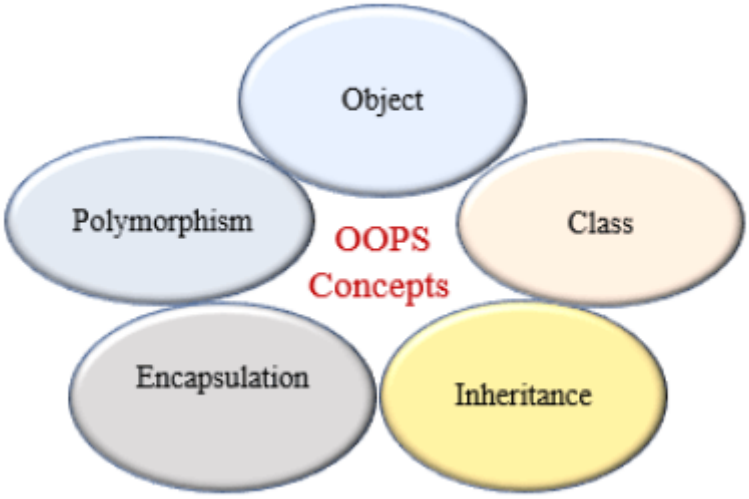
Session 4

OBJECT ORIENTED PROGRAMMING

What is Object Oriented Programming in Python

Object-oriented programming (OOP) is a programming paradigm based on the concept of "**objects**". The object contains both data and code: Data in the form of properties (often known as attributes), and code, in the form of methods (actions object can perform).

An object-oriented paradigm is to design the program using classes and objects. Python programming language supports different programming approaches like functional programming, modular programming. One of the popular approaches is object-oriented programming (OOP) to solve a programming problem is by creating objects



An object has the following two characteristics:

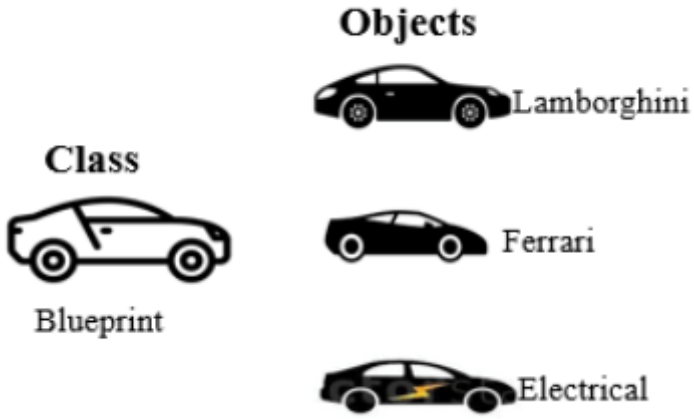
- Attribute
- Behavior

For example, A Car is an object, as it has the following properties:

- name, price, color as attributes
- breaking, acceleration as behavior

One important aspect of OOP in Python is to create **reusable code** using the concept of inheritance. This concept is also known as DRY (Don't Repeat Yourself).

> A class contains the properties (attribute) and action (behavior) of the object. Properties represent variables, and the methods represent actions. Hence class includes both variables and methods.



Python Class and Objects

Object is an instance of a class. The physical existence of a class is nothing but an object. In other words, the object is an entity that has a state and behavior.

Class Attributes and Methods

When we design a class, we use instance variables and class variables.

In Class, attributes can be defined into two parts:

- Instance variables: The instance variables are attributes attached to an instance of a class. We define instance variables in the constructor (the `__init__()` method of a class).
- Class Variables: A class variable is a variable that is declared inside of class, but outside of any instance method or `__init()__` method.

Inside a Class, we can define the following three types of methods.

- **Instance method**: Used to access or modify the object attributes. If we use instance variables inside a method, such methods are called instance methods.
- **Class method**: Used to access or modify the class state. In method implementation, if we use only class variables, then such type of methods we should declare as a class method.
- **Static method**: It is a general utility method that performs a task in isolation. Inside this method, we don't use instance or class variable because this static method doesn't have access to the class attributes.

Constructors in Python

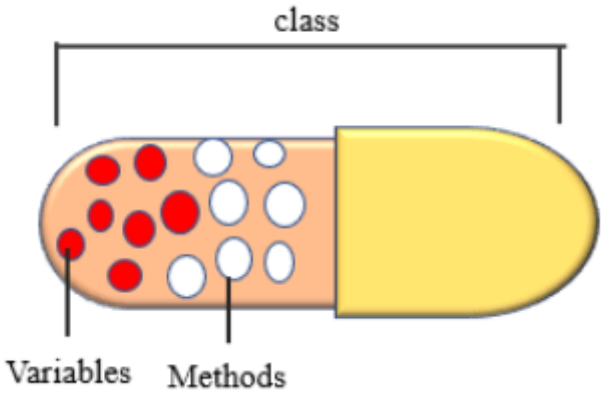
In Python, a **constructor** is a special type of method used to initialize the object of a Class. The constructor will be executed automatically when the object is created. If we create three objects, the constructor is called three times and initialize each object.

The main purpose of the constructor is to declare and initialize instance variables. It can take at least one argument that is `self`. The `__init()__` method is called the constructor in Python. In other words, the name of the constructor should be `**__**init**__**(self)`.

> A constructor is optional, and if we do not provide any constructor, then Python provides the default constructor. Every class in Python has a constructor, but it's not required to define it.

Encapsulation in Python

In Python, encapsulation is a method of wrapping data and functions into a single entity. For example, A class encapsulates all the data (methods and variables). Encapsulation means the internal representation of an object is generally hidden from outside of the object's definition.



Python Encapsulation

Need of Encapsulation

Encapsulation acts as a protective layer. We can restrict access to methods and variables from outside, and It can prevent the data from being modified by accidental or unauthorized modification. Encapsulation provides security by hiding the data from the outside world.

Example: Encapsulation in Python

When you create a class, it means you are implementing encapsulation. A class is an example of encapsulation as it binds all the data members (instance variables) and methods into a single unit.

In Python, we do not have access modifiers, such as public, private, and protected. But we can achieve encapsulation by using prefix **single underscore** and **double underscore** to control access of variable and method within the Python program.

```
class Employee:
    def __init__(self, name, salary):
        # public member
        self.name = name
        # private member
        # not accessible outside of a class
        self.__salary = salary

    def show(self):
        print("Name is ", self.name, "and salary is", self.__salary)

emp = Employee("Jessa", 40000)
emp.show()

# access salary from outside of a class
print(emp.__salary)
```

Output

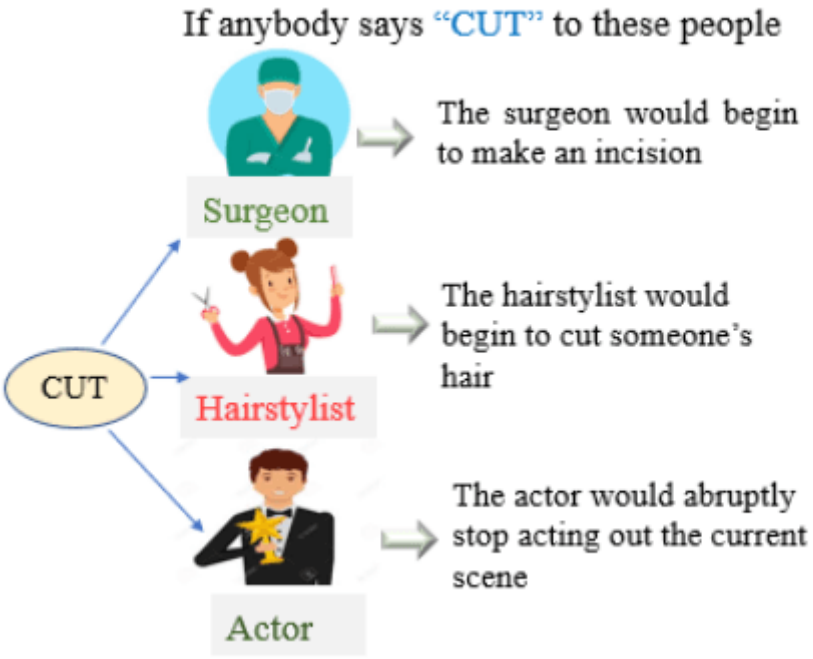
```
Name is  Jessa and salary is 40000
AttributeError: 'Employee' object has no attribute '__salary'
```

Polymorphism in Python

Polymorphism in OOP is the **ability of an object to take many forms**. In simple words, polymorphism allows us to perform the same action in many different ways.

Polymorphism is taken from the Greek words Poly (many) and morphism (forms). Polymorphism defines the ability to take different forms.

For example, The student can act as a student in college, act as a player on the ground, and as a daughter/brother in the home. Another example in the programming language, the + operator, acts as a concatenation and arithmetic addition.



Python

Polymorphism

Example: Using Polymorphism in Python

For example, In the below example, calculate_area() instance method created in both Circle and Rectangle class. Thus, we can create a function that takes any object and calls the object's calculate_area() method to implement polymorphism. Using this object can perform

Polymorphism with class methods is useful when we want objects to perform the same action in different ways. In the below example, both objects calculate the area (same action) but in a different way (different formulas).

```
class Circle:
    pi = 3.14

    def __init__(self, redius):
```

```
        self.radius = redius

    def calculate_area(self):
        print("Area of circle :", self.pi * self.radius * self.radius)

class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def calculate_area(self):
        print("Area of Rectangle :", self.length * self.width)

# function
def area(shape):
    # call action
    shape.calculate_area()

# create object
cir = Circle(5)
rect = Rectangle(10, 5)

# call common function
area(cir)
area(rect)
```

Output

```
Area of circle : 78.5
Area of Rectangle : 50
```

Inheritance In Python

In an Object-oriented programming language, inheritance is an important aspect. In Python, inheritance is the process of inheriting the properties of the parent class into a child class.

The primary purpose of inheritance is the reusability of code. Using inheritance, we can use the existing class to create a new class instead of recreating it from scratch.

Syntax

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

Example: Use of Inheritance in Python

in the below example, From a vehicle class, we are creating a Car class. We don't need to define common attributes and methods again in Car class. We only need to add those attributes and methods which are specific to the Car.

In inheritance, the child class acquires all the data members, properties, and functions of the parent class. Also, a child class can customize any of the parent class methods.

```
# Base class
class Vehicle:

    def __init__(self, name, color, price):
        self.name = name
        self.color = color
        self.price = price

    def info(self):
        print(self.name, self.color, self.price)

# Child class
class Car(Vehicle):

    def change_gear(self, no):
        print(self.name, 'change gear to number', no)

# Create object of Car
car = Car('BMW X1', 'Black', 35000)
car.info()
car.change_gear(5)
```

Output

```
BMW X1 Black 35000
BMW X1 change gear to number 5
```