Gyan4U

__/__/____

10. Object Oriented Programming

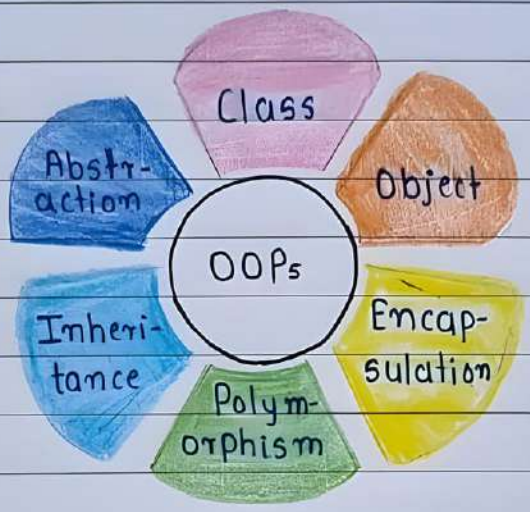
- **OOPs Concepts :**

- Like other general-purpose programming languages, Python is also an object-oriented language. Since its beginning. It allows us to develop applications using an object-Oriented approach. In Python, 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 hous, pencil, book, etc. The OOPs concept focuses on writing the reusable code. It is a widespread technique to solve the problem by creating objects.

➤ OOPs principles are given below.

1) Class
2) Objects
3) Polymorphism
4) Encapsulation
5) Inheritance
6) Data Abstraction



1) 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.

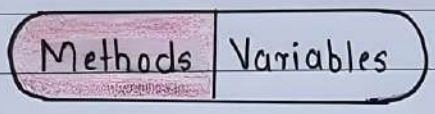
- Syntax -

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

__/__/____

3) Encapsulation :

- Encapsulation is also an essential aspect of object-orient programming. It is used to restrict access to methods and variables. In encapsulation, code and data wrappe together within a single unit from being modified by accident.



Class

- Example :

```
class Computer:

  def __init__(self):
      self.__maxprice = 900

  def sell(self):
      print("Selling Price: {}".format(self.__maxprice))

  def setMaxPrice(self, price):
      self.__maxprice = price

c = Computer()
c.sell()

# Change the price
c.__maxprice = 1000
```

```
# using setter function
c. setMaxPrice(1000)
c. sell()
```

Output :

```
Selling Price : 900
Selling Price : 900
Selling Price : 1000
```

4) Polymorphism :

- Polymorphism contains two words "poly" and "morphs". Poly means many and morph means shape. By polymor-phism, we understand that one task can be performed in different ways.
- Example.

```
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()
```

Output:

```
Rendering Square
Rendering Circle
```

5) Inheritance:

- Inheritance allows us to define a class that inherits all the methods and properties of another class.
- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class.

__/__/____

- Example : Create a Parent Class

```
class Person :
    def __init__(self, fname, lname):
    self.firstname = fname
    self.lname = lname

    def printname(self):
        print(self.firstname, self.lastname)

# Use the Person class to create an object, and
then execute the printname method:

x = Person("John", "Doe")
x.printname()
```

```
John Doe
```

- Create a Child Class -
- To create a class that inherits the functionality from another class, send the parent class as a parameter when creating child class :

```
class Student(Person):
    pass
```

__/__/____

6) 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.

- Data Abstraction in Python can be achieved by creating abstract classes.

• Method Overloading :

- Method overloading refers to defining multiple methods with the same name but with different parameter types or the number of parameters. In Python, method overloading is achieved using default arguments.

- Example -

```
class MyClass
    def my_method (self, arg1, arg2 = None):
        if arg2:
            print (arg1 + arg2)
        else:
            print (arg1)

# Calling the method with one argument
obj = MyClass ()
obj.my_method (10)

# Calling the method with two arguments
obj.my_method (10, 20)
```