# *Python_Notes_Weak_08*

# *Object Oriented Programing(OOP)*

## Procedural Programming

- ❖ Procedural Programming can be defined as a programming model which is derived from structured programming, based upon the concept of calling procedure.
- ❖ Procedures, also known as routines, subroutines or functions, simply consist of a series of computational steps to be carried out.
- ❖ During a program's execution, any given procedure might be called at any point, including by other procedures or itself.

A procedural program is divided into functions each function has a clearly defined purpose & clearly defined interface to the other functions in the program.(Issue: Occupy huge space and time).

## Problems with procedural programming/structural programming

- ❖ Un restricted Access
  - Function have unrestricted access to global data
- ❖ Real world Modeling
  - Un related functions and data, the basics of procedural paradigm, provide a poor model of real world.
- ❖ Difficult of creating new data types
  - Traditional languages are not extensible because they will not let you create new data types.

## What `is Object Oriented Programming?

- ❖ As the name suggests, Object-Oriented Programming or OOPs refers to languages that uses objects in programming.
- ❖ An object is a basic unit of Object Oriented Programming and represents the real life entities.
- ❖ An object represents an entity in real world that can be distinctly identified.
- ❖ For example: student, a desk, mobile, circle, button etc. can be viewed as objects.
- ❖ Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc. in programming.
- ❖ The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.
- ❖ Two key features:
  - Data hiding
  - Encapsulation
- ❖ Data is hidden inside the object, no one can access data, only particular method or member function can access data.

- the data in a class is hidden from other classes, so it is also known as **data-hiding**.
- The process of bundling both data and functions together in a single entity is called **encapsulation** & entity is called object.

**Note:** *Object Oriented approach enable programmer's to **model** real world objects into their software/computer parts.*

## Procedural v/s OOP

| | |
|---|---|
| ➢ The unit in procedural programming is function. | ➢ The unit of object oriented programming is class. |
| ➢ In procedural programming, program is divided into small parts called ***functions***. | ➢ In object oriented programming, program is divided into small parts called ***objects***. |
| ➢ Procedural programming follows ***top down approach***. | ➢ OO programming follows ***bottom up approach***. |
| ➢ Adding new data and function is not easy. | ➢ Adding new data and function is easy. |
| ➢ Procedural programming concentrates on creating functions. | ➢ Object oriented programming starts isolating the classes, and the look for the methods inside the class. |
| ➢ Procedural programming separates the data of the program from the operations that manipulate the data. | ➢ OOP focus both of them. |

## OOP fundamental building blocks.

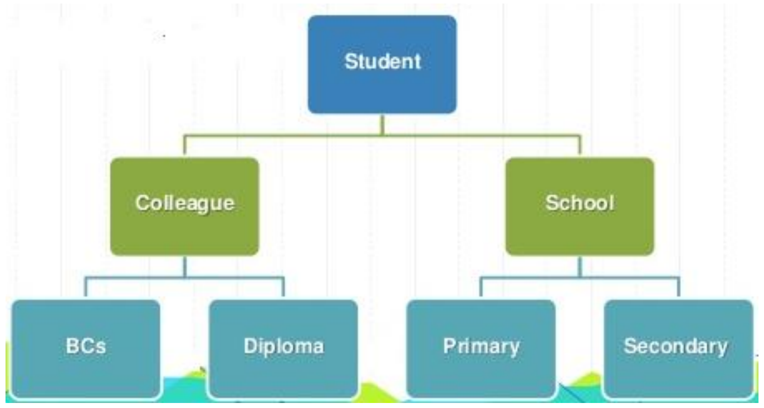
- Four fundamental building blocks/components/concepts of OOP/features
  - Inheritance
  - Polymorphism
  - Encapsulation
  - Abstraction

## Inheritance

Inheritance is the processing of creating new classes from existing class(es).The new created classes are called derived classes or child classes or sub classes and the existing classes are called base classes or super classes
The main advantages:

- Existing classes remain same or un changed but we can add additional features in the derived classes
- The code in base class need not be rewritten in the child class.

- The **variables** and **methods** of the base class can be used in the **child class** as well.
- It provides the idea of reusability.



Three types of Inheritance that python support

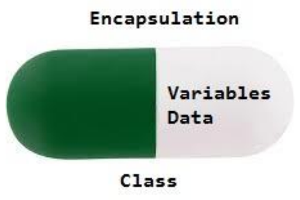
- 1.Single Inheritance: When a child class inherits from single parent class.
- 2.Multi-Inheritance: when a child class inherits from multiple parent classes.
- 3.Multi-Level Inheritance: when a class inherits from a parent class and parent class itself inherits from another parent class.

## Encapsulation

Encapsulation simply means binding object state(fields) and behaviors (methods) together in to single unit (called class). If we are creating class, we are doing encapsulation.

For example:

- capsule, it is wrapped with different medicines
- A python class is example of encapsulation.



## Abstraction

Hiding internal details and showing functionality is known as abstraction. Refers to the set of representing essential features without including the background details or explanations

Example:

- phone call, we don’t know the internal processing.
- A car ‘s gear system abstracts away complexity, allowing drivers to focus on driving.

## Basic Structure of Python class.

```
class Student:
  name = "Ali"
  DOB = "1 jan 2009"

s1 = Student()
print(s1.name)
s2 = Student()
print(s2.name)
```

**Constructor:** special method automatically called/invoke when object of a class is created. The constructor is used to initialize the attributes of a class.

## Constructor Implementation

```
class Student:
  def __init__(self,name,DOB):
    self.name = name
    self.DOB = DOB

s1 = Student("Ali","1 jan 2009")
print(s1.name)
print(s1.DOB)
s2 = Student("Ahmed","1 feb 2009")
print(s2.name)
print(s2.DOB)
```

or!

```
class Student:
  def __init__(self,name,DOB):
    self.name = name
    self.DOB = DOB

s1 = Student("Ali","1 jan 2009")
print(s1.name)
print(s1.DOB)
s2 = Student("Ahmed","1 feb 2009")
print(s2.name)
print(s2.DOB)
```

Example

```
class Teacher:
  def __init__(self,name,salary):
    self.name = name
    self.salary = salary
T1 = Teacher("sara",105000)
print(T1.name,T1.salary)
```

Python program that use the OOP Inheritance concept.

```python
 # Main class
class Person:
    def __init__(self, name, age, gender):
        self.name = name
        self.age = age
        self.gender = gender
# Subclass: Student
class Student(Person):
    def __init__(self, name, age, gender, student_id):
        super().__init__(name, age, gender)
        self.student_id = student_id
# Subclass: Teacher
class Teacher(Person):
    def __init__(self, name, age, gender, subject):
        super().__init__(name, age, gender)
        self.subject = subject
# Subclass: Employee
class Employee(Person):
    def __init__(self, name, age, gender, department):
        super().__init__(name, age, gender)
        self.department = department
# Create objects
student = Student("Muhammad Ibrahim", 20, "Male", "CSE/94")
teacher = Teacher("Maam Farhat Naureen", 40, "Female", "Python")
employee = Employee("Ahmed", 40, "Male", "HR")
# Access and print attributes directly
print("Student Information")
print("Name:", student.name)
print("Age:", student.age)
print("Gender:", student.gender)
print("Student ID:", student.student_id)

print("\nTeacher Information")
print("Name:", teacher.name)
print("Age:", teacher.age)
print("Gender:", teacher.gender)
print("Subject:", teacher.subject)

print("\nEmployee Information")
print("Name:", employee.name)
print("Age:", employee.age)
print("Gender:", employee.gender)
print("Department:", employee.department)
```

## Polymorphism

The word polymorphism means many forms. It comes from Greek word “Poly” which means many and “morphism” which means forms. It is the ability to take more than one form. Polymorphism can be achieved through method overriding or method overloading.

For example:

- to draw shape, it can draw circle, rectangle, oval etc. depend upon the type of object
- In python we use method overloading and method overriding to achieve polymorphism.

### Polymorphism with operator overloading

- Plus (+) operator in python performs many operations according to the context it shows polymorphism concept in object oriented programing.
- + operator save different meaning according to context and performs functions according to context that called operator overloading.

### + operators different operations according to context

```
print( 5 + 10 )      #int addition
print( 7.5 + 9.11 ) # float addition
print( "IMCS" + ",UOS" ) # string concatenation
print( [1,2,3] + [4,5,6] ) # list concatenation(merge list)
```

### Create a class for complex number and overload + operator.

```
class Complex:
  def __init__(self,real,imag):
    self.real = real
    self.imag = imag

  def display(self):
    print(self.real,"i+",self.imag,"j")

  def add(self,num2):
    newReal = self.real + num2.real
    newImag = self.imag + num2.imag
    return Complex(newReal,newImag)
n1 = Complex(5,6)
n1.display()

n2 = Complex(7,8)
n2.display()
n3 = n1.add(n2)
n3.display()
```

## Method overloading

- ➢ Multiple methods in same class with same values but different parameters
- ➢ Python does not directly support traditional method overloading. Instead, it relies on default arguments and variable length argument lists to achieve similar functionality.
- ➢ Default Values overloading
    - ▪ name.leng => return number of characters
    - ▪ Tuple.leng  => return items
- ➢ Default parameter overloading
    - ▪ def add ( *p)

## Method overloading with default arguments

```
class Calculator:
  def add(self,x,y=None,):
    if y is None:
      return x
    else:
      return x+y

c1 = Calculator()
print(c1.add(10))
print(c1.add(10,20))
```

## Method overloading with variable length argument

```
class Calculator:
  def add(self,*args):
    total = 0
    for num in args:
      total += num
    return total

c1 = Calculator()
print(c1.add(10))
print(c1.add(10,20))
print(c1.add(10,20,30))
```

## Method Overriding

Method overriding occurs when the sub class provides a specific implementation for an inherited method from its super class / parent class.

The sub class method must have the same value and parameters as the super class method.

```python
class Vehicle:
  def __init__(self,brand,model):
    self.brand = brand
    self.model = model

  def move(self):
    print("Move")

class Car(Vehicle):
  pass

class Plane(Vehicle):
  def move(self):
    print("Fly")

car1 = Car("Toyota","Camry")

plane1 = Plane("Boeing","747")

for el in (car1,plane1):
  print(el.brand,el.model)
  el.move()
```

```python
class Animal:
  def speak(self):
    print("Generic animal sounds")

class Dog(Animal):
  def speak(self):
    print("Dog barks")

class Cat(Animal):
  def speak(self):
    print("Cat meows")

a = Animal()
d = Dog()
c = Cat()

a.speak()
d.speak()
c.speak()
```

https://github.com/ibraheem-02/Python_Programs/blob/main/OOP_Python.ipynb