**Oops part 2**

* **Polymorphersion**
* **Encapsulation**
* **Data abstraction**

**Single inheritance:**

class Parent:

    def output(self):

         print('this is parent class')

class Child(Parent):

    def outputChild(self): # output

        print('this is child class')

i=Child()

i.output()

i.outputChild()

**multiple inheritance :**

class Father:

    def output(self):

        print('this is parent class')

class Mother:

    def outputM(self):

        print('this is mother class')

class Child(Father,Mother):

    def outputChild(self):

        print('this is child class')

ice=Child()

ice.output()

ice.outputM()

ice.outputChild()

**Multi-level inheritance :**

class GrandFather:

    def output(self):

        print('this is gf class')

class Father(GrandFather):

    def outputf(self):

        print('this is father class')

class Child(Father):

    def outputChild(self):

        print('this is child class')

ice=Child()

ice.output()

ice.outputf()

ice.outputChild()

**Hierarchical inheritance** :

class Father:#100cr

    def output(self):

        print('this is father class')

class Child1(Father):#50cr

    def outputf(self):

        print('this is child 1 class')

class Child2(Father):#50cr

    def outputChild(self):

        print('this is child  2 class')

ice=Child1()

cream=Child2()

ice.output() #child 1 of parent

ice.outputf()

cream.output() # child 2 of parent

cream.outputChild() # child 2

**polymorspherism**

class Methodoverlod:

    def something(self,a=None,b=None,c=None):

        print(a,b,c)

obj=Methodoverlod()

obj.something(1,2,3)

obj.something(1,2)

obj.something(1)

obj.something()

**method overriding**

class Methodoverri:

    def display(self):

        print("this is parent class")

class Child(Methodoverri):

    def display(self):

        print("this is child class")

        super().display()

obj=Child()

obj.display()

**Encapsulation**

Binding of class (method and variables (attributes))

Public

Private \_ \_

Protect \_

Class GFather:

Def\_\_inti\_\_(self,a):

Self.\_\_y=a

Print(self.\_y)

Class Father (GFather):

Def display 1 (self):

Print(self.\_y)

Class child 2(father):

Def display (self)

Print(“child2”,self.\_y)

Obj=child2(12)

Obj=.display2()

Obj.display1()

**Global and local variable**

A=10

Def fun( ):

B= 20

Print(‘this is fun’,b,a)

Print(b)

Fun()

**Abstraction**

Abstraction method there is no body

Abstraction class can not create object

A class contain one or more Abstraction methods then it said to a Abstract base class.

from abc import ABC, abstractmethod

class Car(ABC):

    @abstractmethod

    def mileage(self):

        pass

class Tesla(Car):

    def mileage(self):

        print("The mileage is 30kmph")

class Suzuki(Car):

    def mileage(self):

        print("The mileage is 25kmph ")

class Duster(Car):

     def mileage(self):

          print("The mileage is 24kmph ")

class Renault(Car):

    def mileage(self):

            print("The mileage is 27kmph ")

# Driver code

t= Tesla ()

t.mileage()

r = Renault()

r.mileage()

s = Suzuki()

s.mileage()

d = Duster()

d.mileage()