TASK 25

In [1]:

```
#creating class
class Praveen():#defination
    age=21#atribute
    def display(self):#method
        print('this is inside the class and age is ',self.age)
#object creation to call our class
obj=Praveen()
obj.display()
```

```
this is inside the class and age is  21
```

In [2]:

```
# __init__ it is constructor
class Kumar():
    def __init__(self,name,age,dob):
        self.n=name
        self.a=age
        self.d=dob
    def show(self):
        print('init is constructor',self.n,self.a,self.d)
a=input('enter your name ')
b=input('enter your age ')
c=input('enter your dob ')
obj1=Kumar(a,b,c)
obj1.show()
```

```
enter your name praveen
enter your agw 21
enter your dob 24
init is constructor praveen 21 24
```

Inheritance

In [ ]:

```
'''
1.Single
2.Multiple
3.Multilevel
4.Heirarical
'''
```

In [5]:

```
#single (one paren one child)
class Parent:
    def output(sef):
        print('this is parent class')
class Child(Parent):

    def outputchild(self):
        print('this is child class')
obj2=Child()
obj2.output()#calling parent class
obj2.outputchild()#calling child class
```

```
this is parent class
this is child class
```

In [9]:

```
#Multiple (2 parent one child)

class Father:
    def output(sef):
        print('this is father class')
class Mother:
    def opmother(self):#without def also print

        print('this is mother class')
class Child(Father,Mother):

    def outputchild(self):
        print('this is child class')
obj2=Child()
obj2.output()#calling parent class
obj2.outputchild()
obj2.opmother()
```

```
this is father class
this is child class
this is mother class
```

In [10]:

```
#Multilevel (Grand father , father, child) it level by level

class GrandFather():
    def properties(self):
        print('this is grand father properties')

class Father(GrandFather):
    def properties1(self):
        print('this proprty come from fathers father')
class Child(Father):
    def prop(self):
        print('He got proporty from his father and indirect of grandfather')

land=Child()
land.properties()
land.properties1()
land.prop()
```

```
this is grand father properties
this proprty come from fathers father
He got proporty from his father and indirect of grandfather
```

In [35]:

```
# Hierarical (one parent two child)
class Parent():
    def display1(self):
        print('This is is parent class')
class Child1(Parent):
    def dis(self):
        print('this is child 1 class')
class Child2(Parent):
    def dis2(self):
        print('this is child 2 class')

#here we create 2 child class objects because we have 2 child class
obj4=Child1()
obj5=Child2()
obj4.display1()
obj4.dis()
obj5.dis2()
obj5.display1()
```

```
This is is parent class
this is child 1 class
this is child 2 class
This is is parent class
```

In [2]:

```
#Polymorphisam (many forms )
#METHOD OVERLOADING

class MethodOverloding():
    def disp(self,a=None,b=None,c=None):
        print('this is ',a,b,c)
 # same disp fun are there so overloading here
#     def disp(self,a,b):
#         print('this is ', a,b)
#     def disp(self,a):
#         print('this is ',a)
objec=MethodOverloding()
objec.disp(1,2,3)
objec.disp(1,2)
objec.disp(1)
objec.disp()
```

```
this is  1 2 3
this is  1 2 None
this is  1 None None
this is  None None None
```

In [55]:

```
#Method Overriding
#if function is same in parent and child class then compiler get confused
class MethodOverriding():
    def display1(self):
        print('this is Parent ')
class Child(MethodOverriding):
    def display1(self):
        print('this is child')
        super().display1()
ob=Child()
ob.display1()
```

```
this is child
this is Parent
```

In [56]:

```
#Encapsulation ( binding of methods and variables )

'''
access specifier/ modifiers are...
Public(y) # any one can access the properties
And
Protected(_y) #certain peopele or fun can access
Private(__y) #no access for any one

'''
```

In [5]:

```
#Public
class Encap():
    def __init__(self,a):
        self.y=a
class Child(Encap):
    def encapsu(self,c):
        print('this is encapsulation of public ', self.y,c)
en=Child(21)
en.encapsu(8)#def fun lo access cheyyali ante paina variable iyyali
```

```
this is encapsulation of public  21 8
```

In [11]:

```
#Protect
```

```
class Encap1():
    def __init__(self,a):
        self._y=a
class Child1(Encap1):
    def encapsu(self):
        print('this is encapsulation of protect ', self._y)
en1=Child1(21)
en1.encapsu()
```

```
this is encapsulation of protect  21
```

In [13]:

```
#Private
class Encap2():
    def __init__(self,a):
        self.__y=a
class Child2(Encap2):
    def encapsu(self):
        print('this is encapsulation of private ', self.__y)
en2=Child2(21)
en2.encapsu()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-13-20aafc39f5c6> in <module>
      6         print('this is encapsulation of private ', self.__y)
      7 en2=Child2(21)
----> 8 en2.encapsu()

<ipython-input-13-20aafc39f5c6> in encapsu(self)
      4 class Child2(Encap2):
      5     def encapsu(self):
----> 6         print('this is encapsulation of private ', self.__y)
      7 en2=Child2(21)
      8 en2.encapsu()

AttributeError: 'Child2' object has no attribute '_Child2__y'
```

TASK 26

In [ ]:

```
'''
Abstraction
1.no body in abs
2.no obj creation
3.inheritance is not possible
4.class contain one or more abstraction methods is said to be a abc abstractbase class

'''
```

In [20]:

```
from abc import ABC,abstractmethod
class Car(ABC):
    @abstractmethod #@ is a decarative function if apply the all funs to the body
    def milage(self):
        pass
class Tesla(Car):
    def milage(self):
        print('electric car')
class Audi(Car):
    def milage(self):
        print('Petrol car')


test=Tesla()
test.milage()
```

```
# t=Car()
# t.milage()
```

```
electric car
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```