Oops

In [5]:

```
class person:
    def __init__(self):
        self.name="sam"
        self.gender="female"
        self.age=22

    def talk(self):
        print("hi iam ",self.name)

    def vote(self):
        if self.age<18:
            print("i am not elgible to vote")
        else:
            print("i am elgible to vote")

obj=person()
person.talk(obj)
person.vote(obj)
```

```
hi iam  sam
i am elgible to vote
```

In [16]:
```
class person:
    def __init__(self,name,gender,age):
        self.name=name
        self.gender=gender
        self.age=age

    def talk(self):
        print("hi iam ",self.name)

    def vote(self):
        if self.age<18:
            print("i am not elgible to vote")
        else:
            print("i am elgible to vote")

obj1=person("vaishnavi","female",22)
obj2=person("keerthi","female",12)
obj3=person("karan",'male',29)
obj1.talk()
obj1.vote()
obj2.talk()
obj2.vote()
obj3.talk()
obj3.vote()
```

```
hi iam  vaishnavi
i am elgible to vote
hi iam  keerthi
i am not elgible to vote
hi iam  karan
i am elgible to vote
```

inheritance

single level inheritance

In [18]:
```
#if you want to use methods of 1st class in 2nd class inheritance comes into
class parent():
    def feature1(self):
        print("feature 1 is working")

    def feature2(self):
        print("feature 2 is working")

obj=parent()
obj.feature1()
obj.feature2()
```

```
feature 1 is working
feature 2 is working
```

In [41]:
```
#just by adding parent class in child class i am getting all the features wh
class child(parent):
     def feature3(self):
        print("feature 3 is working")

     def feature4(self):
        print("feature 4 is working")

obj=child()
obj.feature1()
obj.feature2()
obj.feature3()
obj.feature4()
```

```
feature 1 is working
feature 2 is working
feature 3 is working
feature 4 is working
```

In [56]:
```
#another class
class grandchild():
    def feature5(self):
        print("feature 5 is working")

object=grandchild()
object.feature5()
```

```
feature 5 is working
```

multi level in heritance

In [57]:
```
class grandchild(child):
    def feature5(self):
        print("feature 5 is working")

object=grandchild()
object.feature1()
object.feature2()
object.feature3()
object.feature4()
object.feature5()
```

```
feature 1 is working
feature 2 is working
feature 3 is working
feature 4 is working
feature 5 is working
```

multiple inheritance

In [59]:
```
class a():
    def feature6():
        print("feature 6 is working")
```

In [69]:
```
class b():
    def feature7():
        print("feature 7 is working")
```

In [73]:
```
class c():
    def feature8():
        print("feature 8 is working")
```

In [77]:
```
#multiple inheritance
class c(a,b):
    def feature8():
        print("feature 8 is working")
obj=c
obj.feature6()
obj.feature7()
obj.feature8()
```

```
feature 6 is working
feature 7 is working
feature 8 is working
```

encapulisation

In [24]:
```
class car:
    def __init__(self,speed,color):
        self.speed=speed
        self.color=color

    def set_speed(self,value):
        self.speed=value

    def get_speed(self):
        return self.speed

ford=car(200,"black")
honda=car(300,"red")
audi=car(400,"white")
#here we are changing the values of variables so to prevent that encapulisat
ford.speed=250
honda.speed=350
print(ford.get_speed())
print(honda.get_speed())
```

```
250
350
```

In [29]:
```
#public variable
class public:
    def __init__(self,name):
        self.name=name
obj=public("vaishnavi")
obj.name
```

Out[29]: 'vaishnavi'

In [41]:
```
#we cannot cal the private varaiable outside the function
class private:
    def __init__(self,name):
        self.__name=name
obj=private("vaishnavi")
print(obj.__name)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [41], in <cell line: 6>()
      4         self.__name=name
      5 obj=private("vaishnavi")
----> 6 print(obj.__name)

AttributeError: 'private' object has no attribute '__name'
```

In [71]:
```
#we can only call the private variable within the function
class private:
    def __init__(self,name):
        self.__name=name
    print(obj.__name)
obj=private("vaishnavi")
```

```
vaishnavi
```

In [73]:
```
#if you want to call the private variable outside the function you have to c
class private:
    def __init__(self,name):
        self.__name=name

obj=private("vaishnavi")
print(obj._private__name)
```

```
vaishnavi
```

polymorphism

operator overloading

In [74]:
```
#here "+" is used as addition in integer
a=10
b=20
c=a+b
print(c)
```

```
30
```

In [76]:
```
#whereas "+" is used as concatination in strings
a="vaishnavi "
b="abbugari"
c=a+b
print(c)
```

```
vaishnavi abbugari
```

method overloading

In [81]:
```
#using only one function in three ways
class overload:
    def display(self,a=None,b=None):
        print(a,b)

obj=overload()
obj.display()
obj.display(10)
obj.display(10,20)
```

```
None None
10 None
10 20
```

overriding

In [83]:
```
class father:
    def transport(self):
        print("cycle")
obj=father()
obj.transport()
```

```
cycle
```

In [85]:
```
class son(father):
    pass
obj=son()
obj.transport()
```

```
cycle
```

In [86]:

```
class son(father):
    def transport(self):
        print("duke")
obj=son()
obj.transport()
```

```
duke
```