```
six pillars of opps:
    object
    class
    polymorphism
    abstraction
    inheritance
    encapsulation
```

In [ ]:

```
object:
    it is real time entity(exists in real time)
    ex.classroom,fan,laptop

class:
    is a blueprint/template/instance of object
    ex.refrigerator =object , structure of refrigerator
    (doors,containers etc) is class

class=attributes+methods

attributes define features and characteristics
method is function which actions you can perform on objects(verbs)
```

```
encapsulation: binding up data and method together
    ex. capsule:binding various chemical into one capsule
```

```
abstraction: hidding unnecessary information
    showing important info
    ex. airconditioner : we cannot see mechanism but only see the cool air, all builtin function
```

```
polymorphism: poly: many, morphism: forms
            +:used for addition,concatenation
            * : multiplication, repeatation
```

```
inheritance:  inherits the properties of parent class
    ex. property distribution: inherits the property of parent by the
        child

types of in heritance:
    1.single :
        parent-->child
    2.multi level inheritance:
        parentA  +  parentB -->child c
        binding up data and methods together
    3.heirarchical
    parenta=child 1 and child 2
```

Class

In [ ]:

In [ ]:

static data

In [1]:
```
class Student:
    name="abc"
    age=23
    gender="female"


    def info(self):
        print("name is ",self.name)
        print("age is", self.age)
        print("gender is",self.gender)

    def info1(self):
        self.address=input("enter address :")
        self.course=input("enter course name :")

    def disp(self):
        print("my info is")
        print(self.name)
        print(self.age)
        print(self.gender)
        print(self.address)
        print(self.course)

s=Student()
```

In [2]:
```
s.info()
```

```
name is  abc
age is 23
gender is female
```

In [4]:
```
s.info1()
```

```
enter address :Plot no. 16 gat no.40 joshi nagar ,osmanabad
enter course name :Master in datascience
```

In [5]:
```
s.disp()
```

```
my info is
abc
23
female
Plot no. 16 gat no.40 joshi nagar ,osmanabad
Master in datascience
```

constructor

```
__init__
```

In [7]:
```
class Abc:
    def __init__(self):
        print("hello")

a=Abc()
```

```
hello
```

In [11]:
```
class Student:
    def __init__(self,name,age,marks):
        self.name=name
        self.age=age
        self.marks=marks

    def disp(self):
        print("name is ",self.name)
        print("age is ",self.age)
        print("marks is ",self.marks)

st=Student("gayu",34,89)
```

In [12]:
```
st.disp()
```

```
name is  gayu
age is  34
marks is  89
```

```
self:is a required argument
```

Encapsulation

In [14]:
```
class Customer:
    def cs(self):
        self.product=input("enter product name: ")
        self.price=input("enter price of product: ")

c=Customer()
```

In [15]:
```
c.cs()
```

```
enter product name: laptop
enter price of product: 75000
```

Abstraction

In [16]:
```
print("hello")  #print is function
```

```
hello
```

In [17]:
```
l=[1,2,3,4]
print(sum(l))
```

```
10
```

Polymorphism

In [27]:
```
s="abc"
v="xyz"
print(s+v)
a=10+20
print(a)
print(s*2)
10*2
```

```
abcxyz
30
abcabc
```

Out[27]:
```
20
```

inheritance

1. single inheritance

In [31]:
```
class A:
    def geta(self):
        self.a=int(input("enter a value of a: "))

class B(A):
    def getb(self):
        self.b=int(input("enter a value of b: "))

    def add(self):
        print("addition is : ",self.a+self.b)
```

In [32]:
```
b=B()
```

In [33]:
```
b.geta()
```

```
enter a value of a: 32
```

In [34]:
```
b.getb()
```

```
enter a value of b: 22
```

In [35]:
```
b.add()
```

```
addition is :  54
```

In [38]:
```
c=A()
c.geta()
```

```
enter a value of a: 23
```

In [39]:
```
c.getb()   # this error we getting because in single inheritance the class A does not inherit the another class B()propert
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[39], line 1
----> 1 c.getb()

AttributeError: 'A' object has no attribute 'getb'
```

2.multilevel inheritance

In [40]:
```
class A:
    def geta(self):
        self.a=int(input("enter a value of a:"))
class B(A):
    def getb(self):
        self.b=int(input("enter a value of b :"))

class C(B):
    def getc(self):
        self.c=int(input("enter value of c: "))

    def add(self):
        print("addition is ",self.a+self.b+self.c)
        print("value of b is: ",self.b)


c=C()
```

In [41]:
```
c.geta()
```

```
enter a value of a:22
```

In [42]:
```
c.getb()
```

```
enter a value of b :20
```

In [43]:
```
c.getc()
```

```
enter value of c: 20
```

In [44]:
```
c.add()
```

```
addition is  62
value of b is:  20
```

3.Multiple inheritance

In [49]:
```
class A:
    def geta(self):
        self.a=int(input("enter a value of a:"))
class B:
    def getb(self):
        self.b=int(input("enter a value of b :"))

class C(A,B):
    def getc(self):
        self.c=int(input("enter value of c: "))

    def add(self):
        print("addition is ",self.a+self.b+self.c)
        print("value of b is: ",self.b)


c=C()
```

In [50]:
```
c.geta()
c.getb()
c.getc()
c.add()
```

```
enter a value of a:2
enter a value of b :3
enter value of c: 4
addition is  9
value of b is:  3
```

hierarchical inheritance

In [58]:
```
class A:
    def geta(self):
        self.a=int(input("enter a value of a:"))
class B(A):
    def getb(self):
        self.b=int(input("enter a value of b :"))
    def add1(self):
        print("Addition is : ",self.a+self.b)

class C(A):
    def getc(self):
        self.c=int(input("enter value of c: "))

    def add2(self):
        print("addition is ",self.a+self.c)
        print("value of a is: ",self.a)


c=C()
b=B()
```

In [59]:
```
c.geta()
c.getc()
c.add2()
```

```
enter a value of a:3
enter value of c: 4
addition is  7
value of a is:  3
```

In [60]:
```
b.geta()
b.getb()
b.add1()
```

```
enter a value of a:2
enter a value of b :3
Addition is :  5
```

In [61]:
```
l=[2,3,4,5,6]
l1=l
```

In [63]:
```
l1.insert(2,70)
print(l1)
print(l)   #deepcopy
```

```
[2, 3, 70, 70, 4, 5, 6]
[2, 3, 70, 70, 4, 5, 6]
```

In [64]:
```
l2=l.copy()
l2.insert(3,80)
print(l2)
print(l)  #shallow copy
```

```
[2, 3, 70, 80, 70, 4, 5, 6]
[2, 3, 70, 70, 4, 5, 6]
```

In [80]:
```
l=[1,2,2,1,3,4,3,6]
#o/p l=[1,2,3]
rvalue=set()
univalue=set()
for i in l:
    if i in univalue :
        rvalue.add(i)
    else:
        univalue.add(i)

print(rvalue)
print(univalue)
print(l)
#mam solved
s=set(l)
l1=[]
for i in s:
    if l.count(i)>1:
        l1.append(i)
print(l1)
```

```
{1, 2, 3}
{1, 2, 3, 4, 6}
[1, 2, 2, 1, 3, 4, 3, 6]
```

In [92]:
```
l=[1,2,2,1,3,4,3,6]
#o/p {1:2,2:2,3:1,4:1,6:1}
l = [1, 2, 2, 1, 3, 4, 3, 6]

c = {}

# Iterate through the list
for element in l:
    # Check if the element is already in the dictionary
    if element in c:
        # Increment the count if the element is present
        c[element] += 1
    else:
        # Add the element to the dictionary with a count of 1 if not present
        c[element] = 1

# Print the result
print(c)
```

```
{1: 2, 2: 2, 3: 2, 4: 1, 6: 1}
```

In [85]:
```
d={"name":['a','b'],"Age":[12,15]}
#o/p {'a':12,'b':15}
d1=dict(zip(d["name"],d["Age"]))
print(d1)
```

```
{'a': 12, 'b': 15}
```

In [ ]: