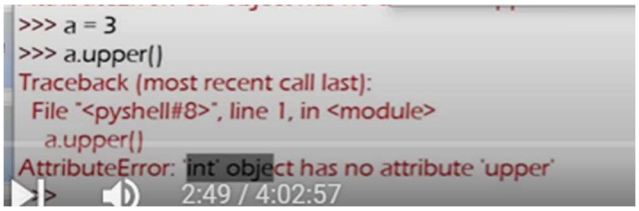
OOPS

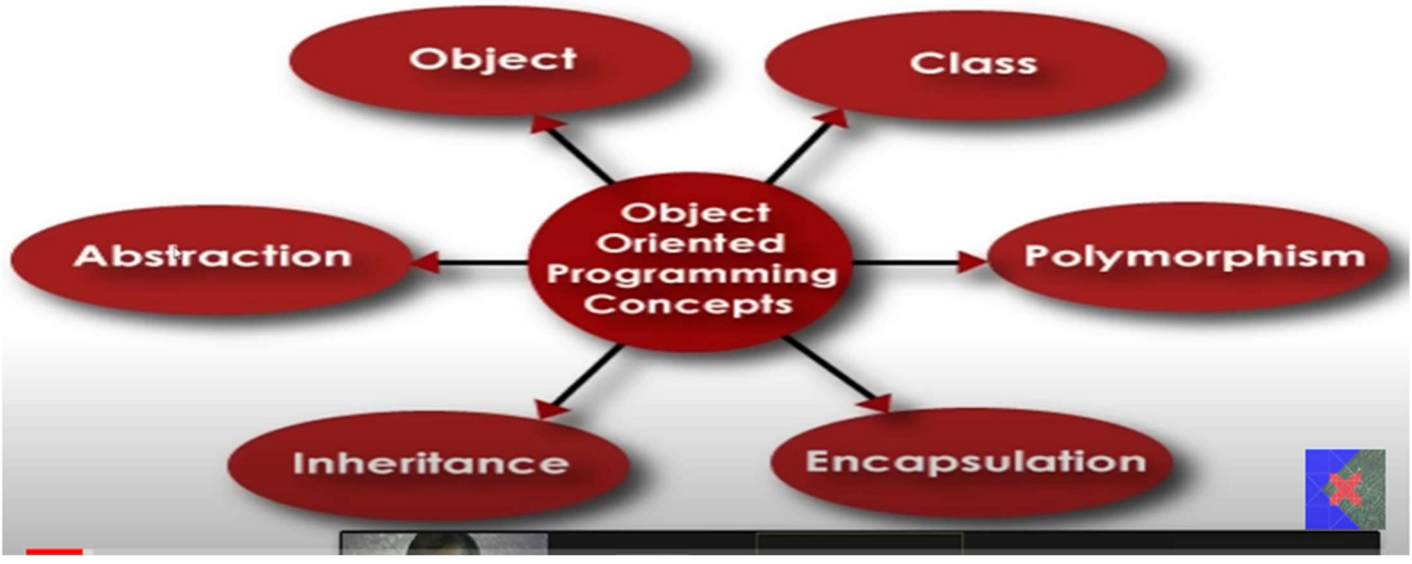
```
>>> L = [1,2,3,4]
>>> L
[1, 2, 3, 4]
>>> L.upper()
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    L.upper()
AttributeError: 'list' object has no attribute 'upper'
>>>
>>>
>>> city = "Kolkata"
>>> city.append("a")
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    city.append("a")
AttributeError: 'str' object has no attribute 'append'
```



int object has no attribute upper means uss class mei upper nahi bana hai isliye voa call nahi kar pa rahe hai.

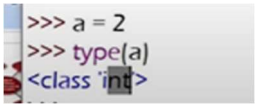
. Everything in python is object.

. oops ka feature yei hai ki aap apna khud ka data type bana sakte hoa.



Class is a blue print.

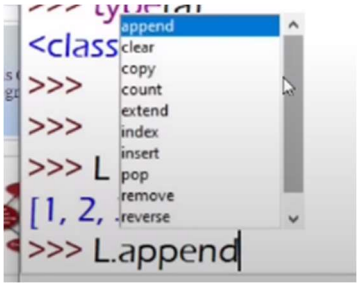
Object hamesha kisi class ka hota hai.



a int class ka object hai

Python mei bahut sare data types hai jaise list,tuples and so o & ussai kisi ne banaya hi hoga

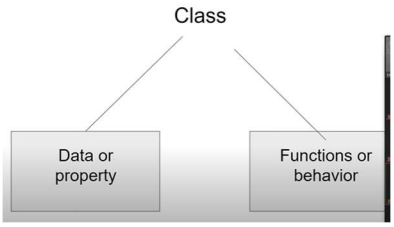
L=[1,2,3]



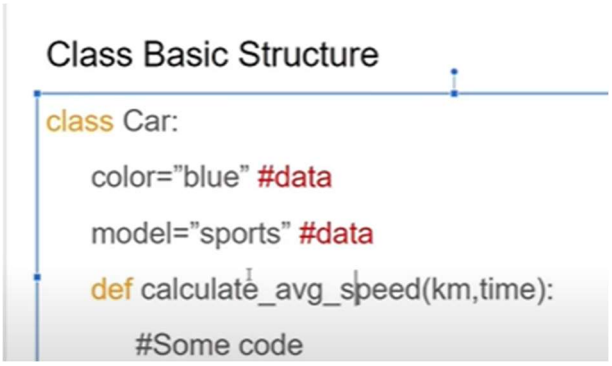
L ke baad dot lagate hai toa joa kaayi function dikhai de rahe hai voa list class mei define . And in function koa only list object access kar sakta hai.

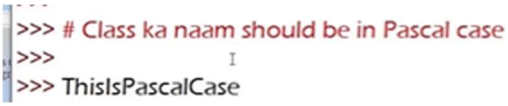
Class koa banane mei jyada time lagega instead of making object.

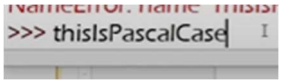
Class is a blueprint of object which tell how the object will behave.



Data means us object ke baare mai whereas function means uss object ka behaviour.





Camel case – mei first vala small 

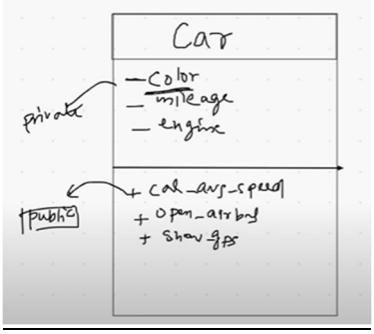
Snakecase-in this two word is joined with underscore eg- snake_cow

Variables and function ka name in class will be in snakecase.

<u>Representation of a class</u>

- **Represent datamembers**

  + represent public method,member function can be accessed outside



Data private hota hai generally class ke bahar aap usai use nahi kar sakte

Object creation

Syntax- objectname=classname()

Accessing syntax- objectname.memberfunction of class()

Object Examples

1. Car----------->WagorR // wagonr=Car()
2. Sports---------->Gilli Danda// gillidanda=Sports()
3. Animals---------->Langoor// langoor=Animals()

Object literal- assani se object banana ka tarika.

```
>>> L = [1,2,3]
>>> L
[1, 2, 3]
>>> # Object Literal
```

Aap neeche ddiye gaye taike se bhi list bana sakte hoa.

```
>>> L = list()
```

ATM MACHINE BANANE KI KHOSIS KAREGE

Function call mei L koa andar pass karenge but this doen't happen in case of mehod calling.

```
>>> len(L)
0
>>> # Function
>>>
>>> L.append(1)
>>> L
[1]
```

it means append is a method joa list class mei defined hai

Len is general which can be used by anyone

init ke andar hum variables koa declare karte hai. Generally in python we don't declare it outside.

```
class ATM:
    def __init__ (self):
        self.pin=0
        self.balance=10
    def pin_change(self):
        self.pin=1000
    def display(self):
        print("pin=",self.pin)
        print("balance=",self.balance)
a=ATM()
a.pin_change()
a.display()
```

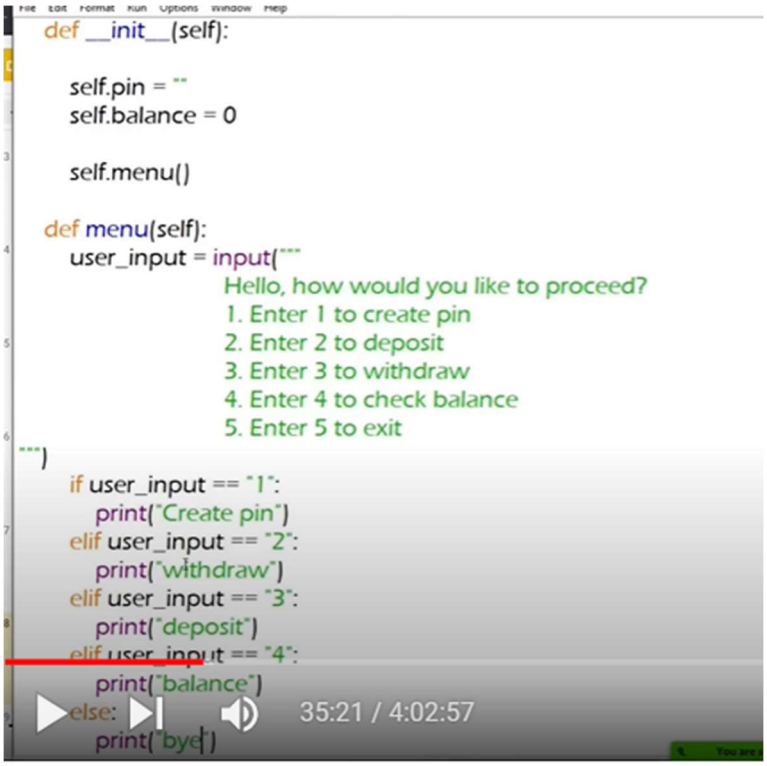
```
pin= 1000
balance= 10
```

init is a constructor. Which means jiske andar Rakha hua code automatically execute hota hai as the object is created.

```
>>> from xyz import Atm
>>> sbi = Atm()
```

class atm is imort from file name xyz

Python mei hamesha constructor ka naam init hi hona chahiye.



```
class ATM:
    def __init__ (self):
        self.pin=""
        self.balance=10
        self.menu()
    def menu(self):
        user_input=input("""Hello how would u proceed?Enter option to create pin
        1. Enter 1 to create pin
        2. Enter 2 to deposit
        3. Enter 3 to withdraw
        4. Enter 4 to check balance
        5. Enter 5 to exit
        """)

        if(user_input=="1"):
            print("Create pin=")
            self.create_pin()
        elif(user_input=="2"):
            print("deposit")
            self.deposit()
        elif(user_input=="3"):
            print("withdraw")
            self.withdraw()
        elif(user_input=="4"):
            print("checkbalance")
            self.checkbalance()
        else:
            print("bye")
    def create_pin(self):
        self.pin=input("Enter ur pin=")
        print("pin set successfully")
    def deposit(self):
        temp=input("enter ur pin=")
        if(temp==self.pin):
            amount=int(input("Enter the amount="))
            self.balance=self.balance+amount
            print("deposit succesful")
        else:
            print("invalid pin")
    def withdraw(self):
        temp=input("enter ur pin=")
        amount=int(input("Enter the amount wanted="))
        if(temp==self.pin and amount<self.balance):
            self.balance=self.balance-amount
            print("withdraw succesful")
        else:
            print("invalid pin /insufficient money")
```

```
 def checkbalance(self):
     temp=input("enter ur pin=")
     if(temp==self.pin):
          print("current balance=",self.balance)
     else:
         print("invalid pin")
```

```
a=ATM()
Hello how would u proceed?Enter option to create pin
        1. Enter 1 to create pin
        2. Enter 2 to deposit
        3. Enter 3 to withdraw
        4. Enter 4 to check balance
        5. Enter 5 to exit
        1
Create pin=
Enter ur pin=5050
pin set successfully
a.deposit()
enter ur pin=5050
Enter the amount=8500
deposit succesful
a.checkbalance()
enter ur pin=5050
current balance= 8510
a.withdraw()
enter ur pin=5050
Enter the amount wanted=32
withdraw succesful
a.checkbalance()
enter ur pin=5050
current balance= 8478
```

Pyhon mei kuch methods koa special method or magic methods or dender methods kahete hai.

Magic methods voa method hoge jiske aage and peeche dono jagah double underscore hoga.

Init is a magic method.

Magic methods can't be called by objects. It is automatically trigerred

What is the utility of the constructor? Yei ek aisi cheej hai jiska control user ke pass nahi hai. App open hote hi yei chakne lagega. Like internet se connect hona when app open hooa iske liye hum koi button nahi daalege so here we use constructor inside it build connectivity of app with internet when open.

Self- it is a object

```
class ATM:
    def __init__ (self):
        self.pin=100
        self.balance=10
        print("self=",id(self))
a=ATM()
print("id of object a=",id(a))
```

input

```
self= 139929066832512
id of object a= 139929066832512
```

it means object of a class hi self hai

Agar self hatadenge toa code error dega
Class ke andar koi bhi cheej koa acess class ka object access kar sakta hai.
Ek class ka method dusre method se directly access nahi kar saKTA.i can be accessed though object

PYHON FRACTION KOA HANDLE NAHI KARTA

Abb hum apna khud ka data banayege joa fraction ke add, subtract ,multiply and divide koa handle kar sake.

```
class Fraction:
    def __init__(self,n,d):
        self.num=n
        self.den=d
a=Fraction(20,50)
type(a)
<class '__main__.Fraction'>
```

Isko pass bhi kar sakte hai list mei

```
L=[1,2,3,a]
L
[1, 2, 3, <__main__.Fraction object at 0x000001DEBFF11450>]
```

Abbyei dikhta kaise hai yei humhe batana padega tabhi vaisa hi dikhega.

So yaha magic method use hoa

```
class Fraction:
    def __init__(self,n,d):
        self.num=n
        self.den=d
    def __str__(self):
        return "Hi anupam"
> a=Fraction(58,72)
> type(a)
  <class '__main__.Fraction'>
> print(a)
  Hi anupam
> l=[1,2,a]
> l
  [1, 2, <__main__.Fraction object at 0x00000182FC509390>]
>
```

Str magic method is used for printing.

```
class Fraction:
    def __init__(self,n,d):
        self.num=n
        self.den=d
    def __str__(self):
        return "{}/{}".format(self.num,self.den)
```

```
= RESTART: C:/Users/Anupam Shukla/OneDrive/Desktop/DXC/pytho
a=Fraction(48,3)
print(a)
48/3
l=[1,2,a]
l
[1, 2, <__main__.Fraction object at 0x0000026129269360>]
```

Now we are trying to add to fraction object

```
b=Fraction(2,3)
c=Fraction(4,5)
print("add=",b+c)
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module>
    print("add=",b+c)
TypeError: unsupported operand type(s) for +: 'Fraction' and 'Fraction'
```

Cant be added directly . so we define it how we can add and then represent.

So to peform add in this we use magic function __add__

```
class Fraction:
    def __init__(self,n,d):
        self.num=n
        self.den=d
    def __str__(self):
        return "{}/{}".format(self.num,self.den)
    def __add__(self,other):
        temp_num=(self.num*other.den)+(other.num*self.den)
        temp_den=self.den*other.den
        return "{}/{}".format(temp_num,temp_den)
    def __sub__(self,other):
        temp_num=(self.num*other.den )-(other.num*self.den)
        temp_den=self.den*other.den
        return "{}/{}".format(temp_num,temp_den)
    def __mul__(self,other):
        temp_num=self.num*other.num
        temp_den=self.den*other.den
        return "{}/{}".format(temp_num,temp_den)
    def __truediv__(self,other):
        temp_num=self.num*other.den
        temp_den=self.den*other.num
        return "{}/{}".format(temp_num,temp_den)


a=Fraction(2,3)
b=Fraction(6,4)
print("a=",a," b=",b)
a= 2/3  b= 6/4
print("add=",a+b,"\n sub=",a-b," \n mul=",a*b,"\n div=",a/b)
add= 26/12
 sub= -10/12
 mul= 12/12
 div= 8/18
```

What is an instance variable? Joa bhi variable aap constructor ke andar banate hoa ussai instance variable kahete hai.

Inki value harr object ke liye alag hota hai.

Encapsulation

Object can access method as well as data members.

```
def __init__(self):

    self.pin = ""
    self.balance = 0
```

So pin and balance can be also accessed . so if any one want then they can directly update it.

```
>>> sbi.balance
0
>>> sbi.balance = "wgwrgrhgw"
>>> sbi.deposit()
Enter your pin1234
Enter the amount50000
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    sbi.deposit()
  File "C:\Users\91842\AppData\Local\Programs\Python\Python36\Lib\xyz.py", li
n deposit
```

So ki koi aur isko update na kar sake we will hide these 2 i.e. pin and balance so that they can't be directly accessed.

For that implementing we use access modifier- private – private laga matlab voa class ke andar hi access hoa sakta hai bahar nahi- private**(jise karna hai uske aage double underscore laga dete hai**).

Output before encapsulation of balance and pin

```
a=ATM()
Hello how would u proceed?Enter option to create pin
        1. Enter 1 to create pin
        2. Enter 2 to deposit
        3. Enter 3 to withdraw
        4. Enter 4 to check balance
        5. Enter 5 to exit
        1
Create pin=
Enter ur pin=2312
pin set successfully
a.deposit()
enter ur pin=2312
Enter the amount=40
deposit succesful
a.balance
50
a.balance=80
a.balance
80
```

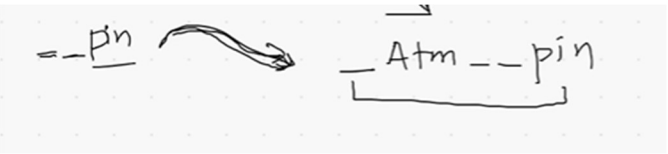
After encapsulation we can't directly access pin and balance

```
class ATM:
    def __init__ (self):
        self.__pin=""
        self.__balance=10
        self.menu()
    def menu(self):
        user_input=input("""Hello how would u proceed?Enter option to create pin
        1. Enter 1 to create pin
        2. Enter 2 to deposit
        3. Enter 3 to withdraw
        4. Enter 4 to check balance
        5. Enter 5 to exit
        """)

        if(user_input=="1"):
            print("Create pin=")
            self.create_pin()
        elif(user_input=="2"):
            print("deposit")
            self.deposit()
        elif(user_input=="3"):
            print("withdraw")
            self.withdraw()
        elif(user_input=="4"):
            print("checkbalance")
            self.checkbalance()
        else:
            print("bye")
    def create_pin(self):
        self.__pin=input("Enter ur pin=")
        print("pin set successfully")
    def deposit(self):
        temp=input("enter ur pin=")
        if(temp==self.__pin):
            amount=int(input("Enter the amount="))
            self.__balance=self.__balance+amount
            print("deposit succesful")
        else:
            print("invalid pin")
    def withdraw(self):
        temp=input("enter ur pin=")
        amount=int(input("Enter the amount wanted="))
        if(temp==self.__pin and amount<self.__balance):
            self.__balance=self.__balance-amount
            print("withdraw succesful")

        else:
            print("invalid pin /insufficient money")
    def checkbalance(self):
        temp=input("enter ur pin=")
        if(temp==self.__pin):
            print("current balance=",self.__balance)
        else:
            print("invalid pin")
```

```
a=ATM()
Hello how would u proceed?Enter option to create pin
        1. Enter 1 to create pin
        2. Enter 2 to deposit
        3. Enter 3 to withdraw
        4. Enter 4 to check balance
        5. Enter 5 to exit
        1
Create pin=
Enter ur pin=2312
pin set successfully
a.deposit()
enter ur pin=2312
Enter the amount=40
deposit succesful
a.balance
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    a.balance
AttributeError: 'ATM' object has no attribute 'balance'
```

Joa cheej aap chahe ki samne vala na dekh paye uske aage double underscore laga doa chahe voa method hoa or data

When we apply double underscore then python usko kisi aur mei convert kar deta hai.



because of this hum ab pin koa access nahi kar sakte

```
>>> sbi = Atm()
1953851704992

                Hello, how would you like to proceed?
                1. Enter 1 to create pin
                2. Enter 2 to deposit
                3. Enter 3 to withdraw
                4. Enter 4 to check balance
                5. Enter 5 to exit
1
Enter your pin1234
Pin set successfully
>>> sbi.__balance = "rsrfhbsrntdejn"
>>> sbi.deposit()
Enter your pin1234
Enter the amount50000
```

```
>>> sbi.check_balance()
Enter your pin1234
50000
>>>
```

why this is not giving the error

It means jab self.__balance write kiye then naya variable ban gaya jiske banne se koi problem nahi hai.

Koi error nahi hai because aapke code ke encapsulation logic ke according self.__balance variable ka use hi nahi hoa raha.so it will not effect existing logic

Agar hum koi cheej private kar chuke hai uske bavjood hum ussai access karna chah rahe hai then syntax to access is given below-

Syntax- objectname.single underscore classname doubleunderscore datamember/function

Eg- `a._ATM__balance=`

When we write this syntax then we can directly manipulate things

```
Hello how would u proceed?Enter option to create pin
        1. Enter 1 to create pin
        2. Enter 2 to deposit
        3. Enter 3 to withdraw
        4. Enter 4 to check balance
        5. Enter 5 to exit
        1
Create pin=
Enter ur pin=2040
pin set successfully
a.deposit()
enter ur pin=2040
Enter the amount=50
deposit succesful
a.checkbalance()
enter ur pin=2040
current balance= 60
a._ATM__balance=1500
a.checkbalance()
enter ur pin=2040
current balance= 1500
```

So from this we learnt that nothing in python is truly private.

GET AND SET PIN

```
class ATM:
    def __init__ (self):
        self.__pin=""
        self.__balance=10
        self.menu()
    def getpin(self):
        return self.__pin
    def setpin(self,newpin):
        self.__pin=newpin
        print("Pin changed successfully")
    def menu(self):
        user_input=input("""Hello how would u proceed?Enter option to create pin
```

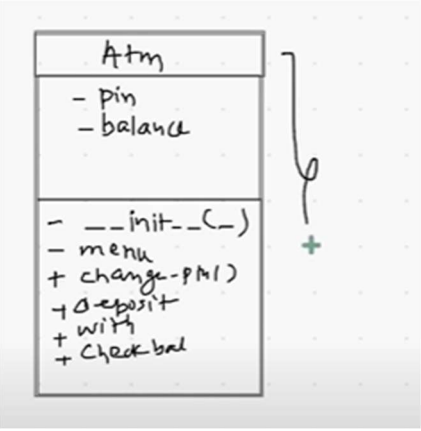
```
    1. Enter 1 to create pin
    2. Enter 2 to deposit
    3. Enter 3 to withdraw
    4. Enter 4 to check balance
    5. Enter 5 to exit
    """)

    if(user_input=="1"):
        print("Create pin=")
        self.create_pin()
    elif(user_input=="2"):
        print("deposit")
        self.deposit()
    elif(user_input=="3"):
        print("withdraw")
        self.withdraw()
    elif(user_input=="4"):
        print("checkbalance")
        self.checkbalance()
    else:
        print("bye")
def create_pin(self):
    self.__pin=input("Enter ur pin=")
    print("pin set successfully")
def deposit(self):
    temp=input("enter ur pin=")
    if(temp==self.__pin):
        amount=int(input("Enter the amount="))
        self.__balance=self.__balance+amount
        print("deposit succesful")
    else:
        print("invalid pin")
```

```
    def withdraw(self):
        temp=input("enter ur pin=")
        amount=int(input("Enter the amount wanted="))
        if(temp==self.__pin and amount<self.__balance):
            self.__balance=self.__balance-amount
            print("withdraw succesful")
        else:
            print("invalid pin /insufficient money")
    def checkbalance(self):
        temp=input("enter ur pin=")
        if(temp==self.__pin):
            print("current balance=",self.__balance)
        else:
            print("invalid pin")
```

```
a=ATM()
Hello how would u proceed?Enter option to create pin
        1. Enter 1 to create pin
        2. Enter 2 to deposit
        3. Enter 3 to withdraw
        4. Enter 4 to check balance
        5. Enter 5 to exit
        1
Create pin=
Enter ur pin=2080
pin set successfully
a.getpin()
'2080'
a.setpin('17')
Pin changed successfully
a.getpin()
'17'
```

GETTER AND SETTER KE THROUGH HUM PRIVATE KOA ACCESS BHI KAR SAKTE HAI AND SET BHI KAR SAKTE HAI.

Class diagram

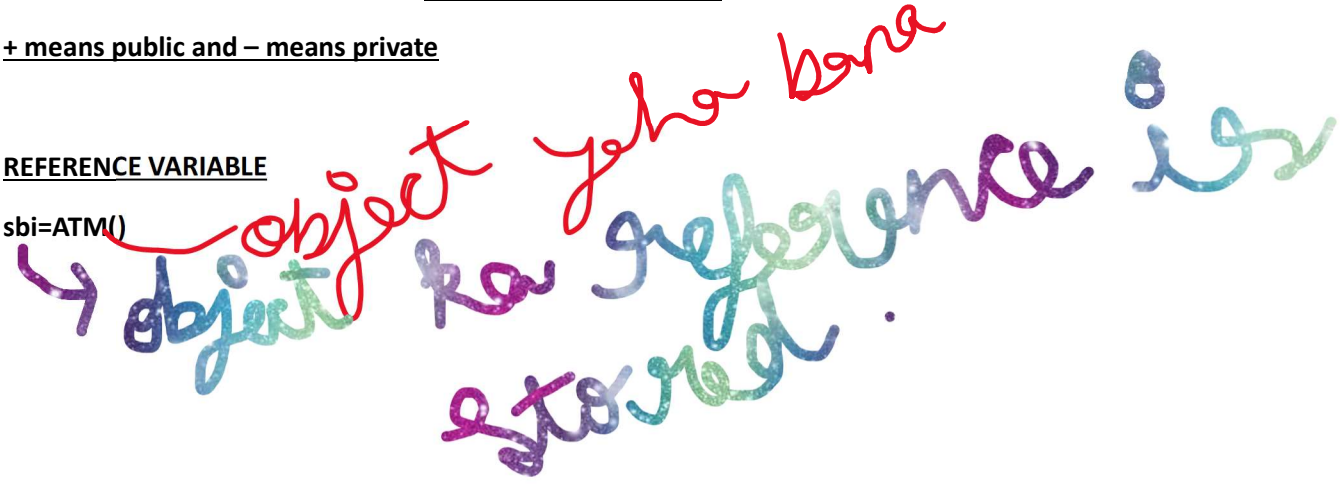


MENU MAI + Kare not -.

+ means public and – means private

REFERENCE VARIABLE

sbi=ATM()



Object create karte time object ko ajis variable mei store karte hai usko reference variable kahete hai.

PASS BY REFERNCE

```
class Customer:
    def __init__(self,name,gender):
        self.name=name
        self.gender=gender
def greet(customer):
    if(customer.gender=="male"):
        print("HELLO",customer.name," sir")
    else:
        print("HELLO",customer.name," maam")

cust=Customer("ANUPAM","male")
#print(cust.name)
greet(cust)
# class ke object koa as an argument kisi function koa de rahe hoa
# and def of greet ke customer mei aap ussai recieve kar rahe hoa.
```

```
HELLO ANUPAM  sir
```

```
class Customer:
    def __init__(self,name,gender):
        self.name=name
        self.gender=gender
def greet(customer):
    if(customer.gender=="male"):
        print("HELLO",customer.name," sir")
    else:
        print("HELLO",customer.name," maam")
    cust2=Customer("Aditi","female")
    return cust2
cust=Customer("ANUPAM","male")
newcust=greet(cust)
print("\n newcustomer greet=",newcust.name)
```

```
HELLO ANUPAM  sir

 newcustomer greet= Aditi
```

```
class Customer:
    def __init__(self,name):
        self.name=name
def greet(customer):
    print("id of customer=",id(customer))
    print("HELLO",customer.name," sir")
cust=Customer("ANUPAM")
print("id of cust=",id(cust))
greet(cust)
```

```
id of cust= 2998381155312
id of customer= 2998381155312
HELLO ANUPAM  sir
```

Aap reference bhej rahe hoa isliye same

```
class Customer:
    def __init__(self,name):
        self.name=name
def greet(customer):
    customer.name="SHEKHAR"
    print("HELLO",customer.name," sir")
cust=Customer("ANUPAM")
greet(cust)
print("cust name=",cust.name)
```

```
HELLO SHEKHAR  sir
cust name= SHEKHAR
```

<u>Aap ne kuch pass kiya hai aur aap ne function ke andar edit kar diya hai toa voa bahar bhi edit hoa jayega.</u>

```
File Edit Format Run Options Window Help
class Customer:

    def __init__(self,name):
        self.name = name

def greet(customer):
    print(id(customer))
    customer.name = "Nitish"
    print(customer.name)
    print(id(customer))

cust = Customer("Ankita")
print(id(cust))

greet(cust)

print(cust.name)
```

```
2591216043792
2591216043792
Nitish
2591216043792
Nitish
```

Class ke objects are mutable like lists,dict and sets.

Pass by refernce agar aap mutable data types bhejo ge toa original function mei change hoga but in case of immutable types original data mei change nahi hoga

```
def change(L):
    print(id(L))
    L.append(5)
    print(id(L))

L1 = [1,2,3,4]
print(id(L1))
print(L1)

change(L1)

print(L1)
```

```
/pass_by_ref.py ========
1825401957320
[1, 2, 3, 4]
1825401957320
1825401957320
[1, 2, 3, 4, 5]
```

list is mutable so original function get change after updation.

Incase of tuple no change in original

```
def change(t):
    print(id(t))
    t=t+(5,6,8)
    print("tuple inside function=",t)
    print(id(t))
t=(1,2,3)
print("original tuple=",t)
print("id=",id(t))
change(t)
print("tuple after func call done=",t)
```

```
original tuple= (1, 2, 3)
id= 2296004489472
2296004489472
tuple inside function= (1, 2, 3, 5, 6, 8)
2296004104960
tuple after func call done= (1, 2, 3)
```

COLLECTION OF OBJECTS

```
class Customer:

    def __init__(self,name,age):
        self.name = name
        self.age = age

c1 = Customer("Nitish",34)
c2 = Customer("Ankit",45)
c3 = Customer("Neha",32)

L = [c1,c2,c3]

for i in L:
    print(i)
```

```
<__main__.Customer object at 0x000002D879AF06A0>
<__main__.Customer object at 0x000002D879AF04A8>
<__main__.Customer object at 0x000002D879B8B358>
```

```
class Customer:

    def __init__(self,name,age):
        self.name = name
        self.age = age

c1 = Customer("Nitish",34)
c2 = Customer("Ankit",45)
c3 = Customer("Neha",32)

L = [c1,c2,c3]

for i in L:
    print(i.name, i.age)
```

```
Nitish 34
Ankit 45
Neha 32
```

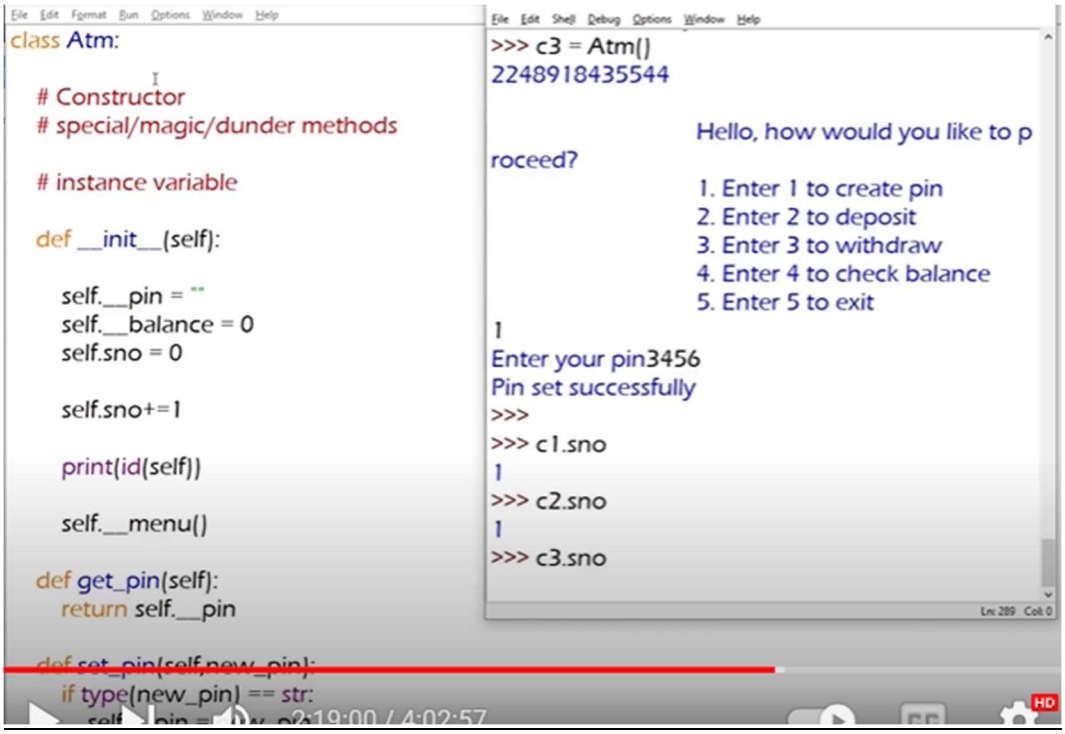
```
class Customer:

    def __init__(self,name,age):
        self.name = name
        self.age = age

    def intro(self):
        print("I am",self.name,"and I am",self.age)

c1 = Customer("Nitish",34)
c2 = Customer("Ankit",45)
c3 = Customer("Neha",32)

L = [c1,c2,c3]

for i in L:
    i.intro()
```

```
I am Nitish and I am 34
I am Ankit and I am 45
I am Neha and I am 32
>>>
```

OBJECTS ARE MUTABLE DATA TYPES IT CANT GO IN SETS LIKE IT EASILY GO IN LIST BECAUSE SETS ARE IMMUTABLE WHEREAS OBJECTS ARE MUTABLE.

STATIC (need for static)

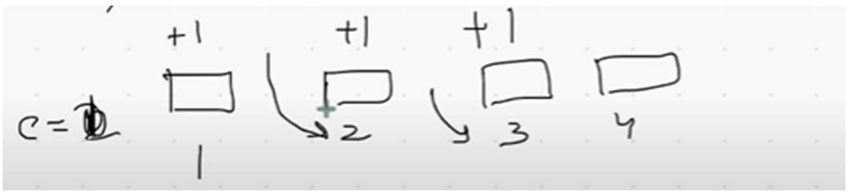
earlier



Har user ka serial no. like- 1,2,3……and so on hona chahiye.

Instance variable- ek aisa variable hai jiska value har object ke liye different hota hai.Like pin and balance

But a scenario comes when we want all variable ki value har object ke liye same hoa



iss tarah ke variable koa aap static or class ka variable kahete hoa. Eg- ifsc code same for everyone

Static variable hamesha class ke baad hote hai and constructor ke uppar hote hai.

Syntax- for accessing static variable=> classname.static variable

```
a=A()
ANUPAM- 1
b=A()
ANUPAM- 2
c=A()
ANUPAM- 3
```

```
class A:
    counter=0  #static/class
    def __init__(self):
        A.counter=A.counter+1
        self.sno=A.counter
        print("ANUPAM-",self.sno)
```

```
a=A()
ANUPAM- 1
b=A()
ANUPAM- 2
c=A()
ANUPAM- 3
b.counter()
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    b.counter()
TypeError: 'int' object is not callable
b.counter
3
a.counter
3
c.counter
3
```

A.counter also work.

memory mei counter ki value 3 hai.

Static variable tab kaam aata hai jab abke share object koa share chahiye.

Joa static method hote hai vaha ke liye humhe self pass karne ke jaroorat nahi hoti. Static method representation- @static method.inko access karne ke liye we don't need object. See getcounter and setcounter in given below example-

```
def __init__(self):
    # instance variable
    self.__pin = ""
    self.__balance = 0
    self.sno = Atm.counter
    Atm.__counter = Atm.__counter + 1

    print(id(self))

    #self.__menu()

@staticmethod
def get_counter():
    return Atm.__counter
```

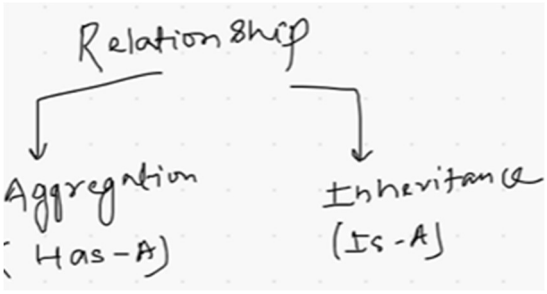
```
@staticmethod
def set_counter(new):
    if type(new) == int:
        Atm.__counter = new
    else:
        print("Not allowed")

def get_pin(self):
    return self.__pin

def set_pin(self,new_pin):
    if type(new_pin) == str:
        self.__pin = new_pin
        print("Pin changed")
    else:
        print("Not allowed")
```

2:30:53 /

Abb chaiye ki hum koi specific variable access na kar paye and can't do any changes then make that variable or function private.

STATIC METHOD IS USED WHEN WE DEAL WITH STATIC VARIABLES

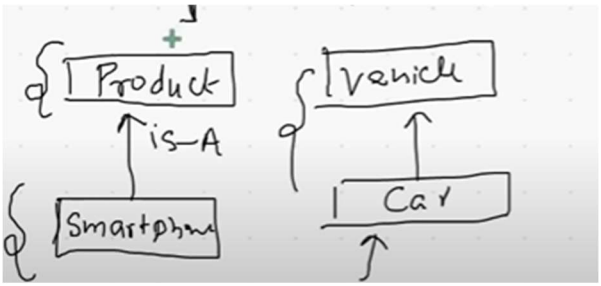
CLASS SHOWS MAINLY TWO TYPES OF RELATION i.e. 1)Aggregation 2)Inheritance



has a relationship.=>aggregation

Is a relationship=> inheritance

Vehicle ki properties car koa mil jaati hai.(inheritance)



Customer has a address.(aggregation)

ute direction mai diamond shape banana . basically jiske pass hai uski taraf banayege.

```
class Customer:

    def __init__(self,name,gender,addr
        self.name = name
        self.gender = gender
        self.address = address

class Address:

    def __init__(self,city,pincode,state):
        self.city = city
        self.pincode = pincode
        self.state = state

add = Address("Kolkata",700156,"WB"
cust = Customer("Nitish","Male",add)

print(cust.address)
```

```
aining/aggregation_demo.py ======
<__main__.Address object at 0x000001605C6904
70>
```

```
class Customer:

    def __init__(self,name,gender,addre
        self.name = name
        self.gender = gender
        self.address = address

class Address:

    def __init__(self,city,pincode,state):
        self.city = city
        self.pincode = pincode
        self.state = state

add = Address("Kolkata",700156,"WB")
cust = Customer("Nitish","Male",add)

print(cust.address.pincode)
```

```
700156
```

```
    def __init__(self,name,gender,address):
        self.name = name
        self.gender = gender
        self.address = address

    def edit_profile(self,new_name,new_city,new_pin,new_state):
        self.name = new_name
        self.address.change_address(new_city,new_pin,new_state)

class Address:

    def __init__(self,city,pincode,state):
        self.city = city
        self.pincode = pincode
        self.state = state

    def change_address(self,new_city,new_pin,new_state):
        self.city = new_city
        self.pincode = new_pin
        self.state = new_state

add = Address("Kolkata",700156,"WB")
cust = Customer("Nitish","Male",add)
print(cust.address.pincode)
```

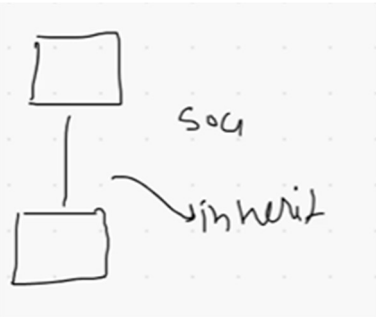

INHERITANCE

. Is a real world concept.

.Reusability

. don't repeat urself for good thing

Apki papa ki property apki hai.



parents ki some property bacche mei bhi aati hai.

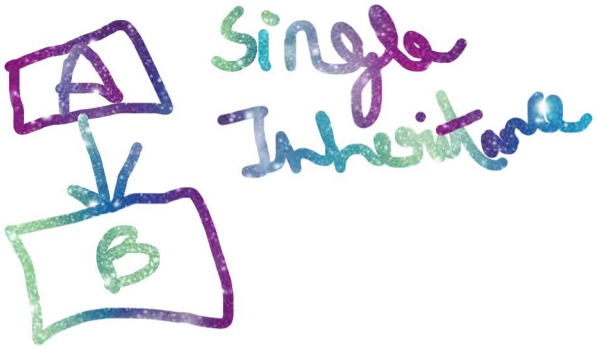
What we can inherit

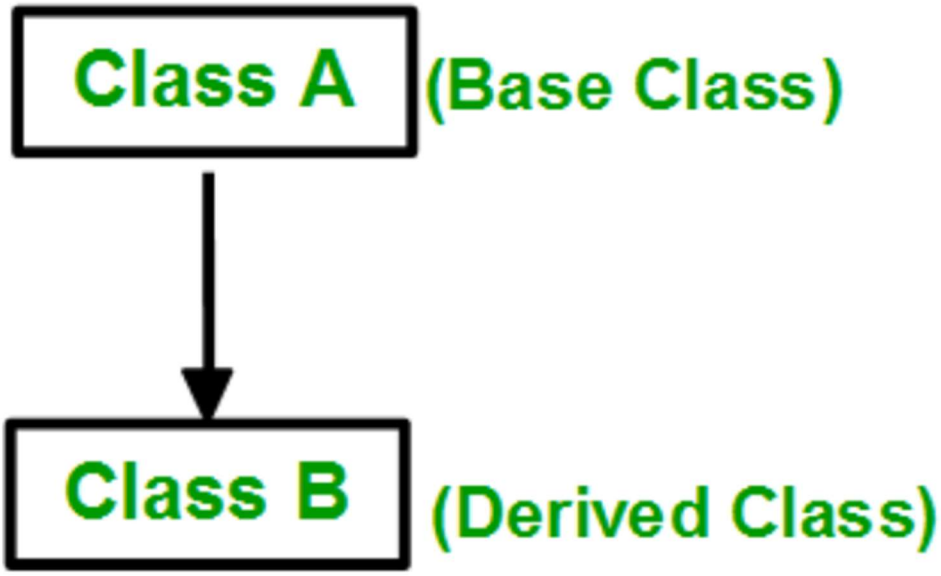
1. Data members 2. Member function 3.constructor
   Remember-> But we can't inherit private members.

STUDENT INHERIT USER

```python
class User:
    def login(self):
        print("Login")
    def register(self):
        print("register")
class Student(User):
    def enroll(self):
        print("ENROLL")
    def review(self):
        print("review")

s=Student()
s.enroll()
s.login()
s.register()
s.review()
```

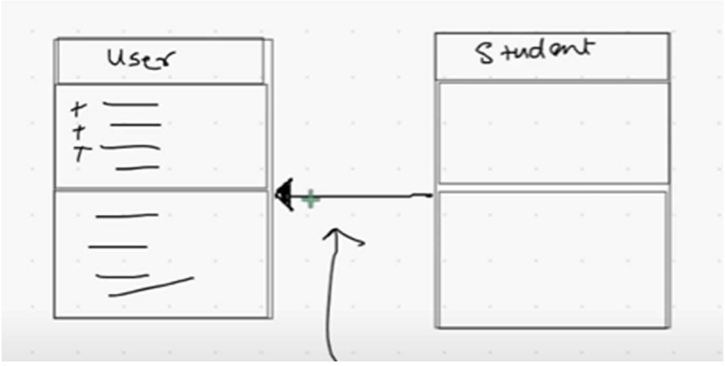
```
ENROLL
Login
register
review
```



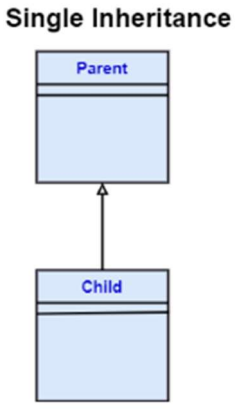


Student can access user function or data members but user can't access student methods.

CLASS DIAGRAM OF INHERITANCE



STUDENT CAN ACCESS USER



```
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.price = price
        self.brand = brand
        self.camera = camera

class SmartPhone(Phone):
    pass

s=SmartPhone(20000, "Apple", 13)
```

O/P=> Inside phone constructor

Inheriting private members (can't be done)

We made band as a private data member.

```
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.price = price
        self.__brand = brand
        self.camera = camera

class SmartPhone(Phone):
    pass

s=SmartPhone(20000, "Apple", 13)
print(s.brand)
```

error-smartphone has no attribute brand. It can't access because brand s private datamember.

```
Traceback (most recent call last):
  File "C:/Users/91842/Desktop/hit_training/inherticance_demo.py", line 12, in <module>
    print(s.brand)
AttributeError: 'SmartPhone' object has no attribute 'brand'
```

```
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.price = price
        self.__brand = brand
        self.camera = camera

class SmartPhone(Phone):
    pass

s=SmartPhone(20000, "Apple", 13)
print(s.__brand)
```

```
Traceback (most recent call last):
  File "C:/Users/91842/Desktop/hit_training/inherticance_demo.py", line 12, in <module>
    print(s.brand)
AttributeError: 'SmartPhone' object has no attribute 'brand'
```

If in 2 class both have a function with same name.then joa inherit kar raha hai voa apne mei check karega function agar hai toa voa khud ka execute karega aur parent vala joa same function hai usko execute nahi karega and vice versa.

```
class Phone:
    def __init__(self, price, brand, camera):

        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):

    def buy(self):
        print ("Buying a smartphone")

s=SmartPhone(20000, "Apple", 13)

s.buy()
```

```
Inside phone constructor
Buying a smartphone
```

This is known as method overriding.

Polymorphism contain 3 things- 1. Method overriding 2. Method overloading 3. Operator overloading.

Method Overloading	Method Overriding
1. Defining multiples methods in same class with different parameters.	1. Defining same methods in different class with same parameters.
2. Method Signature is different.	2. Return type + Method Signature is same.
3. Checked at compile time.	3. Checked at run time.
4. Also called as Compile time polymorphism/Early binding/Static binding	4. Run time polymorphism/Late binding/Dynamic binding.
5. May or may not need inheritance.	5. Must need inheritance.

Anil Jatale

EXAMPLES OF PREVIOUS STUDY CONCEPT

```
class Parent:

    def __init__(self,num):
        self.__num=num

    def get_num(self):
        return self.__num

class Child(Parent):

    def show(self):
        print("This is in child class")

son=Child(100)
print(son.get_num())
son.show()
```

```
100
This is in child class
>>>  3:05:48 / 4:
```

```
class Parent:

    def __init__(self,num):
        self.__num=num

    def get_num(self):
        return self.__num

class Child(Parent):

    def __init__(self,val,num):
        self.__val=val

    def get_val(self):
        return self.__val

son=Child(100,10)
print("Parent: Num:",son.get_num())
print("Child: Val:",son.get_val())
```

O/P-

```
Traceback (most recent call last):
  File "C:/Users/91842/Desktop/hit_training/inherticance_demo.py", line 18, in <module>
    print("Parent: Num:",son.get_num())
  File "C:/Users/91842/Desktop/hit_training/inherticance_demo.py", line 7, in get_num
    return self.__num
AttributeError: 'Child' object has no attribute '_Parent__num'
```

Error occur Because child ka constructor execute kar gaya toa parent ka constructor execute hi nahi hua toa self._num ki value hi nahi mili ussai.

```
File Edit Format Run Options Window Help
class A:

    def __init__(self):
        self.var1=100

    def display1(self,var1):
        print("class A :", self.var1)

class B(A):
    def display2(self,var1):
        print("class B :", self.var1)

obj=B()
obj.display1(200)
```

O/P=>CLASS A:100

Super keyword- when we use super keyword then it means we can access parent methods

Super keyword doesn't work outside the class

```
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):

    def buy(self):
        print ("Buying a smartphone")
        super().buy()

s=SmartPhone(20000, "Apple", 13)

s.buy()
```

```
Inside phone constructor
Buying a smartphone
Buying a phone
```

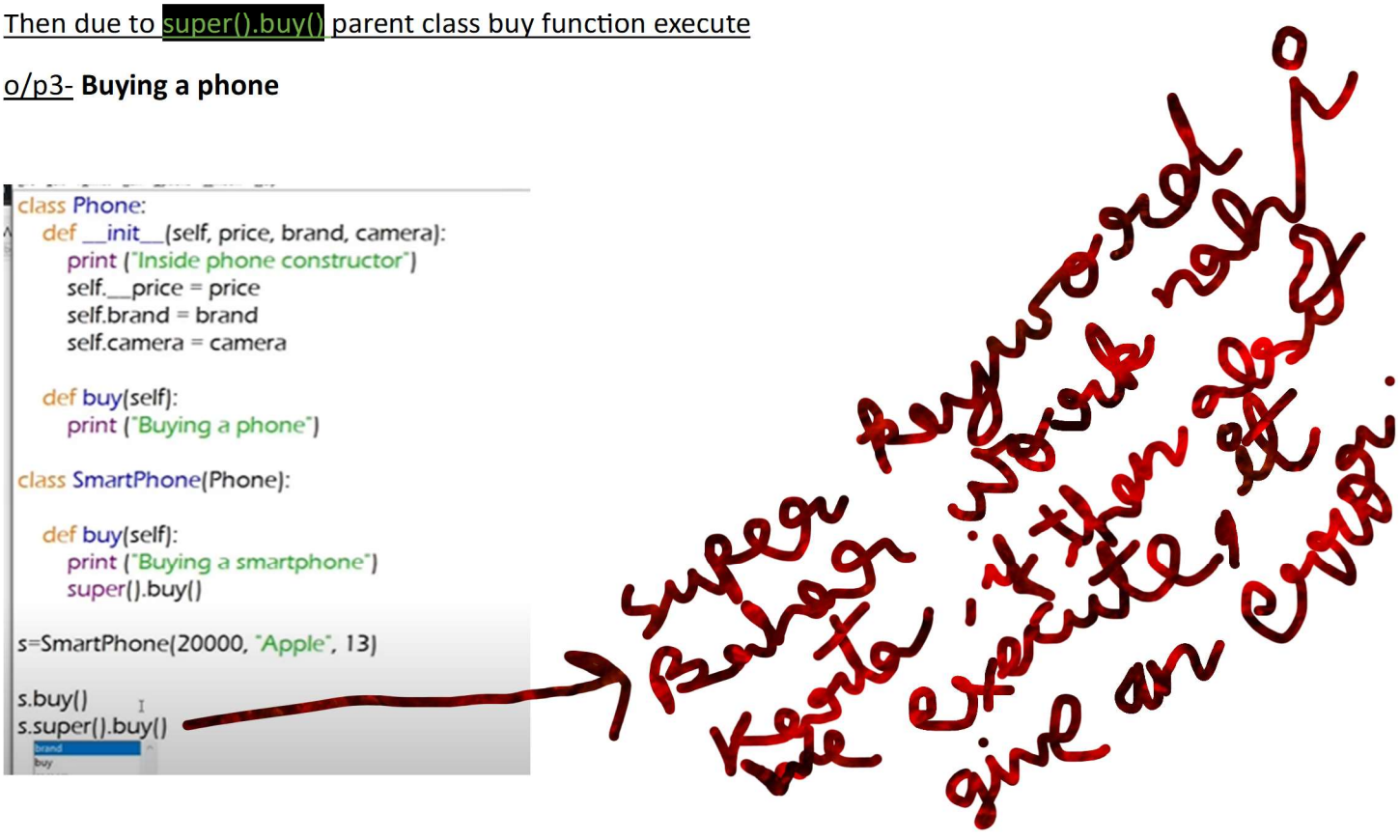
First of due to s.buy() due to method overriding

o/p2- **Buying a smartphone**

Then due to super().buy() parent class buy function execute

o/p3- **Buying a phone**

```
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):

    def buy(self):
        print ("Buying a smartphone")
        super().buy()

s=SmartPhone(20000, "Apple", 13)

s.buy()
s.super().buy()
```




```
s.super().buy()
AttributeError: 'SmartPhone' object has no attribute 'super'
```

Super keywords doa cheej access hoa sakti hai parents ka method and parent ka constructor.

Examples of super keyword

```
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

class SmartPhone(Phone):
    def __init__(self, price, brand, camera, os, ram):
        super().__init__(price, brand, camera)
        self.os = os
        self.ram = ram
        print ("Inside smartphone constructor")

s=SmartPhone(20000, "Samsung", 12, "Android", 2)

print(s.os)
print(s.brand)
```

```
Inside phone constructor
Inside smartphone constructor
Android
Samsung
```

```
class Parent:

    def __init__(self,num):
        self.__num=num

    def get_num(self):
        return self.__num

class Child(Parent):

    def __init__(self,num,val):
        super().__init__(num)
        self.__val=val

    def get_val(self):
        return self.__val

son=Child(100,200)
print(son.get_num())
print(son.get_val())
```

```
100
200
```

Super keyword ka call constructor ke andar first statement hona chahiye.

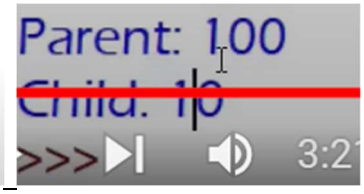
```
class Parent:

    def __init__(self):
        self.num=100

class Child(Parent):

    def __init__(self):
        super().__init__()
        self.var=200

    def show(self):
        print(self.num)
        print(self.var)

son=Child()
son.show()
```

o/p-

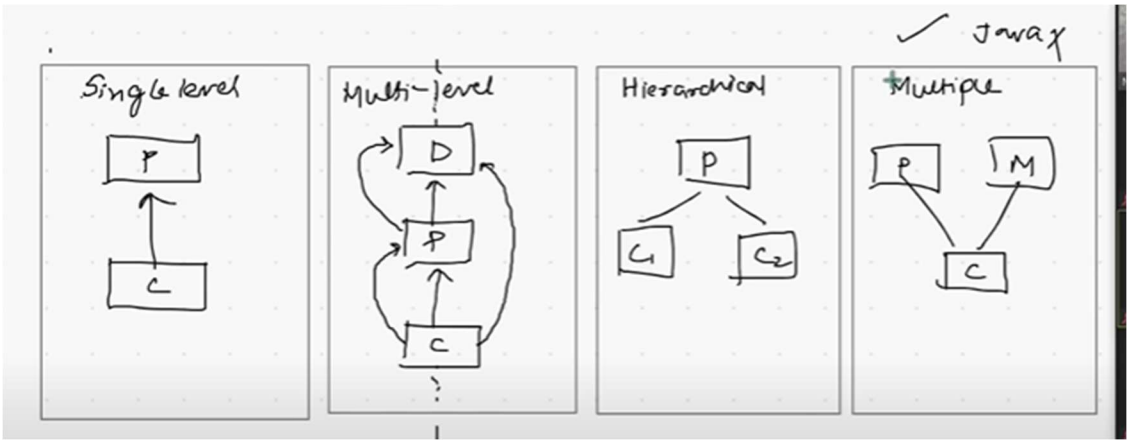
100

200

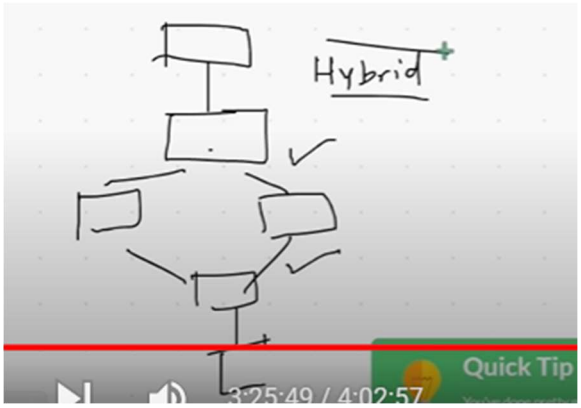
```
class Parent:
    def __init__(self):
        self.__num=100

    def show(self):
        print("Parent:",self.__num)

class Child(Parent):
    def __init__(self):
        super().__init__()
        self.__var=10

    def show(self):
        print("Child:",self.__var)

dad=Parent()
dad.show()
son=Child()
son.show()
```



TYPES OF INHERITNCE



Hybrid inheritance is a combination of any above 4 inheritance.



Multilevel inheritance

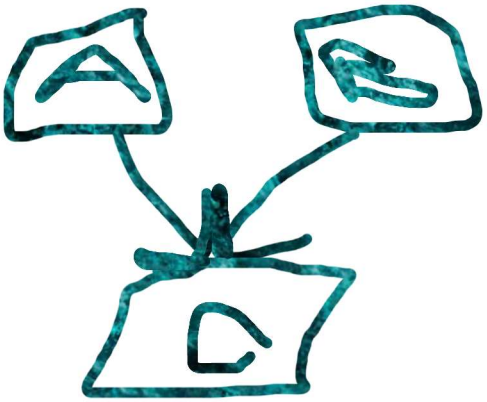
```
class Product:
    def review(self):
        print ("Product customer review")

class Phone(Product):
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class SmartPhone(Phone):
    pass

s=SmartPhone(20000, "Apple", 12)
p = Phone(1000,"Samsung",1)

s.buy()
s.review()
p.review()
```

```
Inside phone constructor
Inside phone constructor
Buying a phone
Product customer review
```

Hierarchical inheritance

```
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

    def return_phone(self):
        print ("Returning a phone")

class SmartPhone(Phone):
    pass

class FeaturePhone(Phone):
    pass

SmartPhone(1000,"Apple","13px").buy()
```

MULTIPLE INHERITANCE



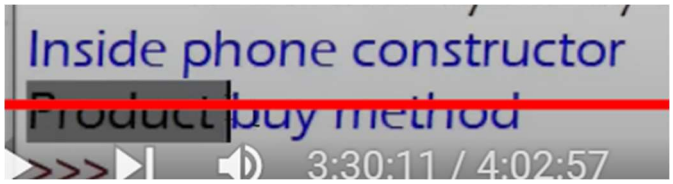
```
Inside phone constructor
Buying a phone
Customer review
```

```
class Phone:

    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class Product:
    def review(self):
        print ("Customer review")

class SmartPhone(Phone, Product):
    pass

s=SmartPhone(20000, "Apple", 12)

s.buy()
s.review()
```

MRO (METHOD RESOLUTION ORDER)

```
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class Product:

    def buy(self):
        print ("Product buy method")

class SmartPhone(Product, Phone):
    pass

s=SmartPhone(20000, "Apple", 12)

s.buy()
```

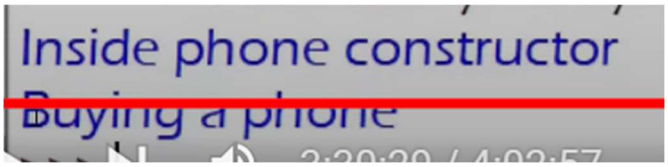
o/p-



Product and phone class dono mei buy method then question is which buy will get executed . toa iska answer hai ki product vala execute hoga i.e. joa child class ke bracket mei pahele likha hogab uska execute karega.

Agar aap phone pahele likhte then phone vala buy execute hota.

```
class Phone:
    def __init__(self, price, brand, camera):
        print ("Inside phone constructor")
        self.__price = price
        self.brand = brand
        self.camera = camera

    def buy(self):
        print ("Buying a phone")

class Product:

    def buy(self):
        print ("Product buy method")

class SmartPhone(Phone,Product):
    pass

s=SmartPhone(20000, "Apple", 12)

s.buy()
```



Jis order mei inherit karege ussai priority decide hoga and vahi execute hoga . this is called MRO.

EXAMPLES OF STUDIED CONCEPTS

```
class A:

    def m1(self):
        return 20

class B(A):

    def m1(self):
        return 30
    def m2(self):
        return 40

class C(B):

    def m2(self):
        return 20

obj1=A()
obj2=B()
obj3=C()
print(obj1.m1() + obj3.m1()+ obj3.m2())
```

20+30+20=70 o/p

```
class A:

    def m1(self):
        return 20

class B(A):

    def m1(self):
        val=super().m1()+30
        return val

class C(B):

    def m1(self):
        val=self.m1()+20
        return val

obj=C()
print(obj.m1())
```

maximum recursion depth exceeded error

Infinite time run- because no breaking conding

Remember-Joa kaamobject bahar karega vahi kaam self class ke andar karega

METHOD OVERLOADING – ek method alag alag tarah se implemented hai.

Technically speaking – method overloading doesn't work in python

par aggar jabarjasti karna hai then neeche diye gaye tarike se hoa jayega.

```
File Edit Format Run Options Window Help
class Geometry:

    def area(self, a,b=0):
        if b==0:
            print("Circle",3.14*a*a)
        else:
            print("Rect",a*b)

obj = Geometry()
obj.area(4)
obj.area(4,5)
```

OPERATOR OVERLOADING

+-> means add two number but agar + ko use kar ke hum koi aur operation kar rahe hai then it is operator overloading.

Similarly in case of +,-,*,/

```
>>> "Hello" + "world"
'Helloworld'
```

```
>>> x = Fraction(3,4)
>>> y=Fraction(5,6)
>>>
>>> print(x + y)
38/24
```

this is not a mathematical addition this is fraction addition

Operator overloading is done using magic methods.