OBJECT ORIENTED PROGRAM

ADVANCE PYTHON

oops:-

class

properties

constructor

method

object

'''

class Add:

a:int

b:int

c:int

constructor():

a=10

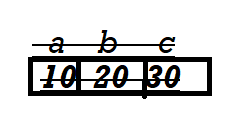
b=20

def add():

c=a+b

def show():

print(c) '''



class Add:

a:int

b:int

c:int

def sum(a,b):

c=a+b

return c

a=10

b=20

al=Add()

c=al.sum(a,b)

print(c)

output====30

====object oriented programing system====

class Bank:

acno:str

name:str

mob:int

bal:int

pin:int

b=Bank()

b.acno='sbi123'

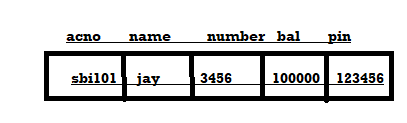
b.name='tani'

b.mob=7000

b.bal=234456

b.pin=12340

print(b.acno," ",b.name," ",b.mob," ",b.bal," ",b.pin)



oops:---

class

object

inheritance

abstraction

encapsulation

polymorphism

class:---class is a blueprint of an object. class contenet data member and member function and we can accsess data of class with the help of an object.

object:---object is instence of class or we can say that its a memory in whitch we can store data member or member function and we can access data of class using object (with the help of an object.)

properties:---data member's or glober variables whitch is declared inside the class is known as properties of an object.we can access properties outside the class using (.) member access opretor.

class Emp:

empno:int

name:str

e=Emp()

e.empno=101

e.name='tani'

print(e.empno," ",e.name)

output:---101 tani

CONSTRUCTOR

Constructor are those member function there name start with \_init\_ it will automatically called where we will create an object of class.

Constructor is mainlly used to initilize properties of an object .As we now Constructor is member function of class. and if want to call a member function of class using object then we will pass self in first argument of no function.

In python we will use two type of constructor.

1.. DEFAULT CONSTRUCTOR :-----

Default constructor is a constructor that content only self constructor and when we will not write any constructor in python class.then python compiler will automatically define default constructor.

1. program

class Emp:

def \_init\_(self):

print("default constructor")

e=Emp()

Output:-------dafault constructor

2.. PARAMETERIZE CONSTRUCTOR:------

A parameterize constructor is constructor that content parameter's other then self.

2. program

class Emp:

def\_init\_(self,a):

print("welcome",a)

e=Emp("ajay")

e1=Emp("jay")

Output:-------------welcome ajay

wlecome jay

===CONSTRUCTOR OVERLOADING====

OR COMPILE TIME POLYMORPHISM=======

we can write more than are constructor .with difference arguments within a python class this process is called constructor overloading But when we will put two constroctor with same argument then last constructor will execute only this process is called polymorphism.

class Emp:

def\_init\_(self,a,b):

print("welcome",a,b):

def\_init\_(self):

print("default constructor")

def\_init\_(self,a):

print("welcome",a)

#e=Emp() #error

e1=Emp("jay")

output:------

welcome jay

\* args non key arguments

\* args keyword is mainly used to achive overloading in python.

when we will pass \* args in argument of any member function then it will internally create a list in witch we can store multiple argument.

class Emp:

def\_init\_(self,\*args):

if ien(args)==0:

print("default construcor is running")

else:

for i in args:

print(i,end=" ")

print()

e=Emp()

e1=Emp("jay")

e2=Emp("jay","singh")

output:------

default constructor is running

jay

jay singh

======== \*\*(star) args====

\*\* args contens key arguments it is also use to achive overloading.

In \*\*args it will store data internally within a dictionary .

We can pass argument values based on key .At the time of calling of any member function.

Class Emp:

Def\_\_init\_\_(self,\*\*args):

if len(args)==0:

print(“default constructor is running”)

else:

print(args)

e1=Emp(name=”ajay”)

e2=Emp(name=”ajay”,age=25,adddress=”indore”)

output:=======

{‘name’:’ajay’}

{‘name’:’ajay’,’age’:25,’address’:’indore’}

====METHOD MEMBER FUNCTION=====

Method or memberfunction of class which we define under the class. In python we will use two type of methods.

1.. statics method 2… nonstatic method

1..static method:---- static method is a method which is define under the class that doesn’t content self in argument .we can call static method of class with the help of class name.method name.

2…..non-static:--- non-static method of class that content self in argument .we can call a not-static method of class with the help of object.

Class Demo:

def display():

Print(“static display id running”)

def show(self):

print(“non-static show is running”)

d=Demo()

#d.display()

d.show()

#Demo.show()

Demo.display()

Output:----

Non-static is running

Static display is running

( d.display() is error,Demo.show() is error )

# Method Overloading🡪we can write more than are constructor .with difference arguments within a python class this process is called method overloading But when we will put two constroctor with same argument then last constructor will execute only this process is called polymorphism.

Class Demo:

Def display(self,\*args):

Print(args)

d=Demo()

d,display(10)

d,display(10,20)

d.display(10,20,30)

output---🡪(10) (10,20) (10,20,30)

#IMP🡪 if we want to use property of an object or globel variable of class in any member function (constructor,method) than we will use self keyword to access globle variable .

Class Emp:

enpno:int

ename:str

job:str

sel:int

def\_\_init\_\_(self,empno,ename,job,sel)

self.empno=empno

self.enmae=name

self.job=job

self.sal=sal

def show(self):

print(self.empno,self.ename,self.job,self.sal)

e=Emp(101,”ajay”,”manager”,2500)

e.show()

################################################3

Selfkeyword:-- self is a keyword which is usually pass as a first argumnent of any

Member function which is mainly used to store carrent object throw which we will call a function.

class Emp:

empno:int

name:str

def show(self):

#print(self)

Self.empno=102

e=Emp()

e1=Emp()

print(e)

print(e1)

#e.show()

e1.show()

e.empno(0)

print(e.empno)

print(e1.empno)

############################################################################

============= INHERITANCE=========

Inheritance is use to achive is a relationship of python.we can access or reuse data members and member function of one python class in anthor class.

With the help of Inheritance.

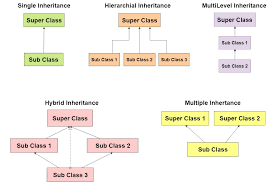
In python we will use 5 types of inheritance.

1.single inheritance

2.. Multilevel inheritance

3.. hieracichal inheritance

4. hybrid inheritance 5.multiple inheritance



We will use syntax:==

@1 class Derived\_class\_name(Base\_class\_name);

Body

Class Calculator:

def add(self,a,b):

Print(“add is ”,a+b)

def sub(self,a,b):

print(“sub is”,a-b)

def mul(“mul is”,a\*b):

print(“mul is”,a\*b)

def div(“div is”,a/b):

print(“div is”,a/b)

class Demo(Calculator):

def show(self):

print(“\*\*\*\*\*calculator\*\*\*\*\*\*”)

d=Demo()

d.show()

d.add(10,20)

d.sub(20,30)

d.mul(10,20)

d.div(20,2)

#SIMPLE INHERITENCE===🡺we will write a class in modul we can import the class from module and we will reverse in other class using inheritence.

Calc.py-----

Class Calculator:

def add(self):

Print(“add is ”,a+b)

def sub(self,a,b):

print(“sub is”,a-b)

def mul(“mul is”,a\*b):

print(“mul is”,a\*b)

def div(“div is”,a/b):

print(“div is”,a/b)

inheritance.py------

from Calc import Calculator

class Demo(Calculator):

def show(self):

print(“\*\*\* CALCULATOR\*\*\*”)

d=Demo()

d.add(10,20)

d.sub(20,10)

d.mul(10,20)

d.div(20,2)

======== Multilevel Inheritence==================

Class Calculator:

def add(self,a,b):

print(“add is”,a+b)

def sub(self,a,b):

print(“sub is”,a-b)

def mul(self,a,b):

print(“mul is”,a\*b)

def div(self,a,b):

print(“div is”,a/b)

class Scalculator (calculator):

def cube(self,n):

print(“cube is”,n\*\*3)

def square(self,n):

print(“square is”,n\*\*2)

def factorial(self,n):

f=1

for i in range(1,n+1):

f=f\*i

print(“factorial of”,n,”is”,f)

class Demo(scalculator):

def show(self):

print(“\*\*\*\*CALCULATOR \*\*\*\*\*”)

d=Demo()

d.show()

d.cube(5)

d.square(5)

d.factorial(5)

d.add(10,20)

d.sub(20,10)

d.mul(10,20)

d.div(20,2)

OUTPUT🡪

\*\*\*\*\*CALCULATOR\*\*\*\*\*

cube is 125

square is 25

factorial of 5 is 120

#imp for read note🡪if class different hogi to method same possible hai.

================HIERARCHIAL==============

class Base:

def show(self):

print(“show is running”)

class Demo(Base):

def display(self):

print(“display is running from Demo”)

class Dummy(Base):

def display(self):

print(“display is running from dummy”)

d=Demo()

d.show()

d.display()

d1.Dummy()

d1.show()

d1.display()

OUTPUT=🡺

Show is running

display is running from Demo

show is running

display is running from dummy

=============MULTIPLE CLASS=============

Class Base1:

def show(self):

print(“show is running”)

class Base2:

def display(self):

print(“display is running from demo”)

class Dammy(Base1,Base2):

def welcome(self):

print(“welcome to dummy”)

d=Dummy()

d.welcome()

d.Display()

d.show()

==============HYBRID=============

Class Base:

def getData(self):

print(“get data from Base”)

class Base1(Base):

def show(self):

print(“show is running”)

class Base2(Base):

def display(self):

print(“display is running from Demo”)

class Dummy(base1,base2):

def welcome(self):

print(“welcome to dummy”)

d=Dummy()

d.welcome()

d.display()

d.show()

output==🡺welcome to dummy

display is running from Demo

show is running

################AMBIGUITY PROBLEM #########################3333

If two parent classes contents same method and with the help of multiple inheritance .we want the call same method using child class then compiler will only one method instead of two method.

This problem is call Ambiguity problem because method is ambiguity define.

Class Base:

def show(self):

print(“show is running from base”)

Class dummy:

def show(self):

print(“show is running from Dummy”)

class Demo(Base,Dummy):

def display(self):

print(“display is running”)

d=Demo()

d.display()

d.show()

output-🡪display is running

show is running from base.

########################################################################3

=========================AGGRIGATION====================

We can access data member of member function of one python class in another python class with the help of an object using aggrigation.

Class Dummy:

def show(self):

print(“show is running from Dummy”)

class Demo(Dummy):

a=Dummy() #aggrigation

def display(self):

# a=Dummy() # aggriagation

# a.show()

self.a.show()

print(“display is running”)

d=Demo()

d.display()

output🡪show is running from dummy display is running

#########################################################333

===========METHOD OVERRIDING and polymorphism================

IF Parent and child class both contents same method with same signature and same parameter and then we will call same method using child class object so at runtime parent class method will replaced by child class method.

This process is called method overriding or runtime polymorphism.

Class Dummy:

def show(self):

print(“show is running from Dummy”)

def display(self):

print(“show is running from Dummy”)

class Demo(Dummy):

def show(self):

print(“show is running from Demo”)

d=Demo()

d.show()

d.display()

output-🡪

show is running from Demo

display is running from Dummy

IMP NOTES-🡪function k under kissi bhi globel variable ko access krne k liye self lagta hai.

# class alag ho to data same ho skta hai.!

# apni class me kisi or class ka object bnay ise aggregatin kahte hai!

====collection with user defined===========

class Emp:

empno:int

ename:str

job:str

def \_\_init\_\_(self,empno,ename,job):

self.empno=empno

self.ename=ename

self.job=job

def show(self):

print(self.empno,self.ename,self.job)

e1=Emp(101,'jay','manager')

e2=Emp(102,'ajay','developer')

e3=Emp(103,'jaya','clerk')

al=[e1,e2,e3]

for i in al:

i.show()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//////////////////\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Emp:

empno:int

ename:str

job:str

sal:int

def \_\_init\_\_(self,empno,ename,job,sal):

self.empno=empno

self.ename=ename

self.job=job

self.sal=sal

def show(self):

print(self.empno,self.ename,self.job,self.sal)

al=[]

s=int(input("enter total no of student"))

for i in range(1,s+1):

print("enter detail for emp",i)

eno=int(input("enter empno"))

enm=input("enter name")

ejob=input("enter job name")

esal=int(input("enter sal"))

e=Emp(eno,enm,ejob,esal)

al.append(e)

print("student detail are")

for i in al:

i.show()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_////////////////////////////\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Emp:

empno:int

ename:str

job:str

sal:int

def \_\_init\_\_(self,empno,ename,job,sal):

self.empno=empno

self.ename=ename

self.job=job

self.sal=sal

def show(self):

print(self.empno,self.ename,self.job,self.sal)

al=[]

s=int(input("enter total no of student"))

for i in range(1,s+1):

print("enter detail for emp",i)

eno=int(input("enter empno"))

enm=input("enter name")

ejob=input("enter job name")

esal=int(input("enter sal"))

e=Emp(eno,enm,ejob,esal)

al.append(e)

print("student detail are")

for i in al:

if i.sal>5000:

i.show()

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

class Emp:

empno:int

ename:str

job:str

sal:int

def \_\_init\_\_(self,empno,ename,job,sal):

self.empno=empno

self.ename=ename

self.job=job

self.sal=sal

def show(self):

print(self.empno,self.ename,self.job,self.sal)

al=[]

s=int(input("enter total no of student"))

for i in range(1,s+1):

print("enter detail for emp",i)

eno=int(input("enter empno"))

enm=input("enter name")

ejob=input("enter job name")

esal=int(input("enter sal"))

e=Emp(eno,enm,ejob,esal)

al.append(e)

print("student detail are")

for i in al:

if i.job==”manager”:

i.show()

#############################################################################

=========SUPER FUNCTION===========

Super is a function which is mainly used to access immediate parent class data members or member function.

=======================================================

=========SUPER CLASS WITH CoNSTRuCTOR=========

We can class parent class with constructor from child class constructor with the help of super function.

class Emp:

def \_\_init\_\_(self):

print("default constructor of emp")

class RegEmp(Emp):

def \_\_init\_\_(self):

print("default constructor of RegEmp")

super().\_\_init\_\_()

e=RegEmp()

output--🡪

default constructor of RegEmp

default constructor of emp

########################################33333

class Emp:

empno:int

ename:str

def \_\_init\_\_(self,empno,ename):

self.empno=empno

self.ename=ename

class RegEmp(Emp):

def \_\_init\_\_(self,empno,ename):

super().\_\_init\_\_(empno,ename)

def show(self):

print(self.empno," ",self.ename)

e=RegEmp(101,'veer')

e.show()

output-🡪

101 veer

==============super function with method=============

If we want to call parent class method from child class method . we will use super function in case of method overriding .if you want to parent class as well as child class method we will use super function in child class method to class parent class method.

class Dummy:

def show(self):

print(“show is running from dummy”)

class Demo:

def show(self):

print(“show is running from Demo”)

super().show()

d=Demo()

d.show()

output-🡪

show is running from Demo show is running from Demo

=============Accese modifir=============

Access Modifier’s are mainly used to decide Accessiblity .And life time of data member’s or member functions of python class.

In python modifiers three tpe of access modifires.

1}🡪public 2}🡪private 3}🡪protected

But adject implementation of protected not supported by python.

Imp note🡪

Protected variables are those data members of a class that can be accessed within the class and the classes derived from that class. In Python, **there is no existence of** “Public” instance variables. However, we use underscore '\_' symbol to determine the access control of a data member in a class.

VARIABLE🡪 Public access within class ,within child class ,outside class.

\_variable🡪 protected access within class ,within child class, not outside.

\_\_variable🡪 private access within class , not within child class, not outside c class.

===========outside class access==========

Class Dummy:

a=10 #public

\_b=20 #protected

\_\_c=30 #private

d=Dummy()

print(d.a)

print(d.\_b)

print(d.\_\_c)

outside🡪

10

20

#2🡪

class Dummy:

a=10

\_b=20

\_\_c=30

def show(self):

print(self.a,” ”,self,\_b,” ”,self.\_\_c)

d=Dummy()

d.show()

=================within child class==================

class Dummy:

a=10 #public

\_b=20 #protected

\_\_c=30 #private

class Demo(Dummy):

def show(self)

print(self.a, “ ”,self. \_b,” ”,self.c)

d=Demo()

d.show()

====================encapsulation================

Wrapping or binding Data of and function of function in single unit is called encapsulation we can achieve encapsulation and we will define private protected with in class and we will write setter public methods to set values in property’s and getter public method to get values of property.

class Emp:

\_empno :int

\_\_ename:str

def set Empno(self,empno):

self. \_\_empno=empno

def set Ename(self,ename):

self. \_\_ename = ename

def get Empno(self):

return self.\_ \_\_empno

def get Ename (self):

return self.\_ \_\_ename

e=Emp()

e.setEmpno(101)

e.setEname(“jay”)

print(e.getEmpno(),” ”,e.getEname())

output🡪

101 jay

######################################################

=============DECORATOR===================

Decorator are one of the most powerful concept of python to which can optimise behaviour of a pyhton function . we can define a decorator with the help of a(@) addtherate symbol. When we will use a decorator before any function. That the function will perform same extra operation’s Before decorater we will use some imp concept related to function like

1. We can pass a function in argument of another function.
2. We can read a function as object.
3. A function can return another function.

def show(name):

return "welcome"+name

# print(show(" ajay"))

display=show

print(display("ajay"))

output🡪

welcome jay

1. -----🡪We can pass a function in argument of another function.

===🡺

def square(n):

return n\*\*2

def cube(n):

return n\*\*3

def factorial(n):

f=1

for i in range(1,n+1):

f=f\*i

return f

def show(fun):

print(fun(5))

show(square)

show(cube)

show(factorial)

output🡪

25

125

120

================%================%============%================

def outer():

def innner():

print("inner function is called")

return innner

temp=outer()

temp()

output🡺

inner function is called

===============%=============%=================%===============

(I)def programmer(fun): #aargument m bhi function denge

def inner():

print("welcome to programmer point")

fun()

print("bye bye")

return inner

#then compile krke chod do

(II)def tanvi():

print("hi i ma tanvi learning python")

temp=programmer(tanvi)

temp()

(III)

@programmer

def tanvi():

print("hi i ma tanviiii learning python")

tanvi()

output==🡺

welcome to programmer point

hi i ma tanviiii learning python

bye bye

In [ ]:

===========ABSTRACTION AND ABSTRACT CLASS=================

ABSTRACTION IS A PROCESS IN WHICH WE WILL SHOW ONLY SPACIAL FEATURE AND HIDE BACKGROUND WE CAN ACHIVE ABSTRACTION IN PYTHON BY INHERITING abc CLASS ABC modle. If we want to construct a abstract class in python then we will create a abstact method in class . we can create a abstract method decorater and if a class that content atlest one abstract method decorater class .If any class will inherit a abstract class will inherit a abstract class then we will overwrite all the abstract method of abstract class in child class.Otherwise child class also we cant create considerd as , we cant create a object of abstract class.

from abc import ABC,abstractmethod

class Rbi(ABC):

def rules(self):

print("rules by Rbi")

@abstractmethod

def deposit(self):

pass

class Sbi(Rbi):

def welcome(self):

print("welcome to sbi")

def deposit(self):

print("you can deposite in Sbi")

#r=Rbi()

s=Sbi()

s.welcome()

s.rules()

s.deposit()

output🡺

welcome to sbi

rules by Rbi

you can deposite in Sbi

=====================final class=========================

FINAL CLASS IS AN IMP CONCEPT IN PYTHON WHICH WE WILL MAKE A CLASS US FINAL WHEN WE WILL DEFINE A CLASS AS FINAL CLASS AS FINAL WE CANT INHERIT THAT FINAL CLASS.IF WE WANT TO MAKE A CLASS AS FINAL .WE WILL USE FINAL DECORATOR BUT IF WE WANT TO USE FINAL DECORATOR TO WE WILL INSTALL A ENTERNAL PACKAGE BY USING FOLLOWING PIP COMMOMD.

# PIP🡺

PACKAGE INSTALLER PYTHON

OR

PYTHON PACKAGE INSTALLER

PIP INSTALL FINAL\_CLASS.

WE CAN USE FINAL CLASS CONCEPT USE AFTER FINAL CLASS.

from final\_class import final

@final

class Dummy:

def show(self):

print("show is running")

def display(self):

print("dispaly is running")

class Demo(Dummy):

def welcome(self):

print("welcome is running")

d=Demo()

d.welcome()

d.show()

d.display()

OUTPUT🡺

**TypeError**: Subclassing final classes is restricted

WHEN WE WILL MAKE CLASS FINAL THEN NO ONE CAN OVERRIDE CONCEPT.

from final\_class import final

from final\_class import final

@final

class Dummy:

def show(self):

print("show is running from Dummy")

def display(self):

print("dispaly is running")

class Demo(Dummy):

def show(self):

print("show is running from Demo")

def welcome(self):

print("welcome is running")

d=Demo()

d.welcome()

d.show()

d.display()

output🡺

**TypeError**: Subclassing final classes is restricted

Note🡺final class ka agrigation possible hai!

from final\_class import final

@final

class Dummy:

def show(self):

print("show is running from Dummy")

def display(self):

print("dispaly is running")

class Demo:

d1:Dummy

def show(self):

print("show is running from Demo")

d1=Dummy()

d1.show()

d1.display()

def welcome(self):

print("welcome is running")

d=Demo()

d.welcome()

d.show()

output🡺

welcome is running

show is running from Demo

show is running from Dummy

dispaly is running

##python iteration#################

Iteration is an object which is used to itert iterable objects like list,set,tuple,dict etc.

Iterototer object inishlize using iter method and we will use next method for itration when data will not found itrable that it will raise Stopiteration.

a=['java','python','angular','react','oracle']

try:

t=iter(a)

print(next(t))

print("our data")

print(next(t))

print("our data")

print("our data")

print("our data")

print(next(t))

print("our data")

print(next(t))

except stapIteration:

print("element end")

output🡪

java

our data

python

our data

our data

our data

angular

our data

react

##############Custom Iterator class ###########

In python we will also creater custom Iterator in which we will define \_\_iter\_\_() method and \_\_next\_\_() method and we will raise Stop iteration exception.

class Myter:

def \_\_iter\_\_(self):

self.roll=101

return self

def \_\_next\_\_(self):

if self.roll==105:

raise StopIteration()

else:

self.roll=self.roll+1

return self.roll

m=MyIter()

t=iter(m)

print(next(t))

print(next(t))

print(next(t))

print(next(t))

output🡪

101

102

103

104

##################### GENERATOR ##########################

IN WHICJ WE WILL USE YEILD KEYWORD INSTIED OF RETURN KEYWORD WE CAN CREATE CONSTRUCT GENERTOR BY USING FUNCTION INSTIED OF CLASS. IT MEANS THAT IS KNOW AS – ITER\_\_(),\_\_NEXT\_\_() METHOD.

MAJOR ADVANTAGE GENRATOR IS IT WILL AUTOMATICALLY RAISE IS STOPITERATION.

def genRoll(m):

for i in range(101,101+m):

yield i

r=genRoll(5)

print(next(r))

print(next(r))

print(next(r))

print(next(r))

print(next(r))

OUTPUT🡪

101

102

103

104

105

===========STR==========(its called magic function)

WHEN we will print a object inside print function then python interpretor enterly call \_\_str\_\_() method it will print a specially string but if we want to print some user define function then we will override \_\_str\_\_() method.

Class Emp:

Empno : int

Ename : str

Job: str

def \_\_init\_\_(self,empno,ename,job):

self.empno=empno

self.ename=ename

self.job=job

def \_\_str\_\_(self):

return str (self.empno)+” “+self.ename+” “ self.job

e=Emp(101,’ajay’,’developer’)

print(e)

output🡪

101 ajay developer

Class Emp:

Ename : str

Job: str

def \_\_init\_\_(self,ename,job):

self.ename=ename

self.job=job

def \_\_str\_\_(self):

return self.ename+” “ self.job

e=Emp(’ajay’,’developer’)

print(e)

output🡪

ajay developer

=================== \_\_DOC\_\_ =======================

Doc string is a special concept of python in which use will define one comment (‘ ‘) or(“ “) .it is also use to describe commont in a function.and we can fetch this commended data using method name . \_\_doc\_\_

def show():

“ “ “

We are learning python

Hello how are you “ “ “

Return none

print(show .\_\_doc\_\_)

outptu🡪

we are learning python

hello how are you.

* Dender wale function ko magic function bolte hen.
* Str function m int ko use krna ho to int ko str m convert krke print kr skte hen.
* Child class se parent class ko use krne k liye super class ka use hoga
* !