Python oops by durga sir

Object oriented programming

Class //plane for the building is non as class or blue print for the building

Object //the physical existence or instances of the class is object eg: building

One class is ready =multiple object

Reference variable // reference variable is always pointing to the object eg: remote is reference variable and t v is object. to change or acces any method any performance for the things

Class to object one to many

Object to reference  one to one or one to many

Class student:

‘’’ this is demo class ‘’’

This is also known as doc string

Eg: class student:

Def \_\_init\_\_(self rollnum, name):

Self.rollnum=rollnum

Self .name =name

Def talk (self)

Print(“hello my roll number is” +self. rollnum)

Print(hello my name is” +self.name)

//this hole thing is blue print or the class

//now we are going to creat the object

Student (sunny ,”101”)

// now to acces all these object we want some remote to access reference variable

S=student(sunny,’101’)

Print(s.name)

You can access the document string of the class

Eg: class student ():

‘’’ this is my first class ‘’’

Print(\_\_doc\_\_) //this is my first class

Help(student)// same output

Self variable :

Self variable is always pointing to current object

With in the python class to refer the current object we should use self variable

The self and reperance variable is pointing to the same object .the id of the both are same

When ever you creat the object for the class .the self variable will be pointing to the same object

eg Def \_\_init\_\_(self Rollnum, name)://this Roll num refer to Rollnum

Self.rollnum=Rollnum // this rollum will become 101

Self .name =name//this name will become surya

Def talk (self)

Print(“hello my roll number is” +self. rollnum)

Print(hello my name is” +self.name)

Student (sunny ,”101”)

S=student(sunny,’101’) //when ever you call student name this will execute (self rollnum,namae)

Print(s.name)

s.talk() //now this refer to talk (self) variable

as long as S is exceuting self refer to that object

as long as s.talk is excuting self reference to that object

every instance method and for constractor method the first argument should be self

we are not required for passing argument for self keyword pyhon vm is responsiable for that

to acces the current object with in the class we need some variable that variable is self variable

instant of self we can use any word self is not keyword you can use s,r,u,self any thing you like what ever the first argument your passing is default its self

**Constractor:**

**Special method in python**

the name of the constractor is always fixed \_\_init\_\_

whene ever we are creating the object the constractor will always execute automatically

we are not required to call explicitly

the main objective is to declare and initialize variable

for every object the constractor will execute only ones

every constractor should take atlest one argument

**overload constractor :**

two method name with different argument is called overload constractor

class constract:

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

print(“no-arg”)

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

print(“one arg”)

t=constractor()

t1=constract(20)

in python constractor overloading is not available

when ever you creat second constractor the first one will destroy

so if we creat the object with no argument thene we will get error

**method constractor**

1.The name can be any thing 1.The name must be \_\_init\_\_()

2.method will exceuted if we 2.will execute automatically if we

Call creat object

3.per object method will exceute 3. Per object constractor will

As many time you want only once

4.business logic 4.to declare and initize instance variable

**Inside python class**:

3 types of variable are there

Instance variable //object level variable

Static variable // class level variable

Local variable //method variable

3 Types of method

Instance method

Class method

Static method

**Instance variable :**

If value of variable wary from object to object such type is called instance variable

For every object the seperate copy will be created

Eg: name and roll number varied from object to object

S1= student(“surya” 101)

S2= student (“vishnu” 101)

S2= student (“sunny “ 600)

Where we can use instances variable ?

1.inside constructor using self

2.inside instance method using self

If you don’t call instance method than instance variable will not assign to object

Class student:

Def \_\_init\_\_(self ,rollnum, name ):

Self rollnum =num

Self name =name

Def info(self):

Self mark =60

S1=student (durga 101)

S1.info ()//if you don’t assign this value this will not print

Print(s1.\_\_dici\_\_)

3.from outside the class by using object refernaces

Class student:

Def \_\_init\_\_(self ,rollnum, name ):

Self rollnum =num

Self name =name

Def info(self):

Self mark =60

S1=student (durga 101)

S1.info ()

S1.age=20 //if age is already present it change from old variable to new variable if there is no age it will creat age variable

Print(s1.\_\_dici\_\_)

4.for every object creation sparate copy will created

Eg : from above program if you creat s2 reperance variable there will be separate copy will be created and s1 variable will not assign in s2

**How to acces instance variable :**

With in the class by using self

From ouside the class by using object reference

Eg :

Class student :

Def\_\_init\_\_(self , name,roll):

Self name =name

Self roll =roll

Def display (self):

Print(“hello my name is “, Self.name )

Print(“hello my name is “, Self.roll)

S= student (durga , 101)

s.display()

print (s.name, s.roll) //you can access instances variable by using object referances S

**How to delete the instance variable :**

Del self.variable name

Del.objectreferance.variable name

Eg: Class student :

Def\_\_init\_\_(self , name,roll):

Self name =name

Self roll =roll

Def delete(self):

Del self.name

Print(“hello my name is “, Self.name )

Print(“hello my name is “, Self.roll)

S=student(durga, 101)

T1.delete()

Del.t1.name

Static variable :

If value of variable does not varied from object to object such type of variable is should be declared at class level

For static variable for all object only one copy is crated at class level memary will be saved

Class student :

cname =’durgasoft’//static variable

Def\_\_init\_\_(self , name,roll):

Self name =name

Self roll =roll

S1=student(durga,`101)

S2=student(surya,203)

Print (S1.name,s1.roll, student.cname)

Print (S2.name,s2.roll, student.cname)

**What are the variable places are there to declare static variable ?**

1.With in the class directly but out side of any method

2.inside the constractor by using class name

3.inside the instance method by using class name

Eg: class student:

X=10

Def\_\_init\_\_(self)

Test.y=10//static in constractor

Def m1(self ):

Test.z=20//static in instant method

@classmethod //4 point

Def m2(cls):

Cls.d=40

Test.e=50

@staicmethod //5 point

Def m3():

Test.f=60

Test.g=70//6point

Print(test.\_\_dicit\_\_) //there will be only 2 staic variable will be there 1with in class and 2 outside the class coz you dident call other method of def

4.inside class method by using class name or cls variable

5.inside staic method by using class name

6.from out side of the class by using class name

**How to access the static variable:**

With in the class using class name or by object reference

With in the class by:

Class name or self or cls

Outside the class by :

Object referances, class name

**How to modify the static variable ?**

With in the class we should use

Class name ,cls variable

From outside the class :

Only class name

Class Test:

A=10

Def \_\_init\_\_(self):

Test .a=20 //modify by class name

@classmethod

Def m1 (cls):

Cls.a=30 //modify by cls variable

Test.a=40

@staticmethod

Def s1():

Test.a=50

T=student()

T1=m1()

T2=S1()

Test.a=60 // from out side of the class

t.a=70 //new instance varirble is going to come

Print (Test.a)

Print(t.a)

How to delete the variable

With in the class by using del

Class A :

a=10

Def \_\_init\_\_(self):

Del A.a

Print(A.\_\_dicit\_\_)

Obj=A()

Print(A.\_\_dicit\_)

A.a

Highly recomanded to use instance variable by object referances

And static vairiable by class name

**Local variable :**

Method level vatible

We should not use self ,cls ,classname

Eg:

Class Test :

Def m1(self):

A=10

Print(a)

Def m2 (self):

B=20

Print(a) //we cant access local variable of another method

But we can return the local variable but if didn’t return we cant access

Print(b)

T=test()

t.m1()

t.m2()

eg:2

class Test:

def average(self ,list):

result=sum(list)/len(list)

print(result)

t=test()

t.average([10,20,30,40,50,])

**globle variable:**

if you access or declare outside of the class than it is globle variable

In genarly it is function level programming not object level programming

eg:

x=10

class Test:

globle x //even you can use globle variable inside the class

def m1(self):

print(x)

t=Test()

t.m1()

‘’’ bank applicatiom’’’

Import sys

Class bank:

Cname =”durgasoft”

Def \_\_init\_\_(self, name,balance=0):

Self.name=name

Self.balance=balance

Def deposite(self ,amt)

Self.balance=self.balnce+amt

Print(“after the deposite amount “,self.balance)

Def withdrow(self ,amt)

If amt<self.balance:

Print( “amount is insuficent than the balance”)

Sys.exit()

Else:

Self .balance=self.balnce+amt

Print(“After the withdraw”,self.balance)

Print(“Welcome to,bank.cname)

Name=input(“enter your name: ”)

B=bank(name)

While True:

Print(‘d- deposite/n w-withdraw/n e-exit’)

Option =input(“enter your choice:” )

If option== “d” or option==”D”:

Amt=input(“enter the amount:”)

b.defosite(amt)

elif option ==”w” or option ==”W”:

amt=input(“enter the amout to withdraw”)

b.withdrow(amt)

elif:

if option ==”e” option ==”E”:

print(“Thank for banking”)

sys.exit()

else:

print(“choice the option”)

**types of method:**

instances method

class method

static method

if we are using only one instances variable than we must uses instances method

if we are using only one static variable but not instances variable than we have to use –class method

if we are not using instances or static variable than we have to use –static method is also called genaral utility method

**instances method:**

we can call by using self inside the class

out side the class we can call by object references

eg: class student:

def \_\_init\_\_(self ,name,marks):

self.name=name

self.mark=mark

def m1(self):

print(“hi”,self.name)

print(your marks”,self.marks)

def m2(self):

if self.marks>=60:

print(“you scored A grade”)

elif self.marks>=50:

print(“you scored B grade”)

elif self.marks =>35:

print(you scored C grade )

else:

print(“you have been failed”)

n=int(input(“enter the number of students”))

for i in range(n):

name =input(“enter the name”)

marks =int(input(“enter the marks”)

s=student (name,marks)

s.m1()  
 s.m2()  
 print('\*'\*20)

setter and getter method

def \_\_init\_\_(self ,name,marks):

self.name=name

self.mark=mark

s=student(‘durga’ ,100) //this is one style

s=student()

s.setName(‘duga’)

s.setMark(‘100’)

print(‘student name=’,getName()) //this is mostly used method

the most advantage of using getter and setter method is by default security is going to increase

syntax of getter and setter method

def setVariablename(self ,variableNamae):

self.varibleName =variableName

def setMark(self,marks)

self.marks=marks

def getMarks(self):

return self.marks

eg:

class student:

def setName(self.namae):

self.name=name

def getNmae(self):

return name

def setMarks(self.marks):

self.marks=marks

def getMarks(self):

return marks

n=int(input(“enter the number of students”))

for i in range(n):

name =input(“enter the name”)

marks =int(input(“enter the marks”)

s=student ()

s.setName(name)

s.getMarks(marks)

Class method:

If you don’t need instances method than you can access class method because instances method is costly than classs method

In class method you no need to creat object instant you can call with class name

@classmethod

You can access class method by:

Def m1(cls):

Print(cls.collage)

Print(cls.school name)

Instances method vs class method

1. Insdie method body if your using atleast one instance variable than compulsory we should declare that method as instances method
2. Inside method body if your using atleast one static variable than compulsory we should declare that method as class method
3. To declar the instances method we are not required to use any decarater
4. To declar the class method we should use @classmethod
5. The first argument to the instances method is self which is referances to current object and by using self you can access instances variable inside method
6. The first argument to the class method is cls which is referances to current class and by using that we can access static variable
7. Inside instances method we can access both instances and static variable
8. Inside class method we can only access static variable
9. We can call instances method by using object reperances
10. We can call classmethod either by object referances or by class name

Eg:

Class Animal:

Legs=4

@classmethod

Def walk(cls .name) :

Print({},walks in .{},legs.format(name, cls.legs))

Animal.walk(‘dogs’)

Animal.walk(‘dogs’)

To track the number of object created in class

Class test:

Count=0

Def\_\_init\_\_(self):

Test.count +=1

Def getNoObject ():

Print(“the number of object created “,test.count)

T1 =test()

T2 =test()

T3 =test()

T4 =test()

Test.getNoObject()

Static variable related method is called classmethod but not static method

Static method:

Just general utility method/helper method

@staticmethod

Def sum(x,y):

Print(“the sum of “,x+y)

Eg:1

Class math:

@staticmethod

def add (x,y):

print(“the sum of :”,x+y)

@staticmethod

def product (x,y):

print(“the product of:” x\*y)

@staticmethod

def average (x,y):

print (“the average :”x\*y/2)

//t=math()//you can directly you can callwith math name

math.add(2,8)

math.product(34,939)

math.average(2,2)

**instances method vs classmethod vs staticmethod :**

**1.**if you are using instances variable inside the method body than we should go for instances method

If you using static variable inside the method body than we should go for classmethod we can call either by object referances and by using class name

If you are using genral utility method inside the method body we should go for staticmethod

**If you are not using any decorator:**

For classmethod , @classmethod is mandatory

For staticmethod , @staticmethod is optional

Without decorator the method can be either instances method or staticmethod .if your calling by using object referances than it is instances method .if your

multhreading in python

executing multiple task is nothing but multhreading

eg: classs studant

listning to class

writing notes

checking mobile

seeing girls etc..

two types of multhreading

process base multi tasking

thread based multi tasking

1.product based multi tasking

Each task is seperate indepentant process

eg:

typing python programming

listing to music

downloading music

All this things will run in our system symantaizily or parlaly with indepentantly

this is basically OS based level multi threading

2. Thread based multi threading

if program as 10k lines and first 50h and secound 50h as no connection than you can use thread based multi tasking

Thread is

an independent part of program

a flow of execution is consider as thread

its python object

when should you go for multhreading ?

Where ever independent jobs is avilable you should go for multi threading

there is name called threading module

import threading

print(current exection thread: , threading.current\_thread().getName())

print("atatment")

print("atatment")

print("atatment")

print("atatment")

print("atatment")

print("atatment")

in three ways we can create thread in our own ways

3 ways:

creating thread without using any class

creating thread by extending thread class

creating thread without using extend thread class

1.creating thread without using any class

always use this method

from threading import \*

#! usr/bin/env python

import requests

import re

import urlparse

from bs4 import BeautifulSoup

import threading

print("current exection thread:" , threading.current\_thread().getName())

def display():

for i in range(10):

print("Child thread")

t= threading.Thread(target=display) #creating thread objectto execute display

t.start() #now thread will start

for i in range(10):

print("child thread1")

now thre is two thread and we will get mixed output

2.creating thread by extending thread class

#! usr/bin/env python

import requests

import re

import urlparse

from bs4 import BeautifulSoup

from threading import \*

class MyThreading(Thread):

def run(self):

for i in range(20):

print("child thread")

t=MyThreading()

t.start()

for i in range(10):

print("main thread")

3.creating thread without using extend thread class

#! usr/bin/env python

import requests

import re

import urlparse

from bs4 import BeautifulSoup

from threading import \*

class test():

def m1(self):

for i in range(10):

print("child class")

obj = test()

t= Thread(target=obj.m1)

t.start()

for i in range(10):

print("m1")

#with threading and without threading

1.with threading

import time

def double(number):

for n in number:

time.sleep(1)

print("Double Value:" ,2\*n)

def square(number):

for n in number:

time.sleep(1)

print("square Value:" ,n\*n)

number=[1,2,3,4,5,6,7,8]

begintime=time.time()

double(number)

square(number)

endtime=time.time()

print("total time taken:", endtime-begintime)

2.without threading

import time

from threading import \*

def double(number):

for n in number:

time.sleep(1)

print("Double Value:" ,2\*n)

def square(number):

for n in number:

time.sleep(1)

print("square Value:" ,n\*n)

number=[1,2,3,4,5,6,7,8]

begintime=time.time()

t1=Thread(target=double, args=(number,))

t2=Thread(target=square, args=(number,))

t1.start()

t2.start()

t1.join()

t2.join()

endtime=time.time()

print("total time taken:", endtime-begintime)

You can get name of thread and set name using

print(current.name().getname())

current.name().setname(‘sunny’)

how get ident number

print(current.thread().ident)

print(t..ident)

active\_count():

enumerate():

l = enumerate()

for I in L:

print(Thread name: , t.name)

print(Thread id , t.ident)