**Advance Python**

Session 2:

OOPS

---------

1)Encapsulation

2)Polymorphism

3)Inheritance

4)Abstraction

C++,JAVA,.Net,Python and ABAP are OOPs Languages

OOPs are secure,flexible and reusable

OOPs principles are nothing but certain rules or guidelines which are supposed to be satisfied by a programming Language inorder to call the programming Languages as OOPs Language.

Different OOPs priciples are Encapsulation,Polymorphism,inheritance and Abstraction. If we develop any business application according to OOPs principles in any programming Language then we will get the security,flexibility and reusability in those applications.

Different Object Oriented Programming Languages are:

c++

JAVA

.NET

Python

ABAP etc

Session 3:

OOPs Priciples

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

1)Encapsulation:-

Concept of binding or grouping related data members along with its related functionalities is known as Encapsulation

Example:-

For Customer

cname,caadr,caccno and cbal are variables

deposit(),withdraw(),transfer() and balenq() are methods

For Employee

eid,ename,eaddr,esal and email are variables

da(),ta(),hra(),pf(),tax() and tsal() are methods

class is a syntax of structure is used to grow related data members along with its related functionalities.

syntax:

class <class\_name>:

""" doc string """

..................

..................

..................

Ex:-

try

..........

..........

..........

Ex:-

try

..........

..........

..........

Exce

..........

..........

..........

finally

..........

..........

..........

Ex:-

if

..........

..........

..........

else

..........

..........

..........

elif

..........

..........

..........

Ex:-

for

..........

..........

..........

while

..........

..........

..........

print("start")

class Test:

""" this is sample class"""

x=10

def m1(self):

print("m1 in Test")

x=Test.x+500

print("end")

t1=Test()

t1.m1()

t1.m1()

t2=Test()

t2.m1()

t2.m1()

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

class Test:

def m1(self):

self.x=10

t1=Test()

t1.m1()

print(t1.x)

t1.x=t1.x+5

print(t1.x)

t2.Test()

t2.m1()

print(t2.x)

t2.x=t2.x+2

print(t2.x)

t3=Test()

t3.m1()

print(t3.x)

Session 4:

Constructor:-

Constructor is a special kind of method , which is used to initialize the non-static variables of a class at the time of creation of the Object.

Differences between the method and the Constructor

Methods Constructors

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

1)Mathod name can be | 1)Constructor name should be

anything | \_ \_init\_ \_

|

2) After creating the object if | 2)Constructor will be executed

we call a method then only method | automatically whenever we

will be executed | created object

|

3)with respect to one object one | 3)with respect to one object

method can be called for any | one constructor will be

number of times. | executed only once.

|

4)Methods are used to represent | 4)Constructors are used to

the business logics to perform | initialize the non-static

the operations. | variables.

|

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

class X:

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

print(" Constructor of X")

def m1(self):

print("In m1 of X")

x1=X()

x1.m1()

x1.m1()

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

class X:

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

self.i=10

self.j=20

def disp(self):

print(self.i)

print(self.j)

x1=X()

x1.disp()

x1.i=30

x1.j=40

x1.disp()

x2=X()

x2.disp()

x2.i=50

x2.j=60

x2.disp()

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

class X:

def \_\_init\_\_(self,i,j):

self.i=i;

self.j=j

def disp(self):

print(self.i)

print(self.j)

x1=X(10,20)

x1.disp()

x2=X(30,40)

x2.disp()

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

class Cust:

"""Customer application"""

cbname="SBI"

def \_\_init\_\_(self,cname,caddr,caccno,cbal):

self.cname=cname

self.caddr=caddr

self.caccno=caccno

self.cbal=cbal

def deposit(self,damt):

self.cbal=self.cbal+damt

def withdraw(self,wamt):

self.cbal=self.cbal-wamt

def balenq(self):

print(self.cbal)

def disp(self):

print(self.cname)

print(self.caddr)

print(self.caccno)

print(self.cbal)

print(self.cbname)

c1=Cust("veer","hyd",1001,100000.00)

c1.balenq()

c1.deposit(20000.00)

c1.withdraw(30000.00)

c1.disp()

print("-----------")

c2=Cust("rao","sec",1002,1000.00)

c2.balenq()

print("-----------")

c2.deposit(40000.00)

c2.disp()

print("-----------")

c2.withdraw(50000.00)

c2.disp()

print("-----------")

c2.withdraw(60000.00)

c2.disp()

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

Session 5:

Garbage Collector:-

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

It is a pre-defined program which is present inside python software.

Garbage collector removes the unused or unreferenced objects from the memory location. If the reference count of any object is 0(zero) then we call that object as a unreferenced or unused object.

Whenever any object reference count is zero then python interpretor calls the Garbage collector and GC(Garbage collector) removes the object.

After executing the GC more memory is available for the program so that rest of the program execution becomes faster.

class X:

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

print("in constructor of X")

def \_\_del\_\_(self):

print("in destructor of X")

x1=X()

x2=x1

x2=X()

x1=X()

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

Before going to delete the objects from the memory location it internally executes destructor.

\_ \_del\_ \_ is known as a destructor.

del is a keyword, which is used for removing the attributes.

class X:

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

print("in constructor of X")

def \_\_del\_\_(self):

print("in destructor of X")

x1=X()

x2=X()

x3=X()

del x1

del x2

del x3

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

class Employee:

'''Common base class for all Employees'''

empCount=0

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

self.name=name

self.sal=sal

Employee.empCount+=1

print("in constructor of Employee")

def \_\_del\_\_(self):

Employee.empCount-=1

print("in destructor of Employee")

def displayEmployee(self):

print("Name=:",self.name,", Salary=",self.sal)

emp1=Employee("zara",2000)

print("Total Employees are:",Employee.empCount)

emp1.displayEmployee()

emp2=Employee("veer",3000)

print("Total Employees are:",Employee.empCount)

emp2.displayEmployee()

emp3=Employee("rao",4000)

print("Total Employees are:",Employee.empCount)

emp3.displayEmployee()

del emp1

print("Total Employees are:",Employee.empCount)

del emp2

print("Total Employees are:",Employee.empCount)

del emp3

print("Total Employees are:",Employee.empCount)

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

class X:

a=10

def m1(self):

print("m1 of X")

print(dir(X))

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

Session 6:

Adding the static and non-static variables to the class and objects.

We can add the static variables to the class from outside of the class by using class name.

We can add the non-static variables to the object from outside of the class by using reference variable.

The non-static variables which are added to one object cannont access to the other object.

class X:

a=10

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

self.b=20

print(X.a)

x1=X()

print(x1.b)

X.c=30

print(X.c)

x1.d=40

print(x1.d)

x2=X()

print(x2.b)

x2.e=50

print(x2.e)

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

Deleting the static variables and non-static variables from the class and object:-

We can remove the static and non-static variables from the class and object by using del keyword

class X:

a=10

b=20

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

self.c=30

self.d=40

print(X.a)

print(X.b)

x1=X()

print(x1.c)

print(x1.d)

x2=X()

print(x2.c)

print(x2.d)

del X.a

print(X.b)

del x1.c

print(x1.d)

print(x2.c)

print(x2.d)

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

Implemetation of user defined Exceptions in the class

class NoFunds(Exception):

pass

class Cust:

"""Customer Application"""

cbname="SBI"

def \_\_init\_\_(self,cname,caddr,caccno,cbal):

self.cname=cname

self.caddr=caddr

self.caccno=caccno

self.cbal=cbal

def deposit(self,damt):

self.cbal=self.cbal+damt

def withdraw(self,wamt):

if wamt>self.cbal:

raise NoFunds

self.cbal=self.cbal-wamt

def balenq(self):

print(self.cbal)

def display(self):

print(self.cname)

print(self.caddr)

print(self.caccno)

print(self.cbal)

print(self.cbname)

c1=Cust('veer','hyd',1001,100000.00)

c1.balenq()

try:

c1.withdraw(30000.00)

except(NoFunds):

print("No Funds in account")

c1.display()

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

Defaulr attributes of a class:-

By default for each and every class some attributes are added by the python interpretor. The default attributes of a class are:

\_ \_dict\_ \_

\_ \_module\_ \_

\_ \_doc\_ \_

\_ \_weakref\_ \_

class X:

"""Sample class"""

a=10

b=20

def m1(self):

print("in m1 of X")

def m2(self):

print("in m2 of X")

print(X.\_\_dict\_\_)

print(X.\_\_module\_\_)

print(X.\_\_doc\_\_)

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

Using the properties of one class into another class:-

We can use the properties of one class into another class in two ways. They are:

1)has-a relationship

2)is-a relationship

1)has-a relationship:-

The concept of using the properties of one class into another class by using class name or by using reference variable name is known as a has-a relationship.

class X:

a=10

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

self.b=20

def m1(self):

print("in m1 of X")

class Y:

c=30

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

self.d=40

def m2(self):

print(X.a)

x1=X()

print(x1.b)

x1.m1()

print(Y.c)

print(self.d)

print("in m2 of Y")

y1=Y()

y1.m2()

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

2)is-a relationship or inheritance:-

The concept of using the properties of one class into another class directly is known as inheritance or is-a relationship.

A class which is extended by other class is known as a super class or base class or Parent class

A class which is extendeding another class is known as a sub class orchild class or Derived class

super class properties directly accessible to the sub class like as a sub class properties.

class X:

a=10

def m1(self):

print("in m1 of X")

class Y(X):

c=30

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

self.d=40

def m2(self):

print(Y.a)

self.m1()

print(Y.c)

print(self.d)

print("in m2 of Y")

y1=Y()

y1.m2()

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

super class properties and sub class properties can be accessed by using subclass name or by using sub class reference variable.

class X:

a=10

def m1(self):

print("in m1 of X")

class Y(X):

c=30

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

self.d=40

def m2(self):

print("in m2 of Y")

print(Y.a)

print(Y.c)

y1=Y()

print(y1.d)

y1.m1()

y1.m2()

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

class X:

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

print("Constructor of X")

def m1(self):

print("in m1 of X")

class Y(X):

def m2(self):

print("in m2 of Y")

y1=Y()

y1.m1()

y1.m2()

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

Session 7:

Object:-

It is a pre-defined class which is defined build in smart.

object class is a super class or base class for every clas in python.

Object class properties can be accessed directly in every class or we can access through any class reference variable.

class X:

def m1(self):

print("in m1 of X")

print(X.\_\_bases\_\_)

x1=X()

print(x1)

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

Types of Inheritances:-

1)Single Inheritance

2)Multi-level Inheritance

3)Multiple Inheritance

4)Hirarchical Inheritance

5)Cyclic Inheritance

1)Single Inheritance:-

Concept of inheriting the properties from only one class into another class is known as single inheritance

class X(object):

"""Single Inheritance"""

def m1(self):

print("in m1 of X")

x1=X()

x1.m1()

2)Multi-level Inheritance:-

The concept of inheriting the properties from multiple classes into single class with concept of one after another is known as multi-level inheritance.

class X(object):

"""Multi-level Inheritance"""

def m1(self):

print("in m1 of X")

class Y(X):

def m2(self):

print("in m2 of Y")

class Z(Y):

def m3(self):

print("in m3 of Z")

z1=Z()

z1.m1()

z1.m2()

z1.m3()

3)Multiple Inheritance:-

Concept of inheriting the properties from multiple classes into single class at a time is known as multiple inheritance.

class X(object):

"""Multiple Inheritance"""

def m1(self):

print("in m1 of X")

class Y:

def m2(self):

print("in m2 of Y")

class Z(Y,X):

def m3(self):

print("in m3 of Z")

z1=Z()

z1.m1()

z1.m2()

z1.m3()

4)Hirarchical Inheritance:-

The concept of inheriting the properties from one class into multiple classes is known as hirearchical inheritance.

class X(object):

"""Hirerarchical Inheritance"""

def m1(self):

print("in m1 of X")

class Y(X):

def m2(self):

print("in m2 of Y")

class Z(X):

def m3(self):

print("in m3 of Z")

y1=Y()

y1.m1()

y1.m2()

z1=Z()

z1.m1()

z1.m3()

5)Cyclic Inheritance:-

The concept of inheriting the properties from sub class to super class is known as cyclic inheritance.

Note:-Python does not support cyclic inheritance.

class X(Z):

"""Cyclic Inheritance is not possible in Python"""

def m1(self):

print("in m1 of X")

class Y(X):

def m2(self):

print("in m2 of Y")

class Z(Y):

def m3(self):

print("in m3 of Z")

z1=Z()

z1.m1()

z1.m2()

z1.m3()

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

Session 8

Polymorphism

poly---->many

morphism--->Logics/Functionalities

The concept of defining multiple methods with the same name, with different no. of parameters within a class is known as method overloading.

Whenever we implement method overloading concept in python by default python interpretor recognizes recently defined method only because python doesnot support method overriding.

class X:

def m1(self,a):

print("1-args")

def m1(self,a,b):

print("2-args")

x1=X()

#x1.m1(10)#error

x1.m1(10,20)

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

class X:

def add(self,instanceOf,\*args):

if instanceOf=='int':

res=0

if instanceOf=='str':

res=''

for i in args:

res=res+i

print(res)

x1=X()

x1.add('int',10,20,30)

x1.add('str','python',' is',' a',' scripting',' language')

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

Method Overriding

The concept of defining multiple methods with the same name and same number of parameters one is in super class and another one is in sub class is known as method overriding. Whenever we override the super class method in sub class always sub class method is executed if sub class object is created.

class X:

def m1(self):

print("m1 in X")

class Y:

def m1(self):

print("m1 in Y")

y1=Y()

y1.m1()

x1=X()

x1.m1()

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

class X(object):

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

self.name=name

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

return self.name

x1=X("Hello")

print(x1)

x2=X("Hai")

print(x2)

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

class X(object):

def m1(self):

print("in m1 of X")

class Y(X):

def m1(self):

#super().m1()

print("in m1 of Y")

y1=Y()

y1.m1()

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

class X(object):

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

self.a=10

class Y(X):

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

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

self.b=20

y1=Y()

#print(y1.a)

print(y1.b)

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

Abstraction

The concept of adding the properties of a class from outside of the class is known as Abstraction. If any variable is preceeded by \_ \_ then it is going to be hidden from that class.

python is providing special syntax to access the hidden properties of a class from outside of the class.

class X(object):

\_\_a=10

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

X.\_\_a=X.\_\_a+20

def disp(self):

print(X.\_\_a)

x1=X()

x1.disp()

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

Session 9

Regular Expressions

Regular Expressions are used to

1)Extracting the required data from the given data

2)To perform data validations

3)To develop the URL patterns in the web applications

In Regular Expressions we use some special characters to define the patterns.

After defining the pattern we can extract that pattern matching data from the given data by using pre-defined functions of RE module.

re is a inbuild module of the python

\* ------->It matches 0 or more occurances of preceding characters

\* ---->ab\*c

abc

ac

abbc

abbbbbbbbbbbbbbc

All the above are correct

+ ------->It matches 1 or more occurances of preceding characters

+ ---->ab+c

abc

abbc

abbbbbbbbbbbbbbc

All the above are correct

? ------->It matches 0 or 1 occurance of preceding character

? ---->ab?c

ac

abc

abbc(wrong)

^ Matches beginning of line.

$ Matches end of line.

. Matches any single character except newline. Using m option allows it

to match newline as well.

[...] Matches any single character in brackets.

b[aeiou]d

bad

bed

bid

bud

bod

All the above are correct

[^...] Matches any single character not in brackets

re\* Matches 0 or more occurrences of preceding expression.

re+ Matches 1 or more occurrence of preceding expression.

re? Matches 0 or 1 occurrence of preceding expression.

re{ n} Matches exactly n number of occurrences of preceding expression.

re{ n,} Matches n or more occurrences of preceding expression.

re{ n, m} Matches at least n and at most m occurrences of preceding expression.

a| b Matches either a or b.

[Pp]ython Match "Python" or "python"

rub[ye] Match "ruby" or "rube"

[aeiou] Match any one lowercase vowel

[0-9] Match any digit; same as [0123456789]

[a-z] Match any lowercase ASCII letter

[A-Z] Match any uppercase ASCII letter

[a-zA-Z0-9] Match any of the above

[^aeiou] Match anything other than a lowercase vowel

[^0-9] Match anything other than a digit

ruby? Match "rub" or "ruby": the y is optional

ruby\* Match "rub" plus 0 or more ys

ruby+ Match "rub" plus 1 or more ys

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

Session 10

(|)--->Matches any one string in the list

Ex:-

(Java|PHP|Python|ASP)

^ ---->Matches start character

$ ---->Matches end character

\d{3} Match exactly 3 digits

\d{3,} Match 3 or more digits

\d{3,5} Match 3, 4, or 5 digits

. Match any character except newline

\d Match a digit: [0-9]

\D Match a nondigit: [^0-9]

\s Match a whitespace character: [ \t\r\n\f]

\S Match nonwhitespace: [^ \t\r\n\f]

\w Match a single word character: [A-Za-z0-9\_]

\W Match a nonword character: [^A-Za-z0-9\_]

\b Word boundary

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

import re

regex=r"[a-zA-Z]+ \d+"

matches=re.findall(regex,"June 24,August 9,Oct 13,Dec")

for match in matches:

print("Full match: ",match)

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

import re

regex=r"([a-zA-Z]+) \d+"

matches=re.findall(regex,"June 24,August 9,Oct 13,Dec")

for match in matches:

print("Full match: ",match)

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

Session 11

import re

regex=r"[a-zA-Z]+ \d+"

matches1=re.findall(regex,"June 24,August 9,Oct 13,Dec")

matches2=re.finditer(regex,"June 24,August 9,Oct 13,Dec")

print(type(matches1))

print(type(matches2))

for match in matches1:

print(match)

for match in matches2:

print(match.start(),match.end())

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

import re

regex=r"([a-zA-Z]+) (\d+)"

print(re.sub(regex,r"\2 of \1","June 24,August 9,Oct 13,Dec"))

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

import re

regex=r"([a-zA-Z]+) (\d+)"

x=re.sub(regex,r"\2 of \1","June 24,August 9,Oct 13,Dec")

regex1="\d+ [a-zA-Z]+ ([a-zA-Z]+)"

matches=re.findall(regex1,x)

for match in matches:

print(match)

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

import re

regex=re.compile(r"(\w+) World")

result=regex.search("Hello World is the easiest")

print(result)

if result:

print(result.start(),result.end())

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

import re

regex=re.compile(r"(\w+) World")

x=regex.findall("Hello World.Hai World")

for result in x:

print(result)

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

import re

regex=re.compile(r"(\w+) World")

print(regex.sub(r"\1 Earth","Hello World"))

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

HomeWork

1)write program to validate Mobile number

2)write program to validate Password

Must containe

atleast 1 uppercase

atleast 1 lowercase

atleast 1 digit

atleast 1 special character

3)write program to validate Email

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

Session 12

MultiThreading

Thread:

A Thread is a functionality or logic which executes simultaneously along with other part of the program

OR

A Thread is a light-weight process

process:

Any program which is under execution is known as a process.

Defining the functionality as a Thread:-

We can define the functionality as a Thread by overriding run() method of Thread class.

Thread is a pre-defined class, which is defining Threading module.

import threading

class X(threading.Thread)

def run(self):

for p in range(10):

print(p)

x1=X()

x1.start()

for z in range(10,20):

print(z)

If we call the run() method directly it will be executed as a normal method.

Inorder to execute the run() method logic as a Thread we have to call the run() method through the start() method of Thread class.

By default python interpretor creates one Thread that is main() Thread.

import threading

class X(threading.Thread)

def run(self):

for p in range(10):

print(p)

class Y(threading.Thread)

def run(self):

for q in range(10,20):

print(q)

x1=X()

x1.start()

y1=Y()

y1.start()