**TYPES OF PROGRAMMING LANGUAGE**

* The Programming Languages are 3 Types
* **POP Languages**
* **OOPS Languages**
* **Scripting Languages**

**1. POP Languages:**

* POP stands for **Procedural Oriented Programming**.
* A Program written using **Functions.**

**Ex: c, pascal, cobol, fortran,....etc.**

**2. OOPS Languages:**

* OOPS stands for **Object Oriented Programming**.
* A Program written using **Classes & Objects**.

**Ex: c++, java, apex, go, move, swift,....etc.**

**3. Scripting Languages:**

* A Program written in simple commands.

**Ex: JavaScript, VB Script, ....etc.**

**Note:**

**PYTHON SUPPORTS POP,OOPS AND SCRIPTING FEATURES.**

**Object Oriented Programming (OOPS)**

* **Classes**
* **Objects**
* **Data Encapsulation**
* **Data Abstraction**
* **Constructors**
* **Destructors**
* **Inheritance**
* **Polymorphism**

**1. Classes:**

* **A Group of Variables and Methods(Functions) are together into a single unit that unit is called a Class.**
* **Python Supports 2 Types of Classes.**

🡺Build-in Classes.

🡺User-defined Classes.

**1. Built-in classes:**

* **The Classes which are coming along with Python Software.These Classes are called**

**Build-in Classes**

**Ex-1:**

a=9

type(a) **#<class 'int'>**

b=1.3

type(b) **#<class 'float'>**

c='ram'

type(c) **#<class 'str'>**

d=True

type(d) **#<class 'bool'>**

a=[11,22,33,44]

type(a) **#<class 'list'>**

t=11,22,33,44

type(t) **#<class 'tuple'>**

s= {111,22,33,44}

type(s) **#<class 'set'>**

x = range (1,11,1)

type(x) **#<class 'range'>**

y=None

type(y) **#<class 'NoneType'>**

**Ex-2:**

def display ():

print ('Hai i am display ....')

type(display) **#<class 'function'>**

**2. User-Defined Classes:**

* **The Classes which are developed by Programmer. These Classes are called**

**User-Defined Classes.**

**Syntax:**

**class classname:**

**list of variables**

**list of methods**

**Ex-1:**

class car:

color

model

price

company name

height

width

start ()

stop ()

move ()

park ()

**Ex-2:**

class door:

color

model

price

company name

height

width

open ()

close ()

**Ex-3:**

class person:

color

name

surname

age

eyes

ears

noise

height

weight

walk ()

talk ()

sleep ()

profession ()

**What is class ?**

* A Group of Variables and Methods Together into a Single Unit.
* i.e. Blueprint

**What is object?**

* The physical appearance of the given class is called object.
* we can create any No. of objects.
* i.e. 'n' number of objects.

**CLASS**

**Syntax:**

**class classname:**

**list of variables**

**list of methods**

**Ex-1: To create Empty Class**

class test:

pass

**Ex-2: To create a class only variables**

class sample:

a=10

b=25

print(sample.a)

print(sample.b)

**Here,**

**a, b** are called ‘**CLASS LEVEL VARIABLES’**.

**How to Access class members**:

* We can access class members by using **‘DOT OPERATOR’ .**
* Class member means it may be either **Variable or Method.**

**Syntax:**

**classname.membername**

**EX:** test.read()

test.m1()

add.sum()

kiran.low()

**Ex-3: To create a class only methods (function)**

class sample:

def m1():

print('I am method-1 in sample class')

def m2():

print('I am method-2 in sample class')

sample.m1()

sample.m2()

**Ex-4: To create a class with variables methods**

class sample:

a=10

b=20

def m1():

print('I am method-1 in sample class')

def m2():

print('I am method-2 in sample class')

print(sample.a)

print(sample.b)

sample.m1()

sample.m2()

**Here,**

**a, b** are called **class level variables**

**m1(), m2()** are called **class level methods**

**Ex-5: Write a program to create rectangle class and find the area of a rectangle**

class rectangle:

def read():

rectangle.l=int(input('Enter length value'))

rectangle.b=int(input('Enter breadth value'))

def find():

rectangle.area = rectangle.l \* rectangle.b

def result():

print('Area of a rectangle = ',rectangle.area)

rectangle.read()

rectangle.find()

rectangle.result()

**Here,**

**l, b, area** are called **class level variables**

**read(), find(), result()** are called **class level methods**.

**Ex-6: Write a program to create circle class and find the area of a circle**

class circle:

def read():

circle.r = int(input('Enter radius value'))

def find():

circle.area = 3.14 \* circle.r \* circle. r

def result():

print('Area of a circle = ',circle.area)

circle.read()

circle.find()

circle. result()

**Here,**

**r, area** are called **class level variables**

**read(), find(), result()** are called **class level methods.**

**OBJECT**

**What is object?**

* The Physical Appearance of the Given class is called object.

Or

* The Instance of the class is called object.

**Syntax:**

**objectname = classname()**

**Ex:**

a=int ()

b=float ()

s=str ()

d=bool ()

c=complex ()

x=list ()

y=tuple ()

z=set ()

d=dict ()

r=rectangle ()

c=circle ()

e=employee ()

s=sample ()

t=test ()

p1 = person ()

**Ex-1: Write a program to Create a Empty class and object**

class sample:

pass

s1 = sample ()

**Ex-2: Write a Program to Create a class and object.**

class sample:

def m1(self):

print(type(self))

print('I am m1 ...')

s1 = sample()

s1.m1()

**Ex-3: Write a Program to Create rectangle class and Find the Area of a rectangle using objects**

class rectangle:

def read(self):

self.l=int(input('Enter length value'))

self.b=int(input('Enter breadth value'))

def find(self):

self.area = self.l \* self.b

def displayresult(self):

print('Area of a rectangle = ',self.area)

r1 = rectangle()

r1.read()

r1.find()

r1.displayresult()

**Here,**

**l, b, area** are called **object level variables.**

**read(), find(), display()** are called **object level methods**

Ex-4: Write a Program to Create Window class and using objects

class window:

def read(self):

self.l=int(input('ENTER THE LENGTH OF THE WINDOW :'))

self.b=int(input('ENTER THE BREADTH OF THE WINDOW :'))

def result(self):

print('WINDOWS ARE READY TO DELIVERY------>>>')

n=int(input('HOW MANY WINDOWS :'))

print('\n')

for i in range(n):

w = window()

w.read()

print('\n')

w.result()

**\_\_INIT\_\_**

**\_\_init\_\_ ()**:

* **init** means **INITIALIZATION**.
* It is a **Built-in Function**.
* It is a **Special Method** or **Dunder Method** or **Double Underscore Method** or **Magic Method.**
* **This Method is Executed Automatically at the time of object creation.**
* **No Need of Calling it.**

**Syntax:**

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

**body of the init function**

**Ex-1: DEMO FOR \_\_INIT\_\_**

class test:

def \_init\_(self):

print('I am init ...')

t1 = test()

**Ex-2: THE INNER CODE FOR THE EMPTY LIST**

class list:

def \_init\_(self):

self = []

print(self)

a = list ()

**Ex-3: THE INNER CODE FOR THE EMPTY DICT**

class dict:

def \_init\_(self):

self = {}

print(self)

d = dict ()

**del Statement in Python:**

* del is a Keyword.
* This Keyword is used to Delete the Object from the Memory.

**Syntax:**

**del objectname**

**or**

**del object-1, object-2, object-3,...object-n**

ex-1:

a = int()

b = float()

c = str()

d = complex()

x=list()

y=tuple()

z=dict()

s=set()

print(a)

print(b)

print(c)

print(d)

print(x)

print(y)

print(z)

print(s)

del a

del b

del c

del d

del x

del y

del z

del s

print(a)

print(b)

print(c)

print(d)

print(x)

print(y)

print(z)

print(s)

**CONSTRUCTORS**

**Constructors in Python:**

* Creating an Object is called as **Constructor.**
* In Python, the Constructor Method is init() Method.

**Syntax:**

**def init(self):**

**body of the Constructor**

* The Constructors are 2 Types:

1. **Default Constructor**
2. **Parameterized Constructor**

**1. Default Constructor:**

* **To Create an Object without a Parameter is called as Default Constructor** or No Arguments Constructor or Parameter less Constructor.

**Ex-1:**

class student:

def \_init\_(self):

print('I am default constructor ...')

s1 = student()

**2. Parameterized Constructor:**

* **To Create an Object with parameter is called as parameterized Constructor.**

**Ex-1:**

class student:

def \_init\_(self,a,b,c,d,e):

print('I am parameterized constructor ...')

self.htno = a

self.name = b

self.m1 = c

self.m2 = d

self.m3 = e

def display(self):

print(self.htno)

print(self.name)

print(self.m1)

print(self.m2)

print(self.m3)

s1 = student(1901,'ram',77,88,99)

s1.display()

print('----------------------------------')

s2 = student(1902,'siva',88,88,88)

s2.display ()

del s1,s2

**Demo for Default Constructor and Destructor:**

**Ex-1:**

class student:

def \_init\_(self):

print('object created ...')

def \_del\_(self):

print('object deleted ...')

s1 = student()

del s1

**Ex-2:**

class test:

count = 0

def \_init\_(self):

print('object created ...')

test.count = test.count + 1

def \_del\_(self):

print('object deleted ..')

test.count = test.count - 1

print('No. of objects = ',test.count)

t1 = test()

t2 = test()

t3 = test()

t4 = test()

t5 = test()

print('No. of objects = ',test.count)

del t3

del t4

print('No. of objects = ',test.count)

s6 = test()

s7 = test()

s8 = test()

s9 = test()

del s8

print ('No. of objects = ',test.count)

**Magic Methods or Double underscore methods or Dunder methods:**

* \_\_ init \_\_ 🡺 Initialization
* This Method is Executed at the of Object Creation.
* This is executed automatically.

* \_ \_ del \_ \_ 🡺 Deletion

* This Method is Executed at the time of Object Deletion.
* This is executed automatically.

**Ex-1: Write a program to create empty constructor and empty destructor**

class test:

def \_init\_(self):

pass

def \_del\_(self):

pass

**Ex-2: Write a program to find the area of a circle**

**Method-1: using parameterized constructors**

class circle:

def \_init\_(self,x):

print('object created ..')

self.r = x

def find(self):

self.area = 3.14\*self.r\*\*2

def display(self):

print('Area of a circle = ',self.area)

def \_del\_(self):

print('object deleted ...')

c1 = circle(5)

c1.find()

c1.display()

del c1

**Method-2: using default constructors**

class circle:

def \_init\_(self):

print('object created ..')

self.r = int(input('Enter radius value'))

def find(self):

self.area = 3.14\*self.r\*\*2

def display(self):

print('Area of a circle = ',self.area)

def \_del\_(self):

print('object deleted ...')

c1 = circle()

c1.find()

c1.display()

del c1

**Ex-3: Write a program to create a student class with parameterized constructor**

class student:

def \_init\_(self,hno,name,mark1,mark2,mark3):

self.htno = hno

self.sname = name

self.m1 = mark1

self.m2 = mark2

self.m3 = mark3

def find(self):

self.total = self.m1+self.m2+self.m3

self.avg = self.total/3

def display(self):

print('Total Marks = ', self.total)

print('Average marks = ',self.avg)

def \_del\_(self):

pass

s1 = student(101,'ram',77,88,99)

s2 = student(102,'venkat',88,88,88)

s3 = student(103,'balut',65,34,11)

s4 = student(104,'abc',11,22,33)

s1.find()

s2.find()

s3.find()

s4.find()

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Object-1 Details ....')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

s1.display()

print('\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Object-2 Details ....')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

s2.display()

print('\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Object-3 Details ....')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

s3.display()

print('\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Object-4 Details ....')

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

s4.display()

**Here,**

**101,'ram',77,88,99** 🡺 **Actual arguments**

**htno, name, mark1, mark2, mark3** 🡺 **Formal arguments**

**self** is **pointing current object.**

**htno, sname,m1,m2,m3** are called **object level variables**

**Ex-1:**

class Dress:

def \_init\_(self, type, price):

self.type = type

self.price = price

def details(self):

return f"A {self.type} costs Rs.{self.price}"

d1 = Dress("shirt", 50)

print(d1.details())

d2 = Dress('Jeans',450)

print(d2.details())

**Constructors and Destructors in Python:**

**What is Constructor?**

* Constructor means, to create an object of any Class.
* Constructor is a Special Method in Python.
* The Name of the Constructor method is \_\_init\_\_(self).
* Constructor will be executed Automatically at the time of Object creation.
* The main purpose of the Constructor is to initialize the Object Level Variables Values.
* Per object Constructor is executed only once.
* Constructor can take atleast one argument i.e. self.

**Syntax:**

def \_init\_(self):

body of the Constructor

* Python supports 2 Types of Constructor:

1. Default Constructor

2. Parameterized Constructor

**1. Default Constructor:**

* It is a Constructor.
* To create an Object without parameters is called a Default Constructor.

**Ex-1:**

class sample:

def \_init\_(self):

print('I am default constructor')

s1 = sample()

**2. Parameterized Constructor:**

* It is a Constructor.
* To create an object with parameters is called a Parameterized Constructor.

**Ex-1:**

class sample:

def \_init\_(self,x):

print('I am parameterized constructor')

print('Hi ....',x)

s1 = sample('ram')

**Ex-2: Write a program to find area of a rectangle, using default constructor**

class rectangle:

def \_init\_(self):

self.l = 10

self.b = 20

def find(self):

self.area = self.l\*self.b

def display(self):

print('Area of a rectangle = ',self.area)

r1 = rectangle()

r1.find()

r1.display()

**Ex-3: Write a program to find area of a rectangle using parameterized constructor**

class rectangle:

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

self.l = x

self.b = y

def find(self):

self.area = self.l\*self.b

def display(self):

print('Area of a rectangle = ',self.area)

r1 = rectangle(10,20)

r1.find()

r1.display()

r2 = rectangle(8,5)

r2.find()

r2.display()

**Ex-4: Write a program to implement parameterized constructor in an employee class**

class employee:

def \_init\_(self,x,y,z):

self.empno=x

self.ename=y

self.sal=z

def details(self):

print('Employee details are :')

print('Employee Number = ',self.empno)

print('Employee Name = ', self.ename)

print('Employee salary = ',self.sal)

e1 = employee(1901,'ram',45000)

e1.details()

e2 = employee(1902,'venkat',99000)

e2.details()

e3 = employee(1903,'siva',56000)

e3.details()

**Ex-5: Write a program to implement parameterized constructor in a student class**

class student:

def \_init\_(self,x,y,a,b,c):

self.htno=x

self.sname=y

       self.m1=a

**INHERITANCE**

* **To Derive the Properties (Variables & Methods) from Old Class into New Class is called Inheritance.**
* **The Advantages of Inheritance :**

**1. Code Reusability.**

**2. Readability**

**Ex:**

Height

Weight

age

name **PARENT**

surname

color

talk()

walk()

profession()

height

weight

age

name **CHILD**

surname

color

talk()

walk()

profession()

* The Old class is called The **Base Class** or **Super Class** or **Parent Class.**
* The New Class is called The **Derived Class** or **Sub Class** or **Child Class.**
* Python Supports 5 Types of Inheritances:

1. **Single Inheritance**.

2. **Multilevel Inheritance.**

3. **Multiple Inheritance.**

4.**Hierarchical Inheritance.**

5. **Hybrid Inheritance.**

**1. Single Inheritance:**

* **To Derive the Properties (Variables & Methods) from only one Base Class is called**

**Single Inheritance.**

**Ex:**

A PARENT

B CHILD

Here,

**A, PARENT** is called **Base class.**

**B, CHILD** is called **Derived class.**

2. **Multilevel Inheritance:**

* **To Derive the Properties (Variables & Methods) from only one Base Class is called Multilevel Inheritance.**

**Ex:**

**A GRAND-PARENT**

**B PARENT**

**C CHILD**

**Here,**

**A, GRAND-PARENT** is called **Base Class.**

**B, PARENT** is called class **Intermediate Base Class.**

**C, CHILD** is called **Derived Class.**

3. **Multiple Inheritance:**

* **The Derived Class is Derived from Multiple Base Classes is called**

**Multiple Inheritance.**

**i.e. More than One Base Class.**

**Ex-1:**

**A B C D**

**E**

**Ex-2:**

**FATHER MOTHER AUNTY UNCLE**

**CHILD**

**Ex-3:**

**CAL-1 CAL-2 CAL-3 CAL-4**

**CALCULATOR**

**4.Hierarchical Inheritance:**

* **Only One Base Class and More than One Derived Classes is called**

**Hierarchical Inheritance.**

**i.e. Only One Base Class and No. of Derived Classes.**

**Ex-1:**

**PARENT**

**CHILD-1 CHILD-2 CHILD-3 CHILD-4**

**Here,**

**PARENT** is a **Base Class.**

**CHILD-1, CHILD-2, CHILD-3, CHILD-4** are called **Derived Classes.**

**Ex-2:**

**COURSES**

**B.SC B.COM B. TECH MBBS**

**Ex-3:**

**B. TECH**

**CSE IT ECE EEE**

5. **Hybrid Inheritance:**

* **A Combination of Two or More Inheritances is called Hybrid Inheritance.**

**Ex:**

**A**

**B C**

**D**

**INHERITANCE DECLARATION SYNTAX:**

**class baseclassname:**

**list of variables**

**list of methods**

**class derivedclassname (baseclassname):**

**list of variables**

**list of methods**

**Ex-1:**

class A:

list of variables

list of methods

class B(A):

list of variables

list of methods

A

B

**Ex-2:**

class sample:

list of variables

list of methods

class test(sample):

list of variables

list of methods

sample

         test

**1. Single Inheritance:**

* **To Derive the Properties (Variables & Methods) from only one Base Class is called**

**Single Inheritance.**

**Ex:**

A PARENT

B CHILD

Here,

**A, PARENT** is called **Base class.**

**B, CHILD** is called **Derived class.**

**Ex-1: Write a Program Demo for Single Inheritance using Variables**

class sample:

a=3

b=5

class test(sample):

x=10

y=99

print(test.a)

print(test.b)

print(test.x)

print(test.y)

print(test.a+test.b+test.x+test.y)

**Ex-2: Write a Program Demo for Single Inheritance using Methods**

class sample:

def m1():

print('I am m1 of sample class ...')

def m2():

print('I am m2 of sample class ...')

class test(sample):

def m3():

print('I am m3 of test class ...')

def m4():

print('I am m4 of test class ...')

test.m1()

test.m2()

test.m3()

test.m4()

**Here,**

**sample** is called **Base Class** or **Super Class**.

**test** is called **Derived Class** or **Sub Class**

**Ex-3: Write a Program Demo for Single Inheritance**

class animal:

def speak(self):

print('Animal speaking ...')

class dog(animal):

def bark(self):

print('dog barking ......')

d1 = dog()

d1.speak()

d1.bark()

2. **Multilevel Inheritance:**

* **To Derive the Properties (Variables & Methods) from only one Base Class is called Multilevel Inheritance.**

**Ex:**

**A GRAND-PARENT**

**B PARENT**

**C CHILD**

**Here,**

**A, GRAND-PARENT** is called **Base Class.**

**B, PARENT** is called class **Intermediate Base Class.**

**C, CHILD** is called **Derived Class.**

**Ex-2: Write a Program Demo for Multilevel Inheritance**

**Using the Following Diagram:**

**ANIMAL**

**DOG**

**DOG CHILD**

class animal:

def speak(self):

print ('Animal speaking ...')

class dog(animal):

def bark(self):

print ('dog is barking')

class dogchild(dog):

def eat(self):

print('dog child eating mutton ...')

d1 = dogchild()

d1.speak()

d1.bark()

d1.eat()

3. **Multiple Inheritance:**

* **The Derived Class is Derived from Multiple Base Classes is called**

**Multiple Inheritance.**

**i.e.More than One Base Class.**

**Ex-1:**

**CAL-1 CAL-2 CAL-3 CAL-4**

**CALCULATOR**

**Syntax:**

**class baseclassname-1:**

**list of variables**

**list of methods**

**class baseclassname-2:**

**list of variables**

**list of methods**

**class baseclassname-3:**

**list of variables**

**list of methods**

**class deriveclassname(baseclassname-1, baseclassname-2, baseclassname-3):**

**list of variables**

**list of methods**

**Ex-1: Write a Program to Implement Multiple Inheritance.**

**CAL-1 CAL-2 CAL-3 CAL-4**

**add() sub() mul() div()**

**CALCULATOR**

**mod (),**

**add(), sub(), mul(), div()**

class calc1:

def add(self,x,y):

print('Addition value = ',x+y)

class calc2:

def sub(self,x,y):

print('Minus value = ',x-y)

class calc3:

def mul(self,x,y):

print('Multipplication value = ',x\*y)

class calc4:

def div(self,x,y):

print('Division value = ',x/y)

class calculator(calc1,calc2,calc3,calc4):

def mod(self,x,y):

print('Remainder value = ',x%y)

c1 = calculator()

c1.add(11,22)

c1.sub(5,2)

c1.mul(9,6)

c1.div(9,3)

c1.mod(9,5)

**4.Hierarchical Inheritance:**

* **Only One Base Class and More than One Derived Classes is called**

**Hierarchical Inheritance.**

**i.e. Only One Base Class and No. of Derived Classes.**

**Ex-1:**

**PARENT**

**CHILD-1 CHILD-2 CHILD-3 CHILD-4**

**Here,**

**PARENT** is a **Base Class.**

**CHILD-1, CHILD-2, CHILD-3, CHILD-4** are called **Derived Classes.**

**Ex-2: Write a Program Demo for Hierarchical Inheritance.**

**A**

**B C D E**

**Here,**

**A** is a **Base Class.**

**B, C, D, E** are called **Derived Classes.**

class A:

def m1():

print('I am m1....')

class B(A):

def m2():

print('I am m2....')

class C(A):

def m3():

print('I am m3....')

class D(A):

def m4():

print('I am m4....')

class E(A):

def m5():

print('I am m5....')

B.m1()

B.m2()

C.m1()

C.m3()

D.m1()

D.m4()

E.m1()

E.m5()

5. **Hybrid Inheritance:**

* **A Combination of Two or More Inheritances is called Hybrid Inheritance.**

**Or**

* **A Combination of Single-Level Inheritance, Multi-Level Inheritance,**

**Multiple Inheritance and Hierarchical Inheritance is called Hybrid Inheritance.**

**Ex-1:**

**A**

m1()

m1() m1()

**B C**

**D**

m1()

class A:

def m1():

print('I am m1 ...')

class B(A):

pass

class C(A):

pass

class D(B,C):

pass

D.m1()

**Ex-2: Write a Program Demo for Hierarchical Inheritance**

**A**

m1()

m1() m1()

m2() m3()

**B C**

**D**

m4(), m1(), m2(), m3()

class A:

def m1():

print('I am m1 ...')

class B(A):

def m2():

print('I am m2 ...')

class C(A):

def m3():

print('I am m3 ...')

class D(B,C):

def m4():

print('I am m4 ...')

D.m1()

D.m2()

D.m3()

D.m4()

**Ex-3: Write a Program Demo for Hierarchical Inheritance**

class P:

a = 30

class Q(P):

b = 50

q = Q()

print(" Value of a: " ,q.a)

print("Value of b: " ,q.b)

**POLYMORPHISM**

* **POLY 🡺 MANY**
* **MORPHISM 🡺 FORMS**
* **Polymorphism Means Single Name Many Forms.**

**i.e. Single Name With Different Types of Behaviours.**

Son

Brother

Lover

Husband

KIRAN --> Student

Guest

Passenger

Audience

Customer

* Python Support 3 Types of Polymorphisms:

1. Operator Based Polymorphism
2. Function Based Polymorphism
3. Class Based Polymorphism

**1. Operator Based Polymorphism:**

* **Only One Operator with Different Behaviours it is called**

**Operator Based Polymorphism.**

**Ex-1: Using ‘+’ Operator**

list concatenation

tuple concatenation

**+** 🡺 addition of two integers

addition of two floats

string concatenation

2+7

9

'python'+'tronix'

'pythontronix'

x= [11,22,33,44]

y= [99,88,77]

x+y

[11, 22, 33, 44, 99, 88, 77]

t= (11,22,33)

t2= (99,88,77)

t+t2

(11, 22, 33, 99, 88, 77)

**Ex-2: Using ‘-‘ Operator**

**-**  🡺 Subtraction of two numbers

difference between two sets.

2-7

-5

s1= {11,22,33,44}

s2= {33,44,55,66,77,88,99}

s1-s2

{11, 22}

**Ex-3: Using ‘\*’ Operator**

list repetition

tuple repetition

**\*** 🡺 multiplication of two numbers

string repetition

2\*7

14

'tronix'\*10

'tronixtronixtronixtronixtronixtronixtronixtronixtronixtronix'

a=[11,22,33,44]

a\*5

[11, 22, 33, 44, 11, 22, 33, 44, 11, 22, 33, 44, 11, 22, 33, 44, 11, 22, 33, 44]

t=11,22,33

t\*4

(11, 22, 33, 11, 22, 33, 11, 22, 33, 11, 22, 33)

**2. Function Based Polymorphism:**

* **Only One Function with Different Behaviours it is called**

**Function Based Polymorphism.**

**Ex-1: Using range () Built-In Function**

range(1,11,1)

range(1,11)

range(6)

**Ex-2: Using extend () Built-In Function**

a = [11,22,33,44]

a.extend({55,66,77})

a.extend([55,66,77])

a.extend((55,66,77))

a.extend(range(1,11,1))

a.extend('tronix')

**Ex-3: Using print () Built-In Function**

print('good mng')

print('good mng','to all')

print(a)

print(a,b)

print(a,b,c)

print(a,b,c,d,e,f)

**Ex-4: Using User-Defined Function, To Implement Function Based Polymorphism**

def maximum(\*x):

print(max(x))

maximum (7,2)

maximum (11,22,6,8,4)

maximum (99,34,67,23)

maximum (11,22,44,66,7,8,3,5,43,3,23,233,323,33,433,34,3)

**Class Based Polymorphism:**

* + To Implement Polymorphism Using Classes is Called **Class Based Polymorphism.**

Ex-1:

class sample:

def m1():

print ('I am m1 of sample class ...')

class test(sample):

def m1():

print ('I am m1 of test class ...')

test.m1()

**Ex-2: Using \_init\_(self)**

class sample:

def \_init\_(self):

print('I am init of base class ...')

class test(sample):

def \_init\_(self):

print('I am init of derived class ...')

t1 = test()

**super ():**

* + super () function is a Built-In Function.
  + This Function is used to Execute super classes functions.

**Syntax:**

**super().functionname()**

**Ex-3: Using \_init\_(self)**

class sample:

def \_init\_(self):

print('I am init of base class ...')

class test(sample):

def \_init\_(self):

super().\_init\_()

print('I am init of derived class ...')

t1 = test()

**Ex-4: Using \_init\_(self)**

class A:

def \_init\_(self):

print('I am init of class--A ')

class B(A):

def \_init\_(self):

super().\_init\_()

print('I am init of class--B ')

class C(B):

def \_init\_(self):

super().\_init\_()

print('I am init of class--C ')

c1 = C()

* **Write a Program Demo for Hierarchical Inheritance using Polymorphism.**

**PERSON**

\_\_init\_\_()

display()

**TEACHER STUDENT**

\_\_init\_\_ () \_\_init\_\_ ()

display () display ()

class person:

def \_init\_(self,n,p):

self.name = n

self.phno = p

def display(self):

print('Name = ',self.name)

print('Phone Number = ',self.phno)

class teacher(person):

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

super().\_init\_(a,b)

self.sal = c

def display(self):

super().display()

print('Salary = ',self.sal)

class student(person):

def \_init\_(self,a,b,c,x,y,z):

super().\_init\_(b,c)

self.htno=a

self.m1=x

self.m2=y

self.m3=z

def display(self):

super().display()

print('Hall ticket Number = ',self.htno)

print('First subject marks = ',self.m1)

print('Second subject marks = ',self.m2)

print('Third subject marks = ',self.m3)

s1 = student(1901,'ram',9988776655,77,88,99)

print('\t\tSTUDNET INFORMATION')

s1.display()

t1 = teacher('siva',6677889922,96000.00)

print('\t\t TEACHER INFORMATION')

t1.display()

**Ex: Using super () Function**

class Animal:

def \_init\_(self):

print('Animal birth...')

class Mammal(Animal):

def \_init\_(self):

# call superclass

super().\_init\_()

print('Mammals give birth directly')

dog = Mammal()

* **What is Attribute?**
* It may be either Variable / Method within the Class.

**Ex-1:**

class sample:

a=10

def m1():

print("I am m1 ..")

print(sample.a)

sample.m1()

#sample.m2()

#print(sample.b)

ACCESS MODIFIERS

* **Python Supports 3 Types of Access Modifiers:**

1. **Public (Variable or Methods)**
2. **Protected (Variable or Methods)**
3. **Private (Variable or Methods)**

**1. Public:**

* By default, **Every Attribute is Public.**
* **We can access from Anywhere either within the Class or from Outside of the Class.**

**Ex-1: To Declare Public Variables.**

class sample:

a=5

b=8

print(sample.a)

print(sample.b)

**Ex-2: To Declare Public Methods:**

class sample:

def m1():

print('I am m1')

def m2():

print('I am m2')

sample.m1()

sample.m2()

**Here,**

**a, b** is called **Public Variables.**

**m1(), m2()** are called **Public Methods.**

**2. Protected Attributes:**

* **The Protected Attributes can be Declared by using Single Underscore Symbol ( ­\_ ) Prefixing Attribute name.**
* **We Can Access from Anywhere within the Class but outside of the Class only in Child Classes.**

**Ex-1: To Declare Protected Variables.**

class sample:

\_a=5

\_b=8

print(sample.\_a)

print(sample.\_b)

**Ex-2: To Declare Protected Methods**

class sample:

def \_m1():

print('I am m1')

def \_m2():

print('I am m2')

sample.\_m1()

sample.\_m2()

**Here,**

**a, b** is called **Protected Variables.**

**m1(), m2()** are called **Protected Methods.**

**3. Private Attributes:**

* The Private Attributes can be Declared by using **Double Underscore Symbol ( \_\_ )** Prefixing Attribute name.
* **Private Attributes can be Accessed only Within the Class.**

**i.e. From Outside of the Class We Cannot Access.**

**Ex-1: To Declare Private Variables.**

class sample:

\_\_a=5

\_\_b=8

ex-2: To declare private methods:

class sample:

def \_\_m1():

print('I am m1')

def \_\_m2():

print('I am m2')

**Here,**

**a, b** is called **Private Variables.**

**m1(), m2()** are called **Private Methods.**

* **Write a Program to Demonstrate Public, Protected and Private Variables.**

class test:

a=10 #public variable

\_b=20 #protected variable

\_\_c=30 #private variable

print(test.a)

print(test.\_b)

print(test.\_\_c)

**Note:**

**c** is a Private Variable. We Cannot access Outside of the Class.

So, it Returns:

**AttributeError: type object 'test' has no attribute '\_\_c'**

* **Write a Program to Demonstrate Public, Protected and Private Methods.**

class test:

def m1():

print('I am m1')

def \_m2():

print('I am m2')

def \_\_m3():

print('I am m3')

test.m1()

test.\_m2()

test.\_\_m3()

**Here,**

m3 is a Private Method. It cannot Access Outside of Class.

So, it returns:

AttributeError: type object 'test' has no attribute '\_\_m3'

* **Write a Program to Demonstrate Public, Protected and Private Methods.**

class test:

a=10

\_b=20

\_\_c=30

def m1():

print('I am m1 of public method')

def \_m2():

print('I am m2 of protected method')

def \_\_m3():

print('I am m3 of private method')

def display():

print(test.\_\_c)

test.\_\_m3()

print(test.a)

print(test.\_b)

test.m1()

test.\_m2()

test.display()