**PYTHON OOPS CONCEPTS**

TASK (06/11/2019)

* + Classes and Objects in Python……………….2

# Constructor in Python………………………….3

# Inheritance in Python…………………………..4

# Method Overriding in Python………………………...9

## Data Abstraction in python………………………….10

## Data Encapsulation in python……………………...11

Classes and Objects in Python :

* Classes and Objects are basic concepts of Object-Oriented Programming which revolve around the real-life entities.
* Class: A Class is a user defined blueprint or prototype from which objects are created.  It represents the set of properties or methods that are common to all objects of one type.
* Object: An Object is a basic unit of Object-Oriented Programming and it is an entity that has state and behavior.

Creating a class and object in python:

Code:

class Ltts\_Employee: #creating a class

ps\_no=40007319;

def display (self):

print("PS.no: %d "%(self.ps\_no))

emp = Ltts\_Employee() #creating an object

emp.display()

Execution:

PS.no: 40007319

# Constructor in Python:

A constructor is a special type of method (function) which is used to initialize the instance members of the class.

Constructors can be of two types.

1. Parameterized Constructor
2. Non-parameterized Constructor

Creating a Constructor in python:

Code 1:

class Ltts\_Employee

ps\_no=0;

def \_\_init\_\_(self, id): #parameterized constructor

         self.id = id;

def display (self):

print("PS.no: %d "%(self.ps\_no))

emp = Ltts\_Employee(40007319)

emp. display()

Execution:

PS.no: 40007319

Code 2:

class Ltts\_Employee

ps\_no=0;

def \_\_init\_\_(self): #non\_parameterized constructor

         self.ps\_no=40007319

def display (self):

print("PS.no: %d "%(self.ps\_no))

emp = Ltts\_Employee()

emp. display()

Execution:

PS.no: 40007319

# Inheritance in Python:

Inheritance is the capability of one class to derive or inherit the properties from some another class.

**Different forms of Inheritance:**

**1. Single inheritance**: When a child class inherits from only one parent class, it is called as single inheritance. We saw an example above.

Example:

Code:

class Person(object): #Parent Class

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

self.name = name

def getName(self):

return self.name

def isEmployee(self):

return False

class Employee(Person):#Child Class

def isEmployee(self):

return True

emp1= Person("Bhaskar")

print (emp1. getName (), emp1. isEmployee ())

emp2 = Employee("Ganesh")

print (emp2.getName(), emp2.isEmployee())

Execution:

Bhaskar False

Ganesh True

**2. Multiple inheritance**: When a child class inherits from multiple parent classes, it is called as multiple inheritance.

Example:

Code:

class Parent1(object): #Parent Class

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

self.str1 = "Bhaskar"

print("Parent1 class")

class Parent2(object): #Parent Class

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

self.str2 = "Ganesh"

print("Parent2 class")

class Derived(Parent1,Parent2): #Child Class

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

Parent1.\_\_init\_\_(self)

Parent2.\_\_init\_\_(self)

print("Derived class")

def printStrs(self):

print(self.str1, self.str2)

ob = Derived()

ob.printStrs()

Execution:

Parent1 class

Parent2 class

Derived class

Bhaskar Ganesh

**3. Multilevel inheritance**: When we have child and grand child relationship.

Example:

Code:

class Base(object): #Parent Class

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

self.name = name

def getName(self):

return self.name

class Child(Base): #Child Class for Base and Parent Class for GrandChild

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

Base.\_\_init\_\_(self, name)

self.age = age

def getAge(self):

return self.age

class GrandChild(Child): #Child Class

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

Child.\_\_init\_\_(self, name, age)

self.address = address

def getAddress(self):

return self.address

g = GrandChild("Bhaskar", 23, "Ap")

print(g.getName(), g.getAge(), g.getAddress())

Execution:

Bhaskar 23 Ap

**4.Hierarchical inheritance** More than one derived classes are created from a single base.

Example:

Code:

class Employee: #Parent Class

def get\_ps(self):

return 0;

class Bhaskar(Employee): #Child Class

def get\_ps(self):

return 40007319;

class Ganesh(Employee): #Child Class

def get\_ps(self):

return 40007320;

e1 = Employee()

e2 = Bhaskar()

e3 = Ganesh()

print("Employee Ps\_Number",e1.get\_ps());

print("Bhaskar Ps\_Number",e2.get\_ps());

print("Ganesh Ps\_Number",e3.get\_ps());

Execution:

Employee Ps\_Number 0

Bhaskar Ps\_Number 40007319

Ganesh Ps\_Number 40007320

**5. Hybrid inheritance**: This form combines more than one form of inheritance. Basically, it is a blend of more than one type of inheritance.

## Method Overriding in Python:

## Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes.

Example:

Code:

class Employee: #Parent Class

def get\_ps(self):

return 0;

class Bhaskar(Employee): #Child Class

def get\_ps(self):

return 40007319;

e1 = Employee()

e2 = Bhaskar()

print("Employee Ps\_Number",e1.get\_ps());

print("Bhaskar Ps\_Number",e2.get\_ps());

Execution:

Employee Ps\_Number 0

Bhaskar Ps\_Number 40007319

## Data abstraction in python:

## In python, we can also perform data hiding by adding the double underscore (\_\_) as a prefix to the attribute which is to be hidden. After this, the attribute will not be visible outside of the class through the object.

Example:

Code:

## class Object\_Count:

## \_\_count = 0;

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

## Object\_Count.\_\_count = Object\_Count.\_\_count+1

## def display(self):

## print("The number of objects created for this class:",Object\_Count.\_\_count)

## ob1 = Object\_Count()

## ob2 = Object\_Count()

## ob1.display()

Execution:

The number of objects created for this class:  2

Data Encapsulation in python:

In an object oriented python program, you can restrict access to methods and variables. This can prevent the data from being modified by accident and is known as encapsulation.

Example:

Code:

Code:

class Car:

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

self.\_\_updateSoftware()

def drive(self):

print('driving')

def \_\_updateSoftware(self):#we can’t access this method using objects

print('updating software')

car1 = Car()

car1.drive()

Execution:

updating software

driving