Python OOPs Concept

An object-Oriented paradigm is to design the program using classes and objects. The object is related to real world entities such as book, house, pencil

class

Syntax

```
class ClassName:
        stmt 1
        -
        -
        -
        stmt n
```

object

- Everything in python is an object and almost everything has attributes and methods.

- All functions have built-in attribute __doc__ which returns the docstring defined in the function source code.

Example

```
class car :
      def __init__(self, modelname, year):
            self.modelname = modelname
            self.year = year

      def display(self):
            print(self.modelname, self.year))

c1 = car("Toyota", 2016)
c1.display
```

Output

```
Toyota 2016
```

Classes and Objects in python

class Syntax

```
class className :
        # statement_ suite
```

Objects Syntax

```
#declare object of a class
  object_name  =  class_Name(arguments)
```

Example

```
class Person :
      def __init__(self, name, age):
           self.name  = name
           Self.age  = age

      def greet(self) :
            print("Hello, my name is" + self.name)

  # create a new instance
   person1 = Person("A", 20)
   person1.great()
```

Date
Page

The Self - parameter

The self parameter refers to the current instance of the class and accesses the class variables. We can use anything instead of self, but it must be the first parameter of any function which belongs to the class.

__init__ method

- The __init__ method is a special method used to initialize the attributes of an object when it's created from a class.
- It's often referred to as a constructor because it's called automatically when a new instance of the class is created.
- The first parameter of the __init__ method named self, which refers to the instance being created. This parameter is required in all instance methods of a class.

Date
Page

Python Constructor

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

- In C++, Java the constructor has the same name as its class, but it trats constructor differently in Python. It is used to create an object.

Constructors can have two types :-
(1) Parameterized constructor
(2) Non - parameterize constructor

creating the constructor

__init__ ()

Example

```
class Employee :
    def __init__ (self, name, id) :
        self.id = id
        self.name = name

    def display (self) :
        print ("ID %d \n Name : %s " % (self.id, self.nam

emp1 = Employee ("John" ,101)

emp1.display ()
```

python built-in class functions

(1) getattr(obj, name, default)

(2) setattr(obj, name, value)

(3) delattr(obj, name)

(4) hasattr(obj, name)

Built in class attributes

(1) __dict__

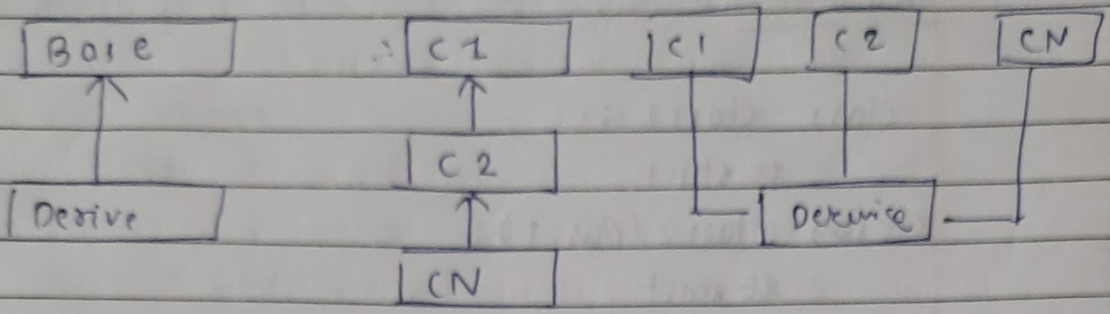
(2) __doc__

(3) __name__

(4) __module__

(5) __bases__

Python Inheritance



Syntax

```
class derived-class(base-class):
        #stmts
```

A class can inherit multiple classes by mentioning all of them inside the brackets.

Syntax

```
class derive-class(<base class 1>, <b-class 2>....):
        #stmt1
```

Example

```
    class Animal:
        def speak(self):
            print("Animal speaking")

    class Dog(Animal):
        def bark(self):
            print("dog barking")
d = Dog()
d.bark()
d.speak()
```

Python Multi-level inheritance

Syntax

```
class class1 :
    #stmt
class class2 (class1):
    #stmt
class class3 (class2) :
    #stmt
```

Example

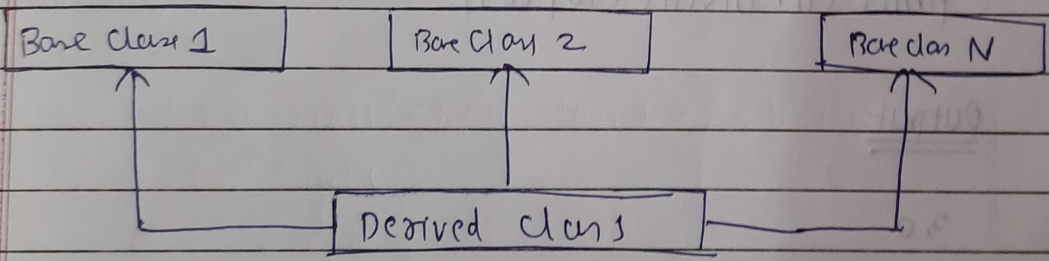
```
class Animal :
    def speak(self):
        print("Animal speaking")

class Dog (Animal) :
    def bark (self) :
        print("dog barking")

class DogChild (Dog) :
    def eat (self) :
        print("Eating bread....")

d = Dogchild ()
d. bark()
d. speak()
d. eat ()
```

Date
Page

Python Multiple Inheritance

Syntax

```
class Base1 :
    #

class Base2 :
    #

-

-

-

class BaseN :
    #

class Derived (Base1, Base 2, Base N) :
    #
```



Example

```
class Calculation1 :
    def Summation (Self, a, b) :
        return a+b;

class Calculation2 :
    def Multiplication (Self, a, b) :
        return a*b;

class Derived (Calculation 1, Calculation2) :
    def Divide (self, a, b) :
        return a/b

d= Derived ()
print (d. Summation(10, 20))
print (d. Multiplication (10, 20))
print (d. Divide (10, 20))
```

Output

```
30
200
0.5
```

Date
Page

Method Overiding

Example

```
class Animal :
        def speak(self):
            print("speaking")
class Dog (Animal):
        def speak(self):
            print("Barking")

d=Dog()
d.speak()
```

Output

```
Barking
```

Abstraction in Python

- Abstraction is used to hide the internal functionality of the function from the users.

- The users only interact with the basic implementation of the function, but inner working is hidden

Date
Page

Data Abstraction in python

- We perform data hiding by adding the double underscore (__) as a prefix to the attribute which is to be hidden.

Example

```
class Employee :
    __Count = 0

    def __init__(self):
        Employee__count = Employee__count + 1

    def display(self):
        print("The number of empolyees", Employes__count)

emp = Employee()
emp2 = Employee()

try :
    print(emp__count)

finally :
    emp.display()
```