Python OOPS Concepts

Class:

- A class is a collection of objects. A class contains the blueprints or the prototype from which the objects are being created. It is a logical entity that contains some attributes and methods.
- Some points on Python class:
- Classes are created by keyword class.
- Attributes are the variables that belong to a class.
- Attributes are always public and can be accessed using the dot (.) operator. Eg.: Myclass.Myattribute

```
Class Person:
        “This is a person class”
        age=10
        def greet(self):
                print(‘Hello’)
Print(Person.age)
Print(Person.greet)
Print(Person.__doc__)
```

```
Class Student:
    Clg = “RVIT”     #class variable
    def __init__(self,rollno,name):
        self.rollno=rollno
        self.name=name
    def display(self):
        print(“student name:”,self.name)
        print(“student rollno:”,self.rollno)
        print(“college:”,student.clg)

Student1=student(“23”, “Tarun”)
Student1.display()

Student2=student(“15”, “”)
```

Object:

- The object is an entity that has a state and behavior associated with it. It may be any real-world object like a mouse, keyboard, chair, table, pen, etc. Integers, strings, floating-point numbers, even arrays, and dictionaries, are all objects.
- **An object consists of :**
- **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
  - State or Attributes can be considered as the breed, age, or color of the dog.
- **Behavior:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
  - The behavior can be considered as to whether the dog is eating or sleeping.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.
  - The identity can be considered as the name of the dog.

Create A Class:

Create a class named MyClass, with a property named x:

- class MyClass:
  x = 5

Create A Object:

Create an object named p1, and print the value of x:

- p1 = MyClass()
  print(p1.x)

```
class Person:
        "This is a person class"
        age = 10
        def greet(self):
                print('Hello')
# create a new object of Person class
harry = Person()
# Output: <function Person.greet>
print(Person.greet)
# Output: <bound method Person.greet of
<__main__.Person object>> print(harry.greet)
# Calling object's greet() method
# Output: Hello
harry.greet()
```

The __init__() Function

- All classes have a function called __init__(), which is always executed when the class is being initiated.
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:
- class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  p1 = Person("John", 36)
  print(p1.name)
  print(p1.age)
- **Note:** The __init__() function is called automatically every time the class is being used to create a new object.

Object Methods

- Objects can also contain methods. Methods in objects are functions that belong to the object.Let us create a method in the Person class:
- Insert a function that prints a greeting, and execute it on the p1 object:
- class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def myfunc(self):
    print("Hello my name is " + self.name)
  p1 = Person("John", 36)
  p1.myfunc()
- **Note:** The self parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.

Self Parameter:

- The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.
- It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:
- class Person:

```
    def __init__(mysillyobject, name, age):
        mysillyobject.name = name
        mysillyobject.age = age
    def myfunc(abc):
        print("Hello my name is " + abc.name)
p1 = Person("John", 36)
p1.myfunc()
```

- Modify Object Properties:

  Set the age of p1 to 40:

  ```
  P1.age = 40
  ```

- Delete Object Properties:

  Delete the age property from the p1 object:

  ```
  del p1.age
  ```

- Delete Objects

  You can delete objects by using the del keyword:

  ```
  del p1
  ```

- The pass Statement

  class definitions cannot be empty, but if you for some reason have a class definition with no content, put in the pass statement to avoid getting an error.

  ```
  class Person:
      pass
  ```

EXERCISE :

1.Creating Classes

```
class Employee:
        'Common base class for all employees'
        empCount = 0
        def __init__(self, name, salary):
                self.name = name
                self.salary = salary
                Employee.empCount += 1
        def displayCount(self):
                print "Total Employee %d" %
Employee.empCount
        def displayEmployee(self):
                print "Name : ", self.name, ", Salary: ",
self.salary
```

2.Creating Instance Objects:

```
#"This would create first object of Employee
class"
emp1 = Employee("Zara", 2000)
#"This would create second object of Employee
class"
emp2 = Employee("Manni", 5000)
```

3. Accessing Attributes

```
 emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" %
Employee.empCount
```

```
#!/usr/bin/python
class Employee:
'Common base class for all employees'
empCount = 0
def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1
def displayCount(self):
        print "Total Employee %d" % Employee.empCount
def displayEmployee(self):
        print "Name : ", self.name, ", Salary: ", self.salary
"This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
"This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
```

```
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" %
Employee.empCount
```

You can add, remove, or modify attributes of classes and objects at any time −

```
emp1.age = 7 # Add an 'age' attribute.
emp1.age = 8 # Modify 'age' attribute.
del emp1.age # Delete 'age' attribute.
```

Exercise:

1. Create a Vehicle class with max_speed and mileage instance attributes.
2. Create a child class Bus that will inherit all of the variables and methods of the Vehicle class.
3. Write a Python class to convert an integer to a roman numeral.
4. Write a Python class to convert a roman numeral to an integer.
5. Write a Python class to find a pair of elements (indices of the two numbers) from a given array whose sum equals a specific target number.
   Note: There will be one solution for each input and do not use the same element twice.
   Input: numbers= [10,20,10,40,50,60,70], target=50
   Output: 3, 4

6.Write a Python class to find the three elements that sum to zero from a set of n real numbers.

Input array : [-25, -10, -7, -3, 2, 4, 8, 10]

Output : [[-10, 2, 8], [-7, -3, 10]]

7. Write a Python class to implement pow(x, n).

8. Write a Python class to reverse a string word by word.

Input string : ‘Nitin'

Expected Output : ‘nitiN'

Inheritance :

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- **Parent class** is the class being inherited from, also called base class.
- **Child class** is the class that inherits from another class, also called derived class.
- Inheritance is the capability of one class to derive or inherit the properties from another class. The benefits of inheritance are:
- It represents real-world relationships well.
- It provides **reusability** of a code. We don’t have to write the same code again and again. Also, it allows us to add more features to a class without modifying it.
- It is transitive in nature, which means that if class B inherits from another class A, then all the subclasses of B would automatically inherit from class A.

Example 1 :

```
class Animal:
    def speak(self):
        print("Animal Speaking")
#child class Dog inherits the base class Animal
class Dog(Animal):
    def bark(self):
        print("dog barking")
d = Dog()
d.bark()
d.speak()
```

EXAMPLE 2

```
# A Python program to demonstrate inheritance
# Base or Super class. Note object in bracket.
# (Generally, object is made ancestor of all classes)
# In Python 3.x "class Person" is
# equivalent to "class Person(object)"
class Person(object):
    # Constructor
    def __init__(self, name):
        self.name = name

    # To get name
    def getName(self):
        return self.name
# To check if this person is an employee
    def isEmployee(self):
        return False
```

```
# Inherited or Subclass (Note Person in bracket)
class Employee(Person):
        # Here we return true
        def isEmployee(self):
                return True
# Driver code
emp = Person("Geek1") # An Object of Person
print(emp.getName(), emp.isEmployee())

emp = Employee("Geek2") # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

EXAMPLE #3

```
# Python code to demonstrate how parent constructors
# are called.
# parent class
class Person( object ):
        # __init__ is known as the constructor
        def __init__(self, name, idnumber):
                self.name = name
                self.idnumber = idnumber
        def display(self):
                print(self.name)
                print(self.idnumber)
# child class
class Employee( Person ):
        def __init__(self, name, idnumber, salary, post):
                self.salary = salary
                self.post = post
                # invoking the __init__ of the parent class
                Person.__init__(self, name, idnumber)
```

```
# creation of an object variable or an instance
a = Employee('Rahul', 886012, 200000,
"Intern")

# calling a function of the class Person using its
instance
a.display()
```

Example #4

```
class Bank:
    def getroi(self):
        return 10;
class SBI(Bank):
    def getroi(self):
        return 7;

class ICICI(Bank):
    def getroi(self):
        return 8;
b1 = Bank()
b2 = SBI()
b3 = ICICI()
print("Bank Rate of interest:",b1.getroi());
print("SBI Rate of interest:",b2.getroi());
print("ICICI Rate of interest:",b3.getroi());
```

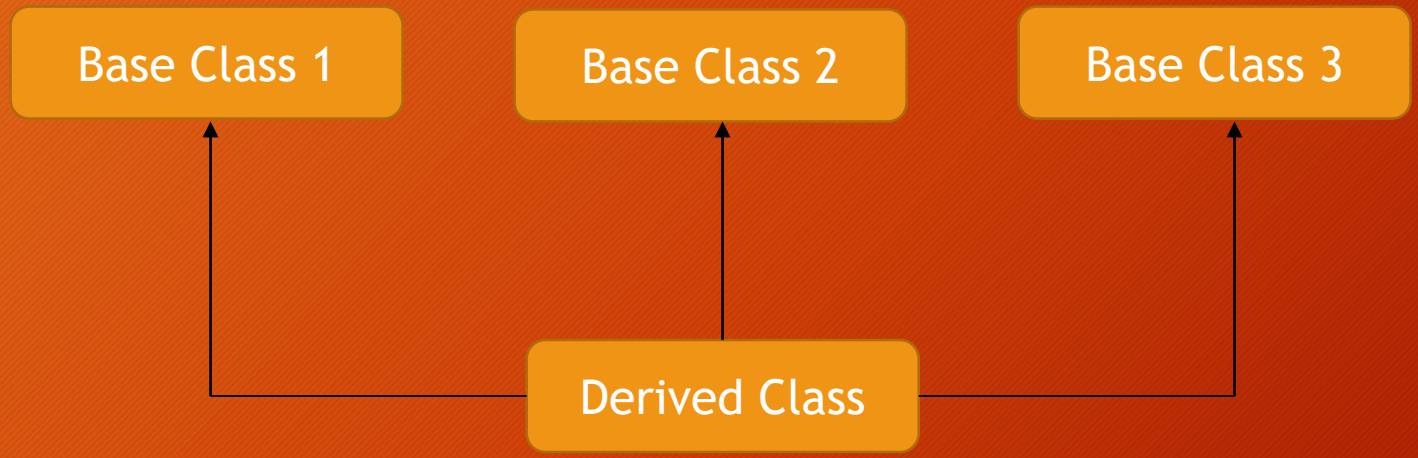
Multilevel Inheritance:

- Multi-Level inheritance is possible in python like other object-oriented languages. Multi-level inheritance is archived when a derived class inherits another derived class. There is no limit on the number of levels up to which, the multi-level inheritance is archived in python.

```
Class Animal:
    def speak(self):
        print("Animal Speaking")
#The child class Dog inherits the base class Animal
class Dog(Animal):
    def bark(self):
        print("dog barking")
#The child class Dogchild inherits another child class Dog
class DogChild(Dog):
    def eat(self):
        print("Eating bread...")
```

Multiple Inheritance:

- Python provides us the flexibility to inherit multiple base classes in the child class.



```
class Calculation1:
    def Summation(self,a,b):
        return a+b;
class Calculation2:
    def Multiplication(self,a,b):
        return a*b;
class Derived(Calculation1,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))
```

The issubclass(sub,sup) method :

- The issubclass(sub, sup) method is used to check the relationships between the specified classes. It returns true if the first class is the subclass of the second class, and false otherwise.

Example :

```
class Calculation1:
    def Summation(self,a,b):
        return a+b;
class Calculation2:
    def Multiplication(self,a,b):
        return a*b;
class Derived(Calculation1,Calculation2):
    def Divide(self,a,b):
        return a/b;
d = Derived()
print(issubclass(Derived,Calculation2))
print(issubclass(Calculation1,Calculation2))
```

The isinstance (obj, class) method :

- The isinstance() method is used to check the relationship between the objects and classes. It returns true if the first parameter, i.e., obj is the instance of the second parameter, i.e., class.

Example :

```
class Calculation1:
    def Summation(self,a,b):
        return a+b;
class Calculation2:
    def Multiplication(self,a,b):
        return a*b;
class Derived(Calculation1,Calculation2):
    def Divide(self,a,b):
        return a/b;
d = Derived()
print(isinstance(d,Derived))
```

Data Abstraction :

- Abstraction is an important aspect of object-oriented programming. 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.

```
class Employee:
    __count = 0;
    def __init__(self):
        Employee.__count = Employee.__count+1
    def display(self):
        print("The number of employees",Employee.__count)
emp = Employee()
emp2 = Employee()
try:
    print(emp.__count)
finally:
    emp.display()
```

1. Get Date And Time :

Python get today's date

- from datetime import date

  today = date.today()

  print("Today's date:", today)

Here, we imported the date class from the datetime module. Then, we used the date.today() method to get the current local date.

2. Current date in different formats:

```
from datetime import date
today = date.today()
# dd/mm/YY
d1 = today.strftime("%d/%m/%Y")
print("d1 =", d1)
# Textual month, day and year
d2 = today.strftime("%B %d, %Y")
print("d2 =", d2)
# mm/dd/yy
d3 = today.strftime("%m/%d/%y")
print("d3 =", d3)
# Month abbreviation, day and year
d4 = today.strftime("%b-%d-%Y")
print("d4 =", d4)
```

3: Get the current date and time:

```
from datetime import datetime
# datetime object containing current date and time
now = datetime.now()
print("now =", now)
# dd/mm/YY H:M:S
dt_string = now.strftime("%d/%m/%Y %H:%M:%S")
print("date and time =", dt_string)
```