Object Oriented Programming

Object-oriented programming (OOP) is a software programming model constructed around objects. This model compartmentalizes data into objects (data fields) and describes object contents and behaviour through the declaration of classes (methods).

Object-oriented programming allows for simplified programming. Its benefits include reusability, refactoring, extensibility, maintenance and efficiency.

OOPs Concept With Real Life Example

An architect will have the blueprints for a house. Those blueprints will be plans that explain exactly what properties the house will have and how they are all laid out. However it is just the blueprint, you can't live in it. Builders will look at the blueprints and use those blueprints to make a physical house. They can use the same blueprint to make as many houses as they want....each house will have the same layout and properties. Each house can accommodate it's own families.

The blueprint is the class...the house is the object. The people living in the house are data stored in the object's properties.

Diff. Between Procedure Oriented and Object Oriented Programming:

1. In procedural programming, a program is divided into portions called functions, while in object oriented programming, the program is divided into portions called objects.
2. In procedural programming, the focus is placed on the functions and sequence of actions to be performed and not on data. In object-oriented programming, however, the focus is placed on the data and not the procedures or functions.
3. Procedural programming follows the "top-down" approach. While object-oriented programming follows the "bottom-up" approach

Advantages of oops

- OOP offers easy to understand and a clear modular structure for programs.
- Objects created for Object-Oriented Programs can be reused in other programs. Thus it saves significant development cost.
- Large programs are difficult to write, but if the development and designing team follow OOPS concept then they can better design with minimum flaws.
- It also enhances program modularity because every object exists independently.

Disadvantages of oops

- The OOP programs design is tricky.Object Oriented programs require a lot of work to create. Specifically, a great deal of planning goes into an object oriented program well before a single piece of code is ever written. Initially, this early effort was felt by many to be a waste of time. In addition, because the programs were larger (see above) coders spent more time actually writing the

program.

- Object Oriented programs are slower than other programs, partially because of their size. Other aspects of Object Oriented Programs also demand more system resources, thus slowing the program down.

Important Terminology

Class: Class can be defined as a collection of objects. A *class* is the blueprint from which individual objects are created. class describes the contents of the objects that belong to it: it describes an aggregate of data fields (called instance variables), and defines the operations (called methods). def is used to define a function, class is used to define a *class*. Class name, begin with capital letter.

Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.

Object:

1. A unique instance of a data structure that's defined by its class. An object consist of both data members (class variables and instance(object) variables) and methods.
2. Objects are the basic run-time entities in an object oriented system.They may represent a person, a place, a bank account, a table of data or any item that the program must handle
3. Objects have two components:
    1. Data (i.e., Attributes)
    2. Behaviours (i.e., Methods)

Object Attributes:

An *attribute* is an element that takes a value and is associated with an object. Apply the power of the almighty dot objectname.attributename .To handle to another object.

Syntax: objectname.attributename = anotherobject.

Example: self.name1=name1

Object methods:

Function that is associated with a class is called object method. This terminology is important because it helps us to differentiate between functions and variables which are independent and those which belong to a class or object.

Object variables

Object variables are owned by each individual object of the class. In this case, each object has its own copy of the field i.e. they are not shared and are not related in any way to the field by the same name in a different instance.

Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class. Instance is a variable that holds the memory address of the object.

Instantiation: The creation of an instance of a class.

Method: Method is a function that is associated with an object. It is a special kind of function that is defined in a class definition. Define a method in a class by including function definitions within the scope of the class block. There must be a special first argument self in all of method definitions which gets bound to the calling instance. There is usually a special method called __init__ in most classes

__init__ Method:

__init__is the default constructor(Constructors are used for allocating memory space for variables. It is special function which gets activated automatically when object of that class is created.) Inside the class body, we define this method – this is our object's method. When we call the class object, a new instance of the class is created, and the __init__ method on this new object is immediately executed with all the parameters that we passed to the class object. The purpose of this method is thus to set up a new object using data that we have provided. This is method usually does some initialization work .An __init__ method can take any number of arguments. However, the first argument self in the definition of __init__ is special.

Self:

in every method definition we have self as the first parameter, and we use this variable inside the method bodies – but we don't appear to pass this parameter in. This is because whenever we call a method on an object, *the object itself* is automatically passed in as the first parameter. This gives us a way to access the object's properties from inside the object's methods. The first argument of every method is a reference to the current instance of the class. In __init__, self refers to the objects currently being created; so , in other class methods, it refers to the instance whose method was called. You do not give a value for this parameter (self) when you call the method, python will provide it. Although you must specify self explicitly when defining the method, you don't include it when calling the method. Python passes it for you automatically.

Example

Write a program with class Employee that keeps a track of the number of employees in an organization and also stores their fist name, last name, email id and salary details?define two methods to return employee full name and salary raise?

```
class Employee:
        num_of_emps=0
        raise_amount=1.04
        def __init__(self,first,last,pay):
                self.f=first
                self.l=last
                self.p=pay
                self.email=first +'.'+last+'@gmail.com'
                Employee.num_of_emps+=1
        def fullname(self):
```

```
            return '{} {}'.format(self.f,self.l)
        def apply_raise(self):
            self.p=int(self.p*Employee.raise_amount)
print(Employee.num_of_emps)
emp_1=Employee('john','s',10000)
emp_2=Employee('test','user',5000)
print(Employee.num_of_emps)
print(emp_1.email)
print(emp_2.email)
print(emp_2.fullname())
print(Employee.fullname(emp_2))
emp_1.apply_raise()
print(emp_1.p)
```

Exercise 1

Write a program with class Person and define a function to greet the person with his/her name in the class?

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

    def say_hi(self):
        print('Hello, Good morning' , self.name)

p = Person('Sai')
p.say_hi()
# The previous 2 lines can also be written as
# Person('Sai').say_hi()
```

Output:

Hello, Good morning Sai

Exercise 2

Write a program with Song class and deifne one function in the class to print the lyrics line by line? Now create two objects to test the function?

```
class Song():

    def __init__(self, lyrics):
        self.ly = lyrics
    def sing_me_a_song(self):
        for line in self.ly:
            print line
happy_bday = Song(["Happy birthday to you",
```

```
"I don't want to get sued",
"So I'll stop right there"])

bulls_on_parade = Song(["They rally around tha family",
"With pockets full of shells"])
happy_bday.sing_me_a_song()

bulls_on_parade.sing_me_a_song()
```

Exercise 3:

Define one class called Person with name, surname, birthdates, address, telephone and email .Define age method in the class block to calculate the age of the person and print the person age(years)?

```
from datetime import date
class Person:
        def __init__(self, name, surname, birthdate, address, telephone, email):
                self.name = name
                self.surname = surname
                self.birthdate = birthdate
                self.address = address
                self.telephone = telephone
                self.email = email

def age(self):
        age = date.today()-self.birthdate
        age= int((age.days)/365.25)
        return age

person = Person(
"abc",
"xyz",
date(1986, 1, 1), # year, month, day
"IIIT",
"Basar",
"abc.xyz@example.com"
)

print(person.name)
print(person.email)
print(person.age())
```

Exercise 4

Create a student class with student name and 3 subject marks as list.Create 3 objects by taking user input?Display the name and average marks of each student?Define the average marks calculation function in the class block?

```
class Student:

        def __init__(self, name,marks):
                self.name = name
                self.maths = marks[0]
                self.english = marks[1]
                self.sl= marks[2]

        def avg(self):
                total=(self.maths+self.english+self.sl)/3
                return total

mylist=[]
for i in range(1,3):
        obj=raw_input("Enter student object")
        name=raw_input("Enter student name")
        marks=[]
        for j in range(0,3):
                m=input("Enter the marks")
                marks.append(m)
        obj=Student(name,marks)
        mylist.append(obj)
for i in mylist:
        print i.name,i.avg()
'''
s1 = Student("abc",[78,99,55])
print s1.avg()
'''
```

Fundamental concepts of OOP in python:

The four major principles of the object orientation are:

1. **Encapsulation**

   Encapsulation is all about binding the data variables and functions together in class.

   Example: Our Legs are binded to help us walk. Our hands, help us hold things.

2. **Data Abstraction**

Abstraction means, showcasing only the required things to the outside world while hiding the details.

Example: Human Being's can talk, walk, hear, eat, but the details are hidden from the outside world. We can take our skin as the Abstraction factor in our case, hiding the inside mechanism.

3. **Polymorphism**

Polymorphism is defined as the implementation of the same method with different attributes in a different context. It is a feature, which lets us create functions with same name but different arguments, which will perform different actions. That means, functions with same name, but functioning in different ways.

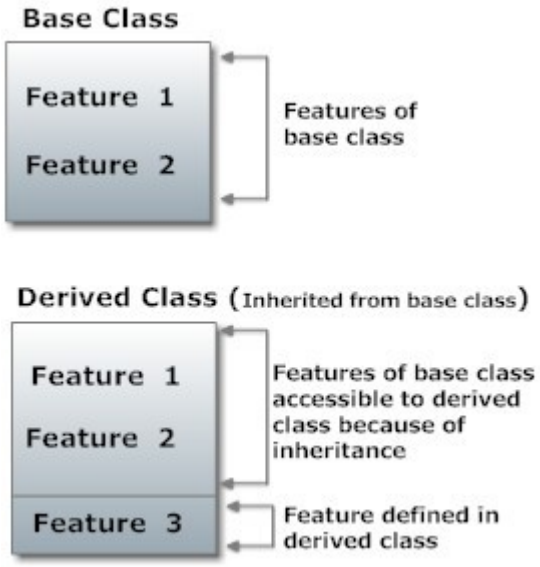
Example

For example, we have a class named shape with a method name buildshape(). We need to use it in the class Circle, class triangle and the class quadrilateral. So for every class we use the buildshape() method but with different attributes.

4. **Inheritance**

Inheritance is a powerful feature in object oriented programming. It refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it inherits is called the base (or parent) class. Derived class inherits features from the base class, adding new features to it. This results into re-usability of code.

Pictorial representation



Syntax

class DerivedClass(BaseClass):
body_of_derived_class

Advantages of inheritance

- Inheritance promotes reusability. When a class inherits or derives another class, it can access all the functionality of inherited class.

- Reusability enhanced reliability. The base class code will be already tested and debugged.
- As the existing code is reused, it leads to less development and maintenance costs.
- Inheritance makes the sub classes follow a standard interface.
- Inheritance helps to reduce code redundancy and supports code extensibility.
- Inheritance facilitates creation of class libraries.

Disadvantages of inheritance

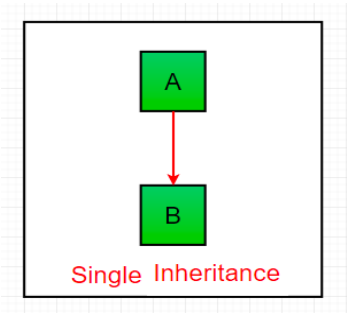
- Inherited functions work slower than normal function as there is indirection.

- Improper use of inheritance may lead to wrong solutions.
- Often, data members in the base class are left unused which may lead to memory wastage.
- Inheritance increases the coupling between base class and derived class. A change in base class will affect all the child classes.

Types of Inheritance

1. Singe inheritance
2. Hierarchical Inheritance
3. Multilevel Inheritance
4. Multiple Inheritance

Single Inheritance : In single inheritance, subclasses inherit the features of one superclass. In image below, the class A serves as a base class for the derived class B.



Example 1

Write a program to inherit the features of employee class to create a developer class. Here inherit the employee first name ,last name and pay from the super class.And add language feature the derived(Developer) class?Now print developer 1 email id and developer 2 programming language?

```
class Employee:
	num_of_emps=0
	raise_amount=1.04
	def __init__(self,first,last,pay):
		self.first=first
		self.last=last
		self.pay=pay
		self.email=first +'.'+last+'@gmail.com'
		Employee.num_of_emps+=1
	def fullname(self):
		return '{} {}'.format(self.first,self.last)
	def apply_raise(self):
		self.pay=int(self.pay*Employee.raise_amount)
class Developer(Employee):#Inheriting the class called employee

	def __init__(self,first,last,pay,prog_lang):
		Employee.__init__(self,first,last,pay)#Inheriting all the  variables from
employee(base)class
		self.prog_lang=prog_lang#And creating onemore instance variable called
prog_lang in derived class called Developer
dev_1=Developer('a','b',50000,'python')
dev_2=Developer('c','d',60000,'java')
print(dev_1.email)
print(dev_2.prog_lang)
```

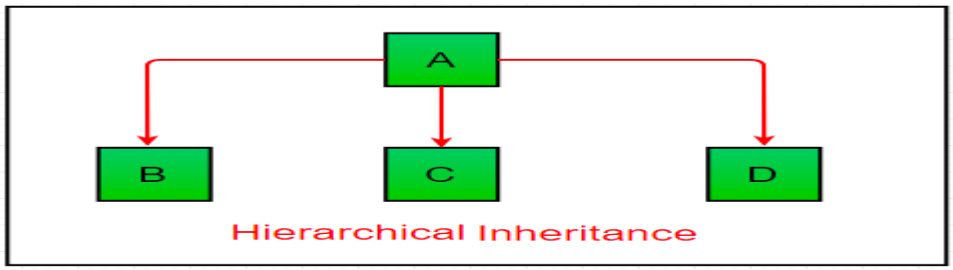
Exercise 1

Create a parent class Car and inherit the properties from parent class to derived class ElectricCar? Print the details of car model, car color?

```
class Car(object):
	condition = "new"
	def __init__(self, model, color, mpg):
		self.model = model
		self.color = color
		self.mpg = mpg

	def display_car(self):
		return "This is a %s %s with %s MPG. " % (self.color, self.model, self.mpg)

	def drive_car(self):
		self.condition = "used"
```

```
class ElectricCar(Car):
        def __init__(self, model, color, mpg, battery_type):
                Car.__init__(self, model, color, mpg)
                self.battery_type = battery_type

        def display_car(self):
                inherit_Str = Car.display_car(self)
                return inherit_Str + "It has a %s battery. " % (self.battery_type)

my_car = ElectricCar("Ford", "black", 95,"molten salt")
print my_car.display_car()
```

Hierarchical Inheritance : In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one sub class.In below image, the class A serves as a base class for the derived class B,C and D.



Example 1

Create a Schoolmember class as a super class and inherit the name and age feature to the derived classes(Student,Teacher).Add salary feature to the Teacher class and Total marks feature to the student class? Display teacher salary and student total marks?

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

class Teacher(Schoolmember):
        def __init__(self,name,age,salary):
                Schoolmember.__init__(self,name,age)
                self.salary=salary
        def output(self):
                return ("teacher name is {}, age is {} and salary is
{}".format(self.name,self.age,self.salary))
class Student(Schoolmember):
        def __init__(self,name,age,totalmarks):
                Schoolmember.__init__(self,name,age)
                self.totalmarks=totalmarks
        def output(self):
                return ("Student name is {}, age is {} and Totalmarks
{}".format(self.name,self.age,self.totalmarks))
std1=Student('a',18,90)
```

```
std2=Student('b',19,85)
teacher1=Teacher('c',30,5000)
teacher2=Teacher('d',31,10000)
print teacher1.salary
print std2.totalmarks
print Teacher.output(teacher2)
print Student.output(std1)
```

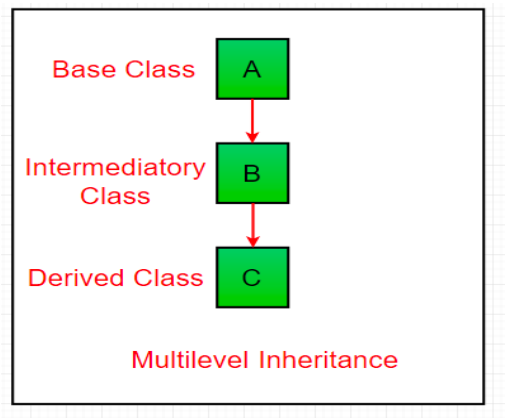
Exercise 1

Create a Companymember class as a super class and inherit the name, designation and age features to the derived classes(Factorystaff,Officestaff).Add Overtime Allowance to the Factorystaff class and Travelling Allowance feature to the Officestaff class? Display the office and factory staff details?

```
# python-inheritance.py
class CompanyMember:
        '''Represents Company  Member.'''
        def __init__(self, name, designation, age):
                self.name = name
                self.designation = designation
                self.age = age
        def tell(self):
                '''Details of an employee.'''
                #print ('Name: ', self.name,'\nDesignation : ',self.designation, '\nAge : ',self.age)

class FactoryStaff(CompanyMember):
        '''Represents a Factory Staff.'''
        def __init__(self, name, designation, age, overtime_allow):
                CompanyMember.__init__(self, name, designation, age)
                self.overtime_allow = overtime_allow
                CompanyMember.tell(self)
                #print ('Overtime Allowance : ',self.overtime_allow)

class OfficeStaff(CompanyMember):
        '''Represents a Office Staff.'''
        def __init__(self, name, designation, age, travelling_allow):
                CompanyMember.__init__(self, name, designation, age)
                self.travelling = travelling_allow
                CompanyMember.tell(self)
                #print('Traveling Allowance : ',self.travelling)
'''
f1=FactoryStaff('abc','Electrician',39,250)
o1=OfficeStaff('xyz','office Assistance',45,202)
print f1.age
'''
fac_staff=[]
off_staff=[]
for i in range(1,3):
```

```
        obj=raw_input("read object")
        name=raw_input("read name")
        designation=raw_input("Enter designation")
        age=input("Enter the age")
        overtime=input("enter overime")
        obj=FactoryStaff(name,designation,age,overtime)
        fac_staff.append(obj)

for i in range(1,3):
        obj=raw_input("read object")
        name=raw_input("read name")
        designation=raw_input("Enter designation")
        age=input("Enter the age")
        travelling=input("enter overime")
        obj=OfficeStaff(name,designation,age,travelling)
        off_staff.append(obj)

for i in fac_staff:
        print ('Name: ', i.name,'\nDesignation : ',i.designation, '\nAge : ',i.age,'Overtime Allowance :
',i.overtime_allow)
for i in off_staff:
        print ('Name: ', i.name,'\nDesignation : ',i.designation, '\nAge : ',i.age,'Traveling Allowance :
',i.travelling)
```

Multilevel Inheritance : In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C.



Example 1

```
class Base:
        pass

class Derived1(Base):
        pass

class Derived2(Derived1):
```

```
    pass
```

Exercise 1

```
class Animal:
    def eat(self):
        print 'Eating...'
class Dog(Animal):
    def bark(self):
        print 'Barking...'
class BabyDog(Dog):
    def weep(self):
        print 'Weeping...'
d=BabyDog()
d.eat()
d.bark()
d.weep()
```

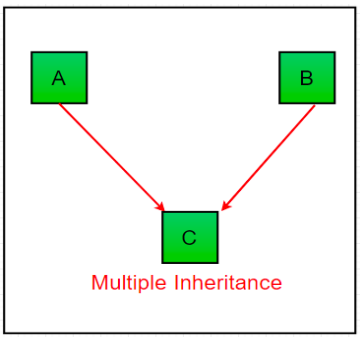
Exercise 2

Write a python code by creating a student class with student details(name,age,gender)as a base class and create test class(with student class and different subject marks)by inherit base class "student". Now create marks class as a child class to "test" class. Define display method in marks class to display student name,age,gender and total marks. Now create an object to call the 3 classes?

```
# Define a class as 'student'
class student:
# Method
    def getStudent(self):
        self.name = raw_input("Name: ")
        self.age = input("Age: ")
        self.gender = raw_input("Gender: ")

# Define a class as 'test' and inherit base class 'student'
class test(student):
# Method
    def getMarks(self):
        self.stuClass = input("Class: ")
        print("Enter the marks of the respective subjects")
        self.literature = input("Literature: ")
        self.math = input("Math: ")
        self.biology = input("Biology: ")
        self.physics = input("Physics: ")

# Define a class as 'marks' and inherit derived class 'test'
class marks(test):
# Method
    def display(self):
        print("Name: ",self.name)
        print("Age: ",self.age)
```

```
                print("Gender: ",self.gender)
                print("Study in: ",self.stuClass)
                print("Total Marks: ", self.literature + self.math + self.biology + self.physics)
m1 = marks()
# Call base class method 'getStudent()'
m1.getStudent()
# Call first derived class method 'getMarks()'
m1.getMarks()
# Call second derived class method 'display()'
m1.display()
```

Multiple Inheritance : In Multiple inheritance ,one class can have more than one super class and inherit features from all parent classes.



Example 1

```
class Base1:
        pass

class Base2:
        pass

class MultiDerived(Base1, Base2):
        pass
```

Exercise 1

Write a Python code by creating two classes called Person and Student. Define name and age methods in Person class and define ID method in Student class. Now create Resident class as a child class to the Person and Student classes. Finally create an object to the Resident class, display the name and ID of the student?

```
#definition of the class starts here
class Person:
#defining constructor
        def __init__(self, personName, personAge):
                self.name = personName
                self.age = personAge
```

```
#defining class methods
        def showName(self):
                print(self.name)

        def showAge(self):
                print(self.age)

#end of class definition

# defining another class
class Student: # Person is the
        def __init__(self, studentId):
                self.studentId = studentId

        def getId(self):
                return self.studentId


class Resident(Person, Student): # extends both Person and Student class
        def __init__(self, name, age, id):
                Person.__init__(self, name, age)
                Student.__init__(self, id)


# Create an object of the subclass
resident1 = Resident('John', 30, '102')
resident1.showName()
print(resident1.getId())
```

Method overriding

Override means having two methods with the same name but doing different tasks. It means that one of the methods overrides the other.

If there is any method in the superclass and a method with the same name in a subclass, then by executing the method, the method of the corresponding class will be executed.

Example 1

```
class Parent(object):
        def __init__(self):
                self.value = 5

        def get_value(self):
                return self.value

class Child(Parent):
        def get_value(self):
                return self.value + 1
c = Child()
```

```
print c.get_value()
```

Exercise 1

Write a python code to Create a class called rectangle with area method and create a square class as a child class to the rectangle class with area method.Display the area of rectangel and area of square?

```
class Rectangle():
        def __init__(self,length,breadth):
                self.length = length
                self.breadth = breadth
        def getArea(self):
                print self.length*self.breadth," is area of rectangle"
class Square(Rectangle):
        def __init__(self,side):
                self.side = side
                Rectangle.__init__(self,side,side)
        def getArea(self):
                print self.side*self.side," is area of square"
s = Square(4)
r = Rectangle(2,4)
s.getArea()
r.getArea()
```