Programming with Python & Java (CS 29008)

Lab 3

Object Oriented Programming in Python
Classes, objects, methods & variables

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)
Deemed to be University U/S 3 of UGC Act, 1956

School of Electronics Engineering
KIIT Deemed to be University
Bhubeneswar, Odisha

Contents

Lab 3: Classes, objects, methods & variables

1 Introduction

Object-Oriented Programming (OOP) is a programming paradigm that revolves around the concept of "objects," which are instances of classes. In Python, OOP allows you to create reusable and organized code by modeling real-world entities as objects with attributes (data) and methods (functions). Each concept has been explained in subsequent sections.

2 Class, Object, Attributes and Methods

1. **Classes and Objects:**
    - Classes are blueprints for creating objects. They define the structure and behavior of objects.
    - Objects are instances of classes. They represent specific instances of the class and contain data and methods defined by the class.
2. **Attributes and Methods:**
    - Attributes or properties are variables that belong to objects. They store data specific to each object.
    - Methods are functions defined within a class. They define the behavior of objects and can manipulate object attributes.

2.1 How to define a class ?

We can define a class by using class keyword and its syntax is as follows

```
    ########## Syntax ##########################
class className:
 """documentation string"""
     variables:instance variables,static and local variables
     methods: instance methods,static methods,class methods

# Documentation string (Doc String) represents description
#of the class. Within the class doc string is always optional.
#We can get doc string by using the following 2 ways.
>> print(classname.__doc__)
>> help(classname)
```

Example:

```
class Student:
 """This is student class with required data"""

>> print(Student.__doc__)
>> help(Student)
```

2.2 What is Object ?

Physical existence of a class is nothing but object. We can create any number of objects for a class using reference variable which refer to an object instance. Moreover, We can access properties and methods of object using reference variable.

```
# Syntax to Create Object:
referencevariable = classname()
>> s = Student()
```

2.2.1 Self variable

- Self is the default variable which is always pointing to current object (like this keyword in Java)
- By using self, we can access instance variables and instance methods of object.
- Self should be first parameter inside constructor. e.g.- def __init__(self):
- Self should be first parameter inside instance methods. e.g. def talk(self):

2.3 Constructor Concept

- Constructor is a special method in python.
- The name of the constructor should be __init__(self)
- Constructor will be executed automatically at the time of object creation.
- The main purpose of constructor is to declare and initialize instance variables.
- Per object constructor will be executed only once.
- Constructor can take atleast one argument (atleast self)

- Constructor is optional and if we are not providing any constructor then python will provide default constructor.

Constructor without any instance variable.....

```
class Test:
    def __init__(self):
        print("Constructor execution...")
    def m1(self):
        print("Method execution...")
>> t1=Test()
>> t2=Test()
>> t3=Test()
>> t1.m1()
```

Constructor with instance variables.....

```
def __init__(self,name,rollno,marks):
    self.name=name
    self.rollno=rollno
    self.marks=marks
```

2.4 Types of Variables

There are 3 types of variables which are used in python.

1. Instance Variables (Object Level Variables)
2. Static Variables (Class Level Variables)
3. Local variables (Method Level Variables)

2.4.1 Instance Variables

1. If the value of a variable is varied from object to object, then such type of variables are called instance variables.
2. For every object, a separate copy of instance variables will be created.

Declaration of Instance Variables:

1. Inside Constructor by using self variable: Once we creates object, automatically these variables will be added to the object.

```
class Employee:
    def __init__(self):
        self.employeeNo=100
        self.employeename='John'
        self.employeeSalary=20000

>> e=Employee()
>> print(e.__dict__)# Accessing the magnitude of all the
                                    instance variables.
```

2. Inside Instance Method by using self variable : We can also declare instance variables inside instance method by using self variable. If any instance variable declared inside instance method, that instance variable will be added once we call that method.

```
class Test:
    def __init__(self):
        self.a=10
        self.b=20
    def m1(self):
        self.c=30

>> t=Test()
>> t.m1()
>> print(t.__dict__)
```

3. Outside of the class by using object reference variable : We can also add instance variables outside of a class to a particular object.

```
class Test:
    def __init__(self):
            self.a=10
            self.b=20
    def m1(self):
            self.c=30

>> t=Test()
>> t.m1()
>> t.d=40
>> print(t.__dict__)
```

Accessing Instance Variables : We can access instance variables with in the class by using self variable and outside of the class by using object reference.

```
class Test:
    def __init__(self):
        self.a=10
```

```
        self.b=20
    def display(self):
        print(self.a)
        print(self.b)
>> t=Test()
>> t.display()
>> print(t.a,t.b)
```

Delete Instance Variable from the Object:

1. Within a class we can delete instance variable as follows

```
        del self.variableName
```

2. From outside of class we can delete instance variables as follows

```
        del objectreference.variableName
```

Example:

```
class Test:
    def __init__(self):
            self.a=10
            self.b=20
            self.c=30
            self.d=40
    def m1(self):
        del self.d
>> t=Test()
>> print(t.__dict__)
>> t.m1()
>> print(t.__dict__)
>> del t.c
>> print(t.__dict__)
```

Note:

1. The instance variables which are deleted from one object,will not be deleted from other objects.
2. If we change the values of instance variables of one object then those changes won't be reflected to the remaining objects, because for every object we are separate copy of instance variables are available

2.4.2 Static Variables

- If the value of a variable is not varied from object to object, such type of variables we have to declare with in the class directly but outside of methods. Such types of variables are called Static variables.

- For total class, only one copy of static variable will be created and shared by all objects of that class.
- We can access static variables either by class name or by object reference. But recommended to use class name.
- In the case of instance variables for every object, a separate copy will be created, but in the case of static variables for total class only one copy will be created and shared by every object of that class.

```
class Test:
    x=10
    def __init__(self):
            self.y=20
>> t1=Test()
>> t2=Test()
>> print('t1:',t1.x,t1.y)
>> print('t2:',t2.x,t2.y)
>> Test.x=888
>> t1.y=999
>> print('t1:',t1.x,t1.y)
>> print('t2:',t2.x,t2.y)
```

Various Places to declare Static Variables

1. In general, we can declare within the class directly but from out side of any method.
2. Inside constructor by using class name
3. Inside instance method by using class name
4. Inside class method by using either class name or cls variable
5. Inside static method by using class name

```
class Test:
    a=10
    def __init__(self):
            print(self.a)
            print(Test.a)
    def m1(self):
            print(self.a)
            print(Test.a)
    @classmethod
    def m2(cls):
            print(cls.a)
            print(Test.a)
```

```
    @staticmethod
    def m3():
            print(Test.a)

>> t=Test()
>> print(Test.a)
>> print(t.a)
>> t.m1()
>> t.m2()
>> t.m3()
```

Where we can modify the Value of Static Variable ? Anywhere either with in the class or outside of class we can modify by using classname. But inside class method, by using cls variable.

```
class Test:
        a=777
        @classmethod
        def m1(cls):
            cls.a=888
        @staticmethod
        def m2():
            Test.a=999

>> print(Test.a)
>> Test.m1()
>> print(Test.a)
>> Test.m2()
>> print(Test.a)
```

Modify the Value of Static Variable:
If we change the value of static variable by using either self or object reference variable, then the value of static variable won't be changed, just a new instance variable with that name will be added to that particular object.

```
class Test:
    a=10
    def m1(self):
        self.a=888
>> t1=Test()
>> t1.m1()
>> print(Test.a)
>> print(t1.a)
```

```
class Test:
    x=10
    def __init__(self):
        self.y=20

>> t1=Test()
```

```
>> t2=Test()
>> print('t1:',t1.x,t1.y)
>> print('t2:',t2.x,t2.y)
>> t1.x=888
>> t1.y=999
>> print('t1:',t1.x,t1.y)
>> print('t2:',t2.x,t2.y)
```

```
class Test:
    a=10
    def __init__(self):
        self.b=20
>> t1=Test()
>> t2=Test()
>> Test.a=888
>> t1.b=999
>> print(t1.a,t1.b)
>> print(t2.a,t2.b)
```

```
class Test:
    a=10
    def __init__(self):
        self.b=20
    def m1(self):
        self.a=888
        self.b=999

>> t1=Test()
>> t2=Test()
>> t1.m1()
>> print(t1.a,t1.b)
>> print(t2.a,t2.b)
```

```
class Test:
    a=10
    def __init__(self):
        self.b=20
    @classmethod
    def m1(cls):
        cls.a=888
        cls.b=999

>> t1=Test()
>> t2=Test()
>> t1.m1()
>> print(t1.a,t1.b)
>> print(t2.a,t2.b)
>> print(Test.a,Test.b)
```

Delete Static Variables of a Class

1. We can delete static variables from anywhere by using the following syntax.

```
del classname.variablename
```

2. But inside classmethod we can also use cls variable

```
del cls.variablename
```

```
class Test:
     a=10
     @classmethod
     def m1(cls):
        del cls.a
>> Test.m1()
>> print(Test.__dict__)
```

```
#By using object reference variable/self we can read static #
                                    variables, but we cannot modify or
                                     delete.
#If we are trying to modify, then a new instance variable #will be
                                    added to that particular object.
#If we are trying to delete then we will get error.
class Test:
a=10

>> t1=Test()
>> del t1.a # AttributeError: a
```

2.4.3 Local Variables

1. Sometimes to meet temporary requirements of programmer,we can declare variables inside a method directly,such type of variables are called local variable or temporary variables.

2. Local variables will be created at the time of method execution and destroyed once method completes.

3. Local variables of a method cannot be accessed from outside of method.

```
class Test:
    def m1(self):
        a=1000
        print(a)
    def m2(self):
        b=2000
```

```
        print(b)
        print(a) #NameError: name 'a' is not defined
>> t=Test()
>> t.m1()
>> t.m2()
```

2.5 Types of Methods

Inside Python class 3 types of methods are allowed

1. Instance Methods
2. Class Methods
3. Static Methods

2.5.1 Instance Method

1. Inside method implementation if we are using instance variables then such type of methods are called instance methods.
2. Inside instance method declaration, we have to pass self variable. def m1(self):
3. By using self variable inside method we can able to access instance variables.
4. Within the class, we can call instance method by using self variable and from outside of the class we can call by using object reference.

```
class Student:
    def __init__(self,name,marks):
            self.name=name
            self.marks=marks
    def display(self):
            print('Hi',self.name)
            print('Your Marks are:',self.marks)
    def grade(self):
            if self.marks>=60:
                    print('You got First Grade')
            elif self.marks>=50:
                    print('Yout got Second Grade')
            elif self.marks>=35:
                    print('You got Third Grade')
            else:
                    print('You are Failed')
```

```
>> n=int(input('Enter number of students:'))
>> for i in range(n):
        name=input('Enter Name:')
        marks=int(input('Enter Marks:'))
>> s= Student(name,marks)
>> s.display()
>> s.grade()
```

2.5.2 Class Method

1. Inside method implementation if we are using only class variables (static variables), then such type of methods we should declare as class method.
2. We can declare class method explicitly by using @classmethod decorator.
3. For class method, we should provide cls variable at the time of declaration
4. We can call classmethod by using classname or object reference variable.

```
 class Animal:
    legs=4
    @classmethod
    def walk(cls,name):
           print('{} walks with {} legs...'.format(name,cls.lEgs))
>>  Animal.walk('Dog')
>>  Animal.walk('Cat')
```

2.5.3 Static Methods

1. In general these methods are general utility methods.
2. Inside these methods we won't use any instance or class variables.
3. Here we won't provide self or cls arguments at the time of declaration.
4. We can declare static method explicitly by using @staticmethod decorator.
5. We can access static methods by using classname or object reference

```
class Computing:
        @staticmethod
        def add(x,y):
            print('The Sum:',x+y)
        @staticmethod
        def product(x,y):
            print('The Product:',x*y)

        @staticmethod
        def average(x,y):
            print('The average:',(x+y)/2)

>> Computing.add(10,20)
>> Computing.product(10,20)
>> Computing.average(10,20)
```

Note:

1. In general, we can use only instance and static methods.Inside static method we can access class level variables by using class name.
2. Class methods are most rarely used methods in python.

3 Advanced Concepts

3.1 Setter and Getter Method

We can set and get the values of instance variables by using getter and setter methods.

1. **Setter Method**: setter methods can be used to set values to the instance variables. setter methods also known as mutator methods.

```
#Syntax:
def setVariable(self,variable):
        self.variable=variable
#Example:
def setName(self,name):
        self.name=name
```

2. **Getter Method:** Getter methods can be used to get values of the instance variables. Getter methods also known as accessor methods.

```
#Syntax:
def getVariable(self):
    return self.variable
```

```
#Example:
def getName(self):
   return self.name
```

Example:

```
class Student:
    def setName(self,name):
        self.name=name
    def getName(self):
        return self.name

    def setMarks(self,marks):
        self.marks=marks

    def getMarks(self):
        return self.marks

>> n=int(input('Enter number of students:'))
>> for i in range(n):
        s=Student()
        name=input('Enter Name:')
        s.setName(name)
        marks=int(input('Enter Marks:'))
        s.setMarks(marks)

>> print('Hi',s.getName())
>> print('Your Marks are:',s.getMarks())
```

3.2 Passing Members of One Class to Another Class

We can access members of one class inside another class.

```
class Employee:
        def __init__(self,eno,ename,esal):
            self.eno=eno
            self.ename=ename
            self.esal=esal
        def display(self):
            print('Employee Number:',self.eno)
            print('Employee Name:',self.ename)
            print('Employee Salary:',self.esal)
class Test:
        def modify(emp):
            emp.esal=emp.esal+10000
            emp.display()

>> e=Employee(121,'John',20000)
>> Test.modify(e)
```

3.3 Inner Classes

Sometimes we can declare a class inside another class, such type of classes are called inner classes. Without existing one type of object if there is no chance of existing another type of object, then we should go for inner classes. Example: Without existing Car object there is no chance of existing Engine object. Hence Engine class should be part of Car class. Hence inner class object is always associated with outer class object.

```
class Car:
 .....
 class Engine:
 ......
```

Note: Without existing outer class object there is no chance of existing inner class object.

```
class Outer:
    def __init__(self):
        print("outer class object creation")
class Inner:
    def __init__(self):
        print("inner class object creation")
    def m1(self):
        print("inner class method")
>> o=Outer()
>> i=o.Inner()
>> i.m1()
# or
>> i = Outer().Inner()
>> i.m1()
# or
>> Outer().Inner().m1()
```

3.4 Destructors:

1. Destructor is a special method and the name should be __del__.
2. Just before destroying an object Garbage Collector always calls destructor to perform clean up activities (Resource deallocation activities like close database connection etc).
3. Once destructor execution completed then Garbage Collector automatically destroys that object.

Note: The job of destructor is not to destroy object and it is just to perform clean up activities.

```
import time
class Test:
    def __init__(self):
        print("Object Initialization...")
    def __del__(self):
        print("Fulfilling Last Wish and performing clean up
                                            activities...")

>> t1=Test()
>> t1=None
>> time.sleep(5)
>> print("End of application")
```

Note: If the object does not contain any reference variable then only it is eligible for GC (Garbage collection). i.e. if the reference count is zero then only object eligible for GC.

```
import time
class Test:
    def __init__(self):
        print("Constructor Execution...")
    def __del__(self):
        print("Destructor Execution...")

>> t1=Test()
>> t2=t1
>> t3=t2
>> del t1
>> time.sleep(5)
>> print("object not yet destroyed after deleting t1")
>> del t2
>> time.sleep(5)
>> print("object not yet destroyed even after deleting t2")
>> print("I am trying to delete last reference variable...")
>> del t3
```

```
import time
class Test:
    def __init__(self):
        print("Constructor Execution...")
    def __del__(self):
        print("Destructor Execution...")

>> list=[Test(),Test(),Test()]
>> del list
>> time.sleep(5)
>> print("End of application")
```

Lab 3 Exercises

The objectives of this lab are

- to apply OOP concepts using Python syntax, including the creation of classes, instantiation of objects, and implementation of methods.
- to design and implement modular and reusable code using classes and objects, promoting code organization and maintainability.

Lab 3 Assignments

1. Create a class for a Student with attributes name, age, and grade. Implement a method to display student information.
2. Write a Python program to create a Student class with details given in previous question. Call the method talk() to display student details.
3. Create a class representing a Rectangle with attributes length and width. Implement methods to calculate its area and perimeter.
4. Define a class for a BankAccount with methods to deposit, withdraw, and check balance. Ensure proper encapsulation of data.
5. Write a Python programme with a class which has attributes like title,actor and actress in its constructor and another method which prints all these properties.
   (a) Prompt the user to enter all this information and add this info in a list.
   (b) Keep the prompt open add all this information until user reply in 'No' and then display all this information of the list.
6. *Implement a class hierarchy for different shapes (e.g., Circle, Square, Triangle). Include methods to calculate area and perimeter for each shape.
7. Develop a class representing a Book with attributes title, author, and price. Implement methods to set and get book details.
8. *Design a class hierarchy for different vehicles (e.g., Car, Truck, Motorcycle). Include methods to calculate mileage and display vehicle information.
9. Define a class for a TodoList with methods to add, remove, and display tasks. Ensure tasks are stored as objects of another class.

10. Implement a class representing a Bank with methods to add accounts, remove accounts, and display total balance across all accounts.
11. Create a class for a Product with attributes name, price, and quantity. Implement methods to calculate total cost and update quantity.
12. *Design a class hierarchy for different employees (e.g., Manager, Developer, Tester). Include methods to calculate salary and display employee details.