Chapter Five: Introduction to OOP in python

5.1 Chapter objectives

By the end of this chapter, the learners should be able to:

- Create objects of type class
- Define a class data type in python
- Declare and define class members

5.2 Introduction

Python has been an object-oriented language since it developed in 1992. Almost everything in Python is an object, with its properties and methods. This chapter will introduce you to OOPs features in python.

An object-oriented paradigm is a programming philosophy where programs use classes and objects. The object is related to real-word entities such as book, house, pencil, etc. The oops concept focuses on writing the reusable code. It is a widespread technique to solve complex problems by creating objects.

Major characteristics associate with object-oriented programming system include; *Class, Object, Method, Inheritance, Polymorphism, Data Abstraction and Encapsulation*. Refer to our chapter one for detailed explanations for each.

5.3 Class

Classes are the fundamental building blocks in any Object-Oriented programming language. A Class is a user defined data type which consists of two elements; attributes and methods.

In OOP languages like C++, Java, python e.t.c, the data and functions (procedures to manipulate the data) are bundled together as a self-contained unit called class. As opposed to structured programming which describes data properties alone, OOP languages, a class describes both the properties (data) and behaviors (functions) of objects. This encapsulation is done by class.

Class attributes are variables in a typical programming language while methods are the functions, we discussed in chapter four. They determine the behavior of the class i.e., provide a definition for supporting various operations on data held in form of an object. Both attributes and methods define the properties and behavior (operations) of the objects in a Class.

We can define a class as an *Abstract Data Type (ADT) which defines how an object of that class will look like and behave*. A class is simple a blue of creating objects later in our program.

Let's consider the following abstract example of a car

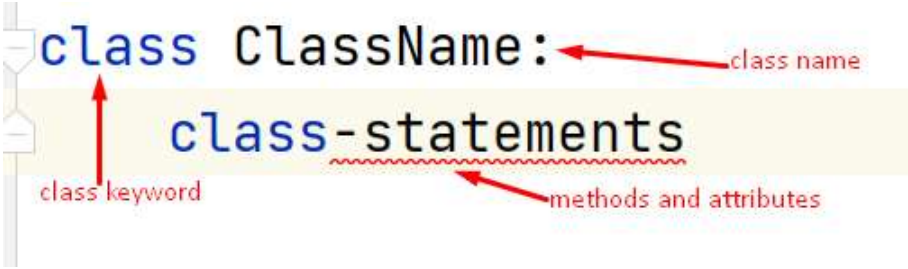
There may be many cars with different names and brand but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range etc. So here, Car is the class and wheels, speed limits, mileage are their properties.

In the above example of class Car, the data member will be speed limit, mileage e.t.c and member functions can be applied brakes, increase speed etc.

5.4 Defining a class in python

By defining a class, we don't define any data, but we define the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

The general form of a class declaration is as follows.



Example of class declaration ;

```
class Student:
    def __init__(self, name, regno):
        self.name = name
        self.regno = regno
    def viewStudent(self):
        print("Name : ", self.name, " Regno: ", self.regno)
s=Student("joel",100)
s.viewStudent()
```

Example explained

The first method __init__( ) is a special method, which is called class constructor/ initialization method that Python calls when you create a new instance of this class.

You declare other class methods like normal functions with the exception that the first argument to each method is self.

Python adds the self-parameter to the list for you; you do not need to include it when you call the methods

5.5 Creating objects

After defining our class, we can now proceed to create an object of that class.

Define an object as instance of a class. This makes the class look like any other data type. When a class is defined, no memory is allocated but when it is instantiated (i.e., an object is created) memory is allocated.

The general form of a class declaration is as follows.

```
Objectname=class_Name()
```

We have already created the class named student, so now we can use this to create object(s).

To create an object of the class student created above, we can use the following python statements

```
s1=Student( )
s1=Student( )
....
sn=Student( )
```

s1, s2…sn are all objects (instances of class student). We can have multiple objects of the same class coexisting without interfering with each other

Objects of a class can be used to access the class attributes and methods using the dot operator as shown below

```
s.name=" Chris" #updating name property
s. viewStudent () #calling viewStudent
```

5.6 Multiple Objects

Its possible to create more than one object of the same class as mentioned above. Below is a modified version of the previous program containing three objects

```
class Student:
    def __init__(self, name, regno):
        self.name = name
        self.regno = regno
    def viewStudent(self):
        print("Name : ", self.name, " Regno: ", self.regno)
s=Student("joel",100)
s.viewStudent()
s1=Student("Mary",101)
s1.viewStudent()
s2=Student("Chris",103)
s2.viewStudent()
```

5.7 Types of class variables

OOPs allows for variables to be used at the class level or the instance level.

There are two types of variables in python in context of class definition

- class variables, This are variables that do not differ between instances of a class and are shared across by all instances of a class. We also call them static variables
- instance variables This are variables that vary from one object to another. They are not shared by objects. Each object has its own copy of the instance variable The type of variables are used if we know variables will change significantly across instances

5.8 The 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. Its not a must for the variable to have the same name as self, we can call it whatever we like, but it has to be the first parameter of any function in the class:

5.9 Constructors

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

They automatically called/invoked when an object of a class is created.

5.9.1 Characteristics of a constructor

1. They automatically invoked/called when the objects of that class are created
2. They do return values
3. Just like other methos, they can have default arguments

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

5.9.2 The __init__ () Function

In Python, the method the __init__() simulates the constructor of the class. This method is called when the class is instantiated. It accepts the self-keyword as a first argument which allows accessing the attributes/method of the class.

We can pass any number of arguments at the time of creating the class object, depending upon the __init__ () definition.

We use it to o initialize the class attributes i.e. assign values to object properties, or other operations that are necessary to do when the object is being created

```
class bank:
  def __init__(self,balance):
    self.balance = balance
  def account(self):
      self.acno=input("Enter account number\n")
      self.name = input("Enter account name\n")
def deposit(self):
      self.depo=int(input("Enter amount to deposit\n"))
      self.balance=self.balance+self.depo
b = bank(0)
print("Your bank balance is",b.balance)
b.account()
b.deposit()
print(b.acno,b.name ,b.balance)
```

5.10 Types of Constructors

Python supports two types of constructors; *parameterized* and *non-parameterized constructors*

Parameterized constructors

These are constructors that have multiple parameters along with the self e.g. the parameter defined above which takes self and balance as parameters can be termed as a parameterized constructor

Non-Parameterized Constructor

This are constructors used when we do not want to manipulate the value or the constructor that has only self as an argument.

```
class bank:
  def __init__(self):
   print("An instance of the class has been created sucessfuly"
```

5.11 Modify Object Properties

You can modify the account name using the following python code

```
b.accno=105
```

5.12 Delete object property

To delete a class instance (object), we can use the following python code

```
del b.accno
```

5.1 Delete object

To delete a class instance (object), we can use the following python code

```
del b
```

5.2 Chapter summary

5.3 Chapter exercise

5.4 Further reading suggestions

i https://pynative.com/python-class-variables/

ii https://www.w3schools.com/python/python_classes.asp

iii https://www.javatpoint.com/python-constructors

iv https://www.tutorialspoint.com/python/python_classes_objects.htm