CHAPTER

12

INTRODUCTION TO OOPS

In this chapter, we will introduce a revolutionary concept called *Object Oriented Programming System* (OOPS) based on which the languages like Smalltalk, Simula-67, C++, Java, Python, etc. are created. In this chapter, however, we are going to have a bird's eye view of the fundamental concepts of OOPS while an in depth discussion will be held only in the subsequent chapters.

The languages like C, Pascal, Fortran etc., are called Procedure Oriented Programming languages since in these languages, a programmer uses procedures or functions to perform a task. While developing software, the main task is divided into several sub tasks and each sub task is represented as a procedure or function. The main task is thus composed of several procedures and functions. This approach is called *Procedure oriented approach*. Consider Figure 12.1:

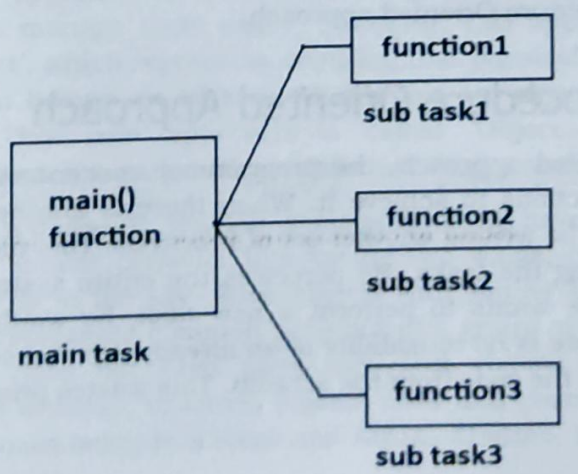

Figure 12.1: Procedure Oriented Approach

On the other hand, languages like C++, Java and Python use classes and objects in their programs and are called Object Oriented Programming languages. A class is a module which itself contains data and methods (functions) to achieve the task. The main task is divided into several sub tasks, and these are represented as classes. Each class can perform several inter-related tasks for which several methods are written in a class. This approach is called *Object Oriented approach*. Consider Figure 12.2:

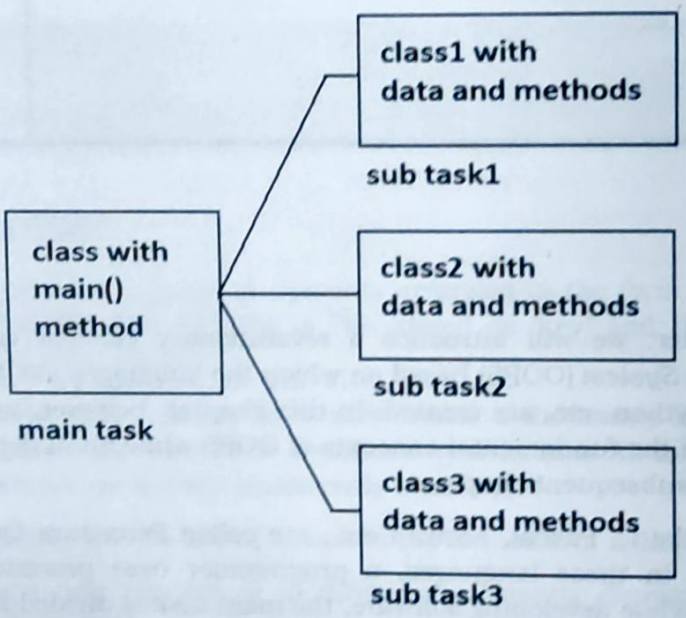

Figure 12.2: Object Oriented Approach

Programmers have followed Procedure Oriented approach for several decades, but as experience and observation teaches new lessons, programmers slowly realized several problems with Procedure Oriented approach.

Problems in Procedure Oriented Approach

In Procedure Oriented approach, the programmer concentrates on a specific task, and writes a set of functions to achieve it. When there is another task to be added to the software, he would be writing another set of functions. This means his concentration will be only on achieving the tasks. He perceives the entire system as fragments of several tasks. Whenever he wants to perform a new task, he would be writing a new set of functions. Thus there is no reusability of an already existing code. A new task every time requires developing the code from the scratch. This wastes programmer's time and effort.

In Procedure Oriented approach, every task and sub task is represented as a function and one function may depend on another function. Hence, an error in the software needs examination of all the functions. Thus debugging or removing errors will become difficult. Any updations to the software will also be difficult.

When the software is developed, naturally code size will also be increased. It has been observed in most of the software developed following the Procedure Oriented approach that when the code size exceeds 10,000 lines and before reaching 100,000 lines, suddenly at a particular point, the programmers start losing control on the code. This means, the programmers could not understand the exact behavior of the code and could neither debug it, nor extend it. This posed many problems, especially when the software was constructed to handle bigger and complex systems. For example, to create software to send satellites into the sky and control their operations from the ground stations, we may have to write millions of lines of code. In such systems, Procedure Oriented approach fails and programmers realized the need of another approach.

There is another problem with Procedure Oriented approach. Programming in this approach is not developed from human being's life. Statements, functions or procedures never reflect the human beings. So, from the human beings' point of view, they are unnatural. Unnatural activities are difficult to perform. For example, as human beings, we can walk or run. But, if we are asked to fly like birds; that would become impossible for us since flying is not natural for human beings. In the same way, if programming is developed from human beings' life, we can adapt to it easily. This idea forced computer scientists to develop a new approach that would be closer to human beings' life.

Due to the preceding reasons, computer scientists felt the need of a new approach where programming will have several modules. Each module represents a 'class' and the classes can be reusable and hence maintenance of code will become easy. When there is an error, it is possible to debug only on that class where error occurred without disturbing the other classes. This approach is suitable not only to develop bigger and complex applications but also to manage them easily. Moreover, this approach is built from a single root concept 'object', which represents anything that physically exists in this world. It means that all human beings are objects. All animals are objects. All existing things will become objects. This new approach is called 'Object Oriented Approach'. Programming in this approach is called Object Oriented Programming System (OOPS).

In OOPS, everything is an object. In real life, some objects will have similar behavior. For example, all birds have similar behavior like having two wings, two legs, etc. Also, all birds have the ability to fly in the sky. Such objects with similar behavior belong to the same class. So, a class represents common behavior of a group of objects. Since a class represents behavior, it does not exist physically. But objects exist physically. For example, bird is a class; whereas, sparrow, pigeon, crow and peacock are objects of the bird class. Similarly, human being is a class and Arjun, Krishna, Sita are objects of the human being class.

Specialty of Python Language

In OOPS, all programs involve creation of classes and objects. This makes programs lengthy. For example, we have to write all the statements of the program inside a class and then create objects to the class. Then use the features of the class through objects. This type of programming requires much code to perform a simple task like adding two numbers. Also, the program execution takes more time. So, for simple tasks, it is still better to go for procedure oriented approach which offers less code and more speed. For example, a C program executes faster than its equivalent program written in Python or Java!

Even though, Python is an object oriented programming language like Java, it does not force the programmers to write programs in complete object oriented way. Unlike Java, Python has a blend of both the object oriented and procedure oriented features. Hence, Python programmers can write programs using procedure oriented approach (like C) or object oriented approach (like Java) depending on their requirements. This is definitely an advantage for Python programmers!

Features of Object Oriented Programming System (OOPS)

There are five important features related to Object Oriented Programming System. They are:

- Classes and objects
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism

Let's move further to have clear understanding of each of these features.

Classes and Objects

The entire OOPS methodology has been derived from a single root concept called 'object'. An object is anything that really exists in the world and can be distinguished from others. This definition specifies that everything in this world is an object. For example, a table, a ball, a car, a dog, a person, etc. will come under objects. Then what is not an object? If something does not really exist, then it is not an object. For example, our thoughts, imagination, plans, ideas etc. are not objects, because they do not physically exist.

Every object has some behavior. The behavior of an object is represented by attributes and actions. For example, let's take a person whose name is 'Raju'. Raju is an object because he exists physically. He has attributes like name, age, sex, etc. These attributes can be represented by variables in our programming. For example, 'name' is a string type variable, 'age' is an integer type variable.

Similarly, Raju can perform some actions like talking, walking, eating and sleeping. We may not write code for such actions in programming. But, we can consider calculations and processing of data as actions. These actions are performed by methods. We should understand that a function written inside a class is called a method. So an object contains variables and methods.

It is possible that some objects may have similar behavior. Such objects belong to same category called a 'class'. For example, not only Raju, but all the other persons have various common attributes and actions. So they are all objects of same class, 'Person'. Now observe that the 'Person' will not exist physically but only Raju, Ravi, Sita, etc. exist physically. This means, a class is a group name and does not exist physically, but objects exist physically. See Figure 12.3:

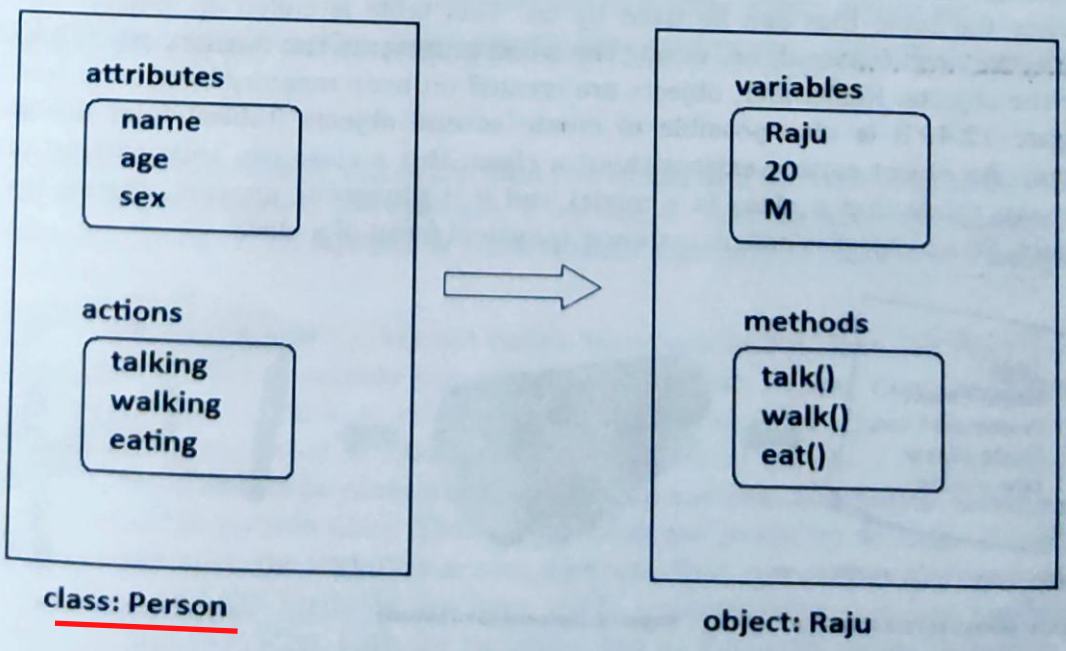

Figure 12.3: Person Class and Raju Object

To understand a class, take a pen and paper and write down all the attributes and actions of any person. The paper contains the model that depicts a person, so it is called a class. We can find a person with the name 'Raju', who got all the attributes and actions as written on the paper. So 'Raju' becomes an object of the class, Person. This gives a definition for the class. A class is a model or blueprint for creating objects. By following the class, one can create objects. So we can say, whatever is there in the class, will be seen in its objects also.

We can use a class as a model for creating objects. To write a class, we can write all the characteristics of objects which should follow the class. These characteristics will guide us to create objects. A class and its objects are almost the same with the difference that a class does not exist physically, while an object does. For example, let's say we want to construct a house. First of all, we will go to an architect who provides a plan. This plan is only an idea and exists on paper. This is called a class. However, based on this plan, if we construct the house, it is called an object since it exists physically. So, we can say that we can create objects from the class. An object does not exist without a class. But a class can exist without any objects.

Let's take another example. Flower is a class but if we take Rose, Lily, and Jasmine - they are all objects of flower class. The class flower does not exist physically but its objects, like Rose, Lily and Jasmine exist physically.

Let's take another example. We want a table made by a carpenter. First of all, the carpenter takes a paper and writes the measurements regarding length, breadth and height of the table. He may also draw a picture on the paper that works like a model for creating the original table. This plan or model is called a 'class'. Following this model, he makes the table that can be used by us. This table is called an 'object'. To make the table, we need material, i.e. wood. The wood represents the memory allotted by the PVM for the objects. Remember, objects are created on heap memory by PVM at run time. See Figure 12.4. It is also possible to create several objects (tables) from the same class (plan). An object cannot exist without a class. But a class can exist without any object. We can think that a class is a model and if it physically appears, then it becomes an object. So an object is called 'instance' (physical form) of a class.

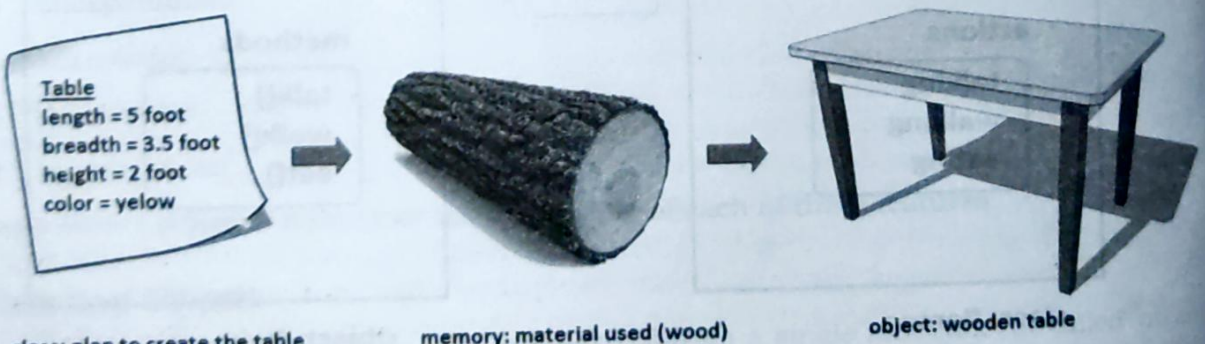

Figure 12.4: Creation of a Class and Object

Creating Classes and Objects in Python

Let's create a class with the name Person for which Raju and Sita are objects. A class is created by using the keyword, class. A class describes the attributes and actions performed by its objects. So, we write the attributes (variables) and actions (functions) in the class as:

```
# This is a class
class Person:
```

```
# attributes means variables
name = 'Raju'
age = 20

# actions means functions
def talk(cls):
    print(cls.name)
    print(cls.age)
```

Observe the preceding code. Person class has two variables and one function. The function that is written in the class is called method. When we want to use this class, we should create an object to the class as:

```
p1 = Person()
```

Here, p1 is an object of Person class. Object represents memory to store the actual data. The memory needed to create p1 object is provided by PVM. Observe the function (or method) in the class:

```
def talk(cls):
```

Here, 'cls' represents a default parameter that indicates the class. So, cls.name refers to class variable 'Raju'. We can call the talk() method to display Raju's details as:

```
p1.talk()
```

Encapsulation

Encapsulation is a mechanism where the data (variables) and the code (methods) that act on the data will bind together. For example, if we take a class, we write the variables and methods inside the class. Thus, class is binding them together. So class is an example for encapsulation.

The variables and methods of a class are called 'members' of the class. All the members of a class are by default available outside the class. That means they are *public* by default. Public means available to other programs and classes. Python follows *Uniform Access Principle* that says that in OOPS, all the members of the class whether they are variables or methods should be accessible in a uniform manner. So, Python variables and methods are available outside alike. That means both are *public* by default. Usually, in C++ and Java languages, the variables are kept *private*, that means they are not available outside the class and the methods are kept *public* meaning that they are available to other programs. But in Python, both the variables and methods are *public* by default.

Encapsulation isolates the members of a class from the members of another class. The reason is when objects are created, each object shares different memory and hence there will not be any overwriting of data. This gives an advantage to the programmer to use same names for the members of two different classes. For example, a programmer can declare and use the variables like 'id', 'name', and 'address' in different classes like Employee, Customer, or Student classes.

Encapsulation in Python

Encapsulation is nothing but writing attributes (variables) and methods inside a class. The methods process the data available in the variables. Hence data and code are bundled up together in the class. For example, we can write a Student class with 'id' and 'name as attributes along with the display() method that displays this data. This Student class becomes an example for encapsulation.

```
# a class is an example for encapsulation
class Student:
    # to declare and initialize the variables
    def __init__(self):
        self.id = 10
        self.name = 'Raju'

    # display students details
    def display(self):
        print(self.id)
        print(self.name)
```

Observe the first method: def __init__(self). This is called a special function since its name is starting and ending with two underscores. If a variable or method name starts and ends with two underscores, they are built-in variables or methods which are defined for a specific purpose. The programmer should not create any variable or method like them. It means we should not create variables or methods with two underscores before and after their names.

The purpose of the special method def __init__(self) is to declare and initialize the instance variables of a class. Instance variables are the variables whose copy is available in the object (or instance). The first parameter for this method is 'self' that represents the object (or instance) of the present class. So, self.id refers to the variable in the object. In the Student class, we have written another method by the name display() that displays the instance variables.

Abstraction

There may be a lot of data, a class contains and the user does not need the entire data. The user requires only some part of the available data. In this case, we can hide the unnecessary data from the user and expose only that data that is of interest to the user. This is called abstraction.

A good example for abstraction is a car. Any car will have some parts like engine, radiator, battery, mechanical and electrical equipment etc. The user of the car (driver) should know how to drive the car and does not require any knowledge of these parts. For example driver is never bothered about how the engine is designed and the internal parts of the engine. This is why the car manufacturers hide these parts from the driver in a separate panel, generally at the front of the car.

The advantage of abstraction is that every user will get his own view of the data according to his requirements and will not get confused with unnecessary data. A bank clerk should see the customer details like account number, name and balance amount in the account. He should not be entitled to see the sensitive data like the staff salaries, profit or loss of the bank, interest amount paid by the bank, loans amount to be recovered, etc. Hence, such sensitive data can be abstracted from the clerk's view. The bank manager may, however, require the sensitive data and so it will be provided to the manager.

Abstraction in Python

In languages like Java, we have keywords like private, protected and public to implement various levels of abstraction. These keywords are called *access specifiers*. In Python, such words are not available. Everything written in the class will come under public. That means everything written in the class is available outside the class to other people. Suppose, we do not want to make a variable available outside the class or to other members inside the class, we can write the variable with two double scores before it as: __var. This is like a private variable in Python. In the following example, 'y' is a private variable since it is written as: __y.

```
class Myclass:
    # this is constructor.
    def __init__(self):
        self.__y = 3   # this is private variable
```

Now, it is not possible to access the variable from within the class or out of the class as:

```
m = Myclass()
print(m.y)   # error
```

The preceding print() statement displays error message as: AttributeError: 'Myclass' object has no attribute 'y'. Even though, we cannot access the private variable in this way, it is possible to access it in the format: instancename._Classname__var. That means we are using Classname differently to access the private variable. This is called *name mangling*. In name mangling, we have to use one underscore before the classname and two underscores after the classname. Like this, using the names differently to access the private variables is called name mangling. For example, to display a private variable 'y' value, we can write:

```
print(m._Myclass__y)   # display private variable y
```

The same statement can be written inside the method as: print(self._Myclass__z). When we use single underscore before a variable as _var, then that variable or object will not be imported into other files. The following code represents the public and private variables and how to access them.

```
# understanding public and private variables
class Myclass:
    # this is constructor.
    def __init__(self):
        self.x = 1   # public var
        self.__y = 2  # private var
```

```
    # instance method to access variables
    def display(self):
        print(self.x)  # x is available directly
        print(self._Myclass__y)   # name mangling required

print('Accessing variables through method:')
m = Myclass()
m.display()

print('Accessing variables through instance:')
print(m.x)   # x is available directly
print(m._Myclass__y)   # name mangling required
```

Output:

```
C:\>python oops.py
Accessing variables through method:
1
2
Accessing variables through instance:
1
2
```

We are planning to write Bank class with 'accno', 'name', 'balance' and 'loan' as variables. Since the clerk should not see the loan amount of the customer, we can write that variable with two underscores before the variable, as: ' __loan'. Then this variable is not available directly outside the class or inside the class to other methods.

In the Bank class, the first method is a special method with the name: __init__(self) is useful to declare variables and initialize them with some data. In the program, 'self' represents current class object. In this method, we are making loan variable as private by writing it as:

```
self. __loan = 1500000.00;
```

This variable is not available outside the class. It is not even available to other methods in the same class. Hence, it is abstracted completely from the user of the class. If the bank clerk calls the display_to_clerk(self) method, he will be able to see account number, name and balance amount only. He cannot see loan amount of the customer. That means some part of the data is hidden from the clerk. See the example code:

```
# accessing some part of data
class Bank :
    def __init__(self):
        self.accno = 10
        self.name = 'Srinu'
        self.balance = 5000.00
        self. __loan = 1500000.00

    def display_to_clerk(self):
        print(self.accno)
        print(self.name)
        print(self.balance)
```

In the preceding class, in spite of several data items, the display_to_clerk() method is able to access and display only the 'accno', 'name' and 'balance' values. It cannot access loan of the customer. This means the loan data is hidden from the view of the bank clerk. This is called abstraction. Suppose, we try to display the loan amount in the display_to_clerk() method, by writing:

```
print(self.loan)
```

This raises an error saying 'loan' is not an attribute of Bank class.

Inheritance

Creating new classes from existing classes, so that the new classes will acquire all the features of the existing classes is called Inheritance. A good example for Inheritance in nature is parents producing the children and children inheriting the qualities of the parents.

Let's take a class A with some members i.e., variables and methods. If we feel another class B wants almost same members, then we can derive or create class B from A as:

```
class B(A):
```

Now, all the features of A are available to B. If an object to B is created, it contains all the members of class A and also its own members. Thus, the programmer can access and use all the members of both the classes A and B. Thus, class B becomes more useful. This is called inheritance. The original class (A) is called the base class or super class and the derived class (B) is called the sub class or derived class.

There are three advantages of inheritance. First, we can create more useful classes needed by the application (software). Next, the process of creating the new classes is very easy, since they are built upon already existing classes. The last, but very important advantage is managing the code becomes easy, since the programmer creates several classes in a hierarchical manner, and segregates the code into several modules.

An Example for Inheritance in Python

Here, we take a class A with two variables 'a' and 'b' and a method, method1(). Since all these members are needed by another class B, we extend class B from A. We want some additional members in B, for example a variable 'c' and a method, method2(). So, these are written in B. Now remember, class B can use all the members of both A and B. This means the variables 'a','b','c 'and also the methods method1() and method2() are available to class B. That means all the members of A are inherited by B.

```
# Creating class B from class A
class A :
    a = 1
    b = 2
```

```
        def method1(cls):
            print(cls.a)
            print(cls.b)

    class B(A):
        c = 3
        def method2(cls):
            print(cls.c)
```

By creating an object to B, we can access all the members of both the classes A and B.

Polymorphism

The word 'Polymorphism' came from two Greek words 'poly' meaning 'many' and 'morphos' meaning 'forms'. Thus, polymorphism represents the ability to assume several different forms. In programming, if an object or method is exhibiting different behavior in different contexts, it is called polymorphic nature.

Polymorphism provides flexibility in writing programs in such a way that the programmer uses same method call to perform different operations depending on the requirement.

Example Code for Polymorphism in Python:

When a function can perform different tasks, we can say that it is exhibiting polymorphism. A simple example is to write a function as:

```
def add(a, b):
    print(a+b)
```

Since in Python, there the variables are not declared explicitly, we are passing two variables 'a', and 'b' to add() function where they are added. While calling this function, if we pass two integers like 5 and 15, then this function displays 15. If we pass two strings, then the same function concatenates or joins those strings. That means the same function is adding two integers or concatenating two strings. Since the function is performing two different tasks, it is said to exhibit polymorphism. Now, consider the following example:

```
# a function that exhibits polymorphism
def add(a, b):
    print(a+b)

# call add() and pass two integers
add(5, 10)  # displays 15

# call add() and pass two strings
add("Core", "Python")  # displays CorePython
```

The programming languages which follow all the five features of OOPS are called object oriented programming languages. For example, C++, Java and Python will come into this category.

Points to Remember

- Procedure oriented approach is the methodology where programming is done using procedures and functions. This is followed by languages like C, Pascal and FORTRAN.
- Object oriented approach is the methodology where programming is done using classes and objects. This is followed in the languages like C++, Java and Python.
- Python programmers can write programs using procedure oriented approach (like C) or object oriented approach (like Java) depending on their requirements.
- An object is anything that really exists in the world and can be distinguished from others.
- Every object has some behavior that is characterized by attributes and actions. Attributes are represented by variables and actions are performed by methods. So an object contains variables and methods.
- A function written inside a class is called method.
- A class is a model or blueprint for creating objects. A class also contains variables and methods.
- Objects are created from a class.
- An object does not exist without a class; however, a class can exist without any object.
- Encapsulation is a mechanism where the data (variables) and the code (methods) that act on the data will bind together.
- Class is an example for encapsulation since it contains data and code.
- Hiding unnecessary data and code from the user is called abstraction.
- To hide variables or methods, we should declare them as private members. This is done by writing two underscores before the names of the variable or method.
- Private members can be accessed using name mangling where the class name is used with single underscore before it and two underscores after it in the form of: instancename. _Classname__variable or instancename. _Classname__method().
- Creating new classes from existing classes, so that new classes will acquire all the features of the existing classes is called Inheritance.

- In inheritance, the already existing class is called base class or super class. The newly created class is called sub class or derived class.
- Polymorphism represents the ability of an object or method to assume several different forms.
- The programming languages which follow all the five features of OOPS namely, classes and objects, encapsulation, abstraction, inheritance and polymorphism are called object oriented programming languages. For example, C++, Java and Python will come into this category.