CHAPTER 9

Classes and Objects

• Class and Objects • Class and Instance Variables • Public and Private Variables • Special Methods • Built-in Attributes and Functions • Garbage Collection • Class Method and Static Method

9.1 INTRODUCTION

In all our programs till now, we have been using the procedure-oriented technique in which our program is written using functions or blocks of statements which manipulate data. However, another and in fact, a better style of programming is called object oriented programming in which data and functions are combined to form a class.

Compared with other programming languages, Python has a very short and simple way to define and use classes. The class mechanism supported by Python is actually a mixture of that found in C++ and Modula-3. As discussed in Chapter 2, Python supports all the standard features of Object Oriented Programming. In this chapter, we will study about these features in detail.

9.2 CLASSES AND OBJECTS

Classes and objects are the two main aspects of object oriented programming. In fact, a class is the basic building block in Python. A **class** creates a new type and object is an instance (or variable) of the class. Classes provides a blueprint or a template using which objects are created. In fact, *in Python, everything is an object or an instance of some class.* For example, all integer variables that we define in our program are actually instances of class int. Similarly, all string variables are objects of class string. Recall that we had used string methods using the variable name followed by the dot operator and the method name. We have already studied that we can find out the type of any object using the `type()` function.

Note The Python Standard Library is based on the concept of classes and objects.

9.2.1 Defining Classes

Python has a very simple syntax of defining a class. This syntax can be given as,

```
class class_name:
    <statement-1>
    <statement-2>
    .
    .
    .
    <statement-N>
```

Programming Tip: A class can be defined in a function or with an `if` statement.

From the syntax, we see that class definition is quite similar to function definition. It starts with a keyword class followed by the `class_name` and a colon `(:)`. The statement in the definition can be any of these—sequential instructions, decision control statements, loop statements, and can even include function definitions. Variables defined in a class are called *class variables* and functions defined inside a class are called *class methods*. Class variables and class methods are together known as *class members*. The class members can be accessed through class objects. Class methods have access to all the data contained in the instance of the object.

Class definitions can appear anywhere in a program, but they are usually written near the beginning of the program, after the import statements. Note that when a class definition is entered, a new namespace is created, and used as the local scope. Therefore, all assignments to local variables go into this new namespace.

Note A class creates a new local namespace where all its attributes (data and functions) are defined.

9.2.2 Creating Objects

Once a class is defined, the next job is to create an object (or instance) of that class. The object can then access class variables and class methods using the dot operator `(.)`. The syntax to create an object is given as,

```
object_name = class_name()
```

Creating an object or instance of a class is known as *class instantiation*. From the syntax, we can see that class instantiation uses function notation. Using the syntax, an empty object of a class is created. Thus, we see that in Python, to create a new object, call a class as if it were a function. The syntax for accessing a class member through the class object is

Programming Tip: Python does not require the new operator to create an object.

```
object_name.class_member_name
```

Example 9.1 Program to access class variable using class object

```
class ABC:
    var = 10    # class variable
obj = ABC()
print(obj.var)   # class variable is accessed using class object
```

OUTPUT

```
10
```

Programming Tip: `self` in Python works in the same way as the "this" pointer in C++.

In the above program, we have defined a class ABC which has a variable `var` having a value of `10`. The object of the class is created and used to access the class variable using the `dot` operator. Thus, we can think of a class as a *factory* for making objects.

9.2.3 Data Abstraction and Hiding through Classes

In Chapter 2, we had learnt that data abstraction refers to the process by which data and functions are defined in such a way that only essential details are provided to the outside world and the implementation details are hidden. In Python and other object oriented programming languages, classes provide methods to the outside world to provide the functionality of the object or to manipulate the object's data. Any entity outside the world does not know about the implementation details of the class or that method.

Data encapsulation, also called data hiding, organizes the data and methods into a structure that prevents data access by any function (or method) that is not specified in the class. This ensures the integrity of the data contained in the object.

Encapsulation defines different access levels for data variables and member functions of the class. These access levels specifies the access rights, for example,

- Any data or function with access level *public* can be accessed by any function belonging to any class. This is the lowest level of data protection.
- Any data or function with access level *private* can be accessed only by the class in which it is declared. This is the highest level of data protection. In Python, private variables are prefixed with a double underscore (__). For example, `__var` is a private variable of the class.

Note Functions defined inside a class are called class methods.

9.3 CLASS METHOD AND SELF ARGUMENT

Class methods (or functions defined in the class) are exactly same as ordinary functions that we have been defining so far with just one small difference. Class methods must have the first argument named as `self`. This is the first argument that is added to the beginning of the parameter list. Moreover, you do not pass a value for this parameter when you call the method. Python provides its value automatically. The `self` argument refers to the object itself. That is, the object that has called the method. This means that even if a method that takes no arguments, it should be defined to accept the `self`. Similarly, a function defined to accept one parameter will actually take two—`self` and the parameter, so on and so forth.

Since, the class methods uses `self`, they require an object or instance of the class to be used. For this reason, they are often referred to as ***instance methods***.

Note If you have a method which takes no arguments, then you still have to define the method to have a `self` argument.

Consider the program given below which has one class variable and one class method. Observe that the class method accepts no values but still has `self` as an argument. Both the class members are accessed through the object of the class.

Example 9.2 Program to access class members using the class object

```
class ABC():
    var = 10
    def display(self):
        print("In class method.....")
obj = ABC()

print(obj.var)
obj.display()
```

OUTPUT

```
10
In class method.....
```

Programming Tip: You can give any name for the `self` parameter, but you should not do so.

Key points to remember

- The statements inside the class definition must be properly indented.
- A class that has no other statements should have a pass statement at least.
- Class methods or functions that begins with double underscore (__) are special functions with a predefined and a special meaning.

9.4 THE __init__() METHOD (THE CLASS CONSTRUCTOR)

The `__init__()` method has a special significance in Python classes. The `__init__()` method is automatically executed when an object of a class is created. The method is useful to initialize the variables of the class object. Note the `__init__()` is prefixed as well as suffixed by double underscores. The `__init__()` method can be declared as, `def __init__(self, [args...])`. Look at the program given below that uses the `__init__()` method.

Example 9.3 Program illustrating the use of `__init__()` method

```
class ABC():
    def __init__(self,val):
        print("In class method.....")
        self.val = val
        print("The value is : ", val)
obj = ABC(10)
```

OUTPUT

```
In class method.....
The value is :  10
```

In the program, the `__init__()` method accepts one argument `val`. Like any other class method the first argument has to be self. In the `__init__()` method we define a variable as `self.val` which has exactly the same name as that specified in the argument list. Though the two variables have the same name, they are entirely different variables. The `self.val` belongs to the newly created object. Note that we have just created an object in the main module and no where have we called the `__init__()` method. This is because the `__init__()` method is automatically involved when the object of the class is created.

Programming Tip: The `__init__()` method is same as constructor in C++ and Java.

Note It is a good programming habit to initialize all attributes in the `__init__()` method. Although values can be initialized in other methods also but it is not recommended.

9.5 CLASS VARIABLES AND OBJECT VARIABLES

We have seen that a class can have variables defined in it. Basically, these variables are of two types—class variables and object variables. As the name suggests, class variables are owned by the class and object variables are owned by each object. What this specifically means can be understood by using the following points.

- If a class has n objects, then there will be n separate copies of the object variable as each object will have its own object variable.
- The object variable is not shared between objects.
- A change made to the object variable by one object will not be reflected in other objects.

- If a class has one class variable, then there will be one copy only for that variable. All the objects of that class will share the class variable.
- Since there exists a single copy of the class variable, any change made to the class variable by an object will be reflected in all other objects.

Note Class variables and object variables are ordinary variables that are bound to the class's and object's namespace respectively.

Example 9.4 Program to differentiate between class and object variables

```
class ABC():
    class_var = 0    # class variable
    def __init__(self,var):
        ABC.class_var += 1
        self.var = var  # object variable
        print("The Object value is : ", var)
        print("The value of class variable is : ", ABC.class_var)
obj1 = ABC(10)
obj2 = ABC(20)
obj3 = ABC(30)
```

OUTPUT

```
The Object value is :  10
The value of class variable is :  1
The Object value is :  20
The value of class variable is :  2
The Object value is :  30
The value of class variable is :  3
```

Programming Tip: Class variable must be prefixed by the class name and dot operator.

In the above program, we have a class variable `class_var` which is shared by all three objects of the class. It is initialized to zero and each time an object is created, the `class_var` is incremented by 1. Since, the variable is shared by all objects, changes made to `class_var` by one object is reflected in other objects as well. Note that class variable is accessed using the class name followed by the dot operator as the variable belongs to the class.

Then we have object variable which is unique for every object. When an object is created and the `__init__()` method is called, the object variable is initialized. The object variable belongs to only a particular object.

Note Class variables are usually used to keep a count of number of objects created from a class.

We have already seen that one use of class variables or class attributes is to count the number of objects created. Another important use of such variables is to define constants associated with a particular class or provide default attribute values. For example, the code given in the following example uses the class variable to specify a default value for the objects. Now, each individual object may either change it or retain the default value.

Example 9.5 Program illustrating the modification of an instance variable

```
class Number:
    even = 0    # default value
    def check(self, num):
        if num%2 == 0:
            self.even = 1
    def Even_Odd(self, num):
        self.check(num)
        if self.even == 1:
            print(num, "is even")
        else:
            print(num, "is odd")
n = Number()
n.Even_Odd(21)
```

OUTPUT

```
21 is odd
```

Programming Tip: Class attributes are defined at the same indentation level as that of class methods.

Name Clashes: Note that in the above program, we had a class variable even with value `0`. We had set an attribute of the object which has the same name as the class attribute. So here, we are actually *overriding* the class attribute with an instance attribute. The instance (or the object) attribute will take precedence over the class attribute. If we create two objects of `Number`, then both the objects will have their own copy of even. Changes made in one object will not be reflected in the other. But this is not true for a mutable type attribute. Remember that, if you modify a mutable object in one place, the change will be reflected in all other places as well. This difference is reflected in the program given below.

Note Overriding means that the first definition is not available anymore.

Example 9.6 Program modifying a mutable type attribute

```
class Number:
    evens = []
    odds = []
    def __init__(self, num):
        self.num = num
        if num%2 == 0:
            Number.evens.append(num)
        else:
            Number.odds.append(num)
N1 = Number(21)
N2 = Number(32)
N3 = Number(43)
N4 = Number(54)
N5 = Number(65)
```

```
print("Even Numbers are : ", Number.evens)
print("Odd Numbers are : ", Number.odds)
```

OUTPUT

```
Even Numbers are :  [32, 54]
Odd Numbers are :  [21, 43, 65]
```

In the aforementioned program, we have defined two lists as class variables which are of mutable types. The class variable is being shared among all objects. So any change made by any of the object will be reflected in the final list. So, whether you write, `Number.evens`, `self.evens`, `N1.evens`, `N2.evens`, `N3.evens`, `N4.evens`, or `N5.evens`, it will all print the same list.

Note A variable defined inside the class is known as class attribute or simply attribute.

9.6 THE __del__() METHOD

In the previous section, we saw the `__init__()` method which initializes an object when it is created. Similar to the `__init__()` method, we have the `__del__()` method which does just the opposite work. The `__del__()` method is automatically called when an object is going out of scope. This is the time when an object will no longer be used and its occupied resources are returned back to the system so that they can be reused as and when required. You can also explicitly do the same using the `del` keyword.

Programming Tip: `__del__()` method is analogous to destructors in C++ and Java.

Example 9.7 Program to illustrate the use of `__del__()` method

```
class ABC():
    class_var = 0    # class variable
    def __init__(self,var):
        ABC.class_var += 1
        self.var = var  # object variable
        print("The Object value is : ", var)
        print("The value of class variable is : ", ABC.class_var)
    def __del__(self):
        ABC.class_var -= 1
        Print("Object with value %d is going out of scope"%self.var)
obj1 = ABC(10)
obj2 = ABC(20)
obj3 = ABC(30)
del obj1
del obj2
del obj3
```

OUTPUT

```
The Object value is :  10
```

Programming Tip: In C++ and Java, all members are private by default but in Python, they are public by default


```
The value of class variable is :  1
The Object value is :  20
The value of class variable is :  2
The Object value is :  30
The value of class variable is :  3
Object with value 10 is going out of scope
Object with value 20 is going out of scope
Object with value 30 is going out of scope
```

Thus, we see that the `__del__()` is invoked when the object is about to be destroyed. This method might be used to clean up any resources used by it.

9.7 OTHER SPECIAL METHODS

In this section, we will read about some other functions that have a special meaning in Python. These functions include:

- `__repr__()`: This method has built-in function with syntax `repr(object)`. It returns a string representation of an object. The function works on any object, not just class instances.
- `__cmp__()`: This method is called to compare two class objects. In fact, the function can even compare any two Python objects by using the equality operator (`==`). For class instances, the `__cmp__()` method can be defined to write the customized comparison logic.
- `__len__()`: This method function has a built-in function that has the syntax `len(object)`. It returns the length of an object.

Example 9.8 Program to illustrate the use of special methods in Python classes

```
class ABC():
    def __init__(self, name, var):
        self.name = name
        self.var = var
    def __repr__(self):
        return repr(self.var)
    def __len__(self):
        return len(self.name)
    def __cmp__(self, obj):
        return self.var - obj.var
obj = ABC("abcdef", 10)
print("The value stored in object is : ", repr(obj))
print("The length of name stored in object is : ", len(obj))
obj1 = ABC("ghijkl", 1)
val = obj.__cmp__(obj1)
if val == 0:
    print("Both values are equal")
elif val == -1:
    print("First value is less than second")
else:
    print("Second value is less than first")
```

OUTPUT

```
The value stored in object is :  10
The length of name stored in object is :  6
Second value is less than first
```

Python has a lot of other special methods that let classes act like numbers so that you can perform arithmetic operations like add, subtract, etc. on them. All those methods cannot be discussed here but other special methods are:

- The `__call__()` method: The method lets a class act like a function so that its instance can be called directly in `obj(arg1,arg2,...)`.
- The `__lt__()`, `__le__()`, `__eq__()`, `__ne__()`, `__gt__()`, and `__ge__()`: These methods are used to compare two objects.
- The `__hash__()` method: It is used to calculate a hash for the object. The hash will decide a placing of objects in data structures such as sets and dictionaries.
- The `__iter__()` method: This method is used for iteration over objects, for example, for loops.
- The `__getitem__()` method: This method is used for indexing. It can be declared as, `def __getitem__(self, key)`
- The `__setitem__()` method: This method is used to assign an item to indexed values. It can be declared as, `def __setitem__(self, key, value)`

Example 9.9 Program to demonstrate the use of `__getitem__()` and `__setitem__()` methods

```
class Numbers:
        def __init__(self,myList):
                self.myList = myList

        def __getitem__(self,index):
                return self.myList[index]
        def __setitem__(self,index, val):
                self.myList[index] = val
NumList = Numbers([1,2,3,4,5,6,7,8,9])
print(NumList[5])
NumList[3] = 10
print(NumList.myList)
```

OUTPUT

```
6
[1, 2, 3, 10, 5, 6, 7, 8, 9]
```

Note Trying to access an attribute of an instance that is not defined or a method that is undefined causes an **AttributeError**.

9.8 PUBLIC AND PRIVATE DATA MEMBERS

Public variables are those variables that are defined in the class and can be accessed from anywhere in the program, of course using the dot operator. Here, anywhere from the program means that the public variables can be accessed from within the class as well as from outside the class in which it is defined.

Private variables, on the other hand, are those variables that are defined in the class with a double score prefix (__). These variables can be accessed only from within the class and from nowhere outside the class.

Example 9.10 Program to illustrate the difference between public and private variables

```
class ABC():
    def __init__(self, var1, var2):
        self.var1 = var1
        self.__var2 = var2
    def display(self):
        print("From class method, Var1 = ", self.var1)
        print("From class method, Var2 = ", self.__var2)
obj = ABC(10, 20)
obj.display()
print("From main module, Var1 = ", obj.var1)
print("From main module, Var2 = ", obj.__var2)
```

OUTPUT

```
From class method, Var1 = 10
From class method, Var2 = 20
From main module, Var1 = 10
From main module, Var2 =
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 11, in <module>
    print("From main module, Var2 = ", obj.__var2)
AttributeError: ABC instance has no attribute '__var2'
```

As a good programming habit, you should never try to access a private variable from anywhere outside the class. But if for some reason, you need to do it, then you can access the private variable using the following syntax,

```
objectname._classname__privatevariable
```

So, to remove the error from the above code, you could have written the last statement as

```
print("From main module, Var2 = ", obj._ABC__var2)
```

9.9 PRIVATE METHODS

Remember that, private attributes should not be accessed from anywhere outside the class. Like private attributes, you can even have private methods in your class. Usually, we keep those methods as private which have implementation details. So like private attributes, you should also not use a private method from anywhere outside the class. However, if it is very necessary to access them from outside the class, then they are accessed with a small difference. A private method can be accessed using the object name as well as the class name from outside the class. The syntax for accessing the private method in such a case would be,

```
objectname._classname__privatemethodname
```

Example 9.11 Program to illustrate the use of a private method

```
class ABC():
    def __init__(self, var):
        self.__var = var
    def __display(self):
        print("From class method, Var = ", self.__var)
obj = ABC(10)
obj._ABC__display()
```

OUTPUT

```
From class method, Var =  10
```

Note Like private attributes, Python also allows you to have private methods to discourage people from accessing parts of a class that have implementation details.

9.10 CALLING A CLASS METHOD FROM ANOTHER CLASS METHOD

You can call a class method from another class method by using the `self`. This is shown in the program given below.

Example 9.12 Program to call a class method from another method of the same class

```
class ABC():
    def __init__(self, var):
        self.var = var
    def display(self):
        print("Var is = ", self.var)
    def add_2(self):
        self.var += 2
        self.display()
obj = ABC(10)
obj.add_2()
```

OUTPUT

```
Var is = 12
```

Key points to remember

- Like functions and modules, class also has a documentation string, which can be accessed using `className.__doc__`. The lines of code given below specifies the `docstring`.

```
class ABC:
    '''This is a docstring. I have created a new class'''
pass
```

- Class methods can reference global names in the same way as ordinary functions.

Example 9.13 Program to show how a class method calls a function defined in the global namespace

```
def scale_10(x):
    return x*10
class ABC():
    def __init__(self, var):
        self.var = var
    def display(self):
        print("Var is = ", self.var)
    def modify(self):
        self.var = scale_10(self.var)
obj = ABC(10)
obj.display()
obj.modify()
obj.display()
```

OUTPUT

```
Var is =  10
Var is =  100
```

Note A class is never used as a global scope.

- Unlike in C++ and Java, Python allows programmers to add, remove, or modify attributes of classes and objects at any time.

Example 9.14 Program to add variables to a class at run-time

```
class ABC():
    def __init__(self, var):
        self.var = var
    def display(self):
        print("Var is = ", self.var)
obj = ABC(10)
obj.display()
obj.new_var = 20  # variable added at run-time
print("New Var = ", obj.new_var)
obj.new_var = 30  # modifying newly added variable
print("New Var after modification = ", obj.new_var)
del obj.new_var   # newly created variable is deleted
print("New Var after deletion = ", obj.new_var)
```

OUTPUT

```
Var is =  10
New Var =  20
New Var after modification =  30
```

```
New Var after deletion =
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 13, in <module>
  print("New Var after deletion = ", obj.new_var)
AttributeError: ABC instance has no attribute 'new_var'
```

9.11 BUILT-IN FUNCTIONS TO CHECK, GET, SET, AND DELETE CLASS ATTRIBUTES

Python has some built-in functions that can also be used to work with attributes (variables defined in class). You can use these functions to check whether a class has a particular attribute or not, get its value if it exists, set a new value, or even delete that attribute. These built-in functions include the following.

`hasattr(obj,name)`: The function is used to check if an object possesses the attribute or not.

`getattr(obj, name[, default])`: The function is used to access or get the attribute of object. Since `getattr()` is a built-in function and not a method of the class, it is not called using the dot operator. Rather, it takes the object as its first parameter. The second parameter is the name of the variable as a string, and the optional third parameter is the default value to be returned if the attribute does not exist. If the attribute name does not exist in the object's namespace and the default value is also not specified, then an exception will be raised. Note that `getattr(obj, 'var')` is same as writing `obj.var`. However, you should always try to use the latter variant.

`setattr(obj,name,value)`: The function is used to set an attribute of the object. If attribute does not exist, then it would be created. The first parameter of the `setattr()` function is the object, the second parameter is the name of the attribute, and the third is the new value for the specified attribute.

`delattr(obj,name)`: The function deletes an attribute. Once deleted, the variable is no longer a class or object attribute.

Example 9.15 Program to demonstrate the use of `getattr()`, `seattr()`, and `delattr()` functions

```
class ABC():
    def __init__(self, var):
        self.var = var
    def display(self):
        print("Var is = ", self.var)
obj = ABC(10)
obj.display()
print("Check if object has attribute var .....", hasattr(obj,'var'))
getattr(obj,'var')
setattr(obj,'var', 50)
print("After setting value, var is : ", obj.var)
setattr(obj,'count',10)
print("New variable count is created and its value is : ", obj.count)
delattr(obj,'var')
print("After deleting the attribute, var is : ", obj.var)
```

OUTPUT

```
Var is =  10
```



```
Check if object has attribute var ..... True
After setting value, var is :  50
New variable count is created and its value is :  10
After deleting the attribute, var is :
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 15, in <module>
    print "After deleting the attribute, var is : ", obj.var
AttributeError: ABC instance has no attribute 'var'
```

9.12 BUILT-IN CLASS ATTRIBUTES

Every class defined in Python has some built-in attributes associated with it. Like other attributes, these attributes can also be accessed using dot operator.

`.__dict__`: The attributes gives a dictionary containing the class's or object's (with whichever it is accessed) namespace.

`.__doc__`: The attribute gives the class documentation string if specified. In case the documentation string is not specified, then the attribute returns `None`.

`.__name__`: The attribute returns the name of the class.

`.__module__`: The attribute gives the name of the module in which the class (or the object) is defined.

`.__bases__`: The attribute is used in inheritance (discussed in Chapter 10) to return the base classes in the order of their occurrence in the base class list. As for now, it returns an empty tuple.

Example 9.16 Program to demonstrate the use of built-in class attributes

```
class ABC():
    def __init__(self, var1, var2):
        self.var1 = var1
        self.var2 = var2
    def display(self):
        print("Var1 is = ", self.var1)
        print("Var2 is = ", self.var2)
obj = ABC(10, 12.34)
obj.display()
print("object.__dict__ - ", obj.__dict__)
print("object.__doc__ - ", obj.__doc__)
print("class.__name__ - ", ABC.__name__)
print("object.__module__ - ", obj.__module__)
print("class.__bases__ - ", ABC.__bases__)
```

OUTPUT

```
Var1 is =  10
Var2 is =  12.34
obj.__dict__ -  {'var1': 10, 'var2': 12.34}
obj.__doc__ -  None
```

```
class.__name__ -  ABC
obj.__module__ -  __main__
class.__bases__ -  ()
```

Note The **__repr__()** special method is used for string representation of the instance.

9.13 GARBAGE COLLECTION (DESTROYING OBJECTS)

Python performs automatic garbage collection. This means that it deletes all the objects (built-in types or user defined like class objects) automatically that are no longer needed and that have gone out of scope to free the memory space. The process by which Python periodically reclaims unwanted memory is known as *garbage collection*.

Python's garbage collector runs in the background during program execution. It immediately takes action (of reclaiming memory) as soon as an object's reference count reaches zero.

Let us recall that an object's reference count increases when we create its aliases. That is, when we assign an object a new name or place it within a list, tuple, or dictionary. Similarly, the object's reference count becomes zero when it is deleted with `del` statement. Moreover, each time the object's reference is reassigned, or its reference goes out of scope, its reference count decreases.

Note When an object's reference count reaches zero, Python recollects the memory used by it.

Consider the following examples which illustrate the way in which reference count changes for a given object.

Programming Tip: Object of a class can be deleted using del statement.

```
var1 = 10       # Create object var1
var2 = var1     # Increase ref. count  of var1 - object assignment
var3 = [var2]   # Increase ref. count  of var1 - object used in a list
var2 = 50       # Decrease ref. count of var1 - reassignment
var3[0] = -1    # Decrease ref. count  of var1 - removal from list
del var1        # Decrease ref. count  of var1 - object deleted
```

PROGRAMMING EXAMPLES

Program 9.1 Write a program that uses class to store the name and marks of students. Use list to store the marks in three subjects.

```
class Students:
    def __init__(self, name):
        self.name = name
        self.marks = []
    def enterMarks(self):
        for i in range(3):
            m = int(input("Enter the marks of %s in subject %d : "%(self.name,i+1)))
            self.marks.append(m)
    def display(self):
        print(self.name," got ", self.marks)
s1 = Students("Anisha")
s1.enterMarks()
s2 = Students("Jignesh")
s2.enterMarks()
s1.display()
s2.display()
```

OUTPUT

```
Enter the marks of Anisha in subject 1 : 89
Enter the marks of Anisha in subject 2 : 88
Enter the marks of Anisha in subject 3 : 87
Enter the marks of Jignesh in subject 1 : 78
Enter the marks of Jignesh in subject 2 : 90
Enter the marks of Jignesh in subject 3 : 87
Anisha  got  [89, 88, 87]
Jignesh  got  [78, 90, 87]
```

Program 9.2 Write a program with class Employee that keeps a track of the number of employees in an organization and also stores their name, designation, and salary details.

```
class Employee:
   empCount = 0
   def __init__(self, name, desig, salary):
      self.name = name
      self.desig = desig
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print("There are %d employees" % Employee.empCount)

   def displayDetails(self):
      print("Name : ", self.name,  ", Designation : ", self.desig, ", Salary :
      ", self.salary)
e1 = Employee("Farhan", "Manager", 100000)
e2 = Employee("Mike", "Team Leader", 90000)
e3 = Employee("Niyam", "Programmer", 80000)
e4 = Employee("Ojas", "Office Assistant", 60000)
e4.displayCount()
print("Details of second employee - \n ")
e2.displayDetails()
```

OUTPUT

```
There are 4 employees
Details of second employee -
Name :  Mike , Designation :  Team Leader , Salary :  90000
```

Program 9.3 Write a program that has a class Person storing name and date of birth (DOB) of a person. The program should subtract the DOB from today's date to find out whether a person is eligible to vote or not.

```
import datetime
class Person():
    def __init__(self, name, dob):
        self.name = name
        self.dob = dob
    def check(self):
        today = datetime.date.today()
        age = today.year - self.dob.year
        if today < datetime.date(today.year, self.dob.month, self.dob.day):
            age -= 1
        if age>=18:
            print(self.name, ", Congratulations... You are eligible to vote.")
        else:
            print(self.name, ", Sorry... You should be at least 18 years of age to
cast your vote.")
P = Person("Saesha", datetime.date(1998, 12, 11))
P.check()
```

OUTPUT

```
Saesha , Congratulations... You are eligible to vote.
```

Program 9.4 Write a program that has a class Circle. Use a class variable to define the value of constant PI. Use this class variable to calculate area and circumference of a circle with specified radius.

```
class Circle:
    PI = 3.14159
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return Circle.PI*self.radius*self.radius
    def circumference(self):
        return 2*Circle.PI*self.radius
C = Circle(7.5)
print("AREA = ", C.area())
print("CIRCUMFERENCE = ", C.circumference())
```

OUTPUT

```
AREA =  176.7144375
CIRCUMFERENCE =  47.12385
```


Program 9.5 Write a program that has a class student that stores roll number, name, and marks (in three subjects) of the students. Display the information (roll number, name, and total marks) stored about the student.

```
class student:
   __marks = []
   def set_data(self,r,n,m1,m2,m3):
      student.__rollno = r
      student.__name = n
      student.__marks.append(m1)
      student.__marks.append(m2)
      student.__marks.append(m3)
   def display_data(self):
      print("Student Details")
      print("Roll Number :",student.__rollno)
      print("Name :",student.__name)
      print("Marks :",self.total())
   def total(self):
      m = student.__marks
      return m[0]+m[1]+m[2]

r = int(input("Enter the roll number : "))
n = input("Enter the name : ")
m1 = int(input("Enter the marks in first subject : "))
m2 = int(input("Enter the marks in first subject : "))
m3 = int(input("Enter the marks in first subject : "))
s1 = student()
s1.set_data(r,n,m1,m2,m3)
s1.display_data()
```

OUTPUT

```
Enter the roll number : 123
Enter the name : Shivan
Enter the marks in first subject : 89
Enter the marks in first subject : 90
Enter the marks in first subject : 92
Student Details
Roll Number : 123
Name : Shivan
Marks : 271
```

Program 9.6 Write a class Rectangle that has attributes Length and Breadth and a method area which returns the area of the rectangle.

```
class Rectangle:
   def get_data(self):
      Rectangle.length = int(input("Enter the length : "))
      Rectangle.breadth = int(input("Enter the breadth : "))
```

```
    def show_data(self):
        print("Length =",Rectangle.length,"\t Breadth =",Rectangle.breadth)
    def area(self):
        print("Area =",Rectangle.length*Rectangle.breadth)

rect = Rectangle()
rect.get_data()
rect.show_data()
rect.area()
```

OUTPUT

```
Enter the length : 10
Enter the breadth : 5
Length = 10         Breadth = 5
Area = 50
```

Program 9.7 Write a program that has a class fraction with attributes numerator and denominator. Enter the values of the attributes and print the fraction in simplified form.

```
class fraction:
    def get_data(self):
        self.__num = int(input("Enter the numerator : "))
        self.__deno = int(input("Enter the denominator : "))
        if(self.__deno == 0):
            print("Fraction not possible")
            exit()

    def display_data(self):
        self.__simplify()
        print(self.__num,"/",self.__deno)

    def __simplify(self):
        print("The simplified fraction is :")
        common_divisor = self.__GCD(self.__num,self.__deno)
        self.__num = self.__num/common_divisor
        self.__deno = self.__deno/common_divisor

    def __GCD(self, a, b):
        if(b==0):
            return a
        else:
            return self.__GCD(b, a%b)

f = fraction()
f.get_data()
f.display_data()
```


OUTPUT

```
Enter the numerator : 20
Enter the denominator : 100
The simplified fraction is : 1.0 / 5.0
```

Program 9.8 Write a program that has a class store which keeps a record of code and price of each product. Display a menu of all products to the user and prompt him to enter the quantity of each item required. Generate a bill and display the total amount.

```
class store:
    __item_code = []
    __price = []

    def get_data(self):
            for i in range(5):
                    self.__item_code.append(int(input("Enter the code of item : ")))
                    self.__price.append(int(input("Enter the price : ")))

    def display_data(self):
        print("ITEM CODE \t PRICE")
        for i in range(5):
            print(self.__item_code[i],"\t\t",self.__price[i])

    def calculate_bill(self, quant):
        total_amount = 0
        for i in range(5):
            total_amount = total_amount+self.__price[i]*quant[i]
        print("************BILL***************")
        print("ITEM \t PRICE \t QUANTITY \t SUBTOTAL")
        for i in range(5):
          print(self.__item_code[i],"\t",self.__price[i],"\t",quant[i],"\t
          t",quant[i]*self.__price[i])
        print("*********************************")
        print("Total =", total_amount)

s = store()
s.get_data()
s.display_data()
q = []
print("Enter the quantity of each item : ")
for i in range(5):
    q.append(int(input()))
s.calculate_bill(q)
```

OUTPUT

```
Enter the code of item : 123
Enter the price : 9876
Enter the code of item : 345
```

```
Enter the price : 8765
Enter the code of item : 456
Enter the price : 7654
Enter the code of item : 567
Enter the price : 6543
Enter the code of item : 890
Enter the price : 5436
ITEM CODE      PRICE
123            9876
345            8765
456            7654
567            6543
890            5436
Enter the quantity of each item :
1
2
1
3
2
***********BILL***************
ITEM    PRICE     QUANTITY      SUBTOTAL
123     9876      1             9876
345     8765      2             17530
456     7654      1             7654
567     6543      3             19629
890     5436      2             10872
*********************************
Total = 65561
```

Program 9.9 Write a program that has a class Numbers with values stored in a list. Write a class method to find the largest value.

```
''' Program to use a constructor to create an array and find the largest element
from that array '''
class Numbers:
    def __init__(self):
        self.values = []

    def find_max(self):
        max = ''
        for i in self.values:
            if(i > max):
                max = i
        print('Maximum element : %r' %max)

    def insert_element(self):
        value = input('Enter value : ')
        self.values.append(value)
```



```
x = Numbers()
ch = 'y'
while(ch == 'y'):
    x.insert_element()
    ch = input('Do you wish to enter more elements?')
x.find_max()
```

OUTPUT

```
Enter value : hi
Do you wish to enter more elements?y
Enter value : bye
Do you wish to enter more elements?y
Enter value : cheer
Do you wish to enter more elements?y
Enter value : smile
Do you wish to enter more elements?n
Maximum element : 'smile'
```

Program 9.10 Write a class that stores a string and all its status details such as number of uppercase characters, vowels, consonants, spaces, etc.

```
class String:
    def __init__(self):
        self.vowels = 0
        self.spaces = 0
        self.consonants = 0
        self.uppercase = 0
        self.lowercase = 0
        self.string = str(input("Enter string : "))

    def count_uppercase(self):
        for letter in self.string:
            if(letter.isupper()):
                self.uppercase+=1

    def count_lowercase(self):
        for letter in self.string:
            if(letter.islower()):
                self.lowercase+=1

    def count_vowels(self):
        for letter in self.string:
            if(letter in ('a','e','i' ,'o','u')):
                self.vowels+=1
            elif(letter in ('A','E','I','O','U')):
                self.vowels+=1

    def count_spaces(self):
        for letter in self.string:
```

```
            if(letter == ' '):
                self.spaces+=1

    def count_consonants(self):
        for letter in self.string:
            if(letter not in ('a','e','i','o','u','A','E','I','O','U',' ')):
                self.consonants+=1

    def compute_stat(self):
        self.count_uppercase()
        self.count_lowercase()
        self.count_vowels()
        self.count_spaces()
        self.count_consonants()

    def show_stat(self):
        print('Vowels : %d'%self.vowels)
        print('Consonants : %d' %self.consonants)
        print('Spaces : %d' %self.spaces)
        print('Uppercase : %d' %self.uppercase)
        print('Lowercase : %d' %self.lowercase)
s = String()
s.compute_stat()
s.show_stat()
```

OUTPUT

```
Enter string : This program must show statistics for this string
Vowels : 11
Consonants : 31
Spaces : 7
Uppercase : 1
Lowercase : 41
```

Program 9.11 Write a program that uses datetime module within a class. Enter manufacturing date and expiry date of the product. The program must display the years, months, and days that are left for expiry.

```
import datetime
class Product:
    def __init__(self):
        self.manufacture  =  datetime.datetime.strptime(input("Enter manufacturing
date (mm/dd/yyyy): "),'%m/%d/%Y')
        self.expiry = datetime.datetime.strptime(input("Enter expiry date (mm/dd/
yyyy): "),'%m/%d/%Y')

    def time_to_expire(self):
        today = datetime.datetime.now()
        if(today > self.expiry):
            print('Product has already expired.')
        else:
            time_left = self.expiry.date() - datetime.datetime.now().date()
            print('Time left : ',time_left)

    def show(self):
        print('Expiry : ',self.expiry)
        print('Manufacturing : ',self.manufacture)

x = Product()
x.time_to_expire()
```


OUTPUT

```
Enter manufacturing date (mm/dd/yyyy): 1/1/2013
Enter expiry date (mm/dd/yyyy): 1/1/2017
Time left :  232 days, 0:00:00
```

Program 9.12 Write a program to deposit or withdraw money in a bank account.

```
class Account:
    def __init__(self):
        self.balance = 0
        print('New Account Created.')

    def deposit(self):
        amount = float(input('Enter amount to deposit : '))
        self.balance+=amount
        print('New Balance : %f' %self.balance)

    def withdraw(self):
        amount = float(input('Enter amount to withdraw : '))
        if(amount > self.balance):
            print('Insufficient balance')
        else:
            self.balance-=amount
            print('New Balance : %f' %self.balance)

    def enquiry(self):
        print('Balance : %f' %self.balance)

account = Account()
account.deposit()
account.withdraw()
account.enquiry()
```

OUTPUT

```
New Account Created.
Enter amount to deposit : 1000
New Balance : 1000.000000
```

```
Enter amount to withdraw : 25.23
New Balance : 974.770000
Balance : 974.770000
```

Program 9.13 Write a menu driven program that keeps record of books and journals available in a library.

```
class Book:
    def __init__(self):
        self.title = ""
        self.author = ""
        self.price = 0

    def read(self):
        self.title = input("Enter Book Title : ")
        self.author = input("Enter Book Author : ")
        self.price = float(input("Enter Book Price : "))

    def display(self):
        print("Title : ",self.title)
        print("Author : ",self.author)
        print("Price : ",self.price)
        print("\n")

my_books = []
ch = 'y'
while(ch == 'y'):
    print('''
    1. Add New Book
    2. Display Books
    ''')
    choice = int(input("Enter choice : "))
    if(choice == 1):
        book = Book()
        book.read()
        my_books.append(book)
    elif(choice == 2):
        for i in my_books:
            i.display()
    else:
        print("Invalid Choice")

    ch = input("Do you want to continue..")
print("Bye!")
```

OUTPUT

```
   1. Add New Book
   2. Display Books

Enter choice : 1
Enter Book Title : OOPS with C++
Enter Book Author : Balaguruswamy
Enter Book Price : 385
Do you want to continue..y

   1. Add New Book
   2. Display Books

Enter choice : 1
Enter Book Title : Computer Networks
Enter Book Author : Forouzan
Enter Book Price : 550
Do you want to continue..y
   1. Add New Book
   2. Display Books

Enter choice : 1
Enter Book Title : Computer Fundamentals
Enter Book Author : P.K. Sinha
Enter Book Price : 250
Do you want to continue..y
   1. Add New Book
   2. Display Books

Enter choice : 2
Title :  OOPS with C++
Author :  Balaguruswamy
Price :  385.0

Title :  Computer Networks
Author :  Forouzan
Price :  550.0

Title :  Computer Fundamentals
Author :  P.K. Sinha
Price :  250.0

Do you want to continue..n
Bye!
```

Programming Tip: The ideal way is to define the classes in a separate file, and then import them in the main program file using `import` statement.

9.14 CLASS METHODS

Till now, we have seen that methods defined in a class are called by an instance of a class. These methods automatically take `self` as the first argument. *Class methods* are little different from these ordinary methods. First, they are called by a class (not by instance of the class). Second, the first argument of the `classmethod` is `cls`, not the `self`.

Class methods are widely used for factory methods, which instantiate an instance of a class, using different parameters from those usually passed to the class constructor. The program code given in the following example illustrates this concept.

Note Class methods are marked with a `classmethod` decorator.

Example 9.17 Program to demonstrate the use of `classmethod`

```
class Rectangle:
        def __init__(self,length, breadth):
                self.length = length
                self.breadth = breadth

        def area(self):
                return self.length * self.breadth
        @classmethod
        def Square(cls,side):
                return cls(side,side)
S = Rectangle.Square(10)
print("AREA = ", S.area())
```

OUTPUT

```
AREA =  100
```

9.15 STATIC METHODS

Static methods are a special case of methods. Any functionality that belongs to a class, but that does not require the object is placed in the static method. Static methods are similar to class methods. The only difference is that a static method does not receive any additional arguments. They are just like normal functions that belong to a class.

Programming Tip: A static method is marked with the `staticmethod` decorator.

Remember that, a static method does not use the `self` variable and is defined using a built-in function named `staticmethod`. Python has a handy syntax, called a *decorator*, to make it easier to apply the `staticmethod` function to the method function definition. The syntax for using the `staticmethod` decorator is given as,

```
@staticmethod
def name(args...):
    statements
```

A static method can be called either on the class or on an instance. When it is called with an instance, the instance is ignored except for its class.

Note A decorator is a syntactic convenience that takes in a function, adds some functionality to it and then returns it. The syntax of a decorator uses the @ character as a prefix to the function. Using decorators is also called *metaprogramming* because a part of the program tries to modify another part of the program at compile time.

Example 9.18 Program to illustrate `static` method

```
class Choice:
        def __init__(self, subjects):
                self.subjects = subjects
```



```
        @staticmethod
        def validate_subject(subjects):
                if "CSA" in subjects:
                        print("This option is no longer available.")
                else:
                        return True
subjects = ["DS","CSA","FoC","OS","ToC"]
if all(Choice.validate_subject(i) for i in subjects):
        ch = Choice(subjects)
        print("You have been allotted the subjects : ", subjects)
```

OUTPUT

```
This option is no longer available.
```

Note A static method does not depend on the state of the object.

__new__() Method: This is a static method which is called to create a new instance of class cls. It takes the class of which an instance was requested as its first argument. The remaining arguments are those passed to the object constructor expression (the call to the class). This method returns the new object instance of cls.

The `__new__()` method allows subclasses of immutable types (like integer, string, or tuple) to customize instance creation. It is also commonly overridden in custom metaclasses in order to customize class creation.

The `__new__()` and `__init__()` methods together are used for constructing objects.

Summary

- Classes and objects are the two main aspects of object oriented programming.
- Classes provides a blueprint or a template using which objects are created.
- Class methods have access to all the data contained in the instance of the object.
- Class definitions can appear anywhere in a program, but they are usually written near the beginning of the program, after the `import` statements.
- Class methods must have the first argument named as `self`.
- The `__init__()` method is automatically executed when an object of a class is created. The method is useful to initialize the variables of the class object.
- The `__del__()` method is automatically called when an object is going out of scope.
- Public variables are those variables that are defined in the class and can be accessed from anywhere in the program.
- Class methods are used for factory methods, which instantiate an instance of a class. It uses different parameters from those usually passed to the class constructor.

Glossary

Attribute Data items that makes up an instance.
Class A user-defined prototype for an object that defines a set of attributes (class variables and instance variables) and methods that are accessed via dot notation.
Class variable A variable defined within a class that is shared by all instances of a class.
Data member A variable (class variable or instance variable) defined within the class that holds data associated with a class and its objects.
Instance Object of a class.
Instance variable A variable that is defined inside a class method and belongs only to the current instance of the class.
Instantiation The process of creating an instance of a class.
Method Function defined in a class definition and is invoked on instances of that class.
Namespace A mapping from names to objects in such a way that there is no relation between names in different namespaces.