Advanced Programming-Python (Module-3)

Presented by

Siddhanta Borah

Contents of Todays Discussion

- Concept of Object-Oriented-Programming (OOP)
  - Data Encapsulation
  - Polymorphism
- Definition of class
- Creating objects
  - Mutable nature of object
- Object as argument
- Objects as return values

CONCEPT OF OBJECT ORIENTED PROGRAMMING

As we know, Python is an object-oriented programming (OOP) language and provides all the features required to support object-oriented programming. OOP mainly focuses on the objects and classes while procedural programming focuses on the functions and methods.

OOP is based on the implementation of real world objects in programming. Such a concept of programming contains objects that contain the data in the form of attributes and classes that contains methods. In this approach, a problem is considered in terms of objects that can be involved in finding the solution to the problem instead of procedures. Hence, through this approach, a person can relate a problem to the real world objects and can work towards its solution with relative ease.

Object is an instance of a class. A class is a collection of data (variables) and methods (functions). A class is the basic structure of an object and is a set of attributes, which can be data members and method members.

Let us understand the concept with an example. We can relate class to a sketch or model of a building. That sketch contains all the information about the structure of the building, such as floors, doorways, exits, rooms, etc. Now, according to our example, the building is an object. Just as various buildings can be based on one model, so too can a class have many objects associated with it.

SOME IMPORTANT TERMS IN OOP

- **Class:** Classes are defined by the user; the class provides the basic structure for an object. It consists of data members and method members that are used by the instances (objects) of the class.
- **Data Member:** A variable defined in either a class or an object; it holds the data associated with the class or object.
- **Instance Variable:** A variable that is defined in a method; its scope is only within the object that defines it.
- **Class Variable:** A variable that is defined in the class and can be used by all the instances of that class.
- **Instance:** An object is an instance of the class.
- **Instantiation:** The process of creation of an object of a class.
- **Method:** Methods are the functions that are defined in the definition of class and are used by various instances of the class.
- **Function Overloading:** A function defined more than one time with different behaviours is known as function overloading. The operations performed by these functions are different.
- **Inheritance:** A class 'A' that can use the characteristics of another class 'B' is said to be a derived class, i.e., a class inherited from 'B'. The process is called inheritance.

DATA ENCAPSULATION

In OOP, restrictions can be imposed on the access to methods and variables. Such restrictions can be used to avoid accidental modification in the data and are known as Encapsulation. Encapsulation is an important feature in OOP.

In fact, we can say that OOP relies strictly on Data Encapsulation.

Most of us are already familiar with the term abstraction. Abstraction means data-hiding. Encapsulation and abstraction can be used as synonyms since both of them relate to the data-hiding concept.

Generally, in the context of programming, we can restrict the access to some of the object's components, ensuring that these components cannot be accessed from outside the object but from inside the object only. For accessing these types of data, some special methods are used.

These methods are known as `getters()` and `setters()`.

POLYMORPHISM

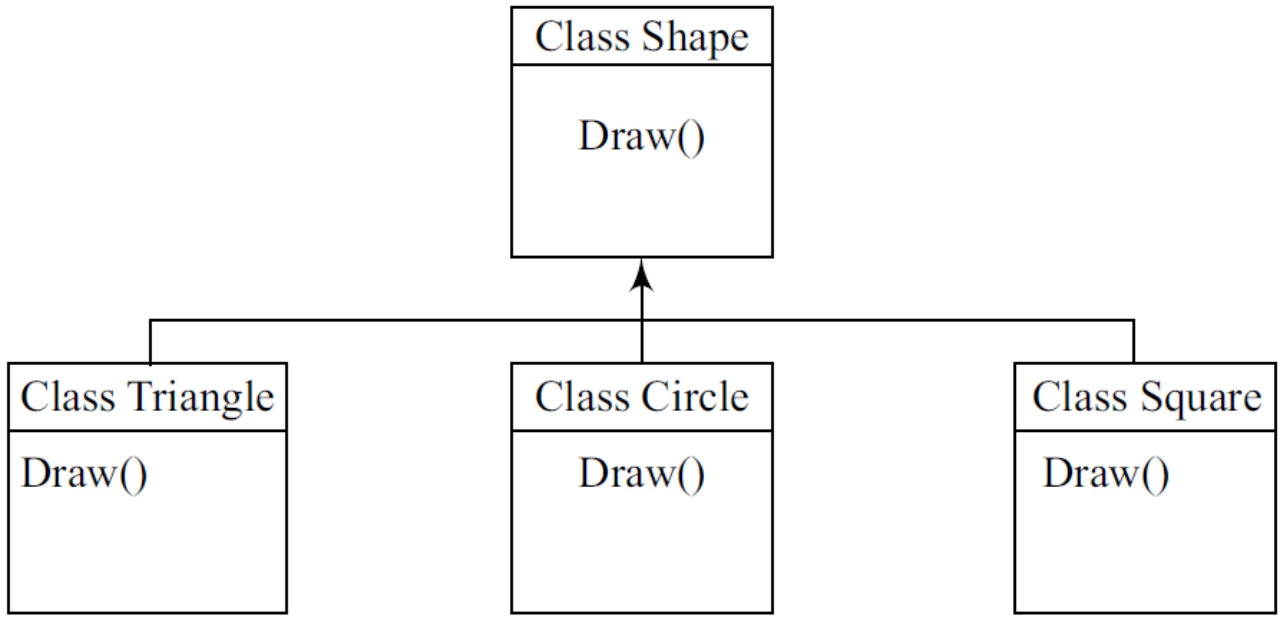
The word 'Poly' means 'many'. Therefore, the term 'polymorphism' means that the object of a class can have many different forms to respond in different ways to any message or action.

In other words, polymorphism is the capability for a message or data to be processed in one or more ways.

Let us look at an example:

If a base class is mammals, then horse, human, and cat are its subclasses. All the mammals can see in the daytime. Therefore, if the message 'see in the daytime' is passed to mammals, all the mammals including the human, the horse and the cat will respond to it. Whereas, if the message 'see during the night time' is passed to the mammals, then only the cat will respond to the message as it can see during the night as well as in the daytime. Hence, the cat, which is a mammal, can behave differently from the other mammals.

This is called polymorphism and is illustrated in Fig right.



DEFINITION OF CLASS

A class can be defined as a blue print or a previously defined structure from which objects are made. It can also be defined as a group of objects that share similar attributes and relationships with each other.

For example:

- Fruit is a class, and apple, mango and banana are its objects. The attributes of these objects can be color, taste, etc.
- Vehicle is a class and car, scooter, bus, truck, etc., can be its objects. The attributes of these objects can be speed, brake, power of engine, etc.

In Python, a class is defined by using a keyword `class`. After that, the first statement can be a docstring (optional) that contains the information about the class. Now, in the body of class, the attributes are defined. These attributes can be data members or method members.

In Python, as soon as we define a class, the interpreter instantly creates an object that has the same name as the class name. Although, we can create more objects of the same class. With the help of objects, we can access the attributes defined in the class.

DEFINITION OF CLASS (CONTD..)

Syntax

```
class class_name:
    'This is docstring which is optional'
    class_suite
```

A new local new space is created by a Class, where all its attributes (data or function) are defined. Special attributes with double underscores (_)are also present, for example- _doc_ gives the docstring of that class. As soon as the class is defined, a new class object is created with same name, which allows access to the different attributes, also to instantiate new object of that class.

Example

```
>>>class Student:
...  'student details'
... def fill_details(self,name,branch,year):
...         self.name = name
...         self.branch = branch
...         self.year = year
...         print("A Student detail object is created")
... def print_details(self):
...         print('Name: ', self.name)
...         print('Branch: ',self.branch)
...         print('Year: ',self.year)
```

DEFINITION OF CLASS (CONTD..)

In the given example, we have created a class `Student` that contains two methods: `fill_details` and `print_details`. The first method `fill_details` takes four arguments: `self`, `name`, `branch` and `year`. The second method `print_details` takes exactly one argument: `self`.

CREATING OBJECTS

An object is an instance of a class that has some attributes and behaviour. The object behaves according to the class of which it is an object.

Objects can be used to access the attributes of the class. The syntax of creating an object in Python is similar to that for calling a function.

Syntax

```
obj_name = class_name()
```

Example

```
s1 = Student()
```

The above statement creates an object `s1` of the class `Student` which we defined earlier.

Now, we can access the methods (attributes) which are defined in the class `Student`. We can use a method from the class `Student` with the `object name` followed by a `dot`, which is then followed by the `method name` with valid arguments.

CREATING OBJECTS (CONTD..)

Example

```
# creating an object of Student class
>>> s1 = Student()
# creating another object of Student class
>>>s2 = Student()
# using the method fill_details with proper attributes
>>> s1.fill_details('John','CSE','2002')
A Student detail object is created
>>>s2.fill_details('Jack','ECE','2004')
A Student detail object is created
# using the print_detail method with proper attributes

>>>s1.print_details()
Name: John          # Output
Branch: CSE         # Output
Year: 2002          # Output

>>>s2.print_details()
Name: Jack          # Output
Branch: ECE         # Output
Year: 2004          # Output
```

CREATING OBJECTS (CONTD..)

In this example, we create two objects (instances) `s1` and `s2` of class `Student`. Then afterwards, we use the `fill_details` method of class with the object names as prefix and passed the valid arguments. The details of the students are stored in the objects. Now, the second method `print_details` is called with the same convention. The method `print_details` prints the details of all stored students.

OBJECTS ARE MUTABLE

Objects are mutable— this statement tells us that the state of an object can be changed at any point of time by making changes to its attributes.

For example, consider the class `Student` which was defined earlier. We created an object `s1` and filled the details of the student using `fill_details` method. If, at any point of time, it is required to change the value of branch from `ECE` to `CSE`, it can be done in the same object by reassigning it a new value.

Example

```
# Create an instance of class Student
>>>s1 = Student()
# Fill details in that object
>>>s1.fill_details('John','ECE',2004)
A Student detail object is created
# Printing details of the object s1
>>>s1.print_details()
```

Output:

```
Name: John
Branch: ECE
Year: 2004
```

OBJECTS ARE MUTABLE (CONTD..)

Now, what if it is required to change the value of `branch` of object `s1` to `CSE`? Will we create a new object? Definitely not. We can change the value in the same object by simply reassigning the value to it.

```
#Change the value of branch from ECE to CSE
>>>s1.fill_details('John','CSE',2004)
A Student detail object is created
# The branch is changed from ECE to CSE
>>>s1.print_details()
```

Output:

```
Name: John
Branch: CSE
Year: 2004
```

Now, the state of object `s1` has been permanently changed.

OBJECTS AS ARGUMENTS

The instance of a class can be passed as an argument to a function in Python.

Let us say, we have a class Triangle. We make an instance of this class and define the attributes that are sides of this triangle. Then, that object or instance of Triangle can be passed to a function which calculates the perimeter of the triangle.

Example

First of all, we will create a class `Triangle` with no statements.

```
>>>class Triangle:
    pass
```

Now, we create an object `t1` of the class `Triangle` and assign the value of sides `a, b, c` of the triangles here.

```
>>>t1 = Triangle()
>>>t1.a = 10
>>>t1.b = 18
>>>t1.c = 23
```

Now, we define a function `perimeter`, which calculates the perimeter of a triangle. This function takes an object or instance of a class as an argument.

```
>>>def perimeter(obj):
    per = t1.a + t1.b + t1.c
    print("Perimeter of triangle: ", per)
```

OBJECTS AS ARGUMENTS (CONTD..)

Now, we pass the object `t1` to the function `perimeter`, which calculates the perimeter of the triangle that is in this object.

```
>>>perimeter(t1)                    # Passing object as argument
Perimeter of triangle: 51           # Output
```

OBJECTS AS RETURN VALUES

The instances of a class can also be returned by a function, i.e., a function can return instances or objects.

Let us say, we are creating an object of `Triangle` class and a function `size_double` that doubles the size of the triangle. Now, when the object of `Triangle` class is passed to this function, it doubles the size of the triangle in that object and returns the Double sized triangle that is in the form of object.

Example

Create a class `triangle` and define two methods: one is `create_triangle`, which will create the triangle, and the other is `print_sides`, which will print the sides of the triangle.

```
>>>class Triangle:
    defcreate_triangle(self,a,b,c):
            self.a = a
            self.b = b
            self.c = c
            print("The triangle is created")
    defprint_sides(self):
            print('Side a: ', self.a)
            print('Side b: ', self.b)
            print('Side c: ', self.c)
```

Create an object `t1` of the `Triangle` class and create a triangle in it.

```
>>>t1 = Triangle()
>>>t1.create_triangle(10,20,30)
The triangle is created        # Output
```

Define a function `size_double` that will take an object as an argument and return a triangle in the form of an object that is double in size.

OBJECTS AS RETURN VALUES (CONTD..)

Define a function `size_double` that will take an object as an argument and return a triangle in the form of an object that is double in size.

```
>>>defsize_double(obj):
    t2 = Triangle()
    t2.a = t1.a *2
    t2.b = t1.b *2
    t2.c = t1.c *2
    return t2                  # Returning object
>>>t2 = size_double(t1)     # Passing object as argument
>>>t2.print_sides()
```

Output:

```
Side a: 20
Side b: 40
Side c: 60
```

Hence, we got a triangle that is double in size with the triangle that was passed as an argument.

Thank You