# Chapter 7 :
# Objects and Classes

Aviral Janveja

## 1 Object

Throughout this course in Python, we have come across various examples of data like numbers, strings, booleans and so on. All of these instances of data belong to a particular data-type like integer, float or string.

In Python, the term object basically refers to any piece of data. Whether we take instances of integer data, float data or string data, they are all essentially objects. Therefore, the term **object** serves a dual purpose in Python language :

1. It refers to an instance of a particular data-type. For example, x = 5. Here 5 is an object of type integer.
2. Object is also the base of all data-types. For example, the integer data-type itself is built on top of the object data-type.

This base object data-type provides core functionality in Python language, such as the ability to assign values to variables. This is why it is often said that **everything in Python is an object** and every object has a type. The type of an object defines how it is represented internally within the Python language and the ways in which we can interact with it. For example, list objects are always defined using square brackets [ ] and we can manipulate them using various built-in methods such as insert, append, remove and sort.

## 2 Class

The next question is : how are these data-types defined behind the scenes and can we define our own type ?

**Answer** : Yes, we can define our own types through **classes**. Defining a class is like defining a blue print for your own data-type, similar to integers, floats and lists. The class definition looks like this :

```
class class_name:
    # Define attributes here
```

We start with the **class** keyword followed by the **name** of our class. Inside the class, we define its attributes which are the data attributes and the procedural attributes that belong to the class.

### 2.1 Data

Data attributes define the data representation of this particular type. For example, a two dimensional coordinate object will be made up of two numbers, in a bracket, separated by a comma.

Coordinate object = (3,5)

One value for the x-coordinate and one value for the y-coordinate. We can further decide on whether these two numbers will be of type int or type float.

### 2.2 Methods

Procedural attributes, better known as methods are functions, that only work with this particular type. For example, you can define a distance method between two coordinate objects, but that method will have no meaning for list objects.

## 3 Defining a new type : Class Coordinate

Let us continue with the class definition above and implement our own class.

```
class Coordinate:
    # Data and Methods
```

### 3.1 Special Method __init__

First, we define a method called **__init__** within our class. In Python __init__ is a special method also known as the constructor. It is automatically called when a new object of the class is created. The __init__ method allows you to **initialize** the data attributes of an object as soon as that object is created.

```
class Coordinate:

    def __init__(self, x, y):
```

While defining this method, a default parameter named **self** is always used. The self parameter refers to the object of the class itself and is used for assigning values to the data attributes of an object. The two underscores on each side of its name imply that the method is invoked automatically and used internally by Python, without needing to be called explicitly.

Any other parameters beyond self can be added, just like a normal function. In this particular case, we are going to initialize a coordinate object with two values, one for the x-coordinate and one for the y-coordinate.

```
class Coordinate:

    def __init__(self, x, y):
        self.x = x
        self.y = y
```

Inside the __init__ method we have two assignments as shown above. It basically says that the "x data attribute of the coordinate object will be assigned to whatever value of x was passed in, while creating that object". Same for the y data attribute.

## 3.2 Creating Coordinate Objects

We have created a simple class above and now we are ready to start creating objects of type Coordinate as follows :

```
class Coordinate:

    def __init__(self, x, y):
        self.x = x
        self.y = y

# Creating an object of type Coordinate
c = Coordinate(3, 4)
```

The following line of code calls the init method with x = 3 and y = 4 :

```
c = Coordinate(3, 4)
```

Notice, when we are creating an object here, we are only giving it 2 parameters, whereas in the init method, we have 3 parameters. This is okay, because implicitly, Python uses the self parameter to refer to the object **c** by default.

Below, we create another Coordinate object named **origin** whose values for x and y are both zero. So now, we have two Coordinate objects and we can access their data attributes using the dot-notation as shown below :

```
class Coordinate:

    def __init__(self, x, y):
        self.x = x
        self.y = y

c = Coordinate(3, 4)
origin = Coordinate(0, 0)

# Accessing data attributes using dot-notation
print(c.x)  # 3
print(origin.x)  # 0
```

The above line prints the x-Coordinates of both objects, which are 3 and 0 respectively.

## 3.3 Distance Method