Lecture 02: Introduction to Class and Objects

Swakkhar Shatabda

B.Sc. in Data Science
Department of Computer Science and Engineering
United International University

January 26, 2024

Classes and Objects

- You have already been using built-in Python objects.
- Strings are an example of a Python object.

```
s = "I am a string"
print(type(s))
```

```
<class 'str'>
```

```
print(dir(str))
```

- dir is a built-in function that tells the user all of the attributes and methods associated with a class.

Vocabulary

Classes

Classes are a collection of data and the actions that can modify the data. Classes act as the blueprint.

Objects

Objects are constructed according to the blueprint that is the class. When a programmer wants to use a class, they create an object.

Instance

An object is an instance of a particular class. For example, s is an instance of the string class.

Instantiation

Instantiation is the process where an object is created according to blueprint of the class.

First Class in Python!

- Assume you want to collect information about actors.
- Creating a class is a good way to keep this data organized. The class keyword is used to define a class.
- For now, use pass as the body of the class.

```
class Student:
    pass
print(Student)
```

```
<class '__main__.Student'>
```

Naming

The convention for naming classes in Python is to use a capital letter. If a class has a name with multiple words, do not use an _ to separate the words. Instead, all of the words are pushed together, and a capital letter is used for the first letter of each word. This is called camel case.

First Object in Python!

- Classes are just the blueprint.
- To you use a class, you need to instantiate an object.
- Here is an object to represent Swakkhar Shatabda.

```
class Student:
    pass

swakkhar = Student()
print(swakkhar)
print(type(swakkhar))
```

```
<__main__.Student object at 0x1055c7130>
<class '__main__.Student'>
```

Adding Attributes!

- The point of having a class is to collect information and define actions that can modify the data.

```
class Student:
    pass

swakkhar = Student()
swakkhar.firstName = "Swakkhar"
swakkhar.lastName = "Shatabda"
print(f'{swakkhar.firstName} {swakkhar.lastName}')
```

```
Swakkhar Shatabda
```

- Adding each attribute individually would require a lot of code.
- Smarter way - Use Constructor.

Class in Python!

```
class Student:
    pass

swakkhar = Student()
swakkhar.firstName = "Swakkhar"
swakkhar.lastName = "Shatabda"
swakkhar.age = 41
swakkhar.adress = "Dhaka"
swakkhar.weight = 71.2
swakkhar.cgpa = 3.75
print(f'{swakkhar.firstName} {swakkhar.lastName}')
```

The Constructor!

- The constructor is a special method (more on methods in another lesson) for a class. Its job is to define all of attributes associated with the object. These attributes are called instance variables.
- In Python, the constructor is coded as __init__. The double underscores before and after init are required.
- The constructor also must have self as a parameter. Use the same syntax for the constructor as you would a function. Be sure to indent the constructor four spaces since it is a part of the class.
- Attributes will be referred to as self.attribute_name.

dunder

Objects often have methods with double underscores referred to as dunder methods. A dunder method is an implicit method and is called behind the scenes by an explicit operation. You never have to call helen.__init__. When you instantiate the object, Python calls init automatically.

The Constructor

```
class Student:
    def __init__(self):
        self.firstName = "Swakkhar"
        self.lastName = "Shatabda"
        self.age = 41
        self.adress = "Dhaka"
        self.weight = 71.2
        self.cgpa = 3.75

swakkhar = Student()

print(f'{swakkhar.firstName} {swakkhar.lastName}')
```

Parameters and Constructor

- Lets instantiate two students.

```
class Student:
    def __init__(self):
        self.firstName = "Swakkhar"
        self.lastName = "Shatabda"
        self.age = 41
        self.adress = "Dhaka"
        self.weight = 71.2
        self.cgpa = 3.75

swakkhar = Student()
farid = Student()

print(f'{swakkhar.firstName} {swakkhar.lastName}')
print(f'{farid.firstName} {farid.lastName}')
```

Parameterized Constructor

```
class Student:
    def __init__(self,fname,lname,age,address,weight,cgpa):
        self.firstName = fname
        self.lastName = lname
        self.age = age
        self.adress = address
        self.weight = weight
        self.cgpa = cgpa
swakkhar = Student("Swakkhar","Shatabda",41,"Dhaka",71.2,3.75)
farid = Student("Dewan Md","Farid",45,"Dhaka",91.2,4.00)

print(f'{swakkhar.firstName} {swakkhar.lastName}')
print(f'{farid.firstName} {farid.lastName}')
```

```
Swakkhar Shatabda
Dewan Md Farid
```

Default Parameters

- Like functions, classes can use default parameters for __init__. Default parameters must come at the end of the parameter list.

```
class Student:
    def __init__(self,fname,lname,age,address,weight,cgpa=0.0):
        self.firstName = fname
        self.lastName = lname
        self.age = age
        self.adress = address
        self.weight = weight
        self.cgpa = cgpa
swakkhar = Student("Swakkhar","Shatabda",41,"Dhaka",71.2)
farid = Student("Dewan Md","Farid",45,"Dhaka",91.2,4.00)

print(f"{swakkhar.firstName}'s cgpa: {swakkhar.cgpa}")
print(f"{farid.firstName}'s cgpa: {farid.cgpa}")
```

```
Swakkhar's cgpa: 0.0
Dewan Md's cgpa: 4.0
```

Class Attributes

```
class Student:
    university = "UIU"
    def __init__(self,fname,lname,age,address,weight,cgpa=0.0):
        self.firstName = fname
        self.lastName = lname
        self.age = age
        self.adress = address
        self.weight = weight
        self.cgpa = cgpa

swakkhar = Student("Swakkhar","Shatabda",41,"Dhaka",71.2)
farid = Student("Dewan Md","Farid",45,"Dhaka",91.2,4.00)

print(f"{swakkhar.firstName} is a student of {swakkhar.university}")
print(f"{farid.firstName} is a student of {farid.university}")
```

```
Swakkhar is a student of UIU
Dewan Md is a student of UIU
```

Class Attributes

```
class Student:
    university = "UIU"
    def __init__(self,fname,lname,age,address,weight,cgpa=0.0):
        self.firstName = fname
        self.lastName = lname
        self.age = age
        self.adress = address
        self.weight = weight
        self.cgpa = cgpa

swakkhar = Student("Swakkhar","Shatabda",41,"Dhaka",71.2)
farid = Student("Dewan Md","Farid",45,"Dhaka",91.2,4.00)

print(f"{swakkhar.firstName} is a student of {swakkhar.university}")
swakkhar.university = "BUET"
print(f"{farid.firstName} is a student of {farid.university}")
```

Shallow Copy!

```
class A:
    pass

a = A()
a.name = "UIU"

b = a
b.name = "United International University"

print(a.name)
```

```
United International University
```

Deep Copy

```
import copy

class A:
    pass

a = A()
a.name = "UIU"

b = copy.deepcopy(a)
b.name = "United International University"

print(a.name)
```

```
UIU
```

Define Class in Python!

```
class <ClassName>():

    def __init__(self, <optional param1>, ..., <optional paramN>):
        #  any initialization code here

    # Any number of functions that access the data
    # Each has the form:

    def <functionName1>(self, <optional param1>, ..., <optional paramN>):
        #  body of function

    #  ... more functions

    def <functionNameN>(self, <optional param1>, ..., <optional paramN>):
        #  body of function
```

Practice...

- Consider a game where we have multiple players. A player could be modeled by a Player class with instance variables for name, points, health, location, and so on. Each player would have the same capabilities, but the methods could work differently based on the different values in the instance variables.
- Imagine an address book. We could create a Person class with instance variables for name, address, phoneNumber, and birthday. We could create as many objects from the Person class as we want, one for each person we know. The instance variables in each Person object would contain different values. We could then write code to search through all the Person objects and retrieve information about the one or ones we are looking for.