BRAC UNIVERSITY

Inspiring Excellence

Course Title: Programming Language II

Course Code: CSE 111

Semester: Summer 2020

Topic: Object-Oriented Programming (Class and Static method)

Table of Contents

In previous lecture, we have been introduced the concepts of class, object, constructor, instance variable and instance method of Object-Oriented Programming (OOP).

A class consists of variables and methods. Variables define the properties of an instance of a class and methods define the behaviors of an instance of a class.

1. Python Variable:

Variables are of 2 types:

1) Instance Variables: Variables which are assigned values inside class methods are instance variables. Instance variables or non-static variable are owned by instances of the class. This means that for each object or instance of a class, the instance variables are different.

2) Class Variables: All variables which are assigned a value in the class declaration are class variables. Class or Static variables are the variables that belong to the class and not to objects. Class or Static variables are shared amongst objects of the class

The variables that we have dealt with so far was the instance variables. Now we will discuss about Class variables.

1.1 Class Variable or Static Variable:

Class variables are defined within the class construction. Because they are owned by the class itself, class variables are shared by all instances of the class. They therefore will generally have the same value for every instance unless you are using the class variable to initialize a variable.
Class variables can defined outside of all the methods and placed right below the class header and before the constructor method and other methods.

In python, it doesn’t require a static keyword. All variables which are assigned a value in class declaration are class variables. And variables which are assigned values inside methods are instance variables.

In the following example, `department` is a class variable because it is defined outside of all the class methods and inside the class definition. `name` and `id` are instance variable as it is defined inside a method.
This is confirmed using the print statement where the `department` variable is referenced using the class name `Student` while the instance variable is referenced using the different object references.
This example shows a scenario where there are different objects each belonging to the same category but are of different types, so each object of the class have the same category

which we have made the class variable and the instance variable is different for all the objects hence it is an instance variable. Also class variables can be accessed using class name also

```python
# Class for Computer Science Student
class Student:
    department = 'CSE'   # Class Variable
    def __init__(self,name, id):
        self.name = name  # Instance Variable
        self.id = id      # Instance Variable

# Objects of CSStudent class
a = Student('ABC', 17101)
b = Student('DEF', 18202)

print(a.department)  # prints "CSE"
print(b.department)  # prints "CSE"
print(a.name)     # prints "ABC"
print(b.name)     # prints "DEF"
print(a.id)     # prints "17101"
print(b.id)     # prints "18202"

# Class variables can be accessed using class name
also
print(Student.department) # prints "CSE"
```

Output:

```
CSE
CSE
ABC
DEF
17101
18202
CSE
```

2. Python Method:

Python method is like a Python function. It must be called on an object and put inside a class. A method has a name, and may take parameters and have a return statement.

There are three types of methods in Python-

1) **Instance Method :** The methods that we will see in this lecture are instance methods. Instance methods can access the attributes and the other methods through the "self" parameter. Thus, instance methods can be only invoked through an instance of the class.
2) **Class Method:** The class method takes a different parameter "cls" instead of "self" that points to the class instead of the object. Hence, it can't modify the state of an instance but can modify the state of a class.

3) **Static method:** The static method does not take any parameter like “cls” or “self”. In fact, it has nothing to do with the state of the class nor the state of the object.

```
class MyClass:

  def method(self):
    return 'instance method called', self

  @classmethod
  def classmethod(cls):
    return 'class method called', cls

  @staticmethod
  def staticmethod():
    return 'static method called'
```

Previously, we have discussed about instance method. So now let’s discuss about class and static method.

2.1 Class Method:

A class method is a method which is bound to the class and not the object of the class. They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.

Method with a `@classmethod` decorator to flag it as a class method. Class methods don’t need a class instance. They can’t access the instance but they have access (self) to the class itself via “cls”.

This method only has access to this “cls” argument, it can’t modify object instance state. That would require access to “self”. Class methods can still modify class state that applies across all instances of the class.

Here in the following example there two class instance creator, a constructor and a birthYear method.

The constructor takes normal parameters name and age. While, birthYear takes class, name and birthYear, calculates the current age by subtracting it with the current year and returns the class instance.

The birthYear method takes Person class (not Person object) as the first parameter cls and returns the constructor by calling cls(name, date.today().year - birthYear), which is equivalent to Person(name, date.today().year - birthYear)

Before the method, we see @classmethod. This is called a decorator for converting birthYear to a class method as classmethod().

```
from datetime import date

# random Person
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    @classmethod
    def birthYear(cls, name, birthYear):
        return cls(name, date.today().year - birthYear)

    def display(self):
        print(self.name + "'s age is: " + str(self.age))

person = Person('Adam', 19)
person.display()
```

Output:

```
Adam's age is: 19

John's age is: 30
```

2.2 Static Method:

Static methods in Python are extremely similar to python class level methods, the difference being that a static method is bound to a class rather than the objects for that class. This means that a static method can be called without an object for that class `MyClass.staticmethod` was marked with a `@staticmethod` decorator to flag it as a static method. This type of method takes neither a `self` nor a `cls` parameter (but of course it's free to accept an arbitrary number of other parameters).
This method can neither modify object state nor class state. Static methods are restricted in what data they can access - and they're primarily a way to namespace your methods. It can be called without an object for that class, using the class name directly. If we want to do something extra with a class we can use static methods.

```
class Math:
    @staticmethod
    def Multiply(one, two):
        return one * two
math = Math()
if(12*72 == math.Multiply(12, 72)):
    print("Equal")
else:
    print("Not Equal")
```

Output:

```
Equal
```

Here above example “Multiply” is a static method because it doesn't need to access any properties of Math itself and only requires the parameters.

Now, Let’s look at an example to understand the difference between both class and static method. Let us say we want to create a class Person. Here, class method is used to create a person object from birth year and static method is to check if a person is adult or not.

```
from datetime import date

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # a class method to create a Person object by birth
 year.
    @classmethod
    def fromBirthYear(cls, name, year):
        return cls(name, date.today().year - year)

    # a static method to check if a Person is adult or
not.
    @staticmethod
    def isAdult(age):
        return age > 18

person1 = Person('mayank', 21)
person2 = Person.fromBirthYear('mayank', 1996)

print(person1.age)
print(person2.age)

# print the result
print(Person.isAdult(22))
```

Output:

```
21

24

True
```