Inspiring Excellence

Course Title: Programming Language II

Course Code: CSE 111

Semester: Summer 2020

Topic: Object-Oriented Programming (OOP) – METHODS

Table of Contents

In the previous lecture, we have been introduced to the basic concepts of Object-Oriented Programming (OOP): Classes and Objects.

Before we jump into Methods. Let us recall briefly about classes, objects and constructors.

A class is a blueprint or a design which is used to create objects. For example, to bake a cake one needs a recipe. So, in this case, the recipe which is not edible is a class and the cake that has been baked and is edible is an object or instance. Using this one recipe, more than one cake can be baked. That is, many objects can be created from one class.

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.

Variables are of 2 types:

1. Instance Variables
2. Class Variables

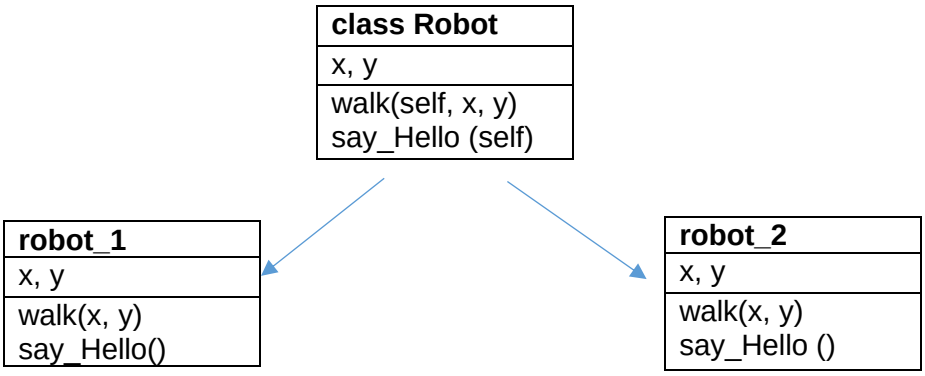
The variables that we have dealt with so far were the instance variables. Class variables are different and will be taught in the next lecture.

Now let us see what are methods!

METHODS

A method is basically like a function. It is a special kind of function which is defined inside the body of a class. It is used to define the behaviors of an object. It is used to manipulate the variables in a class.

Suppose you are making a toy robot. You want the robot to be able to walk and say hello. For this, you can define these 2 behaviors using methods.



In your Robot class, there are 3 data fields and two methods. Invoking these 2 methods, walk() and say_Hello(), will make some actions to be performed. These methods can manipulate the attributes, x and y, of the current instance.

Observe, that methods can be parameterized like walk() that takes 2 parameters and also non-parameterized like say_Hello() that takes no parameters.

The program for such a Robot class would be as shown below.

Code

```
class Robot:
    #Initializing the instance variables
    def __init__(self):
        self.x = 0
        self.y = 0

    # Method that prints hello and the robot's position
    def say_Hello(self):
        print(f"Hello! My position is ({self.x},{self.y})")

    # Method to change position of the robot
    def walk(self,x,y):
        self.x+=x
        self.y+=y

# Creating an instance of Robot class
robot_1 = Robot()

print("First robot: ")
#Calling the methods on robot_1
robot_1.say_Hello()
robot_1.walk(3,2)
robot_1.say_Hello()
robot_1.walk(3,2)
robot_1.say_Hello()

# Creating another instance of Robot class
robot_2 = Robot()

print("\nSecond robot: ")
#Calling the methods on robot_2
robot_2.walk(5,10)
robot_2.walk(4,6)
robot_2.say_Hello()
```

Output

```
First robot:
Hello! My position is (0,0)
Hello! My position is (3,2)
Hello! My position is (6,4)

Second robot:
Hello! My position is (9,16)
```

TYPES OF METHODS

There are 3 types of methods:

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. We will discuss this in depth later.

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. This method will also be discussed in detail later.

INSTANCE METHOD

Instance methods can only be called through instances of a class, i.e. the object name followed by a dot and the method name.

Let us see again in code how methods are called and how they are defined inside the body of a class.

Code	Output

```python
import math

class Circle:
    def __init__(self, r):
        self.radius = r

    def area(self):
        area = math.pi * (self.radius)**2
        print(area)

    def perimeter(self):
        perimeter = 2 * math.pi * self.radius
        print(perimeter)

circle_1 = Circle(2)
circle_1.area()
circle_1.perimeter()
```

```
12.566370614359172
12.566370614359172
```

In the above program, we created a class called **Circle**. This **Circle** class has an attribute: **radius**, and 2 methods: **area** and **perimeter**. These methods are called instance methods because they are called on an instance object i.e. **circle_1**. The method area calculates the area and prints it. Similarly, the method perimeter calculates the perimeter of the instance and prints the result.

Notice, that while calling the methods, area and perimeter, you need to write the name of the instance created and then the method name with a dot in between.

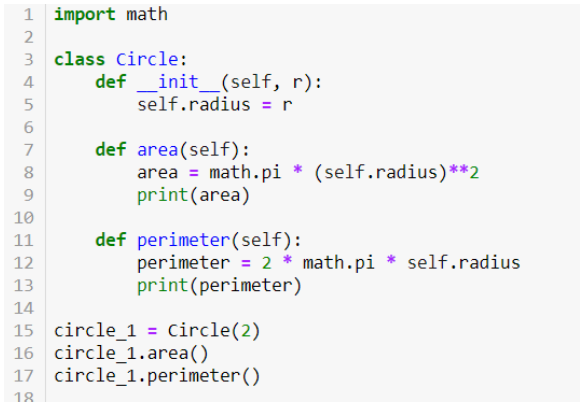
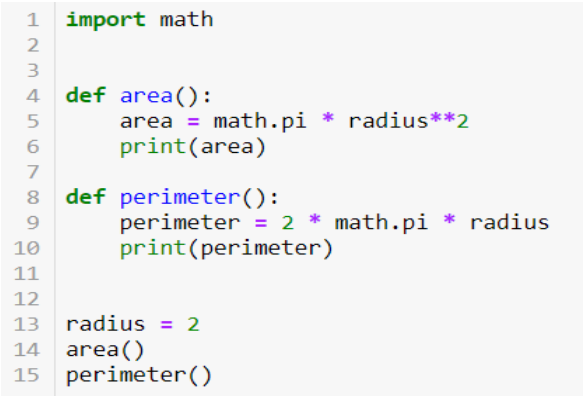
E.g. object_name.method_name
circle_1.area()
circle_1.perimeter()

While writing a method in a class, the first parameter of the method should be “self” that points to the location of the object. Just as done in the constructor. Remember, the constructor is also a method but a special one since it is automatically called whenever an instance of a class is created. However, when a method is called for an object, it is called without the “self” parameter. This is because variable of the object points to the object’s reference.

DIFFERENCE BETWEEN FUNCTIONS AND METHODS

Notice, we are saying a method is *like* a function or a *special* function. A method is different from a function in 2 main ways.

See the codes in the last row to understand this better. The outputs for both the codes are same but one is done using methods and the other using functions.

Methods	Functions
A method is defined within a class and hence belongs to a class.	A function is defined without a class.
A method’s first parameter must be the reference/address/location of the current instance of the class. This parameter is saved in a variable called “self”.	A function does not need any parameters like self as required by methods.
	

```python
import math

class Circle:
    def __init__(self, r):
        self.radius = r

    def area(self):
        area = math.pi * (self.radius)**2
        print(area)

    def perimeter(self):
        perimeter = 2 * math.pi * self.radius
        print(perimeter)

circle_1 = Circle(2)
circle_1.area()
circle_1.perimeter()
```

```python
import math


def area():
    area = math.pi * radius**2
    print(area)

def perimeter():
    perimeter = 2 * math.pi * radius
    print(perimeter)


radius = 2
area()
perimeter()
```

A SIMPLE EXAMPLE OF OOP PROGRAM USING VARIABLES AND METHODS

Let's design a Flight class that has 4 data fields: Name, Destination, Airlines, and Date.

Code

```
class Flight:
    # Initializing the data fields
    def __init__(self, name, destination, airlines, date):
        self.name = name
        self.destination = destination
        self.airlines = airlines
        self.date = date

# Object creation
obj1 = Flight("Sara", "London", "Biman","04/08/2020")
obj2 = Flight("Mark", "Germany", "Emirates", "24/01/2020")

# printing all the attribute values
print("Name:", obj1.name)
print("Destination:",obj1.destination)
print("Airlines:", obj1.airlines)
print("Date:",obj1.date)

print("-------------------------------------")

print("Name:", obj2.name)
print("Destination:",obj2.destination)
print("Airlines:", obj2.airlines)
print("Date:",obj2.date)
```

Output

```
Name: Sara
Destination: London
Airlines: Biman
Date: 04/08/2020
-------------------------
--------------
Name: Mark
Destination: Germany
Airlines: Emirates
Date: 24/01/2020
```

Here, after writing the Flight class, we created 2 instances, obj1 and obj2, of the Flight class. Then we simply printed the values of all the variables for both the objects by printing each attribute separately.

However, now let's re-write this code using a method to print all the attributes of an object.

Code

```
class Flight:
    # Initializing the data fields
    def __init__(self, name, destination, airlines, date):
        self.name = name
        self.destination = destination
        self.airlines = airlines
        self.date = date

    # Method to view all the details
    def get_all_details(self):
        print(f"Name: {self.name}\n"
              f"Destination: {self.destination}\n"
              f"Airlines: {self.airlines}\n"
              f"Date: {self.date}")

# Object creation
obj1 = Flight("Sara", "London", "Biman", "04/08/2020")
obj2 = Flight("Mark", "Germany", "Emirates", "24/01/2020")

# printing all the attribute values
obj1.get_all_details()
print("-----------------------------------")
obj2.get_all_details()
```

Output

```
Name: Sara
Destination: London
Airlines: Biman
Date: 04/08/2020
-----------------------------------
Name: Mark
Destination: Germany
Airlines: Emirates
Date: 24/01/2020
```

As you can see now, we can just simply call the **get_all_details** method for printing all the attributes of the specific instance. The benefit of printing using this method is that we can simply write the codes for printing each attribute once inside the class in a method and then just write a one line code after the creation of an object to print all of its details. Without this method we would have had to write the four lines of code to print each time we wanted to see all the attributes of an instance.