CSE 1321L: Programming and Problem Solving I Lab

Lab 7

Methods

What students will learn:

- Writing their own methods
- Writing their own methods in different files and importing them
- Using built-in methods
- Calling methods

Content

- Overview
- Lab7A: Rectangle Area and Perimeter Calculator
- Lab7B: Max and Min of two numbers

Overview

Sometimes called "functions" or "procedures", methods are blocks of code that have been given a name for the sake of convenience. This is done usually for clarity or to reduce the overall amount of code being written, think of **reusability**, or **reusable code**.

All methods you will write in this class have a "template" that looks like:

```
def <name> (<parameters>):
      <body>
```

In the template specified above, **`<name>`** is the name or identifier of the method (which can be any name except reserved keywords), the **`<parameters>`** list in parentheses (which are the method's inputs), and the **`<body>`** of the method, the code that will be executed once the method is called.

After a method has been defined, we can call it from anywhere else in the code. Once the computer reaches a line of code containing a method, it will halt the execution of the code that is currently running and execute the code inside the method before resuming.

Methods reduce the amount of code we write. Instead of writing the same block of code over and over across our entire program, we can simply place that block apart, give it a name, and then call it whenever we need it.

To call a method, simply write its name or identifier followed with a set of parentheses. If the method includes parameters, place the values inside the parentheses:

```
<name>(<arguments>)
```

You may have noticed that we have used two different words to describe a method's input: parameters and arguments. While some people may use them interchangeably, **they refer to two different things**.
Parameters are what inputs the method can receive.
Arguments are the actual input received.

Let's use a vending machine as an example:
Some vending machines accept bills, if the vending machine is our method, then the slot for inserting the bills would be its **parameter**, while the actual bills we insert are the **arguments**.

```
# `money` is the method's only parameter
def vending_machine(money):
      print(f"You've entered a {money} bill.")

# Call the vending_machine() method using
# "One Dollar" as its argument
vending_machine("One Dollar")
```

Methods are not limited to only having one parameter, you can add as many as you want (even no parameters at all), as long you separate each single parameter with a comma.

Methods can also return a value. This is done through the "return" keyword. Whenever a method reaches a return statement followed by a value, that value is returned as an output.

```
def get_fullname(first_name, last_name):
      return f"{first_name} {last_name}"

get_fullname("Alice", "Smith")
```

Running the code above will not produce any visible output. While we have called the method, supplied it with arguments, and the method has a return statement, we are not doing anything with the value being returned. If we wish to display the value that has been returned, we must send it to the console or store it in a variable:

```
def get_fullname(first_name, last_name):
      return first_name + " " + last_name

full_name = get_fullname("Alice", "Smith")
print(full_name)

# Or we can also print the method call directly
# When a method returns a value, the method call
# becomes an expression of that value.
print(get_fullname("Alice", "Smith"))
```

It is important to make a distinction that **printing something** to the console **does not** constitute **returning a value**. When a method returns a value, that value is being returned to a different part of the code, not to the user.

```
# prints the full name but returns None
def print_fullname(first_name, last_name):
      print(first_name + “ “ + last_name)

# returns a string but prints nothing
def get_fullname(first_name, last_name):
      return first_name + “ “ + last_name
```

Methods can also feature the return keyword without returning a value. This is because **the return keyword primary duty is signaling that the method is done executing**. As soon as the code being executed reaches a return, any code below that return will not be executed:

```
def only_count_to_five(number):
      for x in range(number):
            if x >= 5:
                  return
            print(x)

only_count_to_five(20)
```

If the method above is supplied with a positive integer that is **less than 5**, it will print all integers from 0 up to that number. However, if it is supplied with a number **greater than 5**, it will always stop after having printed **4**.

All methods which terminate naturally (they execute all the code inside of it without hitting a return statement) or which hit a return keyword without an accompanying value always return **None**.

Methods in Python can also accept both optional and required arguments. Required parameters must receive arguments when the method is called, or the program will crash. Optional parameters have default values associated with them; if no arguments are passed, the default values will be used.

```
def greet_user(age, name=”Person”):
      print(“Hello “ + name + “. You are “ + str(age) + “ years old.”)

greet_user(30)
greet_user(40, “Alice”)
# this would crash, as the method requires at least one argument
# greet_user()
```

The code above illustrates this. The first time the method is being called, only the required argument `age` is being passed, so the method is using the default value for `name`, `“Person”`. The second time the method is called, two arguments are being passed: the first one being used as a value for `age`, and the second one for `name`.

Like before, your methods can have any number of required and optional parameters **as necessary**. Please note that in the parameter list, **required parameters** must be listed **before optional parameters**.

Finally, you may have noticed that Python already comes with some methods for you to use out of the box such as `print()`, `input()`, and `len()`. However, it is also possible for you to use methods from other files, either written by your or by other people, **by importing** them into your current file.

```
from file_name import method_name
```

Notice that, for the purpose of this lab, **this will work if the file you are trying to import is in the same folder as the file you are importing to**.

```
# This is inside FileOne
def greet_user(name):
      print(“Hello “ + name)
```

```
# This is inside FileTwo
from FileOne import greet_user

username = input(“Enter your name: “)
greet_user(username)
```

As with previous weeks, all labs should have the appropriate file names:

- Lab7A.py
- Lab7B.py

You will also need to submit an extra file:

- MyMath.py

More on it in Lab7B.

Lastly, make sure you review the sample output and make sure the output of your program follows the exact same format including the input statements, print statement, etc. As always, user input is shown in **red** and **bold**.

Lab7A: Rectangle Area and Perimeter Calculator

This program will calculate the area and perimeter of a rectangle by asking the user for the width and height value of the rectangle. The program should validate the rectangle input by checking if the sum of the width and height is greater than 30.

Requirements:

- For this program, you **must implement and use** the specified methods, or it will not be graded.
- Design and implement a program that implements the following methods:
  - Method `isValid()` which takes in two values: width and height, and returns True if the sum of width and height is greater than 30, otherwise, return False.
    `isValid(width, height):`
  - Method `area()` which takes in two values: width and height, and calculates and returns the area of the rectangle.
    `area(width, height):`
  - Method `perimeter()` which takes in two values: width and height, and calculates and returns the perimeter of the rectangle.
    `perimeter(width, height):`
- The user should be able to input fractional values for width and height.
- Before calculating and outputting the area and perimeter, the program should check if the width and height values form a valid rectangle.
- Print `"This is an invalid rectangle."` if the width and height do not form a valid rectangle. Otherwise output the area and perimeter.
- Allow the user to re-run the program.
- Before termination, output `"Program Ends"`

Hints:

- Remember that we specifically used the word **return** for the specification for each function.
- Remember to use the `isValid()` function to validate the width and height of the rectangle.
- We only calculate the area and perimeter if the rectangle is valid.

Sample Output #1:

```
Enter width: 4.0
Enter height: 5.0
This is an invalid rectangle.

Do you want to enter another width and height (Y/N)?: Y

Enter width: 20.0
Enter height: 15.0
This is a valid rectangle.
The area is: 300.0
The perimeter is: 70.0

Do you want to enter another width and height (Y/N)?: N

Program Ends
```

Lab7B: Max and Min of two numbers

This simple program will ask the user for two numbers, then the program should output which number is the lowest of the two, which number is the largest of the two, and the average of both numbers.

Requirements:

- For this program, you **must implement and use** the specified methods, or it will not be graded.
- Implement two python files, one called **MyMath** which will contain the methods and **Lab7B** which will have the main logic and implementation of the program.
- Inside of **MyMath.py**, write the following methods:
  - `my_max` which takes in two number values and **returns** the largest between the two inputs.
  - `my_min` which takes in two number values and **returns** the smallest between the two inputs.
  - `my_avg` which takes in two number values and **returns** the calculated mean average of the two inputs.
- In **Lab7B**:
  - The program your prompt and read two number values.
  - These values should be taken in as an integer.
  - Import and call the methods implemented in **MyMath.py**.
  - Call the appropriate methods and output the corresponding values.

Hint:

- Remember that we specifically used the word **return** for the specification for each function.

Sample Output #1:

```
Enter number 1: 4
Enter number 2: 9
Min is 4
Max is 9
Average is 6.5
```

Sample Output #2:

```
Enter number 1: 45
Enter number 2: 11
Min is 11
Max is 45
Average is 28.0
```

Submission Instructions:

- Programs must follow the output format provided. This includes each blank line, colons (:), and other symbols.
- Programs must be working correctly.
- Programs must be written in Python.
- Programs must be submitted with the correct **.py** format.

- Programs must be saved in files with the correct file name:
    - Lab7A.py
    - Lab7B.py
    - MyMath.py
- Programs (source code files) must be uploaded to Gradescope by the due date.