CSE 1321L: Programming and Problem Solving I Lab

Lab 11

Introduction to Object-Oriented Programming

What students will learn:

- Creating a class
- The purpose of constructors
- Creating an object
- Understand the difference between a class and an object
- Review of methods

Content

Overview

What you have seen up to this point is called procedural programming. What this means is that we structured programs based on the concepts of functions/methods (which are also called procedures).

For this lab, you are going to start programming using Object-Oriented Programming (OOP). OOP is a completely different way of thinking about programming and may seem unusual at first. You should be well prepared for OOP, since you have already been exposed to the concepts of variables and methods. A class just puts those two things together, which is a technique called **Encapsulation**. We are going to focus on the process of building simple classes.

The main idea behind OOP starts with a class – which is a new data type, just like an `int`, `string`, or a `float`. A class can be just about anything, but usually represents a thing in the real

world, like a Dog, a Bank Account, or a Beverage. Then (very much like working with primitive data types), you have to declare a variable of that type. That variable is an **object** – and hence the term Object-Oriented Programming. The interesting part is that classes have things that they can do in the form of methods, which is why methods sometimes are called behaviors.

Designing and implementing a class usually occurs using the following steps:

1. Declare the class (e.g. "class Dog")
2. Add attributes (variables) in the class (e.g. weight, name, hair color)
3. Add a constructor – a special method used to initialize the attributes in step 2
4. Add methods – these are behaviors of the class, or things it can do

You can then create variables of the class type the same way you create other variables (e.g. `my_num = 0). However, in OOP, you will be calling the constructor of the class, which "brings the object to life":

```
d1 = Dog() # The dog is alive, Dog() is the dog constructor
d1.bark()  # Call the bark method in the dog object
```

In a sense, a class operates as a blueprint to how objects created from it should be described and how they behave. Once you have written a few classes, you will see the pattern.

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

- Lab10A.py
- Lab10B.py

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**.

Lab11A: Having a seat

When learning about objects, you may hear the example “a chair is an object”. For this lab, we are going to implement this example by creating a class called `“Chair”`, which will be used to create `“Chair”` objects.

Chair class specifications:

- Class must be named “Chair”
- Must contain 3 attributes:
    - numOfLeg: Integer value which specifies how many legs the chair has.
    - rolling: Boolean value which specifies if the chair can roll (True) or not (False).
    - material: String value which specifies the material the chair is made of.

Using these attributes you can describe most chairs. For example, you might have a wooden chair with 4 legs that does not roll, or you may have a rolling chair made of leather with 5 legs.

You are going to create a Chair class type object by calling the constructor. Whenever an object is created, the constructor is the first piece of code which runs, and it is used to initialize the fields of the object of a “valid” state. In Python, the constructor is called `__init__()` and looks like any other method, which the exception that it **always** hast at least one parameter called `self`, this `self` parameter is used to refer to the object itself.

Main Program Specifications:

- The main program should ask the user to input the initial state of the chair.
- After taking the input, use those values to create a Chair object.
- Print the attributes of the object you just created
- Change the values of the same object to:
    - Number of legs to 4
    - Rolling to False
    - Material to Wood
- Print the attributes of the object

You can use the syntax below to create your constructor:

```
class Chair:
      def __init__(self, numOfLegs = 4, rolling = False, material = “Wood”):
            self.numOfLegs = numOfLegs
            self.rolling = rolling
            self.material = material
```

Requirements:

- Follow the Chair class specifications.
- You must implement the appropriate construction that takes three parameters for each of the Chair’s attributes.
- In the constructor, specify default values for each attribute:
  - numOfLegs = 4
  - rolling = False
  - material = “Wood”
- Your solution should only contain three attributes and the constructor function.
- The main program should only have one single chair object, you can change the attribute values either by re-initializing it or by using the dot operator.
- The `“rolling”` Chair attribute should always be a Boolean type.

Tip:

- When dealing with “rolling” or “not rolling” you want to use an **IF** statement with different print statement whether the `rolling` chair attribute is **True** or **False**.

Sample Output #1:

```
You are about to create a chair.
How many legs does your chair have: 8
Is your chair rolling (true/false): true
What is your chair made of: plastic
Your chair has 8 legs, is rolling, and is made of plastic.
This program is going to change that.
```

Your chair has 4 legs, is not rolling, and is made of wood.

Sample Output #2:

You are about to create a chair.

How many legs does your chair have: **2**

Is your chair rolling (true/false): **false**

What is your chair made of: **metal**

Your chair has 2 legs, is not rolling, and is made of metal.

This program is going to change that.

Your chair has 4 legs, is not rolling, and is made of wood.

Lab11B: My dog can do tricks

For this lab we are going to create a Dog class.

Dog objects have a few attributes, but this time unlike chair objects they can do some cool things too. Each action is represented by a method, therefore, any action our Dog can do, or we do to the Dog, we are going to create a method.

For example, if you want the dog object to bark, you can create a method to do that in the Dog Class and call that method outside of the class (once you have created an object)

Dog Class Specification:

Variables

- **age**: Integer value that represents the age of the dog.
- **weight**: Float value that represents the weight of the dog in lbs.
- **name**: String value that represents the name of the dog.
- **furColor**: String value that represents the fur color of the dog.
- **breed**: String value that represents the breed type of dog.

Behaviors

- **bark**: This method should just print "Woof! Woof!"
- **rename**: This method **takes in** the new name value and **changes** the Dog's `name` attribute.
- **eat**: This method **takes in** the number of weights of the food consumed and **adds** it to the Dog's `weight` attribute.

Keep in mind that methods inside of a class will always take the `self` parameter. This parameter is always automatically passed whenever you call a method from an object. However, some methods must take in more than just the `self` parameter; in this case, these extra parameters must be passed before the method can be called.

Main Program Specification:

1. Prompt and take the user's input for each of the attributes that are going to be used to construct a Dog object.
2. Create a Dog object using the constructor.
3. Print the Dog object's attributes as shown in the sample output.
4. Call the Dog object's `eat()` function.
5. Ask and take input how much the dog should eat and call the `eat()` function
6. Ask and take input the new name of the dog and call the `rename()` function.
7. Print the Dog object's attributes as shown in the sample output.

Requirements:

- The Dog class should be named "Dog"
- The Dog class should contain four attributes, three methods, and a constructor function.
- The `eat()` and `rename()` function **MUST** take in the new value to be changed or added to the attribute, do not take input inside these functions.
- The main program should only contain a single Dog object.
- Print the Dog object information as shown in the sample output.

Sample Output #1:

```
You are about to create a dog.
How old is the dog: 5
How much does the dog weigh: 30.5
What is the dog's name: Patches
What color is the dog: chocolate
What breed is the dog: lab
Patches is a 5 year old chocolate lab that weighs 30.5 lbs.
Woof! Woof!
Patches is hungry, how much should he eat: 5000.3
Patches isn't a very good name. What should they be renamed to: Sparky
Sparky is a 5 year old chocolate lab that weighs 5030.8 lbs.
```

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:
    - Lab11A.py
    - Lab11B.py
- Programs (source code files) must be uploaded to Gradescope by the due date.