CSE 1321L: Programming and Problem Solving I Lab

Lab 12

Introduction to Object-Oriented Programming Part I

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

Overview: What you’ve seen up to this point is called procedural programming. What this means is that we structured programs based on the concept of functions/methods (which are also called procedures). For this lab, you’re going to start programming using Object-Oriented Programming. OOP is a completely different way of thinking about programming and may seem unusual at first. That being said, you should be well prepared for OOP, since you’ve 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’re going to focus on the process of building simple classes. Next week’s lab will be a bit more complex, so make sure you understand what’s going on.

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 are sometimes 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’ve written a few classes, you’ll see the pattern. So, let’s get started.

As always, your filenames should follow the conventions we’ve used all semester (i.e. Lab12A.py and Lab12B.py).

Lab12A: Having a seat

When learning about objects, you may hear the example “a chair is an object”. Well, for this lab, we are going to implement this example. Create a class called “Chair”, which can create “Chair” objects.

Chair class:

class Chair:

# attributes of a chair

numOfLegs = 4 # how many legs are on the chair

rolling = False # does it roll or not

material = “plastic” # what is the chair 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. The code above will allow you to create Chair objects, each with their own copies of the attributes available to a Chair, as below:

c1 = Chair() # creates a Chair

print(c1.numOfLegs) # prints the numOfLegs attribute of c1 (4)

c1.rolling = True # c1’s rolling attribute is now True

Another way for you to create a Chair class is by initializing the fields inside a 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, with the exception that it always has at least one parameter, called “self”. Its syntax can be found below:

class Chair:

 def \_\_init\_\_(self, numOfLegs = 4, rolling = False, material = “plastic”):

 self.numOfLegs = numOfLegs

 self.rolling = rolling

 self.material = material

For the purposes of this lab, either of the two ways above to create a Chair class will work. However, they can lead to different behavior, which we will see in the next lab.

This class isn’t going to have any methods, so unless you are adding the constructor, your class should only have the three attributes above.

Outside of the class body, create a Chair object. Then, take user input for the attributes for the Chair object and assign them to the object. Once you have done that, print out the information about your chair, as per the sample output below. Finally, inform the user you will be changing the information, change the attributes of the Chair object to have 4 legs, to not be rolling, and to be made of wood, and print the Chair’s information one last time.

Tip: When dealing with rolling/not rolling you may want to use an IF statement with different print statements.

See Sample Output in the next page. The user input is indicated in **bold**.

Sample output:

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.

Lab12B: 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 also do some cool things too. Each action is represented by a method. Therefore, for any action our Dog can do or we do to the Dog, we are going to create a method. For example, if you want my Dog 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:

* Variables (Attributes):
	+ age # current age of the dog, should be an int
	+ weight # weight in lbs, should be a float
	+ name # what is the name of the dog, a string
	+ furColor # color of the dog’s fur/hair, a string
	+ breed # what breed is the dog, a string

* Behaviors (Methods):
	+ bark # prints “Woof! Woof!”
	+ rename # take a string and change the name of the dog
	+ eat # take a float and add that number to weight

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.

Outside the Dog class, create a new Dog object and prompt the user to input the attributes describing this Dog. Once done, print out all the details about the Dog, as per the sample output.

Next, use the methods you created in the Dog class to have it bark, change the name (using the rename method, not the dot operator), and feed it.

Finally print out all the details about the Dog, the object should have changed because of your calls to the various methods.

Sample output (user input in **bold**):

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.