CS 111 – Introduction to Computer Science – Fall 2017

Lab Assignment #11

Classes * *(30pts)*

Due Date: at 11:59pm on Saturday, May 12.

The purpose of this lab is to introduce the use and implementation of user defined classes. During the lab, you will be working with user-defined classes to answer related questions, add methods, and create your own class.

Before getting started with the lab, copy the entire `lab11` folder from the course folder (`H:\Compsci\givens\cs111`) to your `U:\cs111` folder.

Note

Throughout the lab assignment, there are questions related to the lab that you must answer. To avoid using paper, you are to answer each question within a text file named `classes.txt`, which can be opened in *Wing*. Note that the file `classes.txt` is a text file and not a *Word* document. Thus, the lines of text will not automatically wrap to fit on a sheet of paper when printed. You must explicitly press the ENTER key to start a new line if the line of text goes past the vertical red line.

Introduction

In the previous laboratory exercises, you have created simple programs using the various language constructs such as loops, selection statements, and functions. You have also used Python's built-in classes for working with strings, files, and lists and those from the graphics module included with the textbook. But programs can also include *user-defined classes*, which are classes written by the programmer that are then used to create corresponding objects to solve a given problem. Below you will find information about classes that can help you answer the questions in the first part. You should also refer to the lecture slides to help you answer these questions.

Object-Oriented Programming: In object-oriented terms, an *object* is an entity corresponding to an underlying objective. The object stores data related to the given objective. For example, if we wanted to represent the time of day, the object might store values for the hours, minutes, and seconds or simply the number of seconds that have elapsed.

Classes: A *class* represents a collection of related objects and is the blueprint which defines the construction of an object. A class definition contains *data fields* (or attributes) and *methods*. The

*Based on the labs of Dr. Rance Necaise

data fields, or instance variables, are variable definitions for the data stored in each object while the methods are the operations that can be performed on an object to manipulate it's data. Methods contain statements that are executed when the method is called or invoked. The data fields and methods are known collectively as members of the class or simply *class members*.

Constructors: Objects are *instantiated* or created from classes. When an object is created, a special method known as the *constructor* is called to initialize the data fields of the object. In Python, the constructor of a class has the special name `__init__`. The constructor should never be called explicitly by your code; it is automatically called when an object is created.

Encapsulation: In object-oriented programming, the actual details as to how the methods of a class are implemented are typically hidden from the user of the code or the *client* – any segment of code outside the class definition which creates and/or manipulates an object. In addition, classes should be designed such that access and manipulation of the data stored in an object is managed through the methods of the class. In other words, the data fields should not be accessible directly by the client.

Mutability: Objects can either be *mutable* or *immutable*. A mutable object is one in which one or more of the data fields (contents of the object) can be changed by a method after the object has been created and initialized. An immutable object, on the other hand, is one in which the data fields can not be changed. In Python, for example, strings are immutable, but lists are mutable.

Methods: The methods contained in a class are commonly grouped into categories. In addition to the special constructor method, there are *accessor methods* (getters), *mutator methods* (setters) and *helper methods*. Accessor methods provide access to the object, but do not modify its contents while mutator methods (not including the constructor) modify the object in some way. Helper methods are intended for internal use by other class methods and not the client and should be private to the class.

Self: All methods have a special parameter named `self`, which must be the first parameter. Methods are applied to a specific object, but since we can create multiple objects from a single class, there must be a way to indicate the specific object on which the method was applied. When a method is called Python automatically passes a reference to the object as the first argument. Without the `self` reference, there would be no way to distinguish between a variable local to a method and the instance variables (data fields) in the object.

Python Classes

User-defined classes are very useful in Python programs, but the syntax for creating a class can sometimes become overwhelming. Some of these details involve the actual syntax while others are simply recommended practices. In this section, we are going to explore some of the details involved in creating Python classes.

Let's begin by examining the contents of a class definition. Open the `die.py` file in *Wing*. Answer the following questions in `classes.txt`.

Question 11.1 *In the `Die` class definition, how can you identify the class constructor?*

Question 11.2 *Assuming the class has been imported into your program, give the statement to create a* `Die` *object that is assigned to the variable* `myDie`*.*

Question 11.3 *Do the objects created from the class contain one or more instance variable(s)? If so, give the name of the instance variable(s).*

Question 11.4 *Are* `Die` *objects mutable or immutable. If they are mutable, identify a method that modifies an object.*

Question 11.5 *Identify one accessor method and one mutator method in the* `Die` *class.*

Question 11.6 *Identify one local variable (other than* `self`*) used in the* `Die` *class.*

Question 11.7 *Suppose you wanted to add another method called* `reset` *that sets the die back to the value 1. What would that method look like? [Do not actually add this method to the class.]*

Using Objects

In order to use a class, it must be imported into the module that will be using it. For example, the following code segment imports the `Die` class from the `die.py` module in which it's defined and uses the `Die` object created from an earlier question.

```
from die import Die

def main():
    #missing code to create myDie

    value1 = myDie.roll()
    value2 = myDie.roll()
    print("The rolls were", value1, "and", value2)

main()
```

 You are to write a driver module (`snakeEyes.py`) that uses the `Die` class to repeatedly roll two dice until snake eyes are rolled. The value of each roll should be printed. Your program should be written to the following specifications:

- Include an appropriate file prolog and appropriate comments throughout the program. Be sure to use meaningful variable names.
- After each roll, print a message indicating the result of the roll. Print either `Snake Eyes!!` or the value of the two dice (`Value = 8`).

After completing the program, be sure to save the file.

■

Temperature Class

In the next part of the lab, you are going to design and implement a class that stores a temperature value.

 Define and implement a `Temperature` class within a module named `temp.py`. Instances of the class should store a temperature in degrees Celsius, where the temperature can be fractional. The class should provide the following methods:

- The constructor, which initializes the instance with a value of 0 degrees Celsius. The constructor should not take any parameters (other than self), but should still have one instance variable representing the temperature in Celsius.
- `getFahrenheit` - returns the temperature in degrees Fahrenheit.
- `getCelsius` - returns the temperature in degrees Celsius.
- `setFahrenheit` - changes the temperature to a new value which is given in degrees Fahrenheit.
- `setCelsius` - changes the temperature to a new value which is given in degrees Celsius.
- Convert the temperature value to the string representation `#.## C` using the special method `__repr__`. Note that this special method is supposed to return a string version of the temperature. It is automatically called when a `Temperature` object is printed or converted to a string using `str()`.

Use `tempTester.py` to test your class implementation. Be sure to fully document your code, including a file prolog and method comments. The method comments should describe the purpose of the given method and be included immediately before the method header . After implementing and fully testing your class, be sure to save the file.
■

Finishing Up

When you are finished with the lab, you need to show me that your code runs and correctly computes the solution for each part of the lab. Also, you need to submit the source files for grading. To submit the files, find the lab assignment on Canvas and upload the three files:

- `classes.txt`
- `snakeEyes.py`
- `temp.py`

Remember, all of the files must be named exactly as indicated above, with the same case and with no spaces or special characters.