Lab 0 – Warm up, Review, Classes, Exceptions!

Lab 0 is intended to make sure you have a working development environment, can properly define Python classes the way we will be using them in this class, can think through a logical problem, and can successfully commit and push code versions to a GitHub repository.

Part 1

1. Checkout the starter code from GitHub by using the link on PolyLearn.
2. Review the code in `location.py`. Note that there is a class definition for a `Location` class, and an associated `__init__` method. In addition, there is code to create `Location` objects and print information associated with those objects.

```
class Location:
    def __init__(self, name, lat, long):
        self.name = name    # string for name of location
        self.lat = lat      # latitude in degrees (-90 to 90)
        self.long = long    # longitude in degrees (-180 to 180)
```

3. Without modifying the code, run location.py in whatever environment you wish (again, reference the Getting Started document if you need help in doing this)
4. Note the information that is printed out for each Location object – you should see something like this:
   ```
   Location 1: <__main__.Location object at 0x000001F6A2E0C7B8>
   ```
5. Since we haven't provided any specific method to provide a representation for the class, Python uses a default method. What do you notice about the information for `loc1` and `loc4` ?
6. Also note the result of the equal comparisons between the locations, in particular `loc1==loc3` and `loc1==loc4`. Make sure you understand why the results are what they are.
7. Now modify the location.py code, adding in the methods (`__eq__() and __repr__()`). See the location_tests.py to figure out what the repr method should look like.
8. Run the `location.py` code with the modifications made above.
9. Now review the information printed out for each location. The `__repr__` method of `Location` is now being used when printing the object.
10. Examine the results of the equal comparisons. How are they different from before the `__eq__` method is added?

Part 2

1. Create a module called separator.py that has a main function that inputs numbers (integer and/or floats) from the keyboard and when the input stops (see below), outputs them on the screen in two lines:
    a. The first line contains all integers in the order they were input (start the line with "`Integers: `").
    b. The second line contains all non-integer numbers in the order they were input (start the line with "`Floats: `").
2. Allow at most N integer numbers and N non-integer numbers to be entered (*where N is a number obtained from a command-line argument.*)
3. Stop taking input when one of the following occurs:
    a. The user presses <Enter> after an empty line. (In this case, input returns an empty string.)
    b. An invalid value has been entered (something other than a number).
    c. An input number cannot be stored—i.e. either the (N+1)th integer or the (N+1)th float has been input.
4. **Hints:** isdigit(), float(), try…except…
5. **No** prompting should be done during this process. All output occurs after input is done.

Sample Run 1:

```
$ python3 separator.py 6
1 2 1.2 2.3
3
4
7.8 garbage, 12
Integers: 1 2 3 4
Floats: 1.2 2.3 7.8
```

Sample Run 2:

```
$ python3 separator.py 6
1 2 3 4
5 6 7 8
Integers: 1 2 3 4 5 6
Floats:
```

Testing

Test your program thoroughly. Here are some examples of tests:

1. The input contains only integers:
    a. More than N integers
    b. Exactly N integers (followed by a blank line)
    c. Fewer than N integers (followed by a blank line)

2. The input contains only non-integer numbers:
    a. More than N numbers
    b. Exactly N numbers (followed by a blank line)
    c. Fewer than N numbers (followed by a blank line)

3. The input contains only valid values (i.e. numbers of any kind):
    a. More than N integers but fewer than N non-integers
    b. More than N non-integer numbers but fewer than N integers
    c. Exactly N integers and exactly N non-integers (followed by a blank line)
    d. Fewer than N integers and fewer than N non-integer numbers (followed by a blank line)

4. The input contains several invalid values:
    a. First invalid value is preceded by more than N integers and fewer than N non-integers
    b. First invalid value is preceded by more than N non-integers and fewer than N integers
    c. First invalid value is preceded by exactly N integers and exactly N non-integers
    d. First invalid value is preceded by fewer than N integers and fewer than N non-integers

5. The input contains only invalid values

6. The input contains no values at all (blank line right away).

Submission/Grading

Make sure that the code that you want to be graded has been committed **and pushed** to GitHub. Failure to do so will result in receiving no credit for the lab. Not sure if you code has been pushed to GitHub? Use a web browser, go to you GitHub account and check! Once your assignment has been graded, feedback will be automatically emailed to you.