Lab 1: Python, Recursion, and Testing!

Part 0

1. 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)
```

2. 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)
3. Note the information that is printed out for each Location object – you should see something like this:
   `Location 1: <__main__.Location object at 0x000001F6A2E0C7B8>`
4. 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` ?
5. 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.
6. 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.
7. Run the `location.py` code with the modifications made above.
8. Now review the information printed out for each location. The `__repr__` method of `Location` is now being used when printing the object.
9. Examine the results of the equal comparisons. How are they different from before the `__eq__` method is added?

Part 1

1. In the lab1.py file, complete the **iterative function** to find the maximum integer in a list of integers.

```
def max_list_iter(int_list):  # must use iteration not recursion
    """finds the max of a list of numbers and returns the value (not the index)
    If int_list is empty, returns None. If list is None, raises ValueError"""
```

2. In the lab1.py file, complete the **recursive function** to reverse a list of integers:

```
def reverse_rec(int_list):   # must use recursion
    """recursively reverses a list of numbers and returns the reversed list
    If list is None, raises ValueError"""
```

3. In the lab1.py file, complete the **recursive function** to search a list of integers using binary search along with test cases. If the **target** of the search is in the list, the function returns its index.

```
def bin_search(target, low, high, int_list):  # must use recursion
    """searches for target in int_list[low..high] and returns index if found
    If target is not found returns None. If list is None, raises ValueError
    """
```

Test Cases

Many people tend to focus on writing code as the singular activity of a programmer, but testing is one of the most important tasks that one can perform while programming. Proper testing provides a degree of confidence in your solution. Systematic testing helps you to discover and then debug your code. Writing high quality test cases can greatly simplify the tasks of both finding and fixing bugs and, as such, **will save you time during development**. However, testing does not guarantee that your program is correct.

For this part of the lab you will practice writing some simple test cases to gain experience with the unittest framework. I recommend watching the first 20 minutes or so of the following video if you need more guidance on testing in Python. https://www.youtube.com/watch?v=6tNS--WetLI

Using your editor/IDE of choice, open the *lab1_test_cases.py* file. This file defines, using code that we will treat as a boilerplate for now, a testing class with a single testing function.
In the *test_expressions* function you will see some test cases already provided. You must add additional test cases to verify that your functions (max_list_iter, reverse_rec, bin_search) are correct.

Submission/Grading

Ensure that the following file have been pushed to GitHub by the due date:

- **location.py**
  - The location class with __eq__ and __repr__ methods added.

- **location_tests.py**
  - Unit tests for the location class. You do not have to worry about testing the code inside of the main function in location.

- **lab1.py**
  - Correct and well documented iterative **max_list_iter**, recursive **reverse_rec**, and recursive **bin_search** functions based on the template provided

- **lab1_test_cases.py**
  - A complete set of test cases for the functions above. Your test cases should test boundary conditions and other possible errors based on the structure of your program. For each test provide a comment (docstring) that explains what it is testing. Your tests cases will be tested with known incorrect (buggy) versions of the functions in lab1.py and will also be tested for 100% code coverage.

For folks new to Python or who need review

The basic Python Tutorial
https://docs.python.org/3/tutorial/

This site gives the essentials for those familiar with Java.
Python for Java Programmers: http://python4java.necaiseweb.org/Fundamentals/Fundamentals

If you want to use an IDE, you are welcome to. If you do not already have an IDE that you are familiar with, PyCharm is a good choice for Python development.
PyCharm IDE: https://www.jetbrains.com/pycharm/

Videos on specific topics.

Installing Python on Mac/Windows: https://www.youtube.com/watch?v=YYXdXT2l-Gg&list=PL- osiE80TeTt2d9bfVyTiXJA-UTHn6WwU

Strings: https://www.youtube.com/watch?v=k9TUPpGqYTo&list=PL-osiE80TeTt2d9bfVyTiXJA-UTHn6WwU&index=2

Conditionals: https://www.youtube.com/watch?v=DZwmZ8Usvnk&list=PL-osiE80TeTt2d9bfVyTiXJA- UTHn6WwU&index=6

Loops: https://www.youtube.com/watch?v=6iF8Xb7Z3wQ&list=PL-osiE80TeTt2d9bfVyTiXJA- UTHn6WwU&index=7

Functions: https://www.youtube.com/watch?v=9Os0o3wzS_I&list=PL-osiE80TeTt2d9bfVyTiXJA-UTHn6WwU&index=8

Modules: https://www.youtube.com/watch?v=CqvZ3vGoGs0&list=PL-osiE80TeTt2d9bfVyTiXJA- UTHn6WwU&index=9

For folks new to Cal Poly: Unix Environment

Unix

The lab machines run a distribution of the Linux operating system. For simplicity, and to gain experience in a, potentially, new environment, we will do our coursework in this environment.

Open a terminal window. To do so, from the system menu on the desktop toolbar, select **Applications → System Tools → Terminal**. The Terminal program will present a window with a command-line prompt. At this prompt you can type Linux commands to list files, move files, create directories, etc. For this lab you will use only a few commands. Additional commands can be found at:

- Unix Tutorial
    - Tutorials 1 & 2
    - Parts 1-5
- Editors
    - emacs tutorial
    - vi tutorial

In the terminal, type **ls** at the prompt and hit <Enter>. This command will list the files in the current directory. (also know as a folder.) If you type **pwd**, the current directory will be printed (it is often helpful to type **pwd** while you are navigating directories). If you type **tree**, then you will see a tree-like listing of the directory structure rooted at the current directory.

Create a new directory for your coursework by typing **mkdir cpe202**. Use **ls** again to see that the new directory has been created.

Change into this new directory with **cd** by typing **cd cpe202**. To move back "up" one directory, type **cd ..**. To summarize

- **ls** list files in the current directory
- **cd** change to another directory
- **mkdir** create a new directory
- **pwd** print (the path of) the current directory

Though these basic commands are enough for now, consider working through a Unix tutorial.

Editing

There are many options for editing a Python program. On the department machines, you will find vi, emacs/xemacs, nano, gedit, sublime, and others. The editor that one uses is often a matter of taste. You are not required to use a specific editor, but we will offer some advice (and we will try to help with whichever one you choose). There is lots more information here: http://users.csc.calpoly.edu/~akeen/courses/csc101/handouts/labs/lab1.html

Interactive Interpreter

The Python interpreter can be used in an interactive mode. In this mode, you will be able to type a statement and immediately see the result of its execution. Interactive mode is very useful for experimenting with the language and for testing small pieces of code, but your general development process with be editing and executing a file as discussed previously.

Start the interpreter in interactive mode by typing **python** at the command prompt. You should now see something like the following.

```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

The >>> is the interpreter's prompt. You can type an expression at the prompt to see what it evaluates to. Type each of the following (hit enter after each one) to see the result. When you are finished, you can exit the interpreter by typing ctrl-D (i.e., hold the control key and hit d).

- 0 + 1
- 2 * 2
- 19 // 3
- 19 / 3
- 19 / 3.0
- 19.0 // 3.0
- 4 * 2 + 27 // 3 + 4
- 4 * (2 + 27) // 3 + 4