Lab #5

Physics 91SI Spring 2012

Objective: This lab will introduce you to the Python debugger (pdb) and how to go about testing your code.

As usual, log on to `corn` and clone over the starter repository:

```
hg clone /afs/ir.stanford.edu/class/physics91SI/src/lab5 lab5
```

Remember to `hg commit` often to save your changes!

Part 1: Basic pdb examples

Have a look at the short script `debug.py`. It contains 4 functions which are supposed to perform simple operations - look at the source code or use `help(`*`function_name`*`)` to figure out what they're supposed to do. All of these functions are buggy, and your task is to use the command-line debugger `pdb` to find these bugs and squish them. You can start `pdb` by typing:

```
python -m pdb <script_name>.py [arguments]
```

You should be able to find all the bugs by stepping through the code and observing variable names and execution flow. You will need to implement actual function calls to the buggy functions from `main()` before you try to run the debugger.

Part 2: Basic Testing Examples

In the starter repository, you should find a script called `quadratic.py`. In this is a `find_roots()` function that returns the 2 roots of a quadratic equation of the form $ax^2 + bx + c = 0$. It's designed to be run from the command line, with a, b, and c given as arguments. For example,

```
python quadratic.py 1 2 -15
```

should print both roots of $1x^2 + 2x - 15 = 0$. You can also use `import quadratic` from the interpreter to access specific functions. For example:

```
root1 , root2 = quadratic.find_roots(1, 2, -15)
```

will perform the same computation as above.

There's also a file, `test_quadratic.py`, which imports `quadratic` as a module and provides access to its functions (i.e. `quadratic.find_roots(a,b,c)`). The code describes how to write a single test, ie give the a, b, and c of a known quadratic equation and compare the output to what you know to be the roots of that equation. The example in test_quadratic.py gives (1 , 2 , -15) as (a , b, c) and expects to get x = -3 and x = 5 as the roots.

In this file, write several more tests that try to cover all edge cases. For example, test the find_roots function with quadratic equations with non-integer roots (like 2.5 and 3.3), or with b or

c equal to zero. WARNING: do not attempt to test quadratic equations with a = 0 or imaginary roots, as the function is not designed for this, and this relates back to the "Don't go overboard" principle of accounting for edge cases.

There are 4 types of errors in `quadratic.py` , run your test script and see what tests fail. Then go into pdb by typing

```
python -m pdb test_quadratic.py
```

and step into each call to `find_roots()` to try to find the errors. One of the errors only exists when you run `quadratic.py` as a stand-alone script, where you use command-line input, so try finding that by running

```
python -m pdb quadratic.py 1 2 -15
```

Continually search for the errors, correct them in `quadratic.py`, re-run your test script, and repeat, until quadratic.py passes all tests.

Part 3: Testing Your Analysis.py Module

In this part of the lab, we're going to ask you to test the functions you wrote in `analysis.py` as part of Lab #3. We've provided a script that will copy over your submitted copy of analysis.py and add it to the current repository. All you need to do is:

1. Submit your lab 3 in the usual manner (even if it's not done)
2. Run `init.py` in the the lab 5 directory (`./init.py`)

This is an example of when your program is designed to work with a data file, and obviously the data files you have are extremely long (otherwise you probably wouldn't need a computer to tell you the index of the max value). So you can't look through the data files and find the index of the max y index and then use that to check that your functions return the correct answer.

What you'll want to do is write several of your own short data files (in the same format as the old ones!) and then add your tests to the test script we setup for you (named `test_analysis.py`), which imports analysis and then will run the functions you wrote on each of those short data files and check to make sure the outputs are correct. As an example, we give you one new, short data file named `testdata1.dat`.

Write some more short data files of your own, and try to change up the data so as to target any edge cases you could think of, such as the max y value being on the first or last line of the file. You can also try giving bad xmin and xmax values to your find_peak function, such as xmin greater than xmax. What should the function do in these cases?
Write in your tests and then run the test script and fix any errors you find until your code passes all of your tests.