CISC106 Fall 2011 Lab02

- This lab an all subsequent labs will be due Sunday at 11:55 PM EDT on Sakai.
- The preparation problems below are to develop your understanding without creating extra work for you or the TA; these problems will not be graded. Be sure to read and understand them - they will help with the problems you must submit for grading

Preparation (do not submit for grading)

1. Fire up IDLE so that you have a shell window. At the prompt[1] type in:

   ```
   >>> min(5, max(1, 7))
   ```

   This is a simple example of function composition: taking the result of one function and using it as input to another function.

2. Type the following into the shell window (you need to press enter twice at the end of the function to get the prompt back):

   ```
   >>> def plus1(x):  return x + 1
   ```

   Figure out how to compose multiple calls to the `plus1` function to get the number `3` if you start by calling `plus1(0)`.

3. This following exercise walks you through the process of writing a a function which calculates the area of a rectangle. It will not be graded, but you *must submit it as part of the lab.* Download *lab02.py* and *lab02_tests.py* from the Labs folder on the course website (be sure to save them both in the same directory,) as you will be putting the following code in those two files. You can open a file in IDLE by selecting File→Open from any IDLE window.

   (a) The first thing you want to do is think about what a function which computes the area of a rectangle should look like. What parameters does it take and what does it return? Write a test which reflects your decisions. This test should be one you expect to pass when the function is working - even though it won't pass right now because you have yet to write the function:

   ```
   def test_area(self):
       self.assertEqual(area(0, 0), 0)
   ```

   Add the above test to lab02_tests.py, directly beneath the other three tests, indented exactly as those three tests.

   Run your tests. (by going to Run→Run Module in the lab02_tests.py window.) You should notice that this test now fails.

[1] NB: from henceforth, I will preceed any input meant for the Python shell by the Python prompt, `>>>`.

(b) Now add to the bottom of lab02.py a *signature* and *header* for your function:

```
def area(width, height):
    """
    Computes and returns the area of a rectangle
    with width 'width' and height 'height'.
    """
```

The function signature is the def statement line and the header is everything between the """. The point of a header is to describe in English what the purpose of the function is. Describe each parameter and why it is necessary, as well as what the function returns. This way other programmers reading your code (or even yourself after a couple of months) will have an idea of what you were thinking when you wrote it.

If you run your tests now, your test for area will still fail but now the error message will be different. Why?

(c) Now you'll want implement the function such that your test passes. Add the code for the function underneath the header:

```
def area(width, height):
    """
    Computes and returns the area of a rectangle
    with width 'width' and height 'height'.
    """
    return width * height
```

Run your tests again and note that `test_area` now passes.

(d) Modify `test_area` with a new assertion that you expect should hold for a properly implemented area function:

```
def test_area(self):
    self.assertEqual(area(0, 0), 0)
    self.assertEqual(area(10, 10), 100)
```

Run your tests again. In this case, `test_area` still passes. If it didn't, you would need to go back and modify your implementation of area such that the test again passed.

(e) We will add one more assertion to the test and run it again before we're satisified we have the correct implementation:

```
def test_area(self):
    self.assertEqual(area(0, 0), 0)
    self.assertEqual(area(10, 10), 100)
    self.assertEqual(area(1, 10), 10)
```

Programs (to be graded)

1. The first function in the file lab02.py you downloaded is named `messed_up`. Without removing or adding any lines of code to the function, fix all the errors (and rename the function appropriately) so that the first test in lab02_tests.py passes. (Again, run the tests by selecting Run→Run Module from lab02_tests.py's windows.)

2. Note the functions `f`, `g` and `h`. Also note that the tests `test_composition1` and `test_composition2` are still failing (make sure to look carefully at the error messages and understand what they are saying!)

   (a) Correct `test_composition1` by replacing the second parameter 0 in the `assertEqual` with the correct output.

   (b) Complete the function `composition2` by replacing the 0 in the return statement. Use only a composition of function calls to `f`, `g` and `h` (you can call each more than once) with x so that the asserts in `test_composition2` pass.

3. In physics, an object floats if its weight is less than or equal to the weight of the liquid it displaces. Add the following functions to your lab02.py. Implement each of these functions in manner outlined in the prepartion part of the lab: write a test in lab02_tests.py, then make the function pass the test.[2] Repeat this process several times until you're sure that you have a correct function. **Hint:** You want to *return* the values you compute rather than *print* them. All printing does is display a value to the screen for a human to see - if you want your function to do anything to the state of your program, it must return something.

   (a) Write a function, `total_weight`, that calculates the weight of a liquid given its volume (in cubic centimeters) and its density (in grams per cubic centimeter.) Here are some liquid densities you're free to use in your tests:

   ```
   WATER_DENSITY = 1.0
   MILK_DENSITY = 1.03
   GASOLINE_DENSITY = 0.7
   ```

   if you want, you can look up and use other densities as well.

   (b) Assume you have a cubic box that is $100cm \times 100cm \times 100cm$. The box itself weighs $500g$ and is open on the top. Write a function, `max_contents` that takes as a parameter a liquid density (in grams per cubic centimeter) and calculates maximum weight of contents that could be placed in the box such that it still floats in the liquid. Your `max_contents` function should use your `total_weight` function.

   (c) Write a function, `box_orders`, that takes a parameter for liquid density and a parameter for a total weight of inventory. Your function must return the minimum number of boxes required that will keep all of the inventory afloat. You may assume that you can split the inventory into the boxes evenly. However, you cannot order a fractional number of boxes - you will need to round up your order to the closest number of boxes. You should use your `total_weight` function in this function. **Hint:** You can

[2] NB: this is the process you will use to develop code throughout the semester. You will always write tests *before you write* the corresponding code.

use the built-in `ceil` function in the math module of Python to do this, just remember you need to import the math module if you do!

You should submit your lab02.py and your lab02_tests.py and any other docs required by your TA on Sakai.