CIS 480 - Python - Fall 2005
Homework #8
HW #8 due: Thursday, November 3rd, 12:00 noon

(The Week 10 lab exercise will be handed out in lab...)

Until I develop more formal opening comment block standards (which I STILL plan to do), begin **each python module** that you write with at least the following opening comments:

* a comment containing the name of the module (the **file name** of the module, please --- **hw07.py**, for example)
* a comment containing your name, and
* a comment containing the date that your module was last modified

HOMEWORK #8

Create a Python module **hw08.py**. Within it, include the following:

1. Write a function **call_with_2** that takes a function as its parameter. It should then return the result of calling its parameter with --- you guessed it --- the argument **2**.

In a separate module **hw08_test.py**:

* import the **hw08.py** module,
* write a **print** statement that says, **testing call_with_2**.
* write a **print** statement that says, **"True == Passed, False == Failed"**
* write at least two DIFFERENT functions of YOUR choice that can handle a single numeric argument, but do DIFFERENT things with it.
* then write print statements comparing calls to **call_with_2** with those functions as its arguments to their expected values.

2. Write a function **add_amt_fun** that has one parameter, a numeric value, and returns a function that expects one parameter and tries to have as its value the result of adding **add_amt_fun**'s parameter to the new function's parameter. (Note: you must use a **lambda expression** here --- you may not refer to anything outside of **add_amt_fun** in its implementation.)

For example, an example of **add_amt_fun** in action could be as follows:

```
>>> import hw08
>>> looky = hw08.add_amt_fun(4)
>>> looky(3)
7
>>> looky(6)
10
>>> again = hw08.add_amt_fun(1000)
>>> again(5)
1005
>>> again(-5)
995
```

In **hw08_test.py**, test **add_amt_fun** as you tested the function from problem #1, except FIRST call it at least twice with different arguments, assigning the results to some variables of your choice, and THEN call the resulting functions at least twice each, writing print statements comparing the results of those calls to the expected results.

3. Write a function **silly_calc**. It interactively asks the user for two values, and then asks 5 times what action the user would like done to those two values. If the user says **'add'**, it prints the result of adding those two values; if the user says **'subtract'**, it prints the result of subtracting those two values; if the users says **'average'**, it prints the average of those two values, if the user says **'raise'** it prints the result of raising the first value to the power of the second value. If it says anything else, it prints **'Huh? I only know add, subtract, average, and raise!'**

Within silly_calc, you must set up and use a **dict** that has lambda expressions as its keys' values to implement the needed multiway branch. (Note that those lambda expressions will NOT try to print! **silly_calc** will print the results of running the desired lambda expression...)

This being interactive, I'm not sure how to have you test it in **hw08_test.py**. I'll just have to test your function interactively myself... 8-)

4. Write a function **reduce_all**. It should expect two arguments: a list of numbers, and a quantity. It **must** use **map** and a lambda expression to create a new list that consists of the parameter list values each reduced by the given quantity. Also, if NO quantity is given, the quantity should be assumed to be **1** (that is, specify a **default** value for this parameter in its parameter list).

(If done properly, this function's body should consist of only one line --- two lines tops!)

For example,

```
>>> hw08.reduce_all([3, 6, 9, 12], 4)
[-1, 2, 5, 8]
>>> hw08.reduce_all([5, 2, 3])
[4, 1, 2]
```

In **hw08_test.py**, test **reduce_all** as you tested the function from problem #1, calling it for at least 3 different sets of arguments, at least one of which consists ONLY of a list (that consists of only 1 argument, a list).

5. Now, we are going to write a variant of **reduce_all** called **reduce_all2**. This should take a quantity, and then a variable number of arguments --- it should return a list consisting of the results of reducing all but the first argument by the amount of the first argument. This one should NOT have a default value for the quantity parameter.

If written correctly, this should use exactly the same **body** as **reduce_all** --- however, it will be CALLED differently (with the amount to reduce the rest by, then as many values as you'd like; it will NOT expect a list as argument, or react well if one is given!)

For example,

```
>>> hw08.reduce_all2(4, 3, 6, 9, 12)
[-1, 2, 5, 8]
>>> hw08.reduce_all2(1, 5, 2, 3)
[4, 1, 2]
```

In **hw08_test.py**, test **reduce_all2** as you tested the function from problem #1, calling it for at least 3 different sets of arguments (consisting of different numbers of arguments!)

BONUS PROBLEM - +5 POINTS

Write an interesting/nifty function that has as one of its parameters **unmatched keyword arguments** gathered into a dictionary (a ****name** parameter). Add appropriate tests of your function (in the style of problem #1) to your **hw08_test.py**.

And, when you are satisfied with the above problems, you should create a submittable output file **hw08_test.out**:

```
python hw08_test.py > hw08_test.out
```

By the due date and time given at the beginning of this handout, use **~st10/480submit** to submit your final versions of **hw08.py**, **hw08_test.py**, and **hw08_test.out** (And remember --- you can submit more than one version of these before the deadline, if inspiration strikes after a submission. As the syllabus notes, I'll simply grade the latest version that was submitted before the deadline.)