CIS 480 - Python - Fall 2005
WEEK 3 LAB EXERCISE and Homework #2

week 3 lab exercise due: Thursday, September 8th, END of lab
HW #1 due: Thursday, September 15th, 12:00 noon

Until I develop more formal opening comment block standards (which I 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 --- **lab03.py**, for example)
* a comment containing your name, and
* a comment containing the date that your module was last modified

WEEK 3 LAB EXERCISE

For this lab exercise, you are **encouraged** to check your answers with classmates. In fact, you are **required** to do so with at least one classmate who has not yet had his/her work checked yet before you have me check your work; write the name(s) of those you checked your work with below:

____

____

(The point here is to have to argue, er, discuss with others why your answers are correct if they think they are not, or why others' are not correct if they think they are, but you do not...)

1. Answer the following questions in the space provided:

* Write an example of a Python **raw string** ______________________
* Write an example of a Python **Unicode string**

  ____
* Write an example of a Python **multiline block string**

  ____

  ____

  ____

  ____
* Write an example of a Python **string literal** that contains a backslash character.

  ____
* Write an example of a Python **string literal** that contains a single-quote character.

  ____

* Write an example of a Python **string literal** that contains a double-quote character.

  ____

* Write an example of a Python **string literal** that contains a newline character.

  ____

2. Assume that **client_name** is a Python variable to which a string literal has **already** been assigned. Write the specified expression using **client_name** in the space provided:

* write an expression whose value would be the length of **client_name**

  ____

* write an expression that would be the string resulting if you repeated **client_name**'s contents seven times

  ____

* write an expression that would be the 8th letter in **client_name** (assuming that **client_name**'s length is at least 8!)

  ____

* write an expression that would be the last letter in **client_name** --- WITHOUT knowing its length! (You may not call a function that gives you its length, for example --- I want you to show me an alternate approach to this.)

  ____

* write an expression that would be the slice of **client_name** including from the 3rd character to the 6th character, **inclusive** (again, we are assuming that **client_name** has at least 6 characters).

  ____

* write an expression that would be the slice of **client_name** including its first 5 characters.

  ____

* write an expression that would be the slice of **client_name** starting at its 5th character and going to the end of **client_name** (WITHOUT specifically making any reference to its length).

  ____

3. Although we didn't discuss them in lecture on Tuesday, strings in Python have a rich set of **methods** associated with them. (A method is a function associated with an object...) That is, for any string expression --- literal or variable or expression resulting in a string! --- you can type that expression, then a period, then the method name, and then parentheses (containing arguments if needed) to call that method for that string.

What in the heck do I mean? Well, for example, there is a string method **upper()**. It returns a version of the calling string all converted to uppercase. That is,

```
>>> 'hello'.upper()
'HELLO'
>>> name = 'anna'
>>> name.upper()
'ANNA'
>>> name
'anna'
```

(Notice how the calling string itself was not changed --- the upper() method returned a new string modified as I described.)

There's a table (Table 5-4) of the built-in string methods on p. 91 of Learning Python**.**

So, what do I want you to do for the lab exercise?

I want you to create a module **lab03.py**, and write a function **yell_it** in this module. **yell_it** should expect one argument. If...

* ...the argument is a string, that string should be printed to the screen in all-uppercase letters followed by three exclamation points;

* ...the argument is a floating point number, it should be printed to 3 fractional places (hint: try the format **%.3f** for this...) followed by three exclamation points;

* ...the argument is anything else, just print it to the screen followed by three exclamation points.

We're going to add quick-n-sleazy test calls of **yell_it** directly to lab03.py. You see, if you've just defined a function in a module, you can call it in that module without preceding it by the module name (although this is "icky" for convenient re-use of this module and function, so NORMALLY we won't do it...) But, for lab convenience, after you have typed **yell_it**'s definition in **lab03.py**, skip a line, and then type in four calls to **yell_it**:

* one with a string value of your choice, then
* one with a floating point number with 1 fractional place, then
* one with a floating point number with at least 5 fractional places, and then
* one with any non-string and non-floating-point argument of your choice.

When this runs like you would like, check your work with at least one classmate, and then write your name on the 'Next:' list on the board to get your work checked.

All of the above must be completed before the end of your lab time.

HOMEWORK #2

Create a Python module **hw02.py**. Within it, include the following:

1. Write a function **nickname**. **nickname** expects one string parameter, a name for which it will generate a goofy nickname, meeting the following requirements:

* if the length of the name passed is less than 5 letters, it will simply return the original name as

the nickname;

* if the length of the name passed is 5 or more, it will grab the first 5 letters to begin the nickname, and:

    * ...if the 5th letter of the name is a **y**, strip off the **y** and return the result as the nickname;

    * ...otherwise, add **y** to the end of the nickname-so-far and return the result as the nickname;

For example,

```
>>> import hw02
>>> print hw02.nickname('Sally')
Sall
>>> print hw02.nickname('ed')
ed
>>> print hw02.nickname('Horatio')
Horaty
>>> print hw02.nickname('elizabeth')
elizay
```

In a separate module **hw02_test.py**:

* import the **hw02.py** module,
* write a print statement that gives the actual call you are about to make and what it SHOULD return, before a print statement printing the actual result of that call --- for example,

```
print "hw02.nickname('ed') should return ed:"
print hw02.nickname('ed')
```

...for each of at least **three** different testing calls:

* at least one with a name of less than 5 characters,
* at least one with a name of >= 5 characters but with a y as the 5th character,
* at least one with a name of >= 5 characters but with a non-y as the 5th character

2. Note that string method **isupper()** returns True if all of the characters in the calling string are uppercase; it returns false otherwise. For example,

```
>>> 'HELLO'.isupper()
True
>>> greeting = 'Hi'
>>> greeting.isupper()
False
>>> 'hi'.isupper()
False
```

islower() works analogously for returning if a string is all-lowercase. And, like **upper()** returns a version of the calling string in all-uppercase characters, **lower()** returns a version of the calling string in all-lowercase characters.

Write a function **flip_it** that expects a string argument.

* If this argument is in all-uppercase, it should return a version of the argument in all-lowercase.

* If the argument is in all-lowercase, it should return a version of the argument in all-uppercase.

* What if it is neither all-uppercase nor all-lowercase? Then, look at its **first** character's case.

    * If the first character is uppercase, return a version of the argument where only the FIRST letter is changed to lowercase, and the rest are returned unchanged.

    * If the first character is lowercase, return a version of the argument where only the FIRST letter is changed to uppercase, and the rest are returned unchanged.

Add to module **hw02_test.py:**

* write a print statement that gives the actual call you are about to make and what it SHOULD return, before a print statement printing the actual result of that call --- for example,

```
print "hw02.flip_it('hello') should return HELLO:"
print hw02.nickname('hello')
```

...for each of at least **four** different testing calls:

* at least one with an argument that is all-uppercase,
* at least one with an argument that is all-lowercase,
* at least one with an argument that is mixed case, but begins with a lowercase letter,
* at least one with an argument that is mixed case, but begins with an uppercase letter.

3. Let's do some gentle string formatting practice. Write a function **amt_due** that expects a quantity of items and a price per item as its arguments. But, instead of just returning the amount that would be due for that quantity of items at that price per item, it will return a string: that amount formatted in dollars and cents, beginning with a dollar sign ($) and always printed to two fractional places.

That is,

```
>>> hw02.amt_due(3, 2)
'$6.00'
>>> hw02.amt_due(3.5, 2.25)
'$7.88'
```

Add to module **hw02_test.py:**

* write a print statement that gives the actual call you are about to make and what it SHOULD return, before a print statement printing the actual result of that call --- for example,

```
print "hw02.amt_due(3, 2) should return $6.00:"
print hw02.amt_due(3, 2)
```

...for each of at least **two** different testing calls:

* at least one whose two arguments happen to be integers
* at least one whose two arguments will result in a product with at least 3 fractional places

----
BONUS - UP TO 10 points

Within **hw02.py**, write a function that makes practical use of up to **five** of the string methods on p. 91, Table 5-4 that were not mentioned anywhere in this handout nor that you used in problems #1, #2, or #3.

Then, within **hw02_test.py**, write a print statement that gives the actual call you are about to make and what it SHOULD return, before a print statement printing the actual result of that call, for the number of testing calls that you feel adequately show off your function and its capabilities.

(Note that you'll have to research how some of these work --- www.python.org would probably be a good place to start on these, if the textbook isn't immediately helpful...)

You'll receive up to 2 bonus points for each 'new' method --- up to five --- that your function uses in a practical way (as long as you also adequately test your function in your provided example calls in **hw02_test**!)

Really nifty functions **may** be showed off to the rest of the class, if I choose to do so.

And, when you are satisfied, you should create a submittable output file **hw02_test.out**:

```
python hw02_test.py > hw02_test.out
```

By the due date and time given at the beginning of this handout, use **~st10/480submit** to submit your final versions of **hw02.py**, **hw02_test.py**, and **hw02_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.)