CIS 480 - Python - Fall 2005
WEEK 4 LAB EXERCISE and Homework #3

week 4 lab exercise due: Thursday, September 15th, END of lab
HW #3 due: Thursday, September 22nd, 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 --- **lab04.py**, for example)
* a comment containing your name, and
* a comment containing the date that your module was last modified

WEEK 4 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 empty list literal: _____

* Write an example of a Python non-empty list literal: _____

2. Assume that **price_list** is a Python variable to which a list literal has **already** been assigned, and that it currently contains at least 10 (top-level) items.

* write an expression whose value would be the number of (top-level) items in (or, the length of) **price_list**

  _____

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

  _____

* write an expression that would be the 8th item in **price_list**

  _____

* write an expression that would be the last item in **price_list** --- WITHOUT knowing price_list's 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 **price_list** including from the 3rd item to the 6th item, **inclusive**.

  _____

* write an expression that would be the slice of **price_list** including its first 5 items.

_____

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

_____

* write a statement that would **replace** the 7th item in **client_list** with the float value **9.99**.

_____

* write a statement that would replace the 3rd through the 6th items, **inclusive**, in **price_list** with the items 1.99 and 2.99. (Note that the length of **price_list** should decrease by two as a result...)

_____

* write an expression that would be true if 99.89 is an element within **price_list**.

_____

* Assume that **price_sum** is a variable that has been set to 0.0. Write a **for-loop** that, when finished, will have set **price_sum** to be the sum of the (assumed-to-be-numeric) items in the list **price_list**.

3. We did get a chance to sample a few **list** methods in Tuesday's lecture. Still assuming that **price_list** is a Python list variable that currently contains at least 10 items...

(a) write an expression whose value would be the number of times that the item **1.99** occurs within **price_list**.

_____

(b) write a single expression which will remove the item at the end of **price_list** and return it as its value.

_____

(c) write a statement that will add the price 39.98 to the end of **price_list**.

_____

(d) write a statement that will sort **price_list** in-place.

_____

4. There were some interesting string methods that we skipped back in Chapter 5 because they involved lists. But, now that we have lists, it is appropriate to bring them up.

split [Learning Python, p. 92) "chops up a string into a **list** of substrings"

If you call the string method **split** with no arguments, it returns a lists that is the result of splitting up the calling string based on whitespace ... any whitespace is assumed to mark the end of one string and the beginning of the next. That is,

```
>>> name = 'Cleese Gilliam\tChapman     Idle \t Jones'
>>> print name
Cleese Gilliam  Chapman     Idle         Jones
>>> actors = name.split()
>>> actors
['Cleese', 'Gilliam', 'Chapman', 'Idle', 'Jones']
>>> name       # unchanged, of course
'Cleese Gilliam\tChapman     Idle \t Jones'
```

And, the string method **join** can take a list as argument, and it returns a new string consisting of the list elements separated by copies of the calling string:

```
>>> ' '.join(actors)
'Cleese Gilliam Chapman Idle Jones'
>>> 'SPAM'.join(actors)
'CleeseSPAMGilliamSPAMChapmanSPAMIdleSPAMJones'
```

NOW that you know this...

Create a module **lab04.py**, and in it write a function **words_in_order**. **words_in_order** expects a single string of words as its argument. It should split these words into a list (based on white space), convert all the words in the list into all-lowercase, sort the resulting list in-place, and then join the sorted list into a string such that the sorted words are separated by a comma and a blank. That is,

```
>>> lab04.words_in_order('The rain in Spain stays mainly in the plain')
'in, in, mainly, plain, rain, spain, stays, the, the'
```

(HINT: I found it easier to create a new empty list and then add each "lowered" word to the new list, rather than trying to convert each word in a list into lowercase "in place". Your mileage may vary.)

Then create a module **lab04_test.py**. In this module:

* import the **lab04.py** module,
* write print statement(s) that give 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 "lab04.words_in_order('The rain in Spain stays mainly in the plain')"
print "   should return: "
print "in, in, mainly, plain, rain, spain, stays, the, the"
print lab04.words_in_order('The rain in Spain stays mainly in the plain')
```

...for each of at least **TWO** different testing calls (one of which must be the **above** call).

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 #3

Create a Python module **hw03.py**. Within it, include the following:

1. Write a function **pig_latin**. **pig_latin** expects one string parameter, assumed to be a word, and it returns a new string that is the passed string converted into Pig Latin.

These will be the requirements for this particular Pig Latin dialect:

* if the passed string begins with a vowel, return a string that has **-ay** concatenated to the end.
(`hw03.pig_latin('apple') == 'apple-ay'`)

* else if the passed string begins with a consonant followed by an h, return a string that has the first two letters removed, and has a dash, those two letters, and **ay** concatenated to the end
(`hw03.pig_latin('chat') == 'at-chay'`)

* else return a string that has the first letter removed, and has a dash, that first letter, and **ay** concatenated to the end
(`hw03.pig_latin('dog') == 'og-day'`)

HINTS: remember the **in** operator, and how useful **slices** can be...

In a separate module **hw03_test.py**:

* import the **hw03.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 "hw03.pig_latin('apple') should return apple-ay:"
print hw03.pig_latin('apple')
```

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

* at least one with a word that begins with a vowel,
* at least one with a word that begins with a consonant followed by an h,
* at least one with a word that begins with a consonant not followed by an h.

2. Write a function **pig_list** that expects a single list of strings as its argument. It should assume that each string in the list is a single word, and is required to call **pig_latin** appropriately to create, and then return, a new list that contains the argument-list words each transformed into its pig-latin equivalent.

Add to module **hw03_test.py:**

* write print statement(s) that give 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 "hw03.pig_list(['hello', 'should', 'ivory']) should return: "
print "['ello-hay', 'ould-shay', 'ivory-ay']"
print hw03.pig_list(['hello', 'should', 'ivory'])
```

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

3. Write a function **strip_punct**. It expects a single string of words, sentences, or phrases as its argument. It returns a new version of the passed string changed such that:

* any apostrophes (') are **removed**; (replacing each with the **empty** string can work nicely);
* any dashes (-) are **removed**;
* any remaining non-alphabetic, non-white-space characters are replaced with **blanks**;
* (hints: play with or read up on string methods **isalpha** and **isspace**... And remember that Python's boolean operator is **not**)

Add to module **hw03_test.py:**

* write print statement(s) that give 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 "hw03.strip_punct('I don\'t think so, Bucky-lad!') should return: "
print "I dont think so  Buckylad "
print hw03.strip_punct('I don\'t think so, Bucky-lad!')
```

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

* at least one with a string that includes an apostrophe,
* at least one with a string that includes a dash,
* at least one with a string that includes a non-apostrophe, non-dash, non-white-space character,
* at least one with a string that includes white space.

(note that a single testing call could incorporate SEVERAL of the above...!)

4. Write a function **lower_list** that expects a single list of words as its argument. It should return a new version of that list with all of the words converted to all-lowercase.

Add to module **hw03_test.py:**

* write print statement(s) that give 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 "hw03.lower_list(['Hello', 'SHOULD', 'ivory']) should return: "
print "['hello', 'should', 'ivory']"
print hw03.lower_list(['Hello', 'SHOULD', 'ivory'])
```

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

* at least one including a word that is in all-uppercase,
* at least one including a word that is in mixed-case,
* at least one including a word that is in all-lowercase.

(note that a single testing call could incorporate SEVERAL of the above...!)

5. Finally, write a function **pig_latinize**. It should expect a single string (expected to consist of words, phrases, or sentences) as its argument. It should create and return a pig-latinized version of the string by:

* appropriately using **strip_punct** to "clean up" the passed string,
* splitting on white space to get a list of words,
* appropriately using **lower_list** to get a list of the words in all-lowercase,
* appropriately using **pig_list** to obtain a list of the words in pig-latin form,

* joining the list of pig-latin words back into a string (with the pig-latin words separated by a blank) and returning the resuling string.

Add to module **hw03_test.py:**

* write print statement(s) that give 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 "hw03.pig_latinize('Don\'t eat dog-food,    Charlie') "
print "   should return: "
print 'ont-day eat-ay ogfood-day arlie-chay'
print hw03.pig_latinize('Don\'t eat dog-food,    Charlie')
```

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

* at least one including a string with consecutive blanks,
* at least one including a string with a single quote,
* at least one including a string with a dash,
* at least one including a string with other punctuation,
* at least one including a word containing mixed case,
* at least one including a word that begins with a vowel,
* at least one including a word that begins with a consonant followed by an h,
* at least one including a word that begins with a consonant not followed by an h.

(note that a single testing call could incorporate SEVERAL of the above...!)

BONUS - UP TO 10 points

Within **hw03.py**, write a function **fib_create** that takes a length as its single argument, and returns a list of Fibonacci numbers of that length (starting with the elements **1** and **1** if the given length is at least 2) as its result. You may write and use additional "helper" functions as you'd like; you may NOT change the number of arguments to **fib_create**. Be warned that I started playing with this for this homework, then decided it was going to be, ah, more complicated than I wanted, and abandoned the attempt before I came up with a solution. So, I don't know at this point if this is really a reasonable function to ask you to write or not (thus, it is a bonus!)

Then, within **hw03_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 shows off your **fib_create** function and its capabilities.

Really nifty functions **may** be shown 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 **hw03_test.out**:

```
python hw03_test.py > hw03_test.out
```

By the due date and time given at the beginning of this handout, use **~st10/480submit** to submit your final versions of **hw03.py**, **hw03_test.py**, and **hw03_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.)