CS 232 – Programming in Python

Assignment #2

Deadlines

This assignment is due on **Tuesday, February 23, 2015 @ 5:00 PM.**

How to submit your work on the assignment:

- Your assignment will be turned in as a computer file (the "module") containing your Python code. The module will contain only the code that satisfies the problems' requirements. **There is NO NEED to submit a testing module! I will run your code with my own testing module.**

The File Naming Convention

- Properly name the file that contains your work. The file's name will contain the following, with dashes in between each part listed below and no spaces:

  * The course name, `CS232`

  * Then a dash and the assignment number as a two-digit number, in this case `02`

  * Then a dash and your HSU account username with your initials and a number – in this example, I'll use `jqs123` for a typical student named John Q. Smith

  * Windows should automatically add a `.py` at the end of the file. This may be invisible, depending on how Windows is configured on the computer you're using. If you're using a Mac, you'll likely need to add a `.py` to the end of the filename.

  Put all the rules together, and John Q. Smith's Assignment 1 file in CS 232 would be named

  `CS232-02-jqs123.py`

  Your file name will be different from this because your HSU username is different, but that's the **only** difference!

Submitting your file electronically

For this assignment, you will email your file to the instructor as a file attachment. Future assignments may change the way you submit your work, but email will do for now.

Send your email to: `david.tuttle@humboldt.edu`

Type the Subject line of the email as: `CS 232 - Assignment #2`

When I receive your file in the correct way, I will send you an acknowledgment by reply email. If there's a problem, I will let you know by reply email that you need to submit the file again.

If you do NOT get an email reply within 12 hours, try sending the file again. If you wait until too close to the deadline, you run the risk that an improper submission may result in a late penalty!

NOTE: This assignment has a maximum value of 120 points, but will be graded on a 100-point scale. This means there are 20 extra credit points built in!

Documentation of your Python code

For this assignment, please place all functions within a single Python module (file).

Begin **each Python module** (file) that you submit with at least the following opening comments:

* a comment containing the name of the file,
* a comment containing your name, and
* a comment containing the date that your module was last modified

For example:

```
# CS232-02-jqs123.py
# John Q. Smith
# Last modified: February 30, 2099
```

Before each individual function that you've designed, there should be a set of comments that:

* list the data types for the input arguments and the output (use "object" is no specific type is needed)
* give a brief explanation of the purpose of the function, describing what the function expects as input and what it returns as output
* any "side effects" (getting data from the user, printing to the screen) that occurs when the function runs

For example:

```
# Problem 1
# add_them: float float → float
# purpose:  expects two numeric (float) values
#           returns the sum of the two values
# side effects: prints a message containing the sum of the two values
```

PROBLEM 1 – 20 POINTS

Write a function **`pig_latin`** that expects one string parameter, assumed to be a word (string), and it returns a new string that is the passed string converted into Pig Latin as follows:

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

Test this function within the Python interactive window until you are confident that it works. Do NOT include testing code in the Python module when submitting!

PROBLEM 2 – 20 POINTS

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` to create, and then return, a new **list** that contains the argument word list with each transformed into its Pig Latin equivalent. Here is an example:

```
>>> pig_list(['hello', 'should', 'ivory'])
['ello-hay', 'ould-shay', 'ivory-ay']
```

Test this function within the Python interactive window until you are confident that it works. Do NOT include testing code in the Python module when submitting!

PROBLEM 3 – 20 POINTS

Write a function `fib_create` that takes a positive integer (with value at least 2) as its single argument, and returns a **list** of Fibonacci numbers of that length (starting with the elements 0 and 1) as its result. You may write and use additional "helper" functions as you'd like; but you may NOT change the number of arguments to `fib_create`.

Test this function within the Python interactive window until you are confident that it works. Do NOT include testing code in the Python module when submitting!

PROBLEM 4 – 20 POINTS

Write a function `letter_freq` that expects one string parameter (expected to be a multi-word sentence, phrase, or other such language fragment containing only letters and spaces, with no punctuation). It returns a Python **dictionary** containing, for each letter found within the passed string, a **key** consisting of the **lowercase version** of that letter, and a **value** consisting of how many times that letter (uppercase OR lowercase) appears in the passed string. Note that letters that do **not** appear should **not** be keys in the resulting dictionary. You may use a string method like `count()` if you wish. For example:

```
>>> letter_freq('Now is the winter of our discontent'))
{'w': 2, 'n': 4, 'i': 3, 's': 2, 'r': 2, 'd': 1, 'f': 1, 'u': 1, 'e': 3,
 'c': 1, 'h': 1, 'o': 4, ' ': 6, 't': 4}
```

Test this function within the Python interactive window until you are confident that it works. Do NOT include testing code in the Python module when submitting!

PROBLEM 5 – 20 POINTS

Write a function `freq_bar_chart` that expects a **dictionary** of letter frequencies as its argument (such as the one returned by `letter_freq` in Problem 4, and does not return anything. As a side effect, it will print to the screen a simple ASCII horizontal "bar chart" of the letter frequencies that meets the following specifications:

* It should start with an appropriate, eye-catching title of your choice;
* Each letter key followed by a dash, then a number of `X` characters corresponding to the value for that letter key, on its own line. Note that you may incorporate white space as you wish in this to make it look readable and attractive; and
* The letter keys should appear in **alphabetical** order.

Test this function within the Python interactive window until you are confident that it works. Do NOT include testing code in the Python module when submitting!

PROBLEM 6 – 20 POINTS

(This problem is a challenge, and may require writing multiple functions to finish!)

Write a function **plot_points** that expects a **dictionary** as described in the Lab Exercise from Week 4 and plots the points (using the letter key of each entry) onto a "box" of :

* its keys should be the integers from 0 to the maximum y coordinate found in the x-y coordinates in the keys' values in the passed dictionary;

* each key's value should be a list of strings of a single period, whose length is 1 plus the maximum x coordinate found in the x-y coordinates in the keys' values in the passed dictionary.

For example, if the dictionary is as follows:

```
my_points = { 'a':(4, 3), 'b':(1, 2), 'c':(5, 1) }
```

The resulting output to the screen should "plot" the letters **a**, **b**, and **c** at their coordinates like this:

```
3 . . . . a .
2 . b . . . .
1 . . . . . c
0 . . . . . .
  0 1 2 3 4 5
```

This can be done in phases and may involve writing a few auxiliary functions. Here's how I recommend doing it:

* First you can use the **max_coordinate_value** function written during the Lab Exercise in Week 4 to get the maximum values for the X and Y coordinates.

* Then create a list of lists of the appropriate size containing only `'.'` as each element, for example:

```
{[ ['.', '.', '.', '.', '.', '.'],
   ['.', '.', '.', '.', '.', '.'],
   ['.', '.', '.', '.', '.', '.'],
   ['.', '.', '.', '.', '.', '.'] ]
```

* From there, you can fill in the letters, replacing the `'.'` values at the appropriate coordinates:

```
{[ ['.', '.', '.', '.', 'a', '.'],
   ['.', 'b', '.', '.', '.', '.'],
   ['.', '.', '.', '.', '.', 'c'],
   ['.', '.', '.', '.', '.', '.']]}
```

* Then print the dictionary values out to the screen.

This is only a suggestion for how to approach this – there are several ways to tackle this problem. Good luck!