Python Assignment 2

Due: Sunday 5/6/16 at 23:59.
Submit using the online submission system

Assignment aim and objectives

The aim here is to practice writing programs to solve problems using lists, dictionaries, string manipulation and file input.
The objectives of the assignment are to: (1) write correct code applying what was learnt in lectures; (2) gain experience at using a unit testing framework for Python (doctest this week); and, (3) write readable Python code that follows commonly used stylistic rules.
You need to implement the following:

- **print_frequency** reads a string (containing multiple words) and returns a table of the letters of the alphabet in alphabetical order which occur in the string together with the number of times each letter occurs. Do not output a count if the letter does not occur. Case should be ignored. You may want to use the string method isalpha(), use a dictionary data structure to create a general solution (i.e. do not hardcode in the alphabet).
- **verbing** reads a string and for each word -- if the length is at least 3, adds 'ing' to the end, otherwise add 'ly'. You should use iteration, split, concatenation and join to implement this.
- **add_vectors** that takes two lists of numbers of the same length, and returns a new list containing the sums of the corresponding elements of each vector. You must use enumerate when implementing this.
- **add_vectors_file** given a file in the same directory as the script, read the file, apply error checking, extract two columns (‘pay’ and ‘bonus’) from the file, and use the add_vectors function to sum the two extracted columns of the comma separated values (CSV) file. You should make use of the split() function to separate lines based on commas.

Unlike the previous assignment please use the return keyword (multiple returns are allowable).

Correctness is determined by running against regression tests and by the tutors reading your code.

Please use the test cases to guide the error messages you return.

The files including test cases

The required functions, doctest files, and example input files are provided as a zip file on the website.

https://ecs.victoria.ac.nz/Courses/NWEN241_2016T1/Assignments

Python_assignment2.zip

Within the archive you should find:

- assignment2.py : this includes the function prototypes that you must complete.
- add_vectors.txt, print_frequency.txt, verbify.txt and add_vectors_file.txt : doctest tests for the functions.

Run these by typing (not copying from this pdf): 'python3 -m doctest -v add_vectors.txt'

Completion and marking

The basic idea is to implement each function and test it against the regression tests. Submit your code once you are satisfied that you have correctly implemented the functions.
Do not change the file names or function names. Our automated testing will fail and you might not get any marks.
The majority of your grade (90%) is based upon correct implementation as evidenced by passing the regression tests that we have provided and passing our own secret regression tests. This grade might be modified depending upon whether you follow the guidance as to permissible language features (for example, do not use the re module).
The remaining minority of your grade (10%) is based upon adherence to the style guide and general readability of the code. See the notes below on the style rules to be enforced in this assignment.
Roughly speaking, completing **add_vectors** and **print_frequency** make up about 70% of the marks, **verbing** another 20% and **add_vectors_file** makes up the final 10%.

Style rules

When writing your code you should follow some fairly standard stylistic rules for Python

1. Naming conventions Follow these formats: module_name, package_name, ClassName, method_name, ExceptionName, function_name, GLOBAL_CONSTANT_NAME, global_var_name, instance_var_name, function_parameter_name, local_var_name.
2. Use comments to document anything non-obvious You can use the # character for single line comments or use multiline comments that are bracketed by """ (look at the multiple line comments for the functions as a guide).
3. Don't put commands on same line separated by semicolons I do this in the workbooks for brevity but it is considered bad for programs that are more than a line or two long.
4. Parentheses Use these sparingly.
5. Whitespace No whitespace inside parentheses, brackets or braces.
6. Classes If a class inherits from no other base classes, explicitly inherit from object.

7. Strings Use the format method for formatting strings, even when the parameters are all strings. When concatenating strings rather than printing use + or concat operators.
8. Imports formatting Imports should be on one line.