Assignment 2: Advanced data structures, functions, files and string manipulation

Due Wednesday 1/4 @ Midnight

Submit using the online submission system. You only need to submit **`assignment2.py`**.

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:

- **`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.
- **`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()`. **You could solve this using string and count operations with a fixed list of things to count but you should write a more general solution based upon the use of a dictionary!**
- **`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.
- **`verbing_file`** given a file in the same directory as the script, read the film apply `verbing` and write it out to a file with the same name except it has the extension `.verbed` to the end. Use **open**, **read** and **write** to do this, you will also need some string manipulation functions are well.

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.

The Files

The archives containing the files are available here:

- assignment2.tar.gz
- assignment2.zip

Within each archive you should find:

- **`assignment2.py`** : this includes the function prototypes that you must complete.
- **`vector_add.txt`**, **`print_frequency.txt`**, **`verbify.txt`** and **`verbify_file.txt`** : doctest tests for the functions.
- **`monty-python.txt`** : a text file **created under Unix, unsure what will happen under Windows!**

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 (i.e. don't over complicate it!). See the notes below on the style rules to be enforced in this assignment.

Model solutions to this assignments will be released shortly after the assignment deadline.

This means late submissions will NOT be accepted, unless you have made prior arrangement with the course coordinator for valid reasons such as medical and family emergencies.

Style rules

When writing your code you should follow some fairly standard stylistic rules for Python (based upon the Google Python standards).

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 its 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.

This topic: Courses/NWEN241_2015T1 > Assignment2
Topic revision: 25 Mar 2015, ian