CSCI 1100 — Computer Science 1 Homework 3
Word Searching

Overview

This homework is worth **90 points** toward your overall homework grade, and is due Thursday, October 3, 2013 at 11:59:59 pm. In the zip folder associated with this homework you will find `three_double.py` and `words.txt`. The first is a somewhat modified version of the Python code from Lecture 1. The second is the text file containing all of the words of interest.

The goal of this assignment is to work with lists and strings in the context of a more sophisticated program — and to see the structure of a complete program. In order to complete this assignment, you will be modifying `three_double.py` substantially. We strongly suggest that you keep a back up copy. Since this assignment involves developing several separate functions, we also strongly suggest that you write and fully test each function before your work on the next, saving the working code to a backup file before you proceed. Finally, we also suggest that you think about making a smaller, test version of `words.txt`, and add to this words that allow you check the correctness of your program.

Fair Warning About Excess Collaboration

For HW 3 and for the remaining homework assignments this semester we will be using software that compares **all** submitted programs, looking for inappropriate similarities. This handles a wide variety of differences between programs, so that if you either (a) took someone else's program, modified it (or not), and submitted it as your own, or (b) wrote a single program with one or more colleagues and submitted modified versions separately as your own work, this software will mark these submissions as very similar. Both (a) and (b) are beyond what is acceptable in this course — they are violations of the academic integrity policy. The policy is that you are allowed to use any software that we have provided, and you are allowed to discuss the assigned homework problem, the approach to solving the problem, and even the Python techniques that must be applied, doing so both with students from the class and others outside the course. On the other hand, you are not allowed to discuss the details of the code with other students, you are not allowed to show your code to anyone else in the class, you are not allowed to obtain detailed help from anyone outside the class other than the professor, a TA, a programming mentor, or designated (ALAC, dorm, etc) tutors, and you are not allowed to put your code in a publically-accessible location where someone else could get access to it.

Students whose programs are quite similar as discovered by this software tool will be given a chance to explain the similarities to the instructor and to the TAs. Students who have subsequently been found to have violated the academic integrity policy will be punished. The standard penalty for submitting work that is not your own, or for providing code that another student submits (perhaps after modification) will be

- 0 on the homework, and
- an additional overall 5% reduction on the semester grade.

Penalized students will also be prevented from dropping the course. More severe violations, such stealing someone else's code, will lead to an automatic F in the course. A student caught in a second academic integrity violation will receive an automatic F.

By submitting your homework you are asserting that you both (a) understand the academic integrity policy and (b) have not violated it.

Finally, please note that this policy is in place for the small percentage of problems that will arise in this course. Students who follow the rules outlined above and use common sense in doing so will not have any trouble with academic integrity.

Getting Started

Start by looking through the code in `three_double.py` carefully and making sure you understand it. There are just a few things here that you should be aware of:

- In the `three_double` function itself, you will see the loop that starts with

  ```
  for i in range(0, len(s)-5):
  ```

  The novel code here is the function `range`, which generates a list that starts with the first value, 0, and ends just before the last value, `len(s)-5`. For example, with `s == 'successfully'`, the function

  ```
  range(0, len(s)-5)
  ```

  creates the list

  ```
  [0, 1, 2, 3, 4, 5, 6]
  ```

  so that the loop effectively becomes

  ```
  for i in [0, 1, 2, 3, 4, 5, 6]:
  ```

  Using the Python interpreter (without running `three_double.py`), play around with `range` until you are comfortable with what it is doing.

- The operator `in` is used in two different ways in Python. The first is in a `for` as we have already learned and just practiced. The second is as a logical test such as

  ```
  >>> w = "successful"
  >>> 'c' in w
  True
  ```

- In the main part of the code you will see where the program opens file `words.txt`, reads each string, strips off the extra whitespace, and appends it to the words list. The opening and reading of the file are quite similar to what we did in Lab 5. You will not have to change this code at all. When running the code, the crucial thing is to be certain your .py file and the text file are in the same folder.

- Indexing works on strings just like it does on lists, except that you can not change the contents of a string. Thus, for the string

  ```
  w = "successfully"
  ```

  we have `w[0] == 's'`, `w[1] == 'u'`, etc.

Functionality to Add

For each of the following, write a function (or two) to compute and no output the result(s). Each function should be called in turn from the the main code and output its result before returning. As mentioned above, you should solve each problem before moving onto the next.

1. The number of words with at least two consective double letters.

2. The fraction of the words that are both longer than three letters and start and end with the same letter. The output fraction should be accurate to three decimal places, as shown in the example below.

3. The fraction of words containing the letter `'e'`, accurate to three decimal places, and the fraction of words containing the letter `'z'`, again accurate to three decimal places.

4. The letters of the alphabet, in alphabetical order, that appear in at least 25% of the words. Their fractions should be output as well (accurate to three decimal places). As a hint to help you think about how to test all of the letters, the code

   ```
   for letter in 'abcde':
       print letter
   ```

   outputs

   ```
   a
   b
   c
   d
   e
   ```

5. The letters of the alphabet, again in alphabetical order, that appear in less than 10% of the words.

6. The number of words that contain none of the five vowels. Of these, find and output the single word that contains no vowels and three `y`'s.

7. The average, maximum and minimum word lengths. There are a number of ways to do this. We suggest that you create a list containing the length of each word (see the use of `append` when reading in the words from the file), and then apply some of the list functions we have learned. Make the output of the average accurate to two decimal places.

8. As an extra challenge (for only the last five points of this assignment), write a function to find the most common word length. Like the previous problem, you can approach this in many ways. We will leave it to you to figure out and will not be offering much help. Output both the most common length and the number of words of that length.

Output and Testing

Put a line containing 20 `'-'` between the output for each function. For example, here is the output for the first few parts:

```
--------------------
1. At least two consecutive doubles: 163
--------------------
2. Fraction of words that are longer than three and start and end with the same letter: 0.065
--------------------
3. Fraction containing 'e': 0.669
   Fraction containing 'z': 0.030
--------------------
4. Letters appearing in at least 25% of the words:
   'a': 0.497
```

We will not be posting our answers.

If we aren't giving you the answers, how should you know that your answer is correct? You should start by creating small test files for each part, as described above. You will be able to know the correct answers for each of these (although sometimes you do make mistakes in test files that the code will help you fix!). You can also insert additional print statements into each function as you write it and test it to be sure it is making the right decisions; remove these statements when you are done. You are welcome to talk to each other to compare outputs. Finally, postings on Piazza are welcome, although we will not confirm whether or not your answers are correct. We will be talking more about testing later in the semester.

Code structure will become an increasing part of the grade for this assignment. You must put the solution to each part into a function, with all the variables needed by the function passed into it. Place all of the functions together at the top of you file. The code to open and input the file and to call the function should be placed at the end of the file. Use `three_double.py` as your template.