Assessment 01 - Spellchecker

Introduction

The aim of this assessment is to assess your understanding on the material delivered so far in the COMP16321 Introduction to Programming 1 course. You are to create a program which checks the spelling of words in a sentence or file which meets the objectives given below. The program should compile using Python 3.8. You must create the program in a singular .py file. Any additional .py files will not be marked.

The objectives in this assessment have been designed to offer varying levels of complexity (some things are easier to implement than others). If any aspect of this assessment is ambiguous then please post a detailed message in the forum on Blackboard for clarification.

Overview

You are to create a terminal based python program which can do the following:

1. Read in a sentence from the user using the terminal
2. Read in a .txt file saved in the same directory
3. Check the input provided for spelling errors and mark any spelling errors according to the information provided in the objectives
4. Provide suggestions of the correct spelling for any words have been identified as incorrectly spelt

You are also to use your given GitLab for all version control and must apply appropriate levels of commits to the project. The final commit you make for this assessment must be tagged as "Coursework_01" as we will search for the tag in your repositories and assess you work there. You should use the repository that has been created for you (i.e. *programming-practicals_[username]*, within this you should then create a new directory called *coursework_01*. This is where we will check your git history for the coursework when marking.

Guidance on this can be found here:
https://wiki.cs.manchester.ac.uk/index.php/UGHandbook20:Coursework

Objectives

The objectives of the program are given below. You may not be able to implement all the objectives depending on your level of experience, but you should still submit what you have done as you will receive partial credit (providing the program compiles and runs).

The objectives of the program are:

1. The program presents a text based menu to the user, the user is asked to enter 1 if they wish to spell check a sentence, 2 if they wish to spell check a file, 0 to quit the program. The users choice should be validated (i.e. anything entered that is not 0, 1 or 2 would be an invalid choice and the program should inform the user of the invalid entry and give the user the opportunity to enter the choice again).
   - (a) If the user enters 1, the user is prompted to enter a sentence
   - (b) If the user enters 2, the user is prompted for the filename (the filename entered should be validated, i.e. if the file does not exist the user will be informed and given the opportunity to enter the filename again or return to the initial menu).
2. The input (sentence or file) should be split into a list of words (you can assume each new word will always be separated by a white space).
3. The program will read in a list of English words from a text file (`EnglishWords.txt` provided on Blackboard).
4. Each word provided by the input will be checked to see if it is the list of English words.
5. The program will output each word provided by the input (sentence or file)
6. When an incorrectly spelt word is encountered the user should be given the opportunity to 1. ignore, 2. mark, 3. add to dictionary, 4. suggest likely correct spelling.
   - (a) If the user enters 1, the word is just ignored (but still counted as incorrectly spelt)
   - (b) If the user enters 2, the word is marked by including a question mark at the beginning and end of the word (the word is counted as incorrectly spelt).
   - (c) If the user enters 3, the word is added to the dictionary (the word is counted as being correctly spelt).
   - (d) If the user enters 4, the user is presented with a possible suggestion. The user can then either, accept the suggestion which will change the word (and count as correct), or reject the suggested word (and count as incorrect).
7. The program displays summary statistics such as:
   - (a) the total number of words
   - (b) the number of words spelt correctly
   - (c) the number of incorrectly spelt words
   - (d) the number of words added to the dictionary
   - (e) the number of words changed by the user accepting the suggested word

(f) the time and date the input was spellchecked

(g) The amount of time elapsed (the time it took) to spellcheck the input.

8. A new file is to created that contains the original input. The user should be prompted for the new filename. This file should also include the markers (the question marks) for incorrectly spelt words, except those that were ignored. It should also show the summary statistics, these statistics should come at the top of the file before the input text.

9. After the input has been spellchecked the user is given the option to return to the main menu or quit.

Additional Guidance and Help

The following section details additional guidance to help you with your implementation

10. The program should deal with mixed case words (i.e. those that start with an uppercase character). The program should filter out non alpha characters such as commas and full stops.

    Examples:

    “Python’s” would become pythons

    “interpreted,” would become “interpreted”

    “in1991” would become “in”

    “1991” would become “”

    “whitespace.” would become “whitespace”

    N.B. For simplicity you might just want to loop through each character of the word and remove anything that is not an alpha character. You would repeat the process for each word. We understand that grammatically changing a word such as Python’s to pythons is not the best approach, but, for your first coursework we wanted to avoid this extra layer of complexity at this stage.

11. When a word is encountered that is not in the dictionary, then the program should suggest the likely word that the user might have meant.

    Hint:
    ```
    score1 = SequenceMatcher(None, "apple", "appel").ratio()
    score2 = SequenceMatcher(None, "apple", "mango").ratio()

    print(score1)
    print(score2)
    ```

Output:
```
0.8
0.0
```

To use SequenceMatcher you need to import it at the top of your program.

```
from difflib import SequenceMatcher
```

12. Make use of the Unicode character table to format the menus with borders, marks are awarded for a visually pleasing menu.

13. Ensure that input is validated and that the program does not crash if the user enters an inappropriate response.

Preparation and Submission

Your single .py file must be submitted to Blackboard in the appropriate place,
16321-Cwk1-S-Python Basics

The marker will run your program though the terminal and test its functionality. Your code will also be checked for quality and appropriate use of programming principles (e.g. comments, white space, etc.).

Deadline:
18:00 on Friday the 6th November

Assessment Type:
This activity is subject to summative assessments, therefore your submission will be marked and you will receive the associated feedback. The marks you obtain (see below) count for up to 10% of your overall mark for this course unit.

Marks:
Marks are awarded on three main categories:

1. The running & testing of the program
2. Code quality
3. Git usage

If the marker finds that the file submitted is unreadable and/or does not compile then you will receive marks for the code quality & git sections only. Please be aware that we will not install external packages to make your code compile. We should be able to run your code with a basic python setup.

Please note: Your work will not be re-marked because you disagree with the mark you were awarded. The markers all follow the same rubric which has been designed to be as fair and comprehensive as possible. Any queries with your marking can be raised with either Gareth or Stewart within a week of the marks being released, any queries made after this time will not be considered.

End of Assessment