Introduction

The aim of this take-home exam is to practice on file operations and dictionaries in Python. The use of file operations is due to the nature of the problem; that's why you cannot finish this take-home exam without using file operations. You may implement the solution without using a dictionary, but that would be less efficient and much harder for you to code. Besides, beware that if we use a larger file in auto grading, your program may fail if you use lists instead of a dictionary.

Description

In this take-home exam, you will implement a Python program that suggests correct spellings for the string input entered by the user, and it should keep asking for an input until the special input '`exit()`' is entered. Together with this description document, you are also given a ***sample*** input file. Your program will use this file to create a corpus of words by reading and counting the words in the given file. Then, your program will take a string input from the user and check if each word in this string input is spelled correctly. If there are any misspelled words in the string input, then your program will perform a spelling correction operation for each word that are misspelled.
To perform this operation, your program will use the corpus you created and a distance function. In this context, you will be given a pre-implemented function, which takes two string values as its only parameters, calculates the dissimilarity (distance or difference) of these two given words (or sequences), and then returns the value accordingly. This functionality is called Levenshtein distance, and you can check out this link for more information about Levenshtein distance. You can use this function in the spelling correction operation.
If every word within the given string input is spelled correctly, then your program will display a success message. Otherwise, i.e. if any of the words have a mistake in spelling, then your program will display a correction message. The details of the inputs, process and outputs are provided with further details in the below sections.

Input File

You will be given only one input file sample which contains a part from a book, namely "*The Little Prince*". The name of the input file is `book.txt`.

You can assume that there will be no empty lines in the file. However, you cannot make any assumptions on the number of lines of this file. Besides, the lines might end or start with space, tab or other whitespace characters. Keep this in mind while you're preprocessing the content of the file.

The only punctuation marks existing in the file are going to be dots ('.'), commas (','), exclamation marks ('!'), question marks ('?'), colons (':'), double quotations ('"'), and semicolons (';').

For a sample input file, you may check out the `book.txt` provided with this assignment.

Please note that *we will use another file while grading* your take-home exams. The content of the file will most probably change for grading purposes, but the name of the file will remain the same; that is, it will always be `book.txt`.

User Inputs

Your program will process the input file and ask for a string input to be spell checked. The program should keep asking this string input until `'exit()'` is given as input. For simplification, you can assume that no punctuation marks will be used in the string input, and this input will always be given in lowercase. Thus, you do not need to perform any input check.

Preprocessing the File Content and Creating the Corpus

First, and a very important step of this take home exam, is to read the file and store the words with their occurrence counts. To be able to perform this step, you need to apply some preprocessing on the content of the file. By preprocessing, we mean cleaning up the content of the file that might alter the words, which will be stored.

For preprocessing, your program should put the words in canonical form; that is, your program should remove all the punctuation marks and make sure that all the words have the same capitalization. In this regard, your program should remove the punctuation marks (`'.'`, `','`, `'!'`, `'?'`, `':'`, `'"'`, `';'`) from the beginning and from the end of each word. After this step, your program should then convert the text to lowercase, because the user input will always be given in lowercase.

You may assume that, after a particular punctuation mark and before a new word, there will always be at least one whitespace character.

The reason you need to collect the words in the given file by parsing the content is; to perform spelling correction; that is, you need a reference for the correct spellings of the words that your program will check. There might be some words inside your dictionary that are not meaningful, because there may be a few words in the book that are used in daily life. However, this should not scare you because the words that have higher occurrence values will be more important while performing spelling correction.

We highly recommend the use of dictionaries.

We are very aware that this take-home exam can also be done by:

- Using multiple lists instead of a dictionary, which makes the implementation much more complicated.

We would like to remind that dictionaries will be an important part of the exam. Doing this take-home exam by avoiding dictionary use will not help you much in this direction.

Additionally, GradeChecker includes a timeout method which terminates the execution if it takes longer than a predefined threshold. Using lists (multiple lists instead of a single dictionary) may result in your code being not executed entirely by GradeChecker, which would yield a grade of 0. Meanwhile, you can of course use the list data structure in your program for purposes other than storing the dataset(s).

Although it’s not mandatory, we highly recommend the use of functions.

Levenshtein Distance Function

You are given a ready to use function, `levenshtein(str1, str2)`, which takes two string parameters. These two parameters are the words to be compared. This function calculates how many edit operations needed to transfer one string to another. These operations can be insertion, deletion or substitution.

As an example, let's consider that you are comparing the strings "Ankra" and "Ankara". The distance between these two strings (or words) will be equal to 1, because only 1 insertion operation will be required and the location for this modification is between the 2nd and 3rd indices.

On the other hand, as another example, if the strings were "Annkra" and "Ankara", then the distance between these strings would have been equal to 2, since we would need 1 deletion and 1 insertion.

In general, the edge cases for the Levenshtein Distance will have the following values, given the words (i.e. strings), conditions and/or constraints:

- It is zero, *if and only if* the strings are equal.
- It is *at least* the difference of the sizes of the two strings.
- It is *at most* the length of the longer string.

As mentioned before, this function is already shared with you together with this description document, and thus, you do not need to implement it by yourself.

For further information on how it works, you can watch this video.

Spelling Check and Correction Operation

- Your program needs to take input from the user, which will be in a string format.

- Then, each word in the input will be searched in the word dictionary created before.
    - If the word is found in the corpus (dictionary) that you created using the content of the input file, then your program should accept that the word is spelled correctly.
    - If the word cannot be found in the corpus, then your program should calculate the Levenshtein distance between the words in the corpus and the current word in the input string.
    - As explained before, Levenshtein distance indicates how different the words are.
    - By calculating the Levenshtein distance between the misspelled input and words in the corpus created, the main goal is to find the word most similar to the misspelled word.

- After that, your program needs to find the word in the corpus which has a minimum Levenshtein distance between the current word in the input string.
    - Minimum distance means maximum similarity according to Levenshtein distance.

- If there are several words with the same Levenshtein distance, in such a case, your program needs to select the word that has the maximum number of occurrences.
    - There may be more than one word with the same minimum Levenshtein distance. In such a case, the program should choose the most occurred word between these words. Because your program needs to select the most probable word.
    - In case that there are multiple words with the same minimum Levenshtein distance and with the same occurrence frequency, then your program should display the word that comes first in ascending alphabetical order.
        - Sample Run 6 is an example for this particular case, where "`thinfe`" has the same Levenshtein distance with "`think`" and "`thing`", and the number of occurrences of these two words are both 14. When you put these two words in ascending alphabetical order, "`thing`" would come before "`think`", and that's why we only see "`thing`" being suggested for "`thinfe`".

In short, if the program encounters a word in the given input which cannot be found in the constructed corpus, then the most probable word will be the most similar and the most used word.

Outputs

If every word in the string input exists in the corpus, then your program needs to print "`String you entered is correctly written!`". On the other hand, if there

exists a word that cannot be found in the corpus, then your program needs to print the edited version of the input after applying the operations above.
You may check the Sample Run given below for further information.

Important Remarks on Working with Files

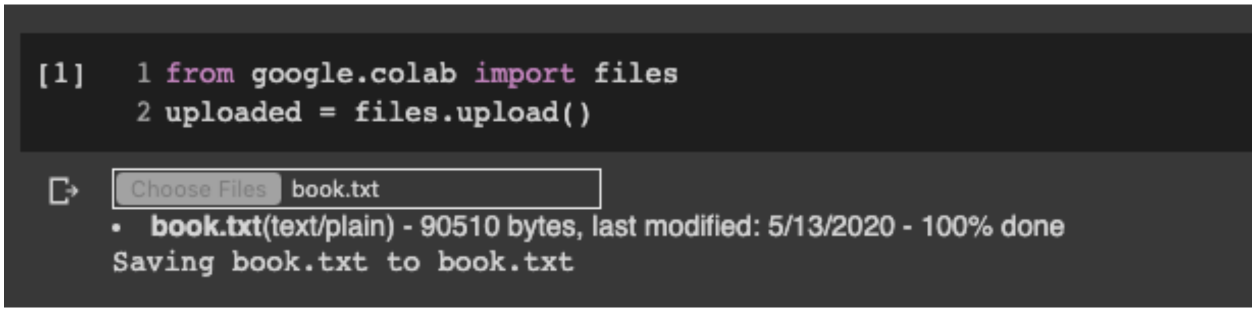
When it comes to working with files, there are some differences in the way we use Google Colab and GradeChecker.

While working in Google Colab, your code does not run on your computer, hence there is no direct access to the files you have on your computer. However, you can upload your text file(s) to Google Colab by running this two line snippet on a separate code cell:

```
from google.colab import files
uploaded                      =                      files.upload()
```

Every time you run this code, you will be prompted to upload a file from your computer. Please note that this snippet may not work for certain browsers, so Chrome/Chromium is recommended. Please also be reminded that you need to repeat the upload process every time you reopen your Colab file or you reconnect to the environment.

If you are going to try your code on GradeChecker, you should remove this code cell (or make them as comment lines as shown below) before you download the .py version of your code, as this snippet only works on Google Colab and it will cause error on GradeChecker.



```
[1]  from google.colab import files
     uploaded = files.upload()
```

```
Choose Files book.txt
• book.txt(text/plain) - 90510 bytes, last modified: 5/13/2020 - 100% done
Saving book.txt to book.txt
```

Another note on GradeChecker is that GradeChecker does not have the txt file(s) related to the take-home exams already. So, while uploading your code to GradeChecker, you should upload the required text file(s) together with your .py file (*2 files in total for this take-home exam: your main `.py` file and `book.txt`*) and select your .py file as the main file to be executed. It also means that you can create your custom text files and test your program on them via GradeChecker, if you wish.

In short, if you use files.upload() method to upload the txt file to Google Colab, do not forget to erase the related lines before you upload your .py to GradeChecker and also to SUCourse.

Sample Runs

Below, we provide some sample runs of the program that you will develop. The *italic* and **bold** phrases are inputs taken from the user. You have to display the required information in the same order and with the same words and characters as below.

Sample Run 1

```
Please enter the word/words you want to check: exit()
See you next time...
```

Sample Run 2

```
Please enter the word/words you want to check: i saw an alephante
whili walkingg to my homee
Did you mean: i saw an elephant while walking to my home
Please enter the word/words you want to check: i saw an elephant while
walking to my home
String you entered is correctly written!
Please enter the word/words you want to check: exit()
See you next time...
```

Sample Run 3

```
Please enter the word/words you want to check: i have a turkisgh
freind
Did you mean: i have a turkish friend
Please enter the word/words you want to check: exit()
See you next time…
```

Sample Run 4

```
Please enter the word/words you want to check: i have a turkish friend
String you entered is correctly written!
Please enter the word/words you want to check: exit()
See you next time...
```

Sample Run 5

```
Please enter the word/words you want to check: i ded not sleeep yet
Did you mean: i did not sleep yet
Please enter the word/words you want to check: exit()
See you next time...
```

Sample Run 6

```
Please enter the word/words you want to check: ii thinfe alot
Did you mean: i thing not
Please enter the word/words you want to check: exit()
See you next time...
```

Sample Run 7

```
Please enter the word/words you want to check: consekuence
Did you mean: consequence
Please enter the word/words you want to check: understanf
Did you mean: understand
Please enter the word/words you want to check: happy
String you entered is correctly written!
Please enter the word/words you want to check: can have serious
consecuence
Did you mean: can have serious consequence
Please enter the word/words you want to check: can have serious
consequence
String you entered is correctly written!
Please enter the word/words you want to check: exit()
See you next time…
```

How to get help?

You can use GradeChecker (http://sky.sabanciuniv.edu:8080/GradeChecker/) to check your expected grade. Just a reminder, you will see a character ¶ which refers to a newline in your expected output.

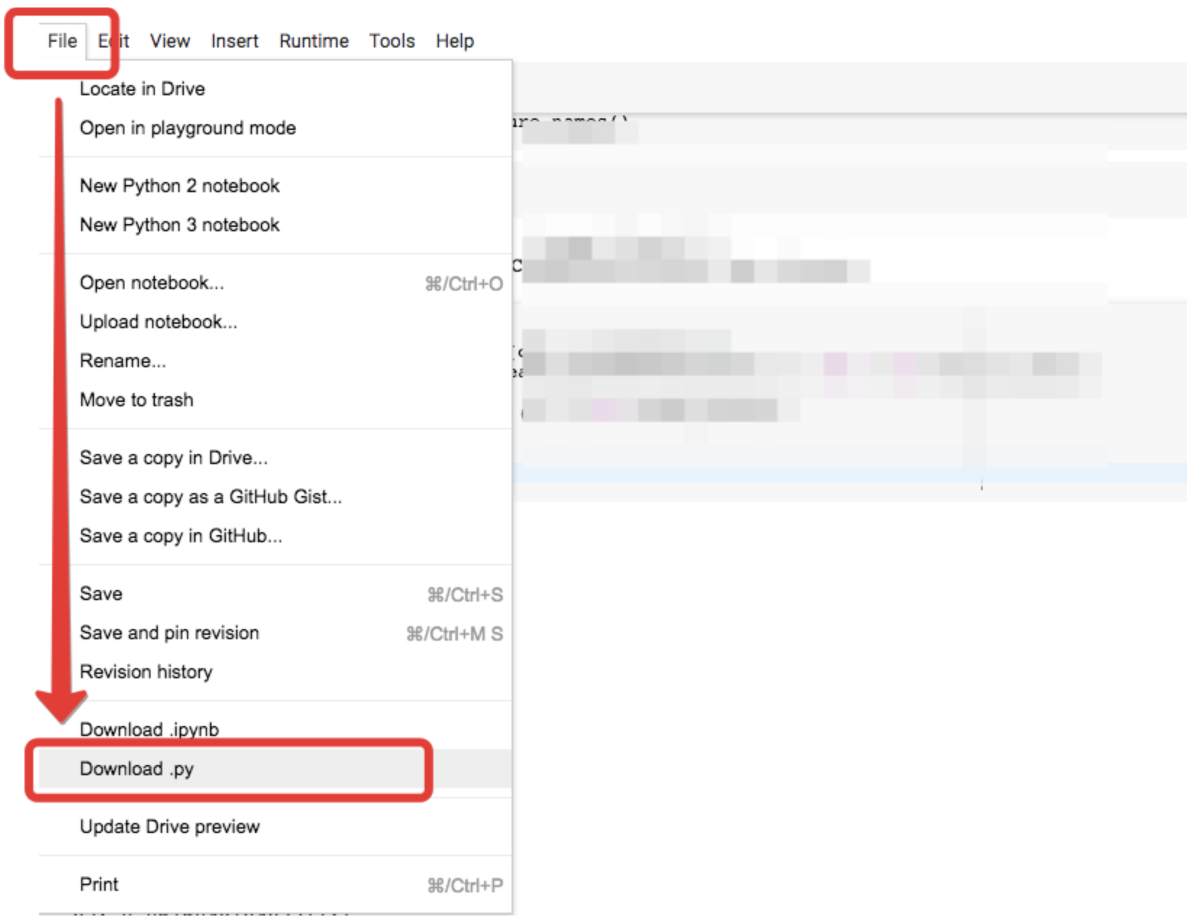
What and where to submit?

You should prepare (or at least test) your program using Python 3.x.x. We will use Python 3.x.x while testing your take-home exam.

It'd be a good idea to write your name and lastname in the program (as a comment line of course). Do not use any Turkish characters anywhere in your code (not even in comment parts). If your name and last name is "İnanç Arın", and if you want to write it as comment; then you must type it as follows:
# Inanc Arin

Submission guidelines are below. Since the grading process will be automatic, students are expected to strictly follow these guidelines. If you do not follow these guidelines, your grade will be 0.

- Download your code as *py* file with "File" -> "*Download .py*" as below:



- Name your *py* file that contains your program as follows:

  "**username_the5.py**"

  For example: if your SuCourse username is "**duygukaltop**", then the name of the *py* file should be: **duygukaltop_the5*.py*** (please only use lowercase letters).

- Please make sure that this file is the latest version of your take-home exam program.

- Submit your work **through SUCourse only**! You can use the GradeChecker only to see if your program can produce the correct outputs both in the correct order and in the correct format. It will not be considered as the official submission. You must submit your work to SUCourse.

- If you would like to resubmit your work, you should first remove the existing file(s). This step is very important. If you do not delete the old file(s), we will receive both files and the old one may be graded.

General Take-Home Exam Rules

- Successful submission is one of the requirements of the take-home exam. If, for some reason, you cannot successfully submit your take-home exam and we cannot grade it, your grade will be 0.
- There is NO late submission. You need to submit your take-home exam before the deadline. Please be careful that SUCourse time and your computer time may have 1-2 minutes differences. You need to take this time difference into consideration.
- Do NOT submit your take-home exam via email or in hardcopy! SUCourse is the only way that you can submit your take-home exam.
- If your code does not work because of a syntax error, then we cannot grade it; and thus, your grade will be 0.
- Please do submit your **own** work only. It is really easy to find out "similar" programs!
- Plagiarism will not be tolerated. Please check our plagiarism policy given in the syllabus of the course.

Good Luck!
Elif Pınar Ön & Ethem Tunal Hamzaoğlu & IF100 Instructors