Python for Informatics

Assignment 4

"Files, Lists, and Split"

Description:

1. This description explains how to develop your assignment as one program. Please understand that you are developing only one program, not multiple programs with each step. ***As you complete this assignment, please do not use single character variable names. Instead, use meaningful names (names that tell the reader what is represented and why).*** Write a program that prompts the user for a filename. Open the file, and read it one line at a time. Do not use hard-coded file paths in your program, and do not require the user to specify the path. Instead, use the %cd magic command in your IPython console (magic commands can only be used in an IPython console; you can't use them in a .py file) to position yourself in the code3 directory. Now run your program, ask the user for the filename, and then open and read it. For each line split the line into a ***list*** of words called ***file_word_list***. For each word in the current ***file_word_list***, convert the word to lowercase and look to see if it is in a list called ***script_list*** (a list that is initially empty). If the word is not in ***script_list***, add it to the ***script_list***. Sort the ***script_list*** alphabetically.
2. Within the same program define a function called ***freq_count()***. This function accepts a ***str*** and a ***list*** of words as arguments. [Note: ***str*** is a Python object type. Hence, you should never use ***str*** as an identifier. ***str*** already refers to the string class object, and by using the name ***str*** to refer to something else, you lose the ability to reference the string class object ***str***! Instead, you can use ***str*** as a compositional element in an identifier name, such as ***search_str***.] I would name my arguments, ***search_str*** and ***word_list***, respectively. But that's just because I like my variable names to tell me what they represent, and why I am using them. Note that are careful to avoid confusion by using two different names, ***file_word_list*** and ***word_list***. These variables are created and used in different scopes, and we don't want to delude ourselves into thinking that they are the same variable—so we give them different names. Your ***freq_count()*** function traverses the ***list*** of words (***word_list***) and searches each word and counts the occurrences of the substring (***search_str***) within each word. **To perform this substring search, you must use the *find(…)* method, and you should use the *find(…)* method is a manner that allows you to specify the position at which the substring search begins within the string.** For this assignment, you are not allowed to use the string count() method or the re.findall(…) method to obtain your counts. Instead, you must use the string ***find(…)*** method. Hint: try specifying the starting position of your *find* operation by passing the starting position in as a variable—i.e., use two arguments when you call the ***find(…)*** method. **The freq_count() function should print each word *along with the number of substring occurrences found within the associated word.***

Please note that you are ***not*** counting the number of occurrences in the list or in the file! Your counts are counts within the words—each unique and separate word, that is, **not all of the words!**

3. Modify this same program so that it accepts the ***filename*** and the substring ***str*** as input from the user. After reading the file to build and sort the ***script_list***, pass the ***script_list*** into a call of the ***freq_count()*** function. ***Test your program with the romeo.txt file that comes as a text file resource with our textbook. You'll find this file in the "code3" folder that you were asked to download onto your system.***

Note: With step 2, you are printing each word, not just words with non-zero occurrences of the substring ***str***. As examples of what the substring search results would be, given the word "there" and a substring ***str*** of "th", the result would be 1, and the print result for that one word would be "there 1"; given the same word "there" and a substring ***str*** of "e", the result would be 2, and the print result for that one word would be "there 2"; and finally, given the same word "there" and a substring ***str*** of "z", the result would be 0, and the print result for that one word would be "there 0". Here is an example run:

```
Enter filename: romeo.txt

Enter substring: o
already 0
and 0
arise 0
breaks 0
but 0
east 0
envious 1
fair 0
grief 0
is 0
it 0
juliet 0
kill 0
light 0
moon 2
pale 0
sick 0
soft 1
sun 0
the 0
```

through 1
what 0
who 1
window 1
with 0
yonder 1

The romeo.txt file is provided within the "code3" directory that you should have downloaded from our textbook website (please see and read the syllabus). If you try to run your program, and it can't find your downloaded romeo.txt file, that usually means that you did not navigate (use the %cd magic command) to the code3 directory before running your program. The %pwd magic command will tell you what your current working directory is, and the %ls magic command will list the contents of the current working directory. To ensure that your magic commands to work in your Spyder IPython console, you need to be sure that your Spyder configuration is correct. In order for your magic command to "stick", do this: Tools --> Preferences --> Current working directory... click the radio button "The current working directory".

By %cd-ing into the "code3" directory, you can just use the simple filename (no directory path) for your data file, and it just works. Here's the syntax:

%cd <pathname>/code3

...where ***<pathname>*** is the path location of your "code3" directory.

Only use '/' as your path separator. Using '\' is highly problematic—you would need to escape it by doing this: \\. And that will only work on Windows systems. '/' is guaranteed to work on all platforms.

The use of magic commands is a best practice when doing informatics or exploratory data analytics. Keep in mind, however, that magic commands will only work in your IPython console. You cannot put them into your .py files and expect them to work—they are IPython console commands only.

Please do not use *os.chdir(...)* in your .py files! That creates an unnecessary dependency upon your local file system. It is exactly this kind of local file system dependency that the use of magic commands avoids.

Deliverable:

Two files as attachments at our course shell assignment page. The first file should be a Python .py file with the specified functionality. The second file should be a screenshot image file (.png or .jpg) demonstrating the correct execution of your program with "romeo.txt" entered by the user as the filename, and the substring 'o' entered by the user as the substring value that is being counted. Please ensure that your full name is specified as a Python comment at the top of your ***.py*** files.

Submission Deadline:

Please see the course schedule in our syllabus for all assignment submission deadlines.

Peerwise Reminder:

In addition to earning 15 points (a full 1.5 letter grade value), creating your 32 required Peerwise questions and answering your 32 questions from other students is a great way of consolidating and integrating your understanding of Python. It will also help you communicate your understanding more effectively to your colleagues. Try it, you'll see! Answering 32 questions created by fellow students will earn you 7.5 points, and will likely help you score better on the quizzes. They are also fun!