Lesson 1: Week 3

Created: 2024-05-27T01:25:46.425696+10:00

Exercise 1: Word frequency

In the workspace on the right is a file called "text.txt" which contains a piece of text (it's the opening passage of Charles Darwin's On the Origin of Species).Write a program that reads the text in the file and lists each unique word, along with its frequency (i.e., how many times it occurs).Example: (These numbers might not be correct - they're just a guide to the sort of output you should generate.)when: 3 we: 3 look: 1 to: 9 the: 11 ... etc.Your program should work not only for this file but also for variations. You may assume the following about the text in a file:First. Words are separated by whitespace. In the sample text given all of the whitespace is just a single space. But sometimes it might be multiple spaces, or tabs, or line breaks, etc. Make sure your program can handle all of these different kinds of whitespace. You will find the string method split very helpful. Second. The text might contain punctuation marks (as the sample text does). Make sure that you don't include punctuation marks in words. You could remove them from the text altogether. It will be good enough if your program can handle the following punctuation marks:. ? ! , ; : ( ) [ ] { } " Don't worry about dashes and single quote marks:- ' These are tricky, because sometimes they are used as parts of words (e.g., hyphenated words, such as "sub-variety", or contractions, such as "aren't") and sometimes they are used not as part of words (e.g., as dashes or quote marks). If you feel like a challenge then you could get your program to deal with them correctly, but you're not expected to.Third. Words might occur both with a capital first letter and without a capital first letter (e.g., the sample text contains both "When" and "when"). You should consider these to be the same word. You could make all words lowercase, turning "When" into "when". Or you could make them all upper case, turning both "When" and "when" into "WHEN". It's up to you.Checking your workNote that you can add your own files to the workspace on the right. So you could create your own text file, called, for example, "my_text.txt", put whatever text you like in that file, and then check your program by getting it to read that file instead of the sample file. You could add something like the following text, which contains things your program should be able to handle:Hello, world, hello! World: hello? GOODBYE. A nice thing about this is that you know what answers you should get: "hello" occurs three times, "world" occurs twice, and "goodbye" occurs once. So, if you're converting words to lowercase then your output should be something like this:hello: 3 world: 2 goodbye: 1Optional extraIf you're feeling up to it, get the words to appear in alphabetical order. Even better, get them to appear in order of frequency, from the most frequent down to the least frequent. Again, you're not expected to.

Exercise 2: Vowel stripper

It is sometimes said that English text is still fairly easy to read even if you remove all of the vowels. To test this, write a program that asks the user for a sentence, and then prints the sentence with all of the vowels removed.Example:What is your sentence? The quick brown fox jumped over the lazy dog Here it is without vowels: Th qck brwn fx jmpd vr th lzy dg It might work best if you only remove vowels from inside words (i.e., not from the beginnings or ends of words). Write an improved version that deals with that.

Exercise 3: Fibonacci lister

The Fibonacci numbers are a famous sequence of numbers that goes as follows:0, 1, 1, 2, 3, 5, 8, 13, ...The rule for generating the sequence is this:The sequence starts with 0, 1The next number is the sum of the previous two numbersWrite a program that prints as many Fibonacci numbers as the user would like. Get the numbers to appear on the same line, separated by commas.Example:How many Fibonacci numbers would you like? 10 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

Exercise 4: ISBN validator

An ISBN (International Standard Book Number) is a 10 character string assigned to every commercial book before 2007. Each character is a digit between 0 and 9, but the last character might also be 'X'.Write a program that asks the user for an ISBN and determines whether it is valid or not.The check for validity goes as follows:Multiply each of the first 9 digits by its position. The positions go from 1 to 9.Add up the 9 resulting products.Divide this sum by 11, and get the remainder, which is a number between 0 and 10.If the remainder is 10, the last character should be the letter 'X'. Otherwise, the last character should be the remainder (a single digit).Examples:Enter ISBN: 1503290565 1503290565 is validEnter ISBN:

938007834X 938007834X is valid Enter ISBN: 2222222224 2222222224 is invalid

Exercise 5: Max element and span in a list

Study the program max_in_list.py and run it in the Terminal window, executing "python max_in_list.py". Then complete the program span.py that prompts the user for a seed for the random number generator, and for a strictly positive number, nb_of_elements, generates a list of nb_of_elements random integers between 0 and 99, prints out the list, computes the difference between the largest and smallest values in the list without using the built-ins min() and max(), prints it out, and check that the result is correct using the built-ins; run it and check your solution with the Run and Mark (or Submit) buttons, respectively.See commands_and_expected_outputs.txt for expected outputs and sample inputs.

Exercise 6: Classifying elements in a list

The operators /, // and % are used for floating point division, integer division, and remainder, respectively.Study the program modulo_4.py and run it in the Terminal window, executing "python modulo_4.py". Then complete program intervals.py that prompts the user for a strictly positive integer, nb_of_elements, generates a list of nb_of_elements random integers between 0 and 19, prints out the list, computes the number of elements strictly less than 5, 10, 15 and 20, and prints those out; run it and check your solution with the Run and Mark (or Submit) buttons, respectively.See commands_and_expected_outputs.txt for expected outputs and sample inputs.

Exercise 7: Mean, median, and standard deviation

Complete the program mean_median_standard_deviation.py that prompts the user for a strictly positive integer, nb_of_elements, generates a list of nb_of_elements random integers between -50 and 50, prints out the list, computes the mean, the median and the standard deviation in two ways, that is, using or not the functions from the statistics module, and prints them out.To compute the median, the easiest way is to first sort the list with the built-in sort() method.See commands_and_expected_outputs.txt for expected outputs and sample inputs.

Exercise 8: Perfect numbers

A number is perfect if it is equal to the sum of its divisors, itself excluded. For instance, the divisors of 28 distinct from 28 are 1, 2, 4, 7 and 14, and 1+2+4+7+14=28, hence 28 is perfect.Insert your code into perfect.py. The program prompts the user for an integer N. If the input is incorrect then the program outputs an error message and exits. Otherwise the program outputs all perfect numbers at most equal to N. Implement a naive solution, of quadratic complexity, so it can deal with small values of N only. Execute your program and check your outputs against the expected outputs with the Run and Mark (or Submit) buttons, respectively.See Perfect number and List of perfect numbers (from Wikipedia) for more details.