Exercise 2 (Date: Jan 06, 2024)

Exercise 1: Slicing strings
Take the following Python code that stores a string:
str = 'X-DSPAM-Confidence: 0.8475'
Use **find** and string slicing to extract the portion of the string after the colon character and then use the **float** function to convert the extracted string into a floating point number.

Exercise 2: Write a program to prompt for a file name, and then read through the file and look for lines of the form:
X-DSPAM-Confidence: 0.8475
When you encounter a line that starts with "X-DSPAM-Confidence:" pull apart the line to extract the floating-point number on the line. Count these lines and then compute the total of the spam confidence values from these lines. When you reach the end of the file, print out the average spam confidence.

Enter the file name: **mbox.txt**
Average spam confidence: 0.894128046745
Enter the file name: **mbox-short.txt**
Average spam confidence: 0.750718518519
Test your file on the ***mbox.txt*** and ***mbox-short.txt*** files.
You can download the file from www.py4e.com/code3/mbox.txt
You can download the file from www.py4e.com/code3/mbox-short.txt

Exercise 3: Find all unique words in a file
Shakespeare used over 20,000 words in his works. But how would you determine that? How would you produce the list of all the words that Shakespeare used?
Would you download all his work, read it and track all unique words by hand?
Let's use Python to achieve that instead. List all unique words, sorted in alphabetical order, that are stored in a file **romeo.txt** containing a subset of Shakespeare's work.
To get started, download a copy of the file www.py4e.com/code3/romeo.txt. Create a list of unique words, which will contain the final result. Write a program to open the file romeo.txt and read it line by line. For each line, split the line into a list of words using the split function. For each word, check to see if the word is already in the list of unique words. If the word is not in the list of unique words, add it to the list. When the program completes, sort and print the list of unique words in alphabetical order.

Enter file: **romeo.txt**
['Arise', 'But', 'It', 'Juliet', 'Who', 'already', 'and', 'breaks', 'east', 'envious', 'fair', 'grief', 'is', 'kill', 'light', 'moon', 'pale', 'sick', 'soft', 'sun', 'the', 'through', 'what', 'window', 'with', 'yonder']

Exercise 4: Write a program to read through a mail log, build a histogram using a dictionary to count how many messages have come from each email address, and print the dictionary.

Enter file name: **mbox-short.txt**
{'gopal.ramasammycook@gmail.com': 1, 'louis@media.berkeley.edu': 3, 'cwen@iupui.edu': 5, 'antranig@caret.cam.ac.uk': 1, 'rjlowe@iupui.edu': 2, 'gsilver@umich.edu': 3, 'david.horwitz@uct.ac.za': 4, 'wagnermr@iupui.edu': 1, 'zqian@umich.edu': 4, 'stephen.marquard@uct.ac.za': 2, 'ray@media.berkeley.edu': 1}

Exercise 5: This program counts the distribution of the hour of the day for each of the messages. You can pull the hour from the "From" line by finding the time string and then splitting that string into parts using the colon character. Once you have accumulated the counts for each hour, print out the counts, one per line,
sorted by hour as shown below.
python timeofday.py

Enter a file name: **mbox-short.txt**
04 3

06 1
07 1
09 2
10 3
11 6
14 1
15 2
16 4
17 2
18 1
19 1

Exercise 6: Write a program to look for lines of the form:
New Revision: 39772
Extract the number from each of the lines using a regular expression and the `findall()` method. Compute the average of the numbers and print out the average as an integer.

Enter file: mbox.txt
38549
Enter file: mbox-short.txt
39756