HOMEWORK #2
Letter and word length frequencies; Zipf’s Law

Purpose

Practice dictionaries, list comprehensions, sorting and output formatting.

Obtain some data on letter frequencies in English. Letter frequency is useful for many applications, including cryptography, data compression, and language identification. Language identification is useful in machine translation (see http://translate.google.com) and rendering of international web pages by browsers.

Obtain some empirical evidence for (or against) Zipf’s Law. Zipf’s Law says that frequency is inversely proportional to rank. We will look at frequency of word lengths.

Specs

Use the ISO-8859-1 version of *The Adventures of Tom Sawyer* provided on turing at `~t90rkf1/dnl/dhw/hw2-zipf/74-2013-0731.txt`. You can tell that this file is intended to be in this character encoding by reading the first few lines of the file. You could also make a guess (many programs do) by looking at the frequencies of bit patterns. However, character files on Unix do not contain metadata to provide an unambiguous identification of encodings.

ISO 8859-1, also called Latin-1, is one of several common extensions of the ASCII encoding scheme. Remember that ASCII is a 7-bit encoding, so any 8-bit scheme must be an extension. Latin-1 contains the accented characters needed for many Western European languages. (Windows-1252, commonly called CP-1252, is an ASCII extension frequently used on Windows machines; it differs from Latin-1 only in the range X'80' through X'9F'.)

This text comes from Project Gutenberg, but please use the version provided on turing so that you can get the specific version and encoding desired.

Write a program to do the following.

Obtain the file name as a command line argument, i.e., I will run your program as follows: `./hw3.py whatever.txt > hw3out.txt`. (That also means you need the #! line at the top of your program.)

Open this file with the `ISO-8859-1` encoding, i.e., with the `encoding=iso-8859-1` keyword argument to `open`. Read this file one line at a time.

In Project Gutenberg files, you don’t have to worry about words being continued from one line to the next. Replace any double hyphen, i.e., '--', by two spaces. Replace all characters other than letters, the apostrophe ("'") and the single hyphen ('-') by a space. We are leaving the apostrophe and the single hyphen in place so that words containing them, e.g. “ting-a-ling-ling” and “don’t” will be treated as single words. Use the `replace` function of `string`, not a loop.

Convert to lower case.

1) Make a table showing the frequency of each of the letters of the alphabet in the input using a defaultdict.

Sort the table in ascending order by key. Print the table 5-across, COLUMNWISE, left-aligning alpha columns, right-aligning numeric columns, and decimal-aligning any real numbers. All columns should have headings. (“Columnwise” means that the first column will have the first few keys. If there is any ambiguity, use the common definition of columnwise, i.e., print with the smallest possible number of rows.) Since you will be printing several sets of information in columnwise format, write a function to do this.

It is possible to do this using only list comprehensions, but you may prefer to use a loop for this. When functional-style programming gets too complicated, it’s like deeply nested `if′`s – it’s a matter of taste and style as to where the cutoff is.

2) Sort the table in descending order by frequency. Print as above.

3) Make a table showing the frequency of each word length, i.e., how many 1-letter, 2-letter, etc. words are found in the input. Sort the table in ascending order by key. Print as above.

4) Sort the table in descending order of frequency. Print it again, in ordinary one-up format, showing the following columns:
- rank (first row is 1, second row is 2, etc.)
- word length
- frequency
- word length*frequency
- rank*frequency
- lg(frequency)/lg(rank), where lg = log base 2, rounded to 2 decimal places.

To avoid division by 0, leave the last column of the first row blank.

Print a total line showing the number of words represented in the table.

5) Print the following summary statistics. The purpose of the summary statistics (except for the last one) is to verify that your program is working correctly.

- Number of records read. This number should match the count from another source, e.g., *less*, *grep* or *wc*.
- Number of characters read. This number should match the count from another source, e.g., *ls* or *wc*.
- Number of characters counted. This number should match the sum of your 26 letter counts (although I didn’t ask you to print a total line for that table).
- Number of words counted. This number should match the total line of your table. In order to use it for accountability, however, you need to count the words when you read them, not just add up the counts in your table.
- Number of distinct characters in the input (i.e., number of table entries).

Test data:

You will find Unix utilities like *less*, *grep* and *wc* very useful.