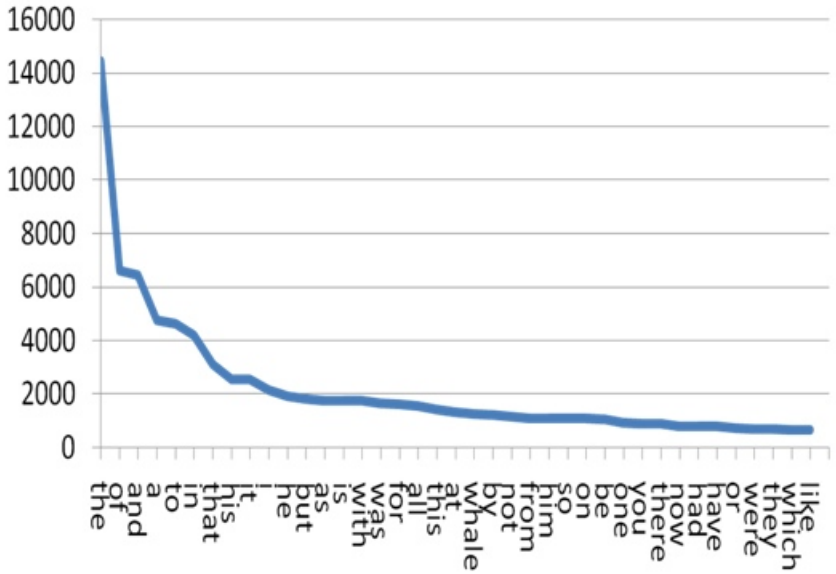
CS150 - Assignment 8
Zipf's Law
Due: Wednesday April 16, at the beginning of class

George Kingsley Zipf was a linguist who noticed an interesting phenomenon regarding the frequencies of words in a corpus (collection of documents). For almost any corpus the frequency of the occurrence of a word (i.e. how many times it occurs) is inversely proportional to the word's frequency rank in the corpus:

$$word\_frequency \propto \frac{1}{word\_rank}$$

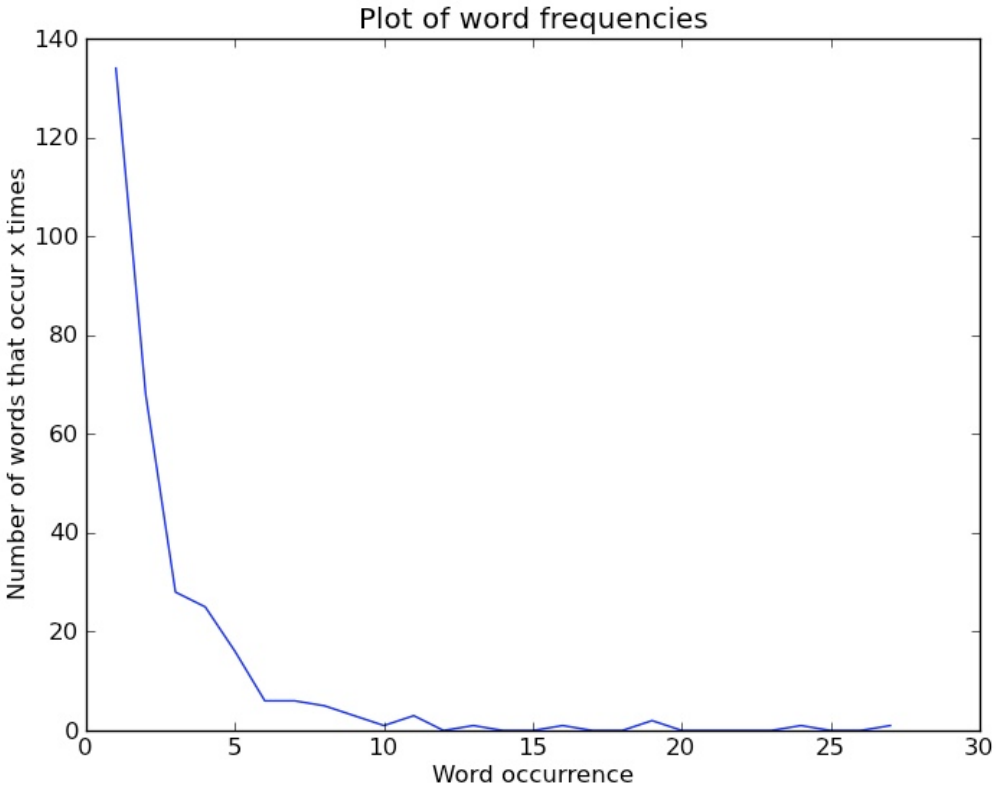
For example, the most frequent word ($word\_rank = 1$) generally occurs twice as many times as the second most frequent word ($word\_rank = 2$), etc. The graph below shows a graph of the word frequencies, sorted by rank/frequency, from *Moby Dick*:



This phenomenon tends to show up not only in text data, but in a variety of naturally occurring data ranging from search engines queries to connections in a social networks. This phenomena is so predominant that is has be coined Zipf's law. In this lab, we will be playing with one corollary of Zipf's law.

1 Zipf's law take two

One of the corollaries of Zipf's law is that if you count the number of words that occur just once, twice, three times, etc. in a corpus and plot them, you see a similar distribution to that above. This is because there are lots of words that occur just once, a few less that occur twice, even less that occur three times, etc. For example, here is a plot of this from some data from one of Dr. Seuss's books:



Here the x-axis represents the words that occur x times in the corpus. Lots of words occur only once (∼135 words), fewer words occur 5 times (∼10 words), and even fewer words occur 19 times (in fact, just 1).

2 The program

For this lab, you will be writing a program that reads text data from a file and generates two things based on the words in the file:

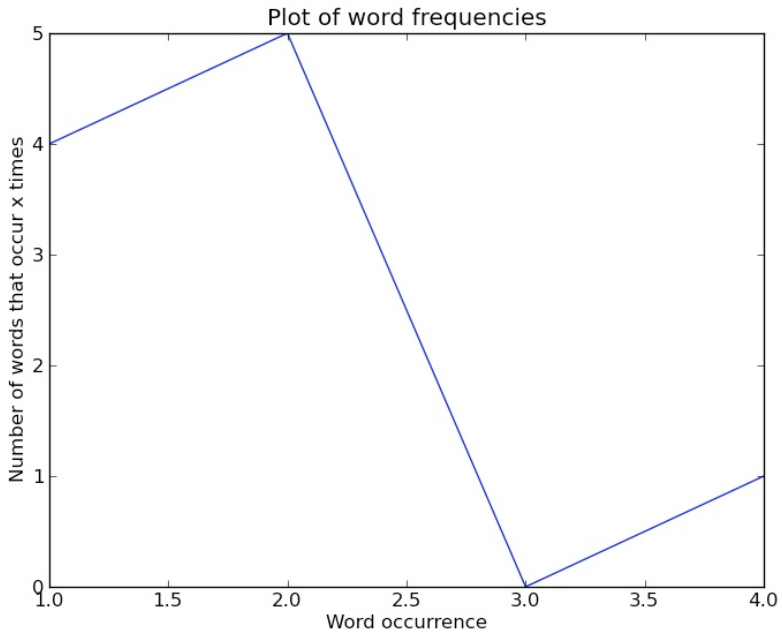
- a graph like the one in the previous section (2nd graph NOT the first) plotting the number of words that occurred once, twice, three times etc. in the file
- a printed list (i.e. printed using `print`) of the 10 most frequent words in the file in order of frequency along with each word's frequency

When the program starts it should ask the user to enter the name of a file. It should then count the frequencies of the words in this file and generate a graph using `matplotlib` *with appropriate x and y labels and title* as well as print out the 10 most frequent words.

For example, given the file:

```
the dog is here
the dog is not here
dog good
dog sometimes not good
i like dogs
```

The program would produce the following graph:



and print out the following list of words:

```
The 10 most frequent words are:
dog   4
good  2
is    2
here  2
not   2
the   2
like  1
i     1
sometimes 1
dogs  1
```

There are 4 words that occur once ("like", "i", "sometimes" and "dogs"), 5 words that occur twice ("good", "is", "here", "not" and "the"), no words that occur three times, and 1 word that occurs four times ("dog"), so the plot has the values `x = [1, 2, 3, 4]` vs `y = [4, 5, 0, 1]`.

Your program should always print just 10 words, so if there are ties for position 10, you can pick any of the tied words for the 10th.

For our purposes you can count anything that is separated by a space as a word so things with punctuation like "ham." will become words. If you want, for extra credit, you can try and remove punctuation.

3 The implementation

For this lab, I am giving you a fair amount of control over how you accomplish the program above. As you design and implement it, think hard about how you can break up your program into smaller functional units (i.e. functions) and how the functions will interact. An important component of your grade for this lab will be based upon how well you do this.

Some hints/suggestions

- When you call `show()` the program waits until you close that window to continue. If you've tried to print out text *it will not show* until you close the plot window if you are running the program with the green arrow (or from the shell). If, however, you run it using the debug mode, it will behave as you'd expect.
- Your first step will be to read the file and store how often each word occurs in the file
  - Dictionaries are a natural way for calculating and storing this information
  - Recall that the `split` method in the string class can be called on a string and gives a list of the words in that string separated by a space
- Once you have a dictionary of words with their frequencies you'll need to use this to calculate the two pieces of information you'll need to output:
  - To generate the graph, you'll need to count how many words occurred once, twice, three times, etc.
    * This will require a traversal of your dictionary from words to word counts to generate a new set of counts. You'll probably need another dictionary here, though there are other ways of doing this.
    * Be careful to make sure that you add in at some point the zero count events. In our toy example above, there were no words that occurred 3 times, but we would still like to associate 0 with the entry 3. You can either do this when you aggregate these counts or as a post processing step.
  - To get the ten most frequent words, you'll need to get the words in a dictionary whose values are largest. There are two ways of doing this. The first way, is to sort the keys (words) by value and then extract the top ten. The book suggests one way of doing this, but it can be complicated (you'll get 1 point of extra credit for doing it this way).

    However, another way to do this is to do the following ten times:
    * Find the largest key/value pair in the dictionary.
    * Remove that key/value pair from the dictionary. The `del` command deletes a key from a dictionary, for example:

```
>>> my_dict = {"banana": 10, "apple": 15}
>>> my_dict
{'banana': 10, 'apple': 15}
>>> del my_dict["banana"]
>>> my_dict
{'apple': 15}
```

One thing to note about with this approach is that it does modify the dictionary, so you need to be careful you're not using the dictionary somewhere later. You can either make sure you don't need to use the dictionary after doing this, or you can create a copy of the dictionary using one of the dictionary classes constructors:

```
>>> my_dict = {"banana": 10, "apple": 15}
>>> dict2 = dict(my_dict)
>>> dict2
{'banana': 10, 'apple': 15}
```

Notice that these two are copies and not references to the same dictionary:

```
>>> del dict2["banana"]
>>> dict2
{'apple': 15}
>>> my_dict
{'banana': 10, 'apple': 15}
```

4 Data

I've posted a few different data set for you to try your program on at:

`http://www.cs.middlebury.edu/~dkauchak/classes/midd_only/cs150/assignment8/`

"green.txt" and "fox.txt" are the text from two Dr. Suess books. Try out these different files and see what the data looks like.

Reading files in Canopy:

If you're using the Canopy editor, when opening files it does NOT assume that that file is in the same location as your .py file. However, you can set the directory where Canopy looks for text files. To do this, in the "interactive shell" panel, click on the black triangle (looks like:

from the menu that drops down select "Change working directory...". This will bring open a file menu. In this menu, select the directory where your .py file is save (and also your .txt files). Now, you should be able to just use relative pathnames, like:

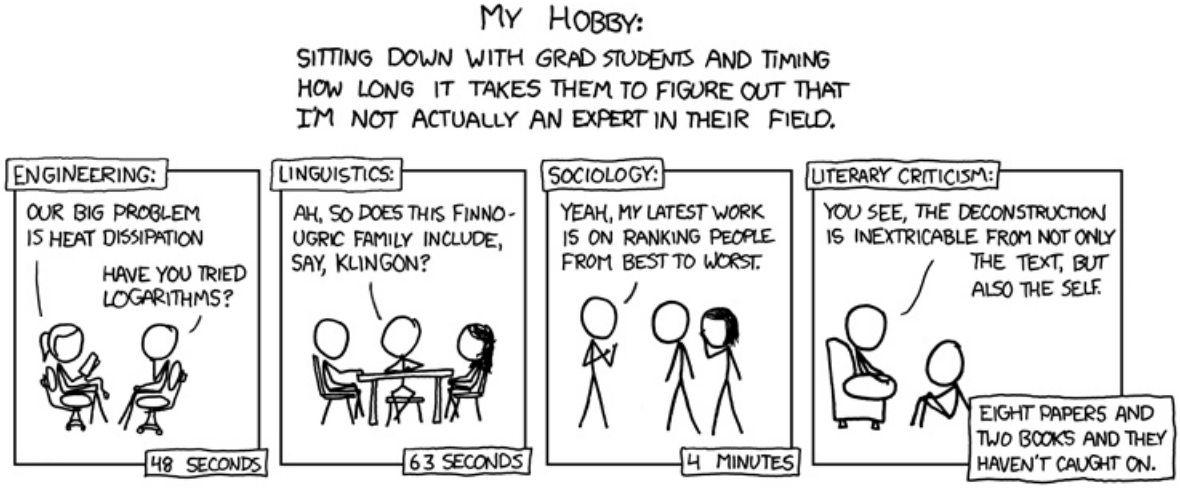
```
file = open("fox.txt", "r")
```

5 Extra credit

You may earn up to 2 points of extra credit on this assignment. Below are some ideas, but you may incorporate your own if you'd like. Make sure to document your extra credit additions in comments at the top of the file.

- Sort the dictionary of counts to find the 10 most frequent words
- Include a function to generate a plot of word verses frequency where the words are sorted by frequency from largest to smallest
- Print out other statistics about the words in the file
- Remove punctionation, etc. from the words to count "real" words

6 When you're done



http://xkcd.com/451/

Make sure that your program is properly commented:

- You should have comments at the very beginning of the file stating your name, course (including section number), assignment number and the date.
- Each function should have an appropriate *docstring*
- Other miscellaneous comments to make things clear

In addition, make sure that you've used good ***style***

Submission procedure

Submit your .py file online using the digital submission link on the course web page. You must have submitted it online before the beginning of class on Wed.

Grading

	points
word frequencies	3
word occurrence counts	3
print top 10 words	4
plot: looks correct	4
plot: labels, etc	2
code organization	2
Comments, style	4
Lab prep	3
extra credit	2
total	25 (+2)