Homework 4

Software Design **Allen B. Downey**
Spring 2007

Due: Wednesday 21 February.
The reading for this assignment is Chapters 9–10 of *How to think...*.

4.1 Textual Analysis

1. Project Gutenberg (PG) is a non-profit organization dedicated to creating a free electronic archive of books whose copyright has expired. More than 20,000 of these books are available for download from

   ```
   http://gutenberg.net
   ```

   Go to PG and download your favorite of the available books.

2. Create a file named `analyze.py` and type in the following code:

   ```
   import sys

   def main(script, filename):
       fp = open(filename)
       for line in fp:
           print line

   if __name__ == '__main__':
       print sys.argv
   ```

   The `sys` module contains functions and variables that pertain to the operating system, including `sys.argv`, which is a list that contains the arguments you provide on the command line[1].

   If you run the program like this:

   ```
   python analyze.py gatsby.txt
   ```

   You should get a list like this:

   ```
   ['analyze.py', 'gatsby.txt']
   ```

[1]The name is inherited from C; it stands for "argument vector".

Of course, instead of gatsby.txt you should use the name of the file where you stored your favorite book.

Now remove the line that says print sys.argv and replace it with main(*sys.argv); in other words, invoke main and use sys.argv as the list of arguments. I'll explain what the * means in class.

If you run the program again, it should print the contents of the file. Notice that PG includes some information at the beginning and end that is not part of the book. Edit the file and remove the extra stuff.

3. The goal of this homework is to write a program that counts the number of words in a book, and the number of *different* words in a book.

   That's pretty much all you need to know to get started, but if it helps, here is a list of suggestions for **incremental development** of the program. You don't have to do all of them. If you are feeling more confident, you can skip steps, but if you end up in debugging heck, you might want to retreat.

   The fundamental principles of incremental development are:

   - Start with a working program.
   - At each step, add the least amount of code that yields a testable result.
   - If the program fails, it should be easy to find the problem.
   - When the program works, go on to the next step.

   The best thing about incremental development is that it minimizes the time you spend debugging. The challenging thing is that it is not always easy to imagine a series of small steps that goes from the starting place to the destination.

   Here are some *suggestions* for steps you might want to follow:

   (a) While you are debugging, work with just the first chapter of the book, or even less. You can use cp to make a copy of the file and then emacs to remove most of the book. Or you can use head to get the beginning of the file and redirect the output into another file:

   ```
   $ head -100 gatsby.txt > short.txt        # get the first 100 lines
   ```

   (b) Modify the program so that instead of printing the lines of the book, it counts the lines of the book and prints the total.

   (c) Write a function called process_line and pass each line of the file to it.

   (d) Make the counter a global variable. It should be initialized outside of any function, incremented in process_line, and printed in main.

   Hint: if you want to *write* a global variable inside a function, you have to declare the variable global:

   ```
   x = 7

   def f():
       global x        # when I say x, I mean the global x
       x = x + 1
   ```

(e) Break each line into words and print the words.

(f) Count the number of words and print the result.

(g) Get rid of punctuation and convert all the words to lower case.

(h) To count the number of different words, you might want to use a dictionary. Make sure you have read Chapter 10, and then take some time to test some of the operations using the interpreter. This is time well spent!

(i) As the program traverses the book, make an entry for each unique word, and then print the dictionary. Again, use a small part of the book to start with.

(j) Instead of printing the dictionary, print just the keys from the dictionary and confirm that at least most of them are legitimate, unique words.

(k) Finally, print the number of different words in the book.

(l) Once you have a working program, review the code and look for opportunities to improve it, including renaming variables and functions to describe their role more accurately, encapsulating code into functions, generalizing functions where appropriate, and refactoring existing functions.

Even if you don't do the JFFEs that follow, you might want to think about them, because they will give you ideas for how you might improve your code.

4. As a JFFE, check to see how many of the words in the book also appear in the list of words we used for the last homework. Are there any typos in the book you downloaded? Are there any legitimate words in your book that aren't in the word list?

5. As another JFFE, count the number of times each word appears in the text, and print the ten most common words.

WHAT TO TURN IN: As always, you should make your printed code as presentable as possible before you turn it in.

You should hand in a copy of `analyze.py`. Please write on the page, or include in the comments, the name of the book you analyzed and the results (total number of words, and number of different words).