CS100: Project 2

Revision Date: October 2, 2015

Preamble

> There Shakespeare, on whose forehead climb
> The crowns o the world; oh, eyes sublime
> With tears and laughter for all time! – Elizabeth Barrett Browning, *A Vision of Poets*

Before you submit this project you must ensure that it runs without failure on a Linux distribution.

Words!

Your task is to read in the entire corpus of Shakespeare's Plays and Sonnets and print out the individual words, each word processed slightly, in sorted order.

The corpus can be obtained with this command:

```
wget troll.cs.ua.edu/cs100/python/projects/shakespeare.txt
```

You will accomplish this task with two programs, one a python program that you write and the other a one-line *shell script*, that calls your python program and pipes its output to the sort utility. The python program will be named *tokenize.py* while the shell script will be named *shakesort*.

An Example

Suppose the file *shakespeare.txt* contained the single line. famously spoken by Juliet in *Romeo and Juliet*:

```
"O Romeo, Romeo!  wherefore art thou Romeo?"
```

Then, running the command:

```
$ shakesort
```

should produce the following output:

```
art
o
romeo
thou
wherefore
```

What are words?

To obtain words from a file of text (*shakespeare.txt* is a file of text), one *tokenizes* the file. Tokenizing a file means repeatedly reading tokens from the file, where each token is composed of a sequence of consecutive non-whitespace characters. Thus, tokenizing the line:

```
"O Romeo, Romeo! wherefore art thou Romeo?"
```

would produce the following tokens:

```
"O
Romeo,
Romeo!
wherefore
art
thou
Romeo?"
```

As previously stated, your program should strip away punctuation from each token and reduce all upper-case letters to lower case.

When completed, you should be able to run your program like this:

```
$ python tokenize.py wherefore.txt
```

which should produce the following output:

```
o
romeo
romeo
wherefore
art
thou
romeo
```

Note that your program neither sorted the output nor removed duplicates. These tasks are left for the second phase of the project. Note also that the file name *wherefore.txt* was passed in as a command-line argument. You can read about how to handle command-line in the chapter entitled "Input and Output" in the textbook.

Reading from files

Once you get the name of the file from the command line, you will need to tokenize the text in the given file. To read the tokens in a file, you should use the *Scanner* class, which you can retrieve with:

```
wget http://troll.cs.ua.edu/cs100/python/projects/scanner.py
```

Run this command in the same directory as your project files. Read the chapter entitled "More on Input" in the textbook on how to read in a file of tokens. Note that the *readtoken* function of a *Scanner* object will break up a quoted string into individual tokens, but the first token will begin with a double quote. Symmetrically, the last token of the string will also contain a double quote.

Processing a token

One processing task is to remove all characters that are not letters. To give you some help on this task, here is some code that replaces each `'x'` character in a token with a `'y'`.

```
result = ""
for i in range(0,len(token),1):      # look at each character
    if (token[i] == 'x'):            # it matches!
        result = result + 'y'        #    so add in the replacement
    else:                            # it doesn't match
        result = result + token[i]   #    so add in that character
```

The code for replacing capital letters would be similar. Removing the non-letter characters would also be similar, except you would not add anything to the resulting string if the character matched.

If the resulting token is the empty string (i.e. the original token contained no letters), the token should not be written to the output.

Using the system *sort* program

Once you have *tokenize.py* working properly, you will now need tor write a shell script, named *shakesort*, to call your tokenizing program and pass its output to the built-in sort utility. This will be a one line "program". In general, to run the output of one program, say *p1* to the input of another, say *p2*, one could type something like:

```
p1 a11 a12 | p2 a21 a22
```

where *a11* and *a12* are command-line arguments to the *p1* program and where *a21* and *a22* are command-line arguments to the *p2* program. Of course, in your case, *p1* represents running *tokenize.py* program and *p2* represents the *sort* utility. Commands of this type can be given at the system prompt or they can be placed in a file.

Once you have placed the proper command in *shakesort* file, you need to make it executable with this command:

```
chmod +x shakesort
```

To learn about what command-line arguments to pass to the built-in sort utility to achieve the desired result, run the command:

```
man sort
```

Program Organization

Your *tokenize.py* program should have a *main* function, of course, plus the following helper functions:

- a function that when given a file name tokenizes the contents of the file, placing the individual tokens into an array – the function returns this array
- a function, when given an uppercase letter, returns the equivalent lowercase letter

- a function that when given a token replaces uppercase letters in the token with lowercase ones, returning the newly generated token
- a function that when given a token removes punctuation from the token, returning the newly generated token
- a function that when given an array of unprocessed tokens, replaces each token in the array with its processed version – this function is a procedure.
- a function that when given an array of tokens, outputs each non-empty token, one token per line

Stepwise refinement

Write a version of the program that:

- prints out the name of the file to be processed
- prints out the first token in the file
- prints out every token in the file
- stores each token in the file into an array – then prints the array
- adds the function that removes punctuation and a function that processes the array with the punctuation-removing function – then prints the array
- adds the function that replaces upper case characters and process the array with this new function – then prints the array
- adds the function that outputs the array, one token per line

Test each version of your program on a file that contains just a few tokens. Name this test data *test.txt*.

Challenge

Use lists instead of arrays. In the tokenizing function, if you prepend the tokens you read onto the front of the growing list, you will find your program runs much faster!

Compliance Instructions

Name the file containing your small amount of test data *test.txt*. You should be able to run your program, like this:

```
shakesort
```

Hint: you can test your *shakesort* shell script on a small amount of data by renaming your *test.txt* to *shakespeare.txt*.

Submission Instructions

Make sure you have the following files in your directory before submitting:

- shakesort
- tokenize.py
- scanner.py
- test.txt
- shakespeare.txt

Change to the directory containing your assignment. If you are working on your USB-stick, run the command:

```
submit cs100 YYY project2
```

Replace *YYY* with your three-digit section number.