Name: Evan Leglar	**Lab Time** Tuesday 12:00pm

Names of people you worked with:
• N/A

Websites you used:
• Stackexchange • Geeks for Geeks

Approximately how many hours did it take you to complete this assignment (to nearest whole number)?	15

The Rules: Everything you do for this lab should be your own work. Don't look up the answers on the web or copy them from any other source. You can look up general information about Python on the web, but no copying code you find there. Read the code, close the browser, then write your own code.

By writing or typing your name below you affirm that all of the work contained herein is your own and was not copied or copied and altered.

Note: Failure to sign this page will result in a 50 percent penalty. Failure to list people you worked with may result in no grade for this lab. Failure to fill out hours approximation will result in a 10-percent penalty.

Turn .zip files of Python code to Gradescope or your assignment will not be graded

BEFORE YOU BEGIN

1. Make a new project for this lab
2. Create compare.py
3. Grab the two text files – war_and_peace.txt and words.txt and put them in your project folder
4. Look up the following string functions: split, crop and their friends
    a. String functions page in Python
    b. Types page in Python
    c. Dealing with punctuation and white space
5. Look up the following on getting arguments from the command line
    a. Command line and systems arguments page

Learning Objectives:

You should be able to do the following:

- Read and write files
- Deal with large data sets
- Be able to use dictionaries/hash functions

Thoughts (come back and read these after you've read the problems):

1. If you find that you're doing the same thing two or more times, consider separating that part of the code out into its own function.

2. Your choice of data structures will seriously affect the amount of time your code takes to run. We've talked about several data structures in class (lists, tuples, strings, dictionaries, and sets). Think about which one of these will make thinks fastest for each part of the lab.

3. There should be no punctuation in words. This means "flibberty-gibbet" is actually two words. This might mean that words that are actually one will get counted as two. That's OK. Also "end." is not a word. It's a word with punctuation attached. You need to remove this punctuation.

4. Testing in this lab is a bit harder. One thing that you could do is to make your own test files, where you know the answers beforehand, and use these to verify that your code is doing the right thing.

5. The numbers for the word count in the examples in this lab are wrong, obviously.

6. When writing the code for this lab, try to do it in as few lines of code as possible. Our reference solution has about 30 Functional Lines of Code (FLOC). A FLOC is a line that does something (i.e. not blank, and not a comment). You don't have to make your code this short to get it to work but, if you're writing substantially more code than this, then you should take a break, and see if there's a simpler way of doing things.

7. Your code should work with any two text files. This is extra credit but good practice. We may pass your program different text files when grading; make sure your code prints out the correct file names, etc. Ask the TAs if you're not familiar with how to run code from the command line.

8. There are many ways within PyCharm to run with command line arguments.
    1. For debugging in the debugger, you can hard code the system argument commands by setting them directly. Make sure you take this out before turning your code in.
    2. You can set the command arguments in the Preferences page
    3. You can run directly from the terminal (python compare.py words.txt)

Grading Checkpoints

1. Files read into code correctly (1 point), correct number of words (1 point) [2 points total]
2. Case insensitive (1 point), punctuation removed (1 point) and correct number of unique words (2 point). [4 points total]
3. Correct values for three classes of words (1 point each). [3 points total]
4. Code runs in under 10 seconds. [1 point]
5. Code can be run via command prompt with any two filenames and guards against bad input. [2 points extra credit]

Remember to hand in the .zip of the python files and this file (as a PDF) to Gradescope.

Problem 1 Word count

For this assignment, create a new file called `compare.py`. First, we're going to write some code that will tell us how many words are in the file.

1. **Write a function** called `word_count` which takes in a single argument, a file name, and returns a list with all of the words in the file.
2. Write some code so that **when you run your file**, it runs the function on the two example files provided, `war_and_peace.txt` and `words.txt`, and prints out the output like this:

   ```
   war_and_peace.txt:
     234 words
   words.txt:
     432 words
   ```

   This code should not run when the file is imported.

3. **Extra credit:** Modify your code so that it can receive two arguments via the command prompt. These arguments should be file names. When the script is run in this manner, it should run the words analysis from 1.2 on the specified files.
   a. If no arguments are specified, it should run the analysis on the default files (war_and_peace and words).
   b. Implement a check to make sure *exactly* two arguments have been received, and that the files exist.
   c. For each of the checks above, print a useful message to the console if the check fails.

For reference, you should be getting about 570,000-ish words for war_and_peace.txt, and 230,000 for words.txt.

Comments for grader/additional information (if any)

Output

`war_and_peace.txt:`
`  566008 words`
`words.txt:`
`  235886 words`

Problem 2 Unique word count

Create a new function called `unique_count` that takes a single file name as an argument, and returns an iterable (list, tuple, or set) with all of the unique words for each file. For the purpose of this lab, words should be **case insensitive** and **remove all non-alphanumeric characters** (punctuation). Therefore, each of the following groups of words is equivalent, and should output the first one only:

- hello, Hello, H'ello, He-LLo
- 1st, 1St, 1-st
- 5551234, 555-1234, (555)1234

Hint: There is a hard way to do this, and an easy way.

Add the count of the unique words function to the output when the file is run as a script. If you did the command line extra credit from above, it should print out the unique words for the passed-in files.

```
war_and_peace.txt:
  234 words
  unique: 12
words.txt:
  432 words
  unique: 15
```

For reference, you should be getting about 17,000-ish unique words for war_and_peace.txt and 230,000-ish unique words for words.txt.

Comments for grader/additional information (if any)

Data structure
To find the unique values in each of the cleaned lists, the lists were put into a set. The set was able to remove any duplicates in the lists. After removing the duplicates, the output was converted back to a basic list to improve handling in any further functions.

Output

war_and_peace.txt:
  566008 words
  unique: 20446
words.txt:
  235886 words
  unique: 234371

Problem 3 Unique words in each file

Create a new function called `unique_compare` which takes in two arguments, each a file name, and returns three values in the following order:

1. An iterable containing the unique words in the first file (i.e. words that the first file has but not the second)
2. An iterable containing the unique words in the second file
3. An iterable containing words in both files

Add the count of the unique compare function to the output when the file is run as a script.

```
war_and_peace.txt:
  234 words
  unique: 12
words.txt:
  432 words
  unique: 15
Only war_and_peace.txt: 54
Only words.txt: 44
Both files: 54
```

Comments for grader/additional information (if any)

compare.py

```
#!/usr/bin/env python3
import sys
import string


def txt_clean(txt_file):
    """
    function to change all characters in string to lower-case, then
use translate table to remove all punctuation,
    and then split the string by the delimiter spaces. The output is
an iterable, clean list
    """
    with open(txt_file, 'r') as f_1:
        txt_file = f_1.read()

    txt_file = txt_file.lower()  # converts each string into lower
case letters
    punctuations = '''!()-[]{};:'"\,<>./?@#$%^’&*_~'''  # list of
punctuation to clean from txt (includes extra ’ )
```

```
    txt_file = txt_file.translate(str.maketrans('', '',
string.punctuation + punctuations))  # removes the punctuation
    clean_list = txt_file.split()  # splits each text file based on
spaces found
    return clean_list


def word_count(txt_file):
    """
    function to return the cleaned list as a list named num_words
    """
    num_words = txt_clean(txt_file)
    return num_words


def unique_count(txt_file):
    """
    function to return an iterable list after duplicates have been
removed by the set function
    """
    unique_words = list(set(txt_clean(txt_file)))
    return unique_words


def unique_compare(x, y):
    """
    takes in 2, cleaned lists from the original txt files and then
uses the python set operations to determine what
    words are unique to each file and the intersection between the
two
    """
    txt1 = set(txt_clean(x))
    txt2 = set(txt_clean(y))
    only_txt1 = list(txt1 - txt2)
    only_txt2 = list(txt2 - txt1)
    both_txt = txt1.intersection(txt2)
    return only_txt1, only_txt2, both_txt


if __name__ == '__main__':

    if len(sys.argv) == 3:
        x = sys.argv[1]
        y = sys.argv[2]
    elif len(sys.argv) == 1:
        x = "war_and_peace.txt"
        y = "words.txt"
    else:
        raise ValueError

    print(str(x) + ":")
    print("  " + str(len(word_count(x))) + " words")
    print("  unique: " + str(len(unique_count(x))))
    print(str(y) + ":")
    print("  " + str(len(word_count(y))) + " words")
    print("  unique: " + str(len(unique_count(y))))
```

```
    j, k, m = unique_compare(x, y)
    print("Only war_and_peace.txt: " + str(len(j)))
    print("Only words.txt: " + str(len(k)))
    print("Both files: " + str(len(m)))
```

Output

```
(base) Evans-MacBook-Pro-2:Lab_3 evanleglar$ python3 compare.py
war_and_peace.txt words.txt
war_and_peace.txt:
  566008 words
  unique: 20446
words.txt:
  235886 words
  unique: 234371
Only war_and_peace.txt: 9406
Only words.txt: 223331
Both files: 11040
```