Lab 6: Strings and File I/O

CSE/IT 107

Introduction

The purpose of this lab is to dig a little deeper into strings and file operations.

Reading

Chapter 9 in Think Python: How to Think Like a Computer Scientist (Think)

Coding Conventions

Follow PEP-8 recommendations for your code.

Problems

Make sure your source code files are appropriately named. Make sure your code has a main function; use `boilerplate.py` you created in Lab 1.

Think Python is available from `http://www.greenteapress.com/thinkpython/`. Page numbers and exercise numbers refer to those found in the PDF file available at the website.

1. Exercise 9.1 *Think Python*, page 82. Name your source code `think_9-1.py`
2. Exercise 9.2 *Think Python*, page 82. Name your source code `think_9-2.py`
3. Exercise 9.3 *Think Python*, page 82. Name your source code `think_9-3.py`. Ignore the *Modify* portion of the problem.
4. Exercise 9.4 *Think Python*, page 82. Name your source code `think_9-4.py`
5. Exercise 9.5 *Think Python*, page 82. Name your source code `think_9-5.py`
6. Exercise 9.6 *Think Python*, page 82. Name your source code `think_9-6.py`

Reading and Writing to a File

Last week we saw it was very easy to open a file in Python. For example, the following program opens the file `integers.txt` for reading, reads one line of the file at a time, prints the contents of the line to the terminal, closes the file, and exits. Note: if you open a file, you are responsible for closing it.

This assumes `integers.txt` is in the current working directory. `integers.txt` is available on Moodle in the tarball `lab6.tar.gz`.

```
#!/usr/bin/env python


def main():
    infile = open('integers.txt', 'r')
    for line in infile:
        print line
    infile.close()


if __name__ == '__main__':
    main()
```

If you run this program, you will notice that the data is printed followed by a new line. For better output, you need to strip off the new line from each line. Can you find the built-in string function that does this? Also, remember you are reading in strings not integers. If you want the data to be integers, you have to convert the string to an integer using the function `int()`.

Now to write to a new file you change the second parameter of `open()`, known as its mode, to a `'w'`. `'w'` stands for write. To make a copy of a file in the current working directory, you can do the following:

```
#!/usr/bin/env python


def main():
    infile = open('integers.txt', 'r')
    outfile = open('copy_integers.txt', 'w')
    for line in infile:
        outfile.write(line)
        # or
        # line = line.rstrip()
        # outfile.write(line + '\n')
    infile.close()
    outfile.close()
```

```
if __name__ == '__main__':
    main()
```

You can check that `integers.py` and `copy_integers` are the same by running the `diff` command. Run the following command at a terminal. It should return nothing.

```
$ diff integers.txt copy_integers.txt
```

Notice that the program did not strip off the new line character as it is making a copy. If you stripped off the new line from the line, you have to add it back in when you write to the out file. The commented code shows how to do that.

To have more complex write statements you can combine format with write calls. For example,

```
l = [1, 2, 3, 4]
outfile.write('number of items = {0}\n'.format(len(l)))
```

7. Write a program that reads in the file `integers.txt`, available on Moodle, and creates a new file `summary.txt` that writes the number of integers in the file, their sum, and the mean, median, mode, minimum and maximum of the integers in the file at the beginning of the file followed by a new line and then the data sorted.

   As you read the data one line at a time, strip off the new line, convert it to an integer and add it to a list. Python has some built-in functions that will be helpful in your calculations: `min(list), max(list), list.sort()` and `list.count(i)`. Look up what these functions/methods do with Ipython's help facility.

   The average is sum / number of integers. This should be a float.

   For the median, first sort the data and then calculate the median:

   If n is odd then Median (M) = value of ((n + 1)/2)th term.

   If n is even then Median (M) = value of (((n)/2)th term + ((n)/2 + 1)th term) / 2

   For the mode, which is the most frequent value in a data set, use the `list.count()` method.

   Your output should look like this:

```
number of items = 500
sum = 244167
avg = 488.334
median = 474
mode is 169 with a count of 4
min = 0
```

```
max = 999

0
1
3
3
5
11
...
```

Stop Words

If you are trying to index a webpage or a book or any text for that matter, one strategy would be to simply store every word on the page. This, of course, allows for duplicate words which would waste space on the server. To save space, you remove all duplicate words from the text to construct your index. While saving significant space with this strategy, you soon realize you can do better if you filter out words that are common and provide little lexical meaning. Such words are called stop words. Example of such words are `'the'` and `'a'`. Searching on stop words doesn't yield useful results as all or a large portion of pages have those words. Try searching on `'the'` in google.

There is little agreement about what constitutes a stop word and filtering based on stop words is problem dependent. But overall the strategy is effective. For example, this strategy has been employed by Stackoverflow to speed up their queries: "One of our major performance optimizations for the related questions query is removing the top 10,000 most common English dictionary words (as determined by Google search) before submitting the query to the SQL Server 2008 full text engine. Its shocking how little is left of most posts once you remove the top 10k English dictionary words. This helps limit and narrow the returned results, which makes the query dramatically faster." `http://blog.stackoverflow.com/2008/12/podcast-32/`

On Moodle is two stop word lists, `stopwords.csv` and `function_words.txt`. You will use these lists to employ two different stop word strategies when you filter a text document. The stop word list is based on MySQL's list of words that are excluded from full text queries. Function words are words that provide grammatical connections between words in a sentence. The list of function words is incomplete as it doesn't account for two or more word combinations.

8. Write a program that takes a filename from the command line and
   (a) finds all unique words in the text
   (b) writes the list of unique words one word per line to a file (argv[1] + '_unique.txt')
   (c) reads in two types of stop words lists and creates a list for each type
   (d) removes all stop words from the unique word list using words in `stopword.csv`

(e) write the unique word list with stop words removed one word per line to a file (argv[1] + '_stop.txt')

(f) removes all function words from the unique word list, using words in `function_words.txt`

(g) write the unique word list with stop words removed one word per line to a file (argv[1] + '_function.txt')

(h) Calculates the percent of 1) unique words to all words, 2) the percent of unique words with stop words removed to all words, 3) the percent of unique words with stop words removed to unique words, 4) the percent of unique words with function words removed to all words, and 5) the percent of unique words with function words removed to unique words. Print the percentages to the terminal.

Name your source code file `stopwords.py`. Test with the files `alice.txt`, `metamorphosis.txt`, and `gadsby.txt`. All files are available on Moodle.

Scrabble

You are working on developing a electronic version of Scrabble. You are tasked with writing a variety of functions to this end. `scrabble.txt` is available on Moodle. `scrabble.txt` is a modified version of the Tournament Word List used in the US and Canada and based on the Official Scrabble Players Dictionary.

9. Write a program that finds and prints all two letter words in the Scrabble dictionary. Name your program `twoletter.py`

10. Write a program that finds and prints all words that consist of only consonants in the Scrabble dictionary. Consider `y` a vowel in all cases. Name your program `consonants.py`

11. Write a program that determines if a word entered by a user is a valid scrabble word. While you could perform a linear search strategy this is inefficient. Employ a binary search strategy to find if a word is valid or not. To test your search, in `main()` write an infinite while loop that keeps asking for words to test. Use a digit to exit out of the program. Name your file `binsearch.py`.

12. Write a program that finds your best play given a list of characters (the letter tiles you are holding) and a letter(s) on the board. You can think of this as a two part strategy. First, find all words in the dictionary that consist only of characters in the given character set. You have to be careful that you do not return words with duplicate letters. For instance, if a user enters `'capjekit'` your list should not include `'jeep'` or `'aa'` or `'tat'` as you only have one `'e'`, `'a'`, and `'t'` tile. Second, given what the letter tiles are worth find the word that returns the maximum letter score. You do not need to account for double letter or triple letter scores on the board. Write a loop in `main()` that asks the user to input his/her tiles and a character from the board and

prints out the list of all possible words plus the best play. What is the best play given the letters `'qzxjkaeiou'`? Name your program `bestplay.py`

Remember strings are immutable. So you may want to use lists when you find words.

Word tiles in Scrabble have the following numeric values:

```
A is worth 1
B is worth 3
C is worth 3
D is worth 2
E is worth 1
F is worth 4
G is worth 2
H is worth 4
I is worth 1
J is worth 8
K is worth 5
L is worth 1
M is worth 3
N is worth 1
O is worth 1
P is worth 3
Q is worth 10
R is worth 1
S is worth 1
T is worth 1
U is worth 1
V is worth 4
W is worth 4
X is worth 8
Y is worth 4
Z is worth 10
```

Submission

Create a tarball of your *.py files.

```
tar czvf cse107_firstname_lastname_lab6.tar.gz *.py
```

To check the contents of your tarball, run the following command:

```
tar tf cse107_firstname_lastname_lab6.tar.gz *.py
```

You should see a list of your Python source code files.

Upload your tarball in Moodle before the start of you next lab.