Week 10: In-class Exercises

Resource: http://tinyurl.com/is111-ice10

1. **[Difficulty: *]** Write a program that prompts the user for a filename (assume valid filename). Then prints the sum of all the integers read from the file. A sample run is shown below:

```
Enter filename>q1-numbers.txt
The sum is 37
```

You are given **q1-numbers.txt** with the following content:

```
10
5
-2
21
3
```

2. **[Difficulty: *]** Write a program that prompts the user for a filename (assume valid filename). The file contains n integers per line.

```
10,20,30
5,9
-2
21,2,4
3,7,9,1,3
```

For each row in the data file, print the sum of the n integers to the console. A sample run is shown below (for **q2-numbers.txt** with content shown above):

```
Enter filename>q2-numbers.txt
60
14
-2
27
23
```

3. **[Difficulty: **]** You are given a file called **q3-tester.py**. Define the following functions in **q3.py**:

   a. **get_english_dictionary()**: The function takes in a filename and returns the list of words from the specified file. The file will contain one word per line. Do remember to remove the trailing end of line character.
   b. **check_spelling()**: The function takes in a sentence and returns a list of words from the text that are possibly misspelled, i.e., a list of words that are not found in the English dictionary. For example, check_spelling("I sturddy at Singapore Managment Univercity") should return ["sturddy", "Managment", "Univercity"].

(**words_alpha.txt** is taken from https://github.com/dwyl/english-words. Note how the functions in **q3.py** are used in **q3-tester.py**)

4. **[Difficulty: *]** Write a program that prompts the user for two **str** values indicating two file names say, input_filename and output_filename. The program reads the input_filename , and writes into output_filename. Each line in the output file will have a line number prefixed to the line content in input. For example, if the following is the content of the input file:

```
A for apple
O for orange
P for pear
```

The output file should look like this:

```
1 A for apple
2 O for orange
3 P for pear
```

Note: if the output file exists, overwrite the existing file. You could use the **q4-data.txt** as input file.

-- OPTIONAL --

5. **[Difficulty: **]** Write a python file called **q5.py** that reads a file and prints all the lines that contain a specific word (passed to the Python script).

Hint:
When values are passed to the Python script during execution, the arguments are stored in this variable called **sys.argv**. As an example of how to use **sys.argv** variable, observe the code in **sample.py** script below:

```
import sys

if len(sys.argv) == 3:
    print("first:", sys.argv[0])
    print("second:", sys.argv[1])
    print("third:", sys.argv[2])
```

A sample run of the **sample.py** is shown below:

```
c:\is111>python sample.py apple orange
first: c:\is111\sample.py
second: apple
third: orange
```

We have given you seashells.txt in the resources. You can pass the filename and word to be searched for on the command line as shown below: (Observe the first line in the sample shown below.
The filename we would want to read is “seashells.txt” and the word we are looking for is “sure”)

```
c:\is111>python q5.py sure seashells.txt
The shells she sells are surely seashells.
I'm sure she sells seashore shells.

c:\is111>python q5.py python seashells.txt
No line found.
```

Question q5 is trying to implement code for a command available in Unix, called grep. The grep command takes a string and a file as arguments and prints all lines in the file which contain the specified string.

6. **[Difficulty: **]** Write a program **q6.py** that reads from **q6-basic.txt**, multiplies corresponding elements in 2 lists, and writes those products whose value is larger or equal to the <check value> into **output.txt** .

   The input file format is as follows:
   Format is as follows: A_1,A_2,...,A_n@B_1,B_2,...,B_n#<check value>

   Below is an example:

```
1,2,3@3,4,5#8
1,2@3,4,5#4
1,2,3@4,5#10
```

   Program **q6** does the following:
   a. read **q6-basic.txt**
   b. multiply A_1 with B_1, A_2 with B_2, ..., A_n with B_n
   c. It checks the result of each multiplication. For results whose value is greater than or equal to the <check value>, writes the multiplication value to the output file.

   For example:
   a. The result of the first line is 1 x 3 = 3, 2 x 4 = 8, and 3 x 5 = 15. Only the numbers 8 and 15 will be written to output.txt (as the check value is 8).
   b. The result of the second line is 1 x 3 = 3, 2 x 4 = 8, 5 = 5. Only 5 and 8 will be written to output.txt

   The result of **output.txt** is as follows:

```
[8, 15]
[8, 5]
[10]
```

7. **[Difficulty: ***]** Write the program **q7.py**. Attempt this question after **q6**. This program is similar to q6 but now the program allows multiple lists, and the number of tokens in each list is not fixed, also the check value is optional. The program must read from **q7-advance.txt** that allows flexible number of lists to be multiplied. If check value is not available, all values are written to output.txt. An example input.txt is shown below:

   Note: It would be better to write functions to perform smaller tasks, such as function to read the file and create a list, another function to find the product of the multiple list given a check number etc. This way it will be easy to test the modular function before putting them all together to solve the overall problem.

```
1,5,7
1,3#5

1,2,3@4,5@11,13@30,2,1,6#10
1,2,3@3,4,5#6
1,2@3,4,5@7,8,9#7
1,2@3,2
```

Line 1: No check Value
Line 2: 1 list, with check value
Line 3: 0 list, no check value
Line 4: 4 lists, with check value
Line 5: 2 lists, with check value
Line 6: 3 lists, with check value
Line 7: 2 lists, no check value

The output for the above input would be as follows:

```
[1, 5, 7]
[]
[]
[1320, 260]
[8, 15]
[21, 64, 45]
[3, 4]
```

8. **[Difficulty: ***]** Write a python script prettifier named **q8.py**. The program will prompt the user for a python script (filename that ends with .py), and the number of spaces to use for a tab (stored in a variable named **n)**. It will repeatedly prompt the user until he enters a valid filename (ends with .py).

   It will then prettify your code in the following manner:

   a. All leading tabs are replaced by spaces. Each tab is replaced by **n** spaces, as specified by the user.
   b. A single space character should replace any other extra spaces/tabs in the statement (other than the leading space/tab).
   c. All trailing whitespaces should be removed.
   d. The processed content should be saved in a file named <original-filename>-cleaned.py (e.g. if the file used is q5-tabs.py, then the output file will be q5-tabs-cleaned.py).

   Note: 3 test files are provided. **q8-tabs.py**, **q8-spaces.py, q8-messy.py**

For example, given the content of **q8-messy.py** :

```
def rot (sentence, offset)                         :
    result = ''
    for ch in sentence:
        if ch == ' ':
            result     += ' '
        else:
            result += chr( (ord(ch)-   ord('a') + offset) % 26 + ord('a'))

    return                                                     result

print (rot('xyz',2))
```

The generated **q5-messy-cleaned.py** should be as follows:

```
def rot (sentence, offset) :
    result = ''
    for ch in sentence:
        if ch == ' ':
            result += ' '
        else:
            result += chr( (ord(ch) - ord('a') + offset) % 26 + ord('a'))

    return result

print (rot('xyz',2))
```

MORE EXERCISES

9. **[Difficulty: *]** Write a program to read from the given file called **words.txt**. Every line in words.txt contains one single word. The program is supposed to read words in the file and print all words along with the count of distinct vowels present in the word as shown in the sample run. You may want to write additional functions in the file, one function perhaps to count the number of distinct vowels in every word etc.

   Sample run (with the given words.txt file):

```
python 1
dynamic-typed 3
function 3
parameter 2
argument 3
str 0
int 1
float 2

… rest not shown for brevity
```

10. **[Difficulty: **]** Write a program that reads inputs from the given file called **phone.txt**. Every line in phone.txt contains one phone number. The program is supposed to verify if lines in the text file contain valid phone numbers and write out only valid phone numbers one number in a line into a file named valid.txt

    A valid phone number has the format
    `Nddd-dddd` where `N` is 6 or 8 or 9 and `d` is any digit 0-9

    Sample run (with the given phone.txt file):

```
Invalid phone numbers:
7346-8895
123445384
846-7894
6457-q088

Writing valid phone numbers to out.txt ...
Done.
```

11. **[Difficulty: **]** Write a program that reads inputs from the given file called **sales.txt**. Every line in sales.txt contains sales amount of a specific month in the format.
    Month name, Sales amount
    Your program is supposed to print the total sales amount of all months in the file on the screen.

    Sample run (with the given phone.txt file):

```
Total sales amount for the whole year $64481.90
```

12. **[Difficulty: ***]** There is a belief that humans are able to read even when the order of the letters are misplaced. Try reading this:

    I cnduo't bvleiee taht I culod uesdtannrd waht I was rdnaieg

Interesting read here
https://www.mrc-cbu.cam.ac.uk/people/matt.davis/cmabridge/

Write a program that reads text from the given file called **talk.txt** and prints the content on the output screen scrambling the words in the content as described below:

- Scramble every word in the text leaving untouched the first and last letter
- You may assume that the text has words separated by a single space. Maintain single space between words in the scrambled version
- **Challenge** yourself to omit the punctuations [, ' ! .] in the content untouched in the original place

(Adapted from the book "The practice of Computing using Python" - Programming Project on Page 340)

As an example, if the text reads "`Programming is really fun`"
One version after scrambling could be "`Pgmnarirmog is rlelay fun`"
Note that since we scramble only the middle letters, "is" and "fun" in the example are not scrambled.

As you think of designing a solution for this question, define and make use of functions to do simpler tasks. Test the function before building on to complete the solution.

Note:
1. To scramble middle letters of a 6-lettered word, you could take each letter in the indices 1-4 and put them in positions randomly between 1 and 4. A 6-letter word after scrambling should contain all the six letters.
2. To make this exercise simpler, after scrambling a word if output happens to remain the same as the original, you may ignore.
3. You may want to use `random` module to help you with scrambling letters.

Sample run (with the given talk.txt file):

```
… partial output shown for brevity
We do not need miagc to chgnae the wldor, we crray all the pwoer we
need inisde oleeursvs aedyral: we have the pweor to imngiae betert.
```

-- OUT OF SCOPE --

13. **[Difficulty: *]** In Python, the `zip` function takes n number of iterables, and returns an iterator of tuples. The i-th element is created using the i-th element from each of the iterables. If the iterables are of unequal lengths, then the returned list is truncated to the length of the shortest iterable.

```
>>> for a_tuple in zip([1,2], [3,4]):  # zip object is iterable!
...     print(a_tuple)
...
(1, 3)
(2, 4)

>>> list(zip([1,2], [3,4]))  # alternatively, you can pass into the list
[(1, 3), (2, 4)]              # function to get a list of tuples

>>> list(zip([1,2,3], [4,5]))# return list is truncated to the
[(1, 4), (2, 5)]              # length of the shorter input list

>>> list(zip('abc', '123'))  # str objects are iterable too!
[('a', '1'), ('b', '2'), ('c', '3')]

>>> list(zip('abc',[1,2,3])) # doesn’t have to be the same type of iterable
[('a', 1), ('b', 2), ('c', 3)]

>>> dict(zip(['apple', 'orange'], [23,4])) # can convert it into a dict
{'apple': 23, 'orange': 4}
```

In Python, the `zip` function takes n number of iterables, and returns a iterable object that returns a tuple in each iteration. The i-th element of the tuple is created using the i-th element from each of the iterables. If the iterables are of unequal lengths, then the returned list is truncated to the length of the shortest iterable.

You are given the following data file(q9-data.csv). Each line consists of the student's name and his/her quiz scores. Print the average of the 3 quizzes. Try using the zip function to solve this problem.

```
name,quiz1,quiz2,quiz3
amy,7,7.5,8
bill,8,10,9
charles,9,9,6
dill,7,6,4
elise,10,10,10
fred,9.5,10,8.5
gregory,4.5,5,6
```