CSC1015F Assignment 10: File Input and Output

Assignment Instructions

This assignment involves constructing Python programs that perform file input and output.

Furthermore, your solutions to this assignment will be evaluated for correctness and for the following qualities:

- Documentation
  - Use of comments at the top of your code to identify program purpose, author and date.
  - Use of comments within your code to explain each non-obvious functional unit of code.
- General style/readability
  - The use of meaningful names for variables and functions.
- Algorithmic qualities
  - Efficiency, simplicity

These criteria will be manually assessed by a tutor and commented upon. Up to 10 marks will be deducted for deficiencies.

Question 1 [35 marks]

Write a Python program called '`anagramsearch.py`' that searches a file for anagrams of a given word, printing the results in alphabetical order.

Given two words, each is an anagram of the other if they contain the same letters in the same quantities. For example, 'green' and 'genre'.

Here are 4 examples of intended program behaviour (user input in BOLD):

Sample I/O:

```
***** Anagram Finder *****
Enter a word:
triangle
['alerting', 'altering', 'integral']
```

Sample I/O:

```
***** Anagram Finder *****
Enter a word:
Orange
['onager']
```

Sample I/O:

```
***** Anagram Finder *****
Enter a word:
back
Sorry, anagrams of 'back' could not be found.
```

Sample I/O:

```
***** Anagram Finder *****
Sorry, could not find file 'EnglishWords.txt'.
```

A lexicon (words file) has been provided on the Vula assignment page (‘`EnglishWords.txt`’).

NOTE: This file begins with a copyright notice that should not be removed. Your program must skip this notice by reading lines until it encounters one consisting of the word "START".

NOTE: The last example of program behaviour shows what the program should do if it does not find the words file.

HINT: To determine if one word is an anagram of another, construct a dictionary of letter frequencies for each word and compare e.g. given ‘act’, ‘cat’, and ‘tact’:

```
>>> w1 = { 'a':1, 'c':1, 't':1}
>>> w2 = {'c':1, 'a':1, 't':1}
>>> w3 = { 'a':1, 'r':1, 't':2}
>>> w1==w2
True
>>> w1==w3
False
```

Question 2 [35 marks]

Write a program called ‘`tracer.py`’ that may be used to assist with debugging Python programs.

- Given the name of a Python program as input, ‘`tracer.py`’ will insert a trace statement at the beginning of each function definition.
- Given the name of a program that already contains trace statements, ‘`tracer.py`’ will remove them.

Say, for example, we have the following program, ‘`rfunction.py`’:

```
# reverse a string, Hussein Suleman, 18 march 2015.

def reverse_string (sentence):
    new_sent = ""
    for i in range (len (sentence)-1,-1,-1):
        new_sent = new_sent + sentence[i]
    return new_sent

def main ():
    sent = input ("Enter a sentence: ")
    print (reverse_string (sent))
    print (reverse_string (sent+sent))

main()
```

Here’s a sample of its behaviour:

```
Enter a sentence: how now brown cow
woc nworb won woh
woc nworb won wohwoc nworb won woh
```

Given the file name ‘rfunction.py’ as input, ‘tracer.py’ will produce the following transformation:

```
"""DEBUG"""
# Reverse a string, Hussein Suleman, 18 march 2015.

def reverse_string (sentence):
    """DEBUG""";print('reverse_string')
    new_sent = ""
    for i in range (len (sentence)-1,-1,-1):
        new_sent = new_sent + sentence[i]
    return new_sent

def main ():
    """DEBUG""";print('main')
    sent = input ("Enter a sentence: ")
    print (reverse_string (sent))
    print (reverse_string (sent+sent))
```

The program inserts the docstring """DEBUG""" at the start of the file. After the signature line of each function it inserts the same docstring followed by a semicolon, followed by a statement that prints the name of the function.

Running the transformed ‘rfunction.py’ (with the same input as before) produces this transcript:

```
main
Enter a sentence: how now brown cow
reverse_string
woc nworb won woh
reverse_string
woc nworb won wohwoc nworb won woh
```

The transcript now shows what function was executed at which point.

Given ‘rfunction.py’ as input for a second time, ‘trace.py’ will return the text to its original form.

To complete the example, here is the expected user interaction for each execution of ‘trace.py’:

```
***** Program Trace Utility *****
Enter the name of the program file: rfunction.py
Inserting...Done

***** Program Trace Utility *****
Enter the name of the program file: rfunction.py
Program contains trace statements
Removing...Done
```

Question 3 [30 marks]

Write a program called '`anagramsets.py`' that asks the user to enter a word length and a filename. The program will search the '`EnglishWords.txt`' file (of question 1) for sets of words that are that length and are anagrams of each other, and will write the results to a file with the given filename.

Sample User I/O:

```
***** Anagram Set Search *****
Enter word length:
12
Searching...
Enter file name:
twelve.txt
Writing results...
```

Expected output to '`twelve.txt`':

```
['abolitionism', 'mobilisation']
['accouterment', 'accoutrement']
['alterability', 'bilaterality']
['amphitheater', 'amphitheatre']
['behaviourism', 'misbehaviour']
['commissioned', 'decommission']
['conservation', 'conversation']
['discreetness', 'discreteness']
['impressively', 'permissively']
['inactivation', 'vaticination']
['microcephaly', 'pyrochemical']
['paradisaical', 'paradisiacal']
['restrengthen', 'strengthener']
['unimpressive', 'unpermissive']
```

NOTE: Results are presented in alphabetical order.

Submission

Create and submit a Zip file called '`ABCXYZ123.zip`' (where ABCXYZ123 is YOUR student number) containing `anagramsearch.py`, `tracer.py`, and `anagramsets.py`.

END