CSE 20
Beginning Programming in Python
Programming Assignment 6

In this project you will write a Python program that creates a dictionary whose values are lists of anagrams of words read from a file. The program source file for this assignment will be called Scrabble.py, since it is designed to be of use in the game of Scrabble. An *anagram* of a string is simply a rearrangement (or permutation) of the characters in that string. Two strings are said to be *anagrams* (of each other) if some rearrangement of the characters in one string transforms it into the other. Another way to define this relation is to require the two strings to have the same characters with the same frequencies.

For instance, the strings 'abacas' and 'casaba' are anagrams because they each contain three a's, one b, one c, and one s. Note that nothing in this definition requires the two strings to be English language words, although both are in this case. We could just as well say that 'abacas' is an anagram of 'aaabcs', which is not a word. This suggests an easy way to tell if two strings are anagrams. Rearrange the characters in both strings so that they appear in alphabetical order. If the resulting strings are identical, then the two original strings were anagrams. This is in fact where the example 'aaabcs' comes from.

	alphabetize	
'abacas'	→	'aaabcs'
'casaba'	→	'aaabcs'

We define the *norm* of a string s, denoted $\mathrm{norm}(s)$, to be the string obtained by alphabetizing its characters. Then two strings s and t are anagrams if and only if $\mathrm{norm}(s) = \mathrm{norm}(t)$. You will write a function with heading

```
def norm(s):
```

that takes a string `s` as input and returns the norm of `s`.

These norms will serve as **keys** in the Python dictionary that you will construct. A **value** in this dictionary will be a list of words all having the same norm, which will be the corresponding key. For instance, the dictionary will contain the following key-value pairs (along with many others.)

key	value
'aaabcs'	['abacas', 'casaba']
'cdeinrs'	['cinders', 'discern', 'rescind']
'cinos'	['cions', 'coins', 'icons', 'scion', 'sonic']
'eeppprs'	['peppers']
'eimrt'	['merit', 'miter', 'mitre', 'remit', 'timer']
'abdeirs'	['abiders', 'braised', 'darbies', 'seabird', 'sidebar']
'aabdeegnt'	['abnegated']
'defilt'	['flited', 'lifted']
'hilnoopsstxy'	['xylophonists']
'aehinntx'	['xanthein', 'xanthine']

Where will all of these words come from? The file scrabble.txt is posted on the class webpage in /Examples/pa6, and contains 178,691 words, one word on each line. These words are considered to be valid in a the game of Scrabble, and in crossword puzzle solutions. Download scrabble.txt and place it in the directory in which you are buliding this project.

You will write another function with heading

```
def AnagramDictionary(f):
```

that takes as input a file object `f`, then returns a dictionary whose keys are the norms of the words in `f`, and whose values are lists of anagrams, as above. Study the example FileCopy.py, also in /Examples/pa6, to see how to iterate over the lines in a file. This function will get each line in `f`, strip off the newline character to obtain a single word, compute the norm of that word, then append the word to one of the lists (values) in the dictionary, or create the list if necessary. See the example DictionaryFunctions.py in /Examples to learn how to assemble a dictionary. A program called TestAnagramDictionary.py is included in /Examples/pa6 which is a weak test of your function AnagramDictionary(). You should of course conduct your own more stringent tests before you submit this program.

Function main() will open the file scrabble.txt, then pass the resulting file object to AnagramDictionary() to create a dictionary object. It will then enter a loop that repeatedly prompts the user for a string, queries the dictionary, then prints out all the anagrams of that string. The entered string s need not be a word in the dictionary, or even the norm of such a word, but if $\text{norm}(s)$ is not a key, then a message to that effect will be printed. The user will enter an empty string to quit the loop. A sample interactive session is included below.

```
$ python3 Scrabble.py

Enter a string (or nothing to quit): happy

The anagrams of happy are:
happy

Enter a string (or nothing to quit): squidly

The letters in 'squidly' do not form a word in the dictionary

Enter a string (or nothing to quit): squid

The anagrams of squid are:
quids, squid

Enter a string (or nothing to quit): stable

The anagrams of stable are:
ablest, bleats, stable, tables

Enter a string (or nothing to quit): aeistilors

The anagrams of aeistilors are:
solitaires, solitaries

Enter a string (or nothing to quit):

Bye!

$
```

As usual your output must match that above exactly to obtain full credit. Notice that the last input string 'aeistilors' is neither a word, nor a key in the dictionary, but norm('aeistilors') = 'aeiilorsst' is a key. See the file anagrams.txt in /Examples/pa6 to get a good picture of the dictionary your program creates.

There is one more required function for this project that will be helpful in producing the above output. Its heading is

```
def printWordList(L):
```

This function will print the words in list `L` on a single line, separated by commas, and ending with a newline character. Below is a rough outline of function main() for this project.

```
def main():

   # open a word file, assemble the dictionary using words, close file

   # repeatedly get input from user, and query the dictionary

      # get a string from the user

      # look up string in the dictionary, print its anagrams if they exist

   # end while

# end main()
```

As usual, you should follow /Examples/GeneralTemplate.py when building this project. I've also included a file called ScrabbleTemplate.py in /Examples/pa6 that you can use as a starting point if you like.

This project is perhaps not as difficult as the last one, but nevertheless, be sure to start early if you think you will need help from TAs, RTs or myself. When many students are seeking help at the last minute, there is invariably not enough help to be had. Submit the file Scrabble.py to Gradescope before the due date. Be sure to *not* submit any other files, especially the very large files used in this project.