Elvis Lives! … Anagram Finder

anagram |ˈanəˌgram|, noun
a word, phrase, or name formed by rearranging the letters of another, such as *cinema*, formed from *iceman*.

Write a Python program to find all possible anagrams of a given input string. A "valid" anagram must use all letters in the input string. Your program should (flexibly) work with different possible dictionaries (files) of English words.

Input (`stdin`, keyboard):
A user enters a text string: word, phrase, or sentence *or alternatively during experimental runs* a file of words, phrases, or sentences to repeatedly test one at a time

Output (`stdout`, console):
A list of unique anagrams formed from the input string and total anagrams found *or alternatively during experimental runs* a file of anagram possibilities for each input string.

Sample Runs

```
def main():
  a = Anagram("wheaton", "english.txt")
  a.findAnagrams()
                                          … partial output …
      [039]: ['ha', 'not', 'we']
      [053]: ['heat', 'won']
      [067]: ['noah', 'wet']
      [070]: ['oh', 'we', 'tan']          (words rearranged here to read better)
      [072]: ['one', 'thaw']
      [073]: ['what', 'one']              (words rearranged here to read better)

      Total Found: 77
```

```
a = Anagram("wheaton college", "english.txt")

      Teaching related                    (words rearranged here to read better)
      [9041]:   we challenge too
      [17750]:  teach well, no ego
      [19558]:  go teach one well

    Genomics related
      [8548]:   ha, we got one cell
      [19003]:  we hot local gene
      [19018]:  whole octal gene

    Random
      [11349]:  on gale, we clothe
```

Public Relations
On some assignments (like this one!), I will ask you to *promote* our AI course, the assignment, and/or your solution. In general, you should consider yourself part of a small start-up company that in addition to cutting code, needs to promote itself. Public Relations (PR) is everyone's job. In this case, you earn points on this assignment by promoting anagrams, the algorithm(s), and/or solutions. **Once you decide on your idea for PR, please run it past me before you do it.** Innovative ideas score highest. I will take "amount of effort" into consideration when grading. For example, you might do:

(i) write a story in the Wheaton Wire about finding anagrams
(ii) collect anagrams into a children's book (perhaps influenced by Jon Agee's books)
(iii) Conduct a literature review and summary of algorithms for finding anagrams
(iv) Create a Word Cloud of the anagrams found on certain words and find a creative way to share
(v) *<insert your cool idea here>*

Algorithm
I expect you to come up with your own, unique solution. In class, I will share some insights … and together we will share some of the details of our algorithms. You should expect to work on paper for a considerable length of time, slowly refining your algorithm. To start, find some simple anagrams on paper. How did *you* do it? What data structures will you need to help you solve the problem? (Read that last sentence again!)

Hints:

(a) Make a *very (very)* small dictionary file for testing purposes, e.g., to test anagrams of "Elvis", use a dictionary of words that only includes: [Bob, Elvis, Eli, vs, lives, done]. (See the StarterKit for such a test input file.)
(b) Remove all punctuation and whitespace from the starting, input string.
(c) We *love* regular expressions, right? `^c.rre.?t$`
(d) Recursion is your friend. On successful recursive calls, the possible words still available should shrink, right?
(e) In one solution, I used the following data structures: (i) **list** (for subset of words in the dictionary that are potential matches to be used in an anagram), (ii) **dictionary** (for keeping track of letters not yet used in my current anagram solution attempt), and (iii) **set** (for identifying unique solutions).
(f) I encourage you to use Python classes (objects). Originally, I just wrote a collection of functions, but my code quickly turned into a mess. My solution was much cleaner once I made an Anagram object and started to refer to "private" (self) data members in my methods, e.g., `self.originalText, self.goodAnagramAnswers, self.totalFound.` See `anagramMain.py`

Experiment(s)
So, your program runs … (*yawn*) … um, I mean "Great!" Now the important (empirical) work begins. Decide on a set of words or phrases and search for "cool" anagram (solutions). Your program should produce results that will appear in a Final Report (a professional looking **.pdf** file) that gives (a) summary of the program/task, (b) input set(s), (c) the total runtime for finding all anagrams in your input set(s), (d) sample solutions (anagrams), (e) your "fav" anagrams. Remember: your Final Report *is* the most significant part of your submission!

Submission: .zip of a folder (named with YourLastName.zip) containing `_README.txt` , *documented* source code (use Python's triple-quote `"""`class and method documentation), input sets, sample output, and Final Report (.pdf). Please submit a hardcopy of your Final Report.

Grading

Average (C):	you get the algorithm working; show *correct* results, success on various trials, for example, finding anagrams of "Wheaton"
Above Average (B):	you use Python objects, you perform a Public Relations effort, run experiment(s)
Superior (A):	you time experimental runs, *then* optimize your algorithm to speed it up, commenting on improvements made, etc.