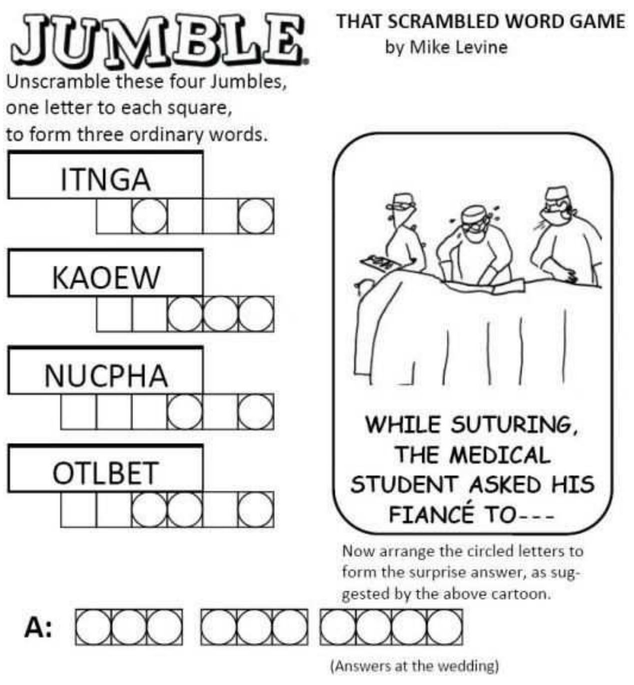
Computer Science I CSCI-141
Anagrams and the Word Jumble Homework 9

10/07/2020


JUMBLE
THAT SCRAMBLED WORD GAME
by Mike Levine
Unscramble these four Jumbles, one letter to each square, to form three ordinary words.
ITNGA
KAOEW
NUCPHA
OTLBET
WHILE SUTURING, THE MEDICAL STUDENT ASKED HIS FIANCÉ TO---
Now arrange the circled letters to form the surprise answer, as suggested by the above cartoon.
A:
(Answers at the wedding)

1 Introduction

In this homework you will use a Python dictionary to help answer questions related to anagrams in the English language. An anagram is a word formed by rearranging the letters of another word. In the case of the Jumble word game illustrated above, the given sequence of characters don't form a word, but can be rearranged to form a word.

2 Provided Data

Provided in the same folder along with this homework document is the following file: `american-english.tx`
This file was copied from the CS machines at: `/usr/share/dict/american-english`.

It contains a listing of American English words, one per line. This file contains more than we need - there are capitalizations and apostrophes and other special characters. That's ok. **You can ignore that detail for this assignment.**

3 Tasks

You will write a Python program that completes these four tasks.

3.1 Task 1

Create a project and the Python file named `anagram.py`.

Read and store the `american-english.txt` word file into a Python dictionary. What are the appropriate key and value for the dictionary to help enable analysis related to anagrams?

For keys, use the lexicographical ordering of the characters in each word. For example, if you read the word "bread", it would form the lexicographical ordering "abder", and you would use "abder" as the key.

For the value, store a list of all American English words formed from those letters (using all of them). For example, if the first word encountered is the word "ate", you will create this entry in your dictionary: `'aet': ['ate']`. Later, after you encounter the words "eat", "eta", and "tea", the dictionary will contain the entry: `'aet': ['ate', 'eat', 'eta', 'tea']`.

The following steps can convert a word to its lexicographically ordered version. Each of these steps can be done with one single line of code, but you don't have to do it with a single line.

1. Convert the string to a list of individual characters.
2. Sort the list of characters lexicographically. You may use the built-in list `sort` function.
3. Join the sorted characters of the list back into a single string.

The `str.join` function is a bit odd; it returns a string from its list argument, but you must call it on a string object as this example shows:

```
>>> lst = ["b", "o", "o"]
>>> value = "".join( lst)
>>> value
'boo'
>>>
```

3.2 Task 2

Prompt the user to enter a string. The string might or might not correspond to an existing word. Determine and print all American English anagrams of the given string.

3.3 Task 3

For words of a given length, find the longest list of words that are all anagrams of each other.

Prompt the user to enter a positive integer for the word length, and use it to identify the maximum size list of words that are anagrams of each other. Print the size of the list, and print the list of words itself. See the output in Section 4 for an example.

There will be only one correct numerical output value, but there could be multiple correct output lists of words. For example, given user input of the word length `8`, there is only one correct numeric answer: `5`. Depending on the order in which the elements of the dictionary are processed, the program might report the list `['restrain', 'retrains', 'strainer', 'terrains', 'trainers']` or the list `['alerting', 'altering', 'integral', 'relating', 'triangle']`.

3.4 Task 4

Prompt the user to enter a positive integer, corresponding to a word length. Identify how many words in the American English text file of that word length qualify as *good Jumble words*. To be a good Jumble word, the word should have no anagrams that are also in the American English text file. (This guarantees that if the letters to that word are scrambled, there is only one correct way to unscramble them.) Print the number of good Jumble words of the specified word length.

4 General Requirements and Example Output

Do not prompt for the name of the input file; hard-code the string `american-english.txt` instead. The program does not need to perform any error handling for bad user input.

Because the input file contains *Unicode characters*, you need to open the file with the proper encoding; here is an example using `with` syntax:

```
with open( filename, encoding="utf-8" ) as fp:
```

Design the program so that each individual task (2, 3 or 4) repeats as often as the user desires, until the user enters a specified input to signal completion of that task.

The example output below illustrates this requirement. The output does not need to exactly match this, but should provide the same information and capability.

```
Enter input string (hit enter key to go to task 3): pleta
Corresponding words: ['leapt', 'petal', 'plate', 'pleat']
Enter input string (hit enter key to go to task 3): fwetora
No words can be formed from: fwetora
Enter input string (hit enter key to go to task 3): branch
Corresponding words: ['branch']
Enter input string (hit enter key to go to task 3):

Enter word length (hit enter key to go to task 4): 4
Max anagrams for length 4: 6
Anagram list:  ['opts', 'post', 'pots', 'spot', 'stop', 'tops']
Enter word length (hit enter key to go to task 4): 9
Max anagrams for length 9: 3
Anagram list:  ['dissenter', 'residents', 'tiredness']
Enter word length (hit enter key to go to task 4):

Enter word length (hit enter key to quit): 5
Number of jumble usable words of length 5: 5046
Enter word length (hit enter key to quit):
```

5 Grading

- 25%: Task 1
- 20%: Task 2
- 50%: Tasks 3, 4
- 5%: Appropriate repeating user prompts in the `main`.

Style flaws, documentation errors and omissions, or misnamed files will incur a 10% grade deduction.

6 Submission Instructions

Zip the `anagram.py` file, and name the zip file: `hw09.zip` and submit it to the appropriate MyCourses dropbox by the deadline. **Use zip only. Do not use another compression mechanism.**