**CS 2302 Data Structures**

**Spring 2019**

**Lab Report #1**

Due: September 9th, 2019

Professor: Olac Fuentes

TA: Anindita Nath

**Introduction**

Lab #1’s purpose was to find anagrams of a word that is specified by the user. These anagrams are located in a large text file, and the file had to be read to find all possible anagrams of different words. Along with this lab came the use and importance of sets, a data structure that holds the anagrams in this lab.

**Proposed Solution Design and Implementation**

**Part 1:** To solve this anagram problem, I divided the program into three different parts for every operation that needed to be done. I first began by creating a function that reads the text file that is given. I opened the file, then went line by line and added each word to a set. At the same time, I also added each word to a prefix set that another function creates by building prefixes of each word. I also made the program so that the user is prompted with a question that asks for a word so that anagrams can be found. I made sure that the word that the user submitted exists within the word set while reading the file. If it did not exist, then part 2 was executed, however zero anagrams are found, and the user is notified that the word does not exist within the text file. At the same time, a timer is started to time the execution of the program.

**Part 2:** For the second part, I used a function that scrambles and creates permutations based off of the word that the user chose. I modified code from the Zybooks textbook which contained a function called scramble. I modified the code by adding checks that make sure that the prefixes of each word exist in the prefix set. This allowed me to complete the optimization in part 2 of the lab where recursion needed to be stopped if the partial word was not in the word set. I also checked if the completed, scrambled word was in the original word set to make sure that it is an anagram. If the word was in the word set, it was sent to another function that checks that the original anagram is not printed and that adds the anagrams to a set.

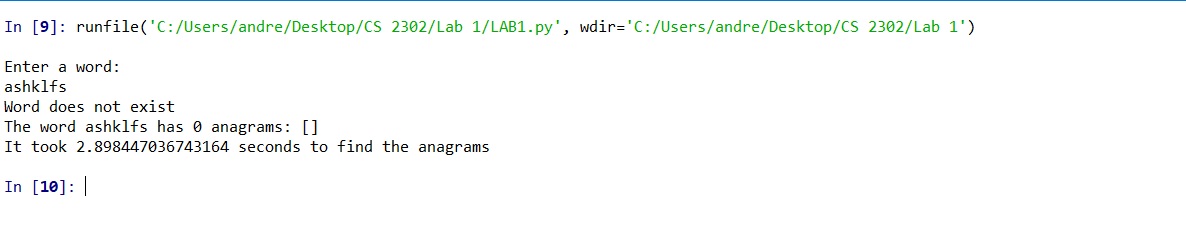
**Part 3:** For the third part, I created a function that checks for duplicates in every word. I did this by modifying the scramble method further and sending every character that is found to a function that checks for duplicates. I added the character to a set and checked if the character that was sent to the function already existed within the set. If it was found to be already in the set, then a duplicate existed. If a duplicate is found, the recursive call to search for anagrams is not done since that will cause the same anagram to be generated.

**Experimental Results**

**Part #1:**

For the first part I tested it by entering a word that does not exist within the text file. This determines if the program will handle the problem correctly, and to see if any anagrams are still generated despite the word not existing within the file.

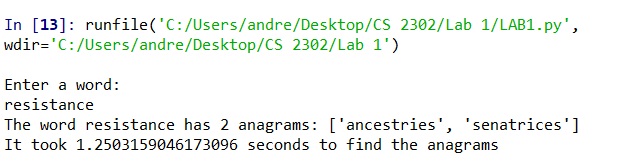
**Test 1: Word does not exist -** No error, the program informed the user that the word did not exist, and did not generate any anagrams.

****

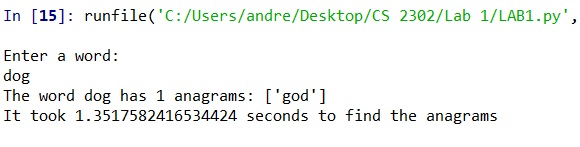
**Part #2:**

I tested the second part by entering very large words, and very small words. This will allow me to determine how fast or how slow the program finds the anagrams, no matter how long or how short the word is.

**Test 1: Long word -** Despite being a long word, the program only took about 1.25 seconds to find the anagrams.

****

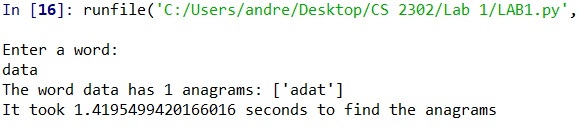
**Test 2: Short word -** The shorter word took about the same time as the longer word. The time stays pretty consistent no matter what the word is as long as it exists within the file.

****

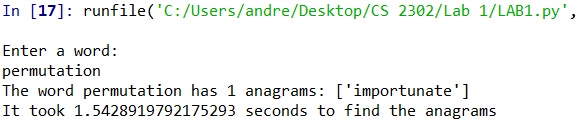
**Part #3:**

The third part was tested with words that contained duplicate characters. The words that were tested were “data”, and “permutation”. This test determines whether recursion is stopped to prevent the same anagrams from being generated each time.

**Test 1: The word “data” -** Extra anagrams were not printed even though the word contained duplicate characters

****

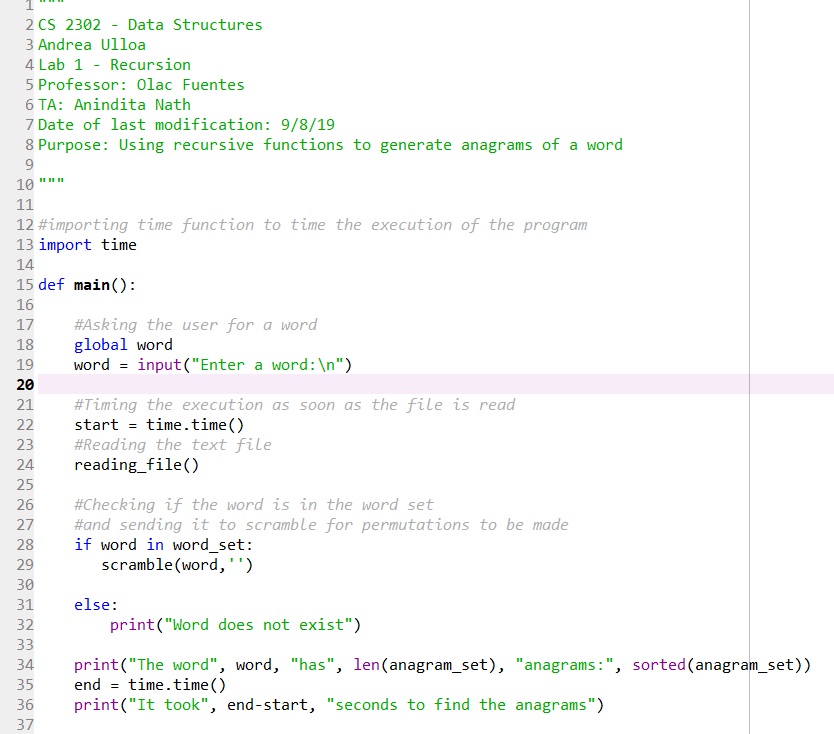
**Test 2: The word “permutation” -** This word also contained duplicate characters, however it did not have any repeating anagrams.

****

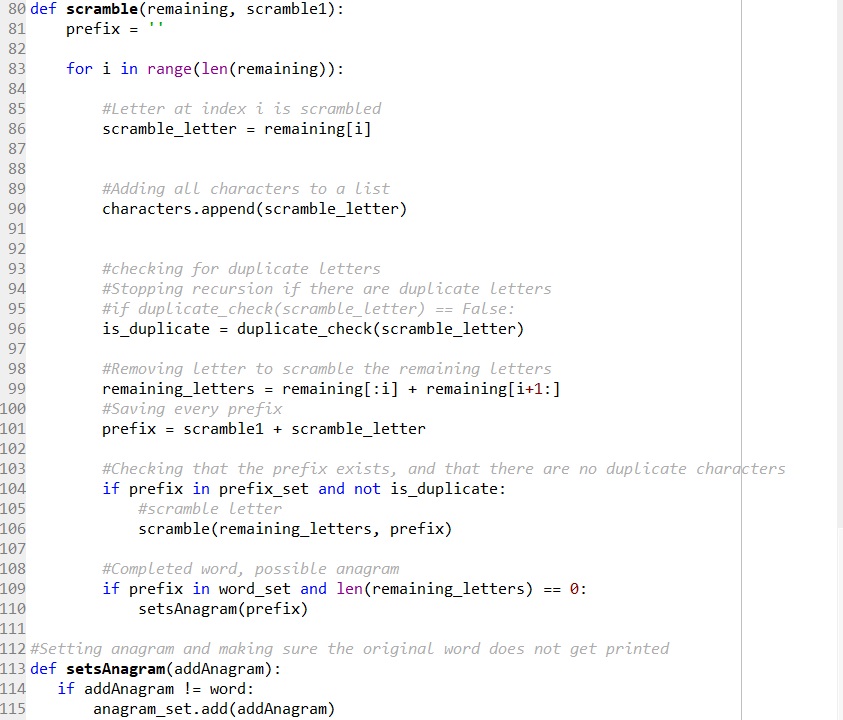
**Conclusion**

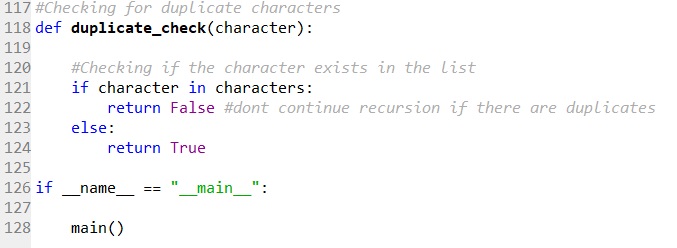
By completing this project, I learned that recursion is a better way to make your program shorter. This program helped me think of how to solve problems with recursion instead of with iterative solutions. I also learned a lot more about Python and how it makes programming much easier and reduces the amount of code in a program. Creating different functions for every operation also helped me understand recursion. I got a better understanding of base cases and what to incorporate within base cases so that the program is more efficient. I also learned about sets and how to manipulate them without having access to each element.

**Appendix**

****

****

****

****

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.