**CS 2302 Data Structures**

**Spring 2019**

**Lab Report #1**

Due: September 6th, 2019

Professor: Olac Fuentes

TA: Anindita Nath

Author: Kevin Ramirez

**Introduction**

For our first lab assignment, the objective was to create an anagram generator, that is Recursively implemented. There are two part to this assignment, the first part will ask the user for a word, it will use this word to compare it to multiple anagrams from a txt file. If there is any anagram in the txt file it will return that anagram, and continue its search until it finds another one, or the list runs out. For the second part, it will ask the user for a word and instead of looking through a list, it will generate all the possible anagrams for this word.

**Proposed Solution Design and Implementation**

**Operation #1:**

The first operation, I first made a method called “Is\_anagram”, the basic function of this method is to turn two word into a list and compare the two list to see in they have the same length and letter on the same spot. If not, then they are not anagrams, so it would return false. Next the second method named “Done\_anagram”, it takes two lists, one is the word that the user input, and the other is reserved for each of the 400,000 words. Base case, if the list of all words from the txt run out, and the list is empty return an empty set. Else, use the method “Is\_anagram” to check if both the user’s word, and the one being held, are anagrams. Then just before calling the recursive method, start your timer, and once it is done subtracts the starting time minus the end time.

**Operation #2:**

For the second part of this lab, I decide to make a method, “Anagram\_Permutations”, that required no other methods, and instead of using list or sets, I was just going to use the user’s input as a string. For the base case it was simple, if the length of the string was 0, return an empty set. Next, I had a “holder” set that would gather all the permutations of the user’s string. Next, we use the for loop in order to save all the permutations that start with the first letter of the original word, once this is done, we push that first letter to the back of the of the string, and now the first letter of the word is the second one. It repeats this process until it has used all the letters from the word. All the new permutations are being saved by the holder, and once done we return the holder set. Set the timer and once the method is done, print the time it took.

**Operation #3:**

Lastly, we needed a more optimize version of the second part of the lab, so I decide to follow the example from the zybook, on the recursion chapter. Following this example, I implemented a method, “Anagram\_Final”, that did the same thing as the previous method. The base case was the same as the others, if the user’s input string was zero, print an empty string, then by using a for loop we could obtain all the permutations for each of the letter in the word, without losing any previous letters.

Example:

Presto 🡪 Restop 🡪 Estopr

| | | | |

V | V | V

All permutations | All permutations | All permutations

of the letter ‘P’ | of the letter ‘R’ | of the letter ‘E’

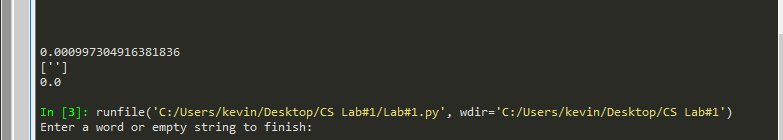
This process happens with the rest of the letters, and then it would print out all of the permutations once the base case was reached.

**Experimental Results**

**Operation #1:**

My test cases for this first part were, an empty txt file, the full txt file, and a smaller txt file. With this I tried to hard test all the options and see if it would be better or worse over time. Also, for the user’s input test, it tried the empty string, “presto”, and “abcd”

Case #1: The empty set, for the first method it should return nothing



Case #2: The full txt file, and presto word

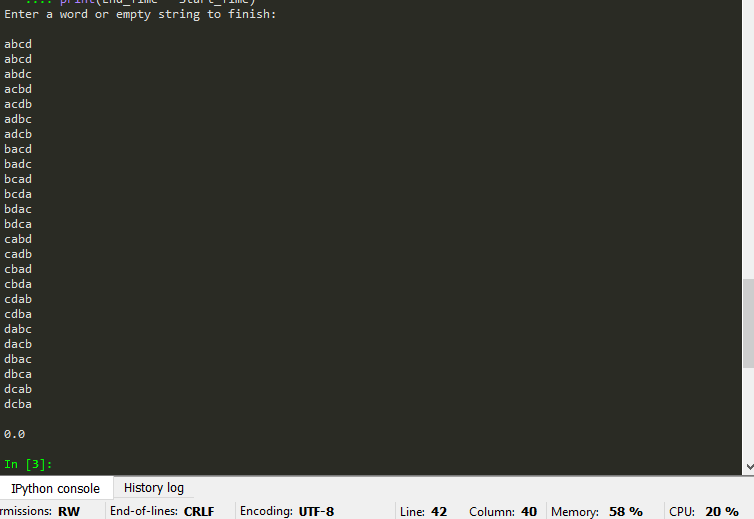
Sadly, for this case I had a lot of troubles, every time I would run the “Done\_method” my laptop would slow down to a crawl and crash, so I was not able to record any data for this case, my code is just sloppy and badly optimized.

Case #3: Smaller txt file, and “abcd”

This time, instead of using all 400,000 words from the original txt file, I only used 25 from all the permutations of the string “abcd”, and we are still using “Done\_anagram”, but this time it wont crash my laptop.



Smaller txt file (25 permutations)



Results

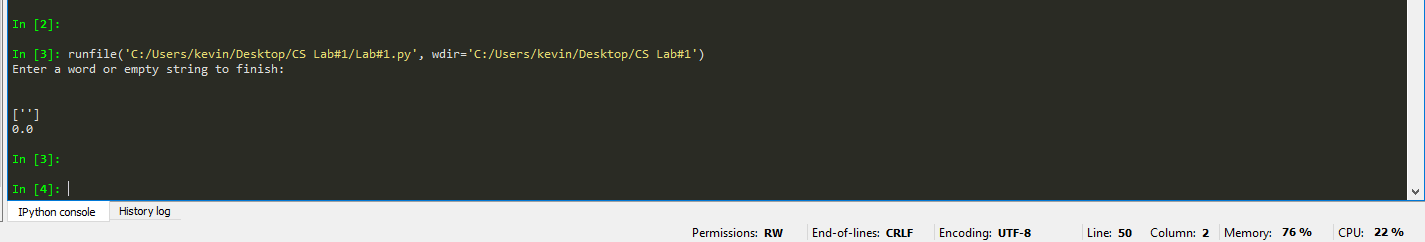
User input: “abcd”

Time: 0.0

**Operation #2:**

For the next operation I tested the following: Empty string, and “presto”

Case #1: The empty string.

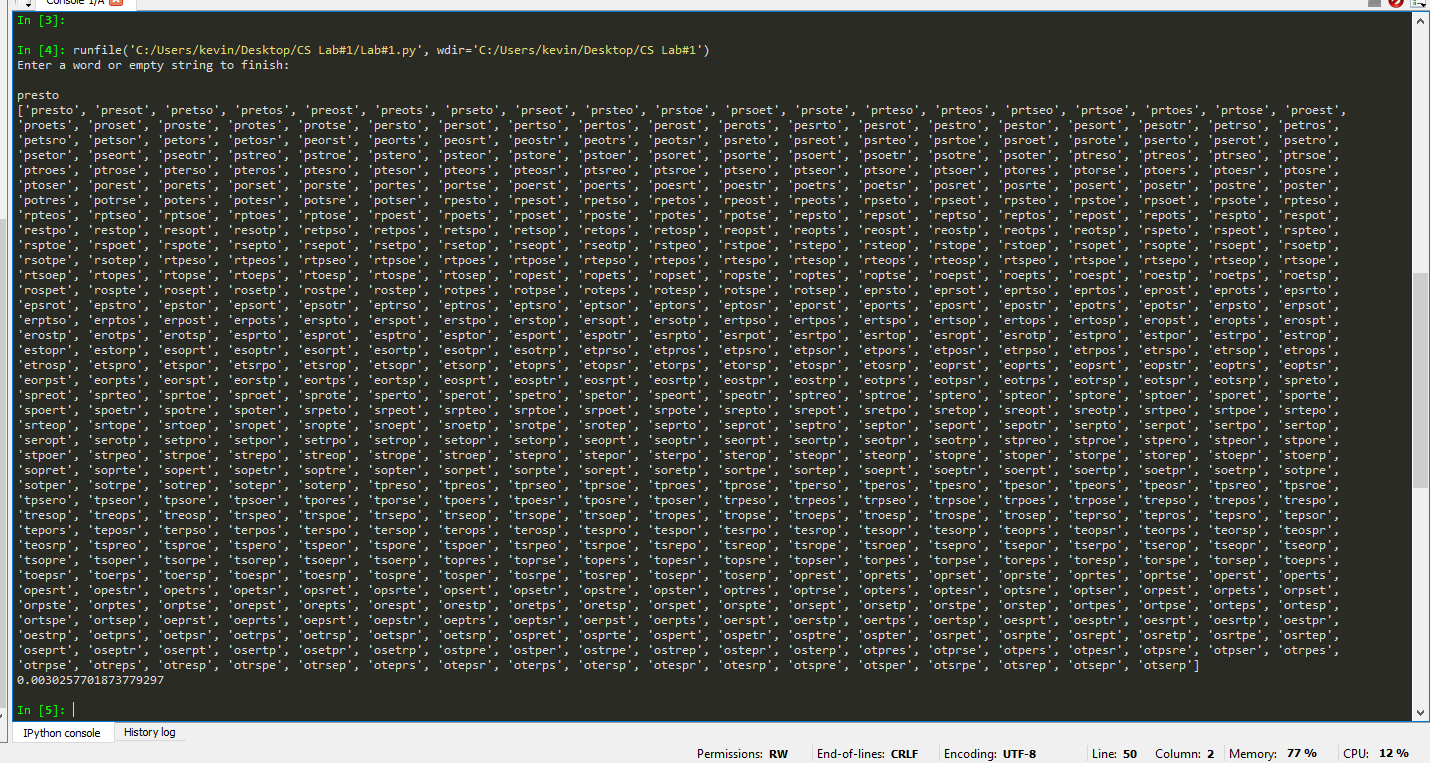


As expected, the method returns and empty set.

User Input: empty string

Time: 0.0

Case #2: “presto”



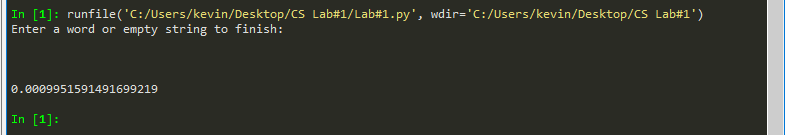
User Input: presto

Time: 0.0030

**Operation #3:**

For the final operation, test cases will be the same as for operation #2, the empty string, and “presto”.

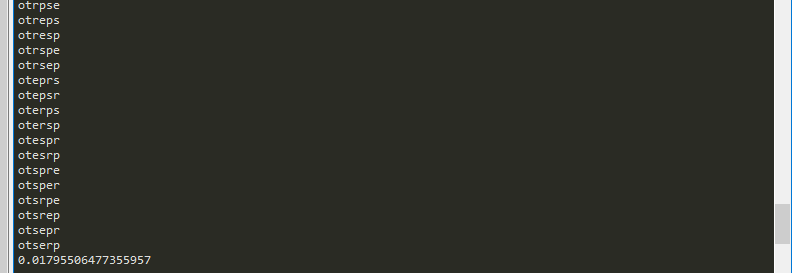
Case #1: Empty string



User Input: empty string

Time: 0.00099

Case #2: Presto



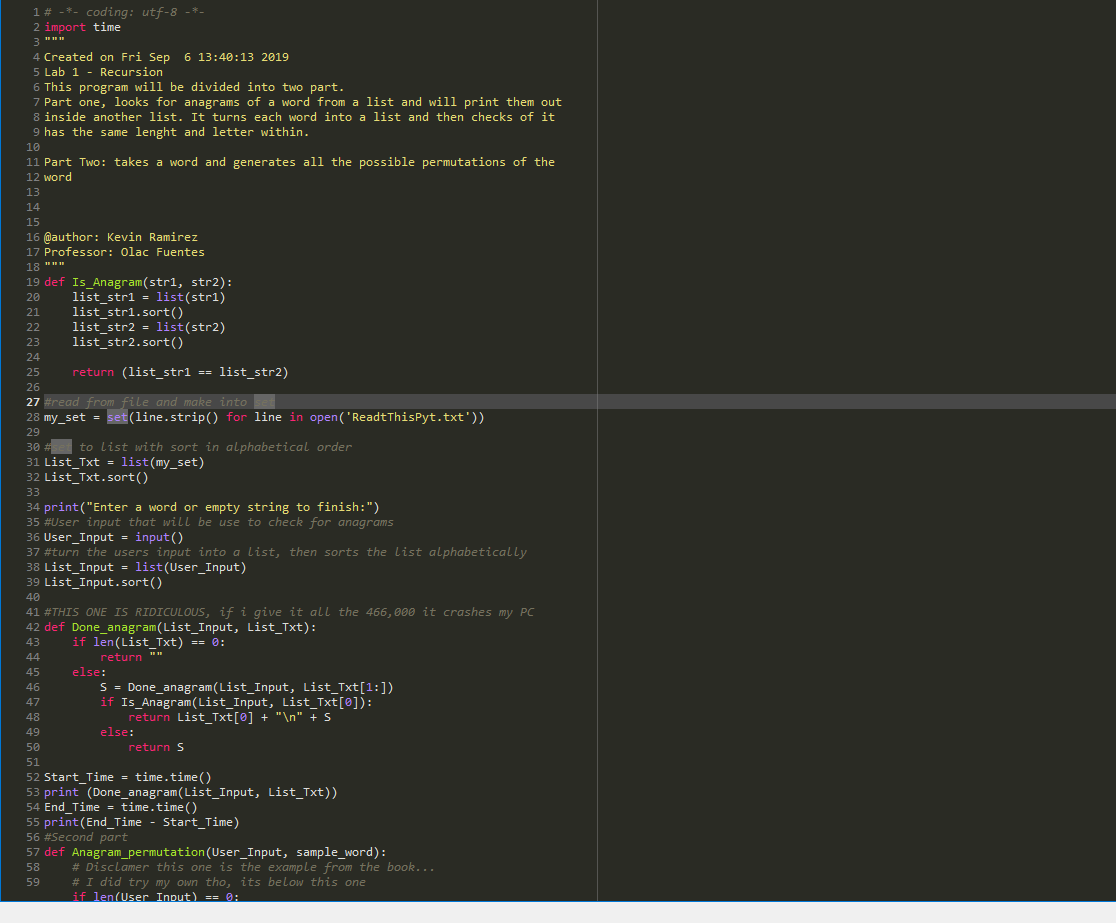
User Input: presto

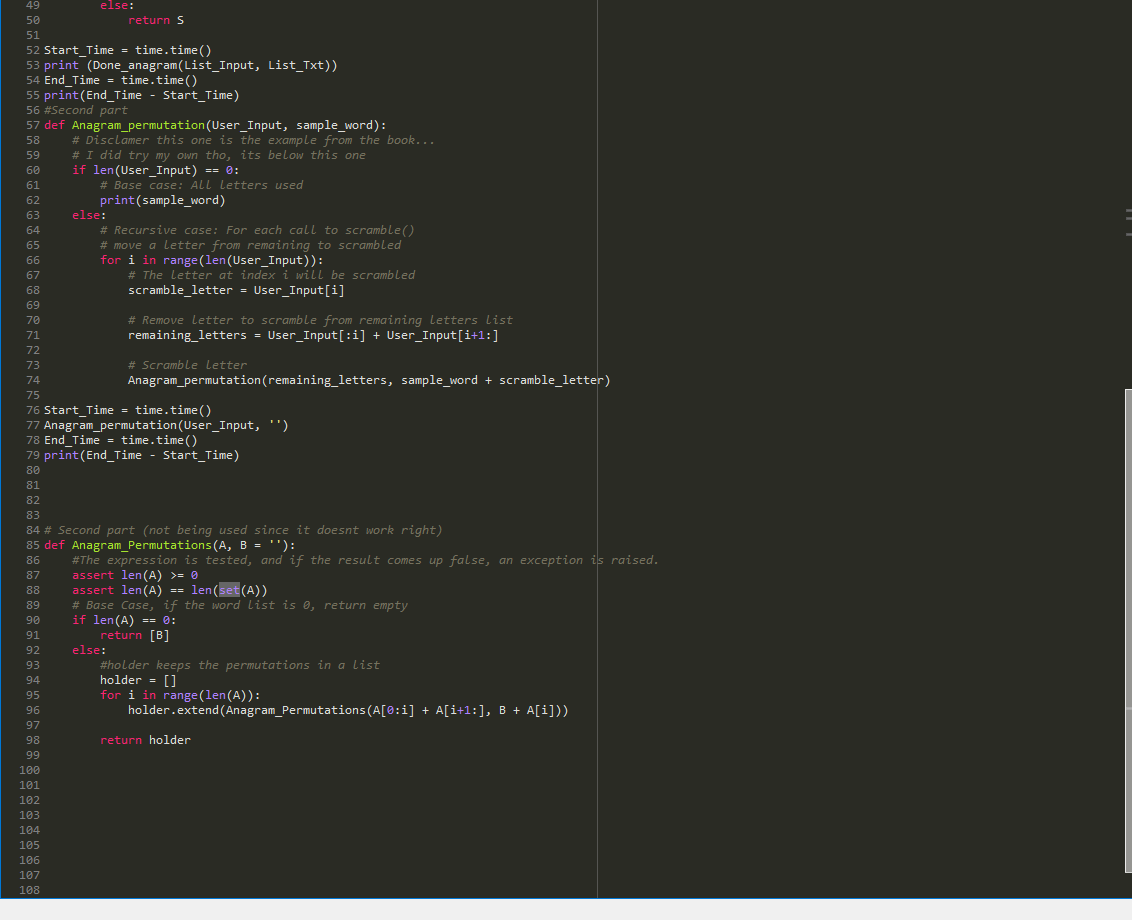
Time: 0.017

**Conclusion**

This lab helped me understand two things, I need to study hard, and that I don’t understand recursions as well as I thought. I would say that this lab was a failure, due to my bad time management, and personal situations outside of my control. Next lab I will not waste time to start it. The code itself seems pretty simple. Therefore, I have to admit that I don’t deserve a passing grade for this lab.

**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