CS 2302 Data Structures

Fall 2019

Lab Report #1

Due: September 6th, 2019

Professor: Olac Fuentes

TA: Anindita Nath

Programmer: Miriam Olague

Introduction

For this lab we were asked to find the anagrams of a word using recursion. The purpose of this lab is to get familiarized and practice recursion. The main objective of this lab is to learn how to manipulate sets and arrange the words the sets contain to obtain the anagrams of the word the user entered recursively.

Proposed Solution Design and Implementation

Part #1:

For this operation, I used the code from Chapter 2.6.1 on Zybooks as a reference. With this code I iterated through every word in ‘words_alpha.txt’ (I created it into a set). I first saved the first character of the word into a temporary variable and the remaining of the word was saved into a different variable. After I iterated through the entire word, I saved the word into a temporary set and then compared it to the original set. I then updated the set by comparing the words (all the anagrams) with the original set and see if the anagrams of the words existed inside of the original set. I also called the method (scrambled is a recursive method) and iterated through all the word set I created.

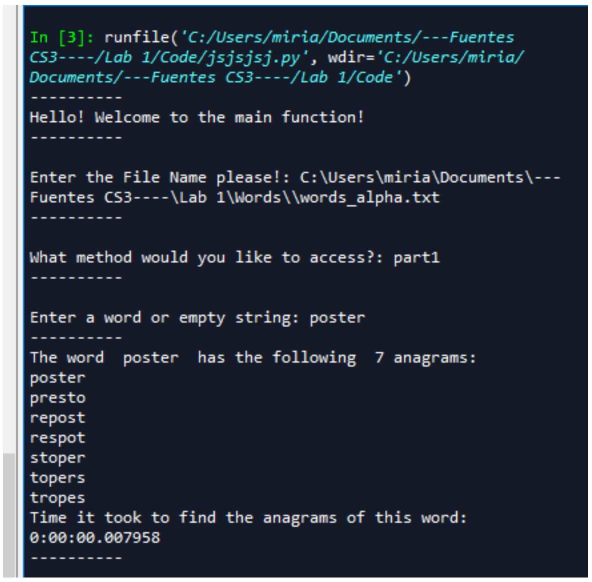
Part #2:

For this operation, I approached it in a very similar way as **Part #1**. I iterated through every word in ‘words_alpha.txt’ (I created it into a set). I created an if statement, in which I allowed it to make recursive calls if and only if the word the user inputs is inside of the set I created, in which it contains all the prefixes of the original word set (words_alpha.txt). Inside of that if statement, I also compared the first character of the word with the rest of the characters inside that word to prevent it from making recursive calls if the character repeats itself. If the conditions are true for this word, I save the first character inside of a temporary variable. Then, I saved the rest of the word inside of another temporary variable. After iterating through the entire word, I then saved the word into a temporary set and compared it to the original set. I continuously updated the set each time if the new anagram existed inside of the original word set.

Experimental Results

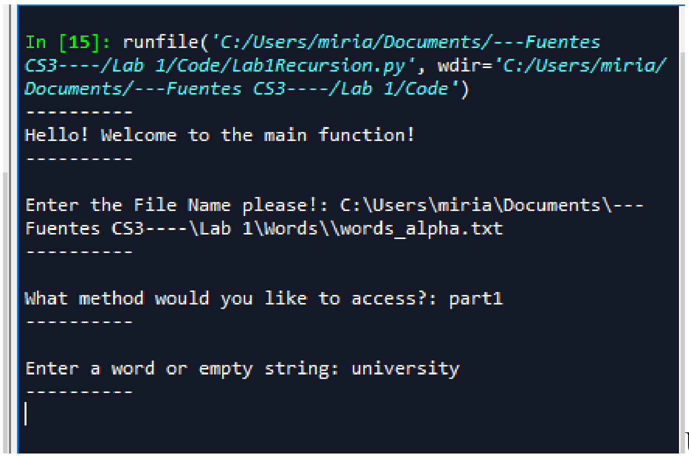
Part #1:

For this operation, I created an empty set, in which all the existing anagrams of the user’s input were added into. I first thought that I had to start the time inside of the recursion method but then realized that the time would reset. I attempted to print all of the existing anagrams inside of the recursion method, but then decided it would be better to put them in a set and then print them outside of the recursion method. I tried printing the anagrams inside of the recursion call, but I found it cleaner to print them outside.



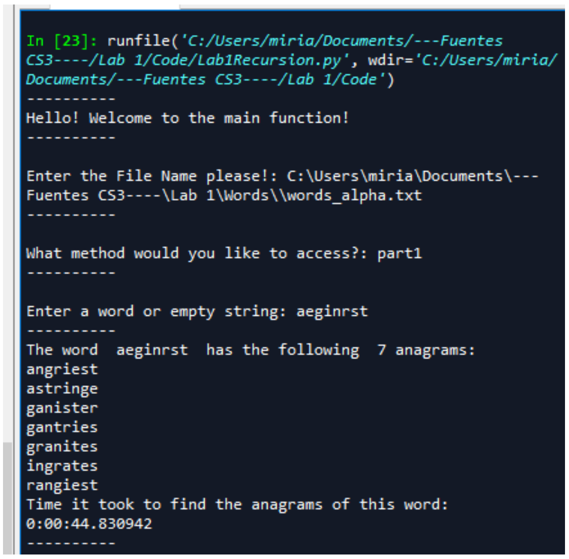
```
In [3]: runfile('C:/Users/miria/Documents/---Fuentes
CS3----/Lab 1/Code/jsjsjsj.py', wdir='C:/Users/miria/
Documents/---Fuentes CS3----/Lab 1/Code')
----------
Hello! Welcome to the main function!
----------

Enter the File Name please!: C:\Users\miria\Documents\---
Fuentes CS3----\Lab 1\Words\\words_alpha.txt
----------

What method would you like to access?: part1
----------

Enter a word or empty string: poster
----------
The word  poster  has the following  7 anagrams:
poster
presto
repost
respot
stoper
topers
tropes
Time it took to find the anagrams of this word:
0:00:00.007958
----------
```

User’s input = poster



```
In [15]: runfile('C:/Users/miria/Documents/---Fuentes
CS3----/Lab 1/Code/Lab1Recursion.py', wdir='C:/Users/miria/
Documents/---Fuentes CS3----/Lab 1/Code')
----------
Hello! Welcome to the main function!
----------

Enter the File Name please!: C:\Users\miria\Documents\---
Fuentes CS3----\Lab 1\Words\\words_alpha.txt
----------

What method would you like to access?: part1
----------

Enter a word or empty string: university
----------
```

User’s input = university (none found)

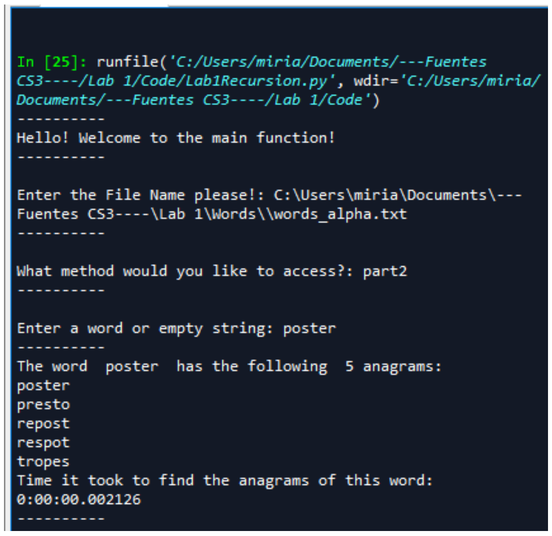


```
In [23]: runfile('C:/Users/miria/Documents/---Fuentes
CS3----/Lab 1/Code/Lab1Recursion.py', wdir='C:/Users/miria/
Documents/---Fuentes CS3----/Lab 1/Code')
----------
Hello! Welcome to the main function!
----------

Enter the File Name please!: C:\Users\miria\Documents\---
Fuentes CS3----\Lab 1\Words\\words_alpha.txt
----------

What method would you like to access?: part1
----------

Enter a word or empty string: aeginrst
----------
The word  aeginrst  has the following  7 anagrams:
angriest
astringe
ganister
gantries
granites
ingrates
rangiest
Time it took to find the anagrams of this word:
0:00:44.830942
----------
```

User’s input = aeginrst

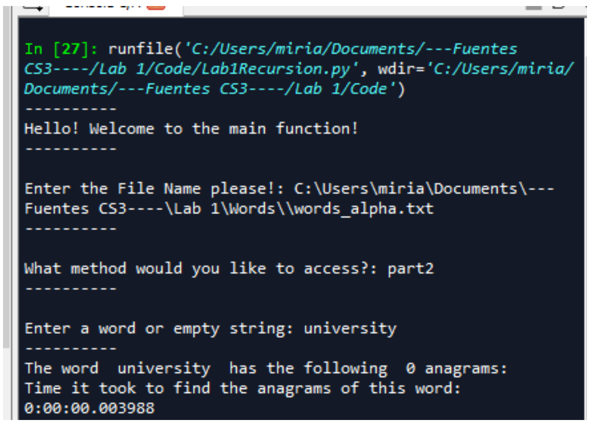
Part #2:

For this operation, I had a hard time understanding how I could compare a character with the rest of the characters. All the experiments I made were the same ones as **Part #1**. The only different experiment I had to do was how to compare the characters with the rest and how to make sure that the word was a prefix of any of the words in the set I created for prefixes. The way I created a set with all of the prefixes of the original set (words_alpha.txt) was with a list. I made a for loop in which I added the prefixes of the set and then converted it to a set in order to compare it to the original set (words_alpha.txt), however, I tried adding all of the elements that were inside the list into the set with a for loop, I also tried 'pref_set = update(pref_list)' but it would not add the elements that were inside of the list. I made it work with a list but not a set. Since many of the words that were in the list were added incorrectly into the set, it did not find all of the existing anagrams inside of the original set (words_alpha.txt).



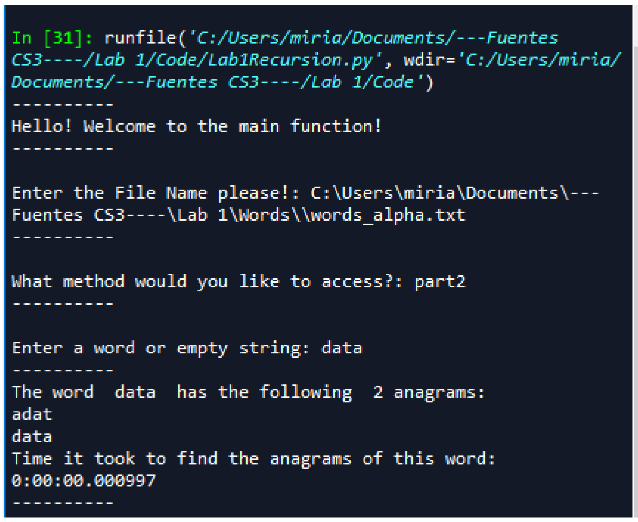
```
In [25]: runfile('C:/Users/miria/Documents/---Fuentes
CS3----/Lab 1/Code/Lab1Recursion.py', wdir='C:/Users/miria/
Documents/---Fuentes CS3----/Lab 1/Code')
----------
Hello! Welcome to the main function!
----------

Enter the File Name please!: C:\Users\miria\Documents\---
Fuentes CS3----\Lab 1\Words\\words_alpha.txt
----------

What method would you like to access?: part2
----------

Enter a word or empty string: poster
----------
The word  poster  has the following  5 anagrams:
poster
presto
repost
respot
tropes
Time it took to find the anagrams of this word:
0:00:00.002126
----------
```

User's input = poster



```
In [27]: runfile('C:/Users/miria/Documents/---Fuentes
CS3----/Lab 1/Code/Lab1Recursion.py', wdir='C:/Users/miria/
Documents/---Fuentes CS3----/Lab 1/Code')
----------
Hello! Welcome to the main function!
----------

Enter the File Name please!: C:\Users\miria\Documents\---
Fuentes CS3----\Lab 1\Words\\words_alpha.txt
----------

What method would you like to access?: part2
----------

Enter a word or empty string: university
----------
The word  university  has the following  0 anagrams:
Time it took to find the anagrams of this word:
0:00:00.003988
```

User's input = University



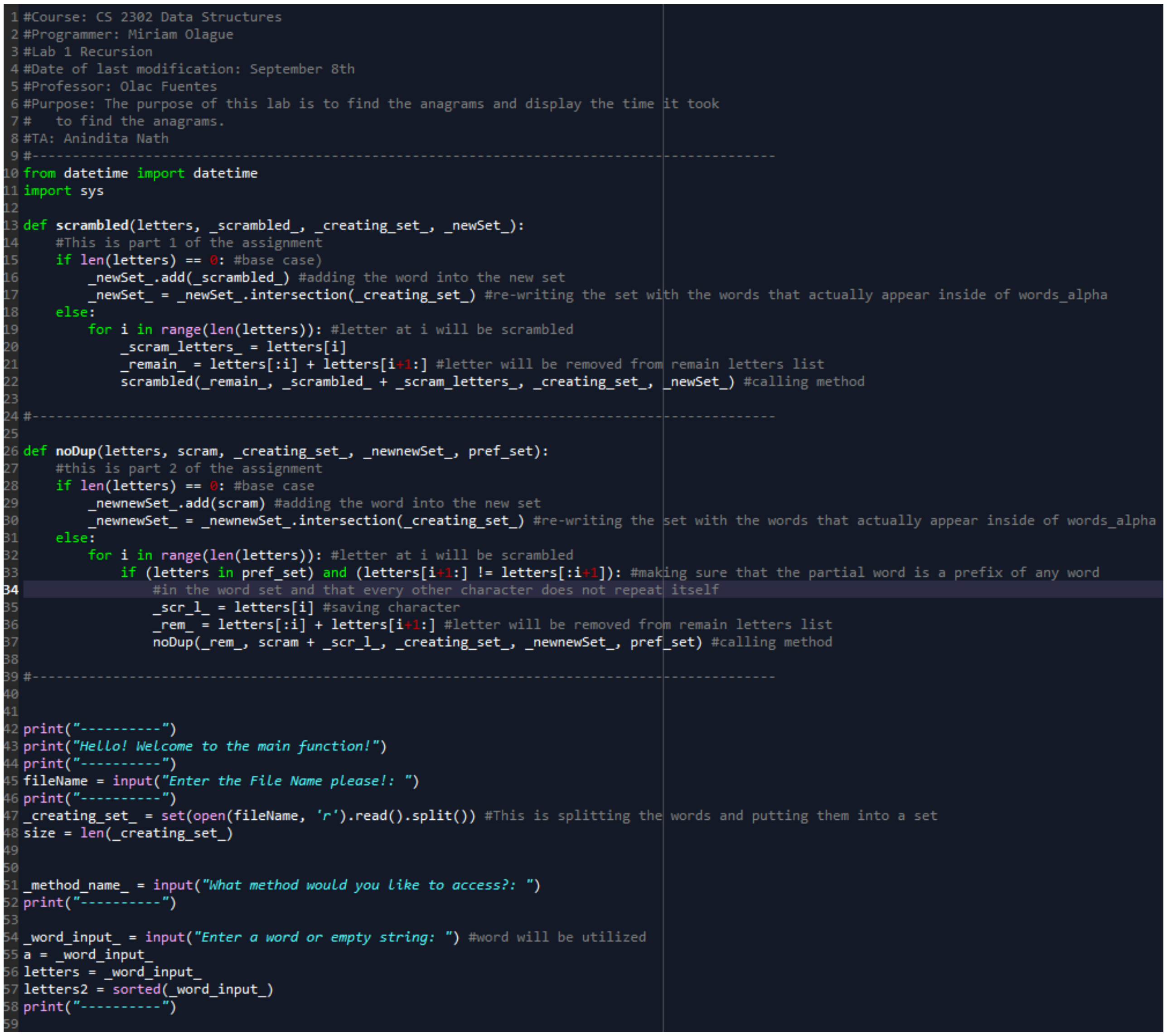
```
In [31]: runfile('C:/Users/miria/Documents/---Fuentes
CS3----/Lab 1/Code/Lab1Recursion.py', wdir='C:/Users/miria/
Documents/---Fuentes CS3----/Lab 1/Code')
----------
Hello! Welcome to the main function!
----------

Enter the File Name please!: C:\Users\miria\Documents\---
Fuentes CS3----\Lab 1\Words\\words_alpha.txt
----------

What method would you like to access?: part2
----------

Enter a word or empty string: data
----------
The word  data  has the following  2 anagrams:
adat
data
Time it took to find the anagrams of this word:
0:00:00.000997
----------
```

User’s input = data

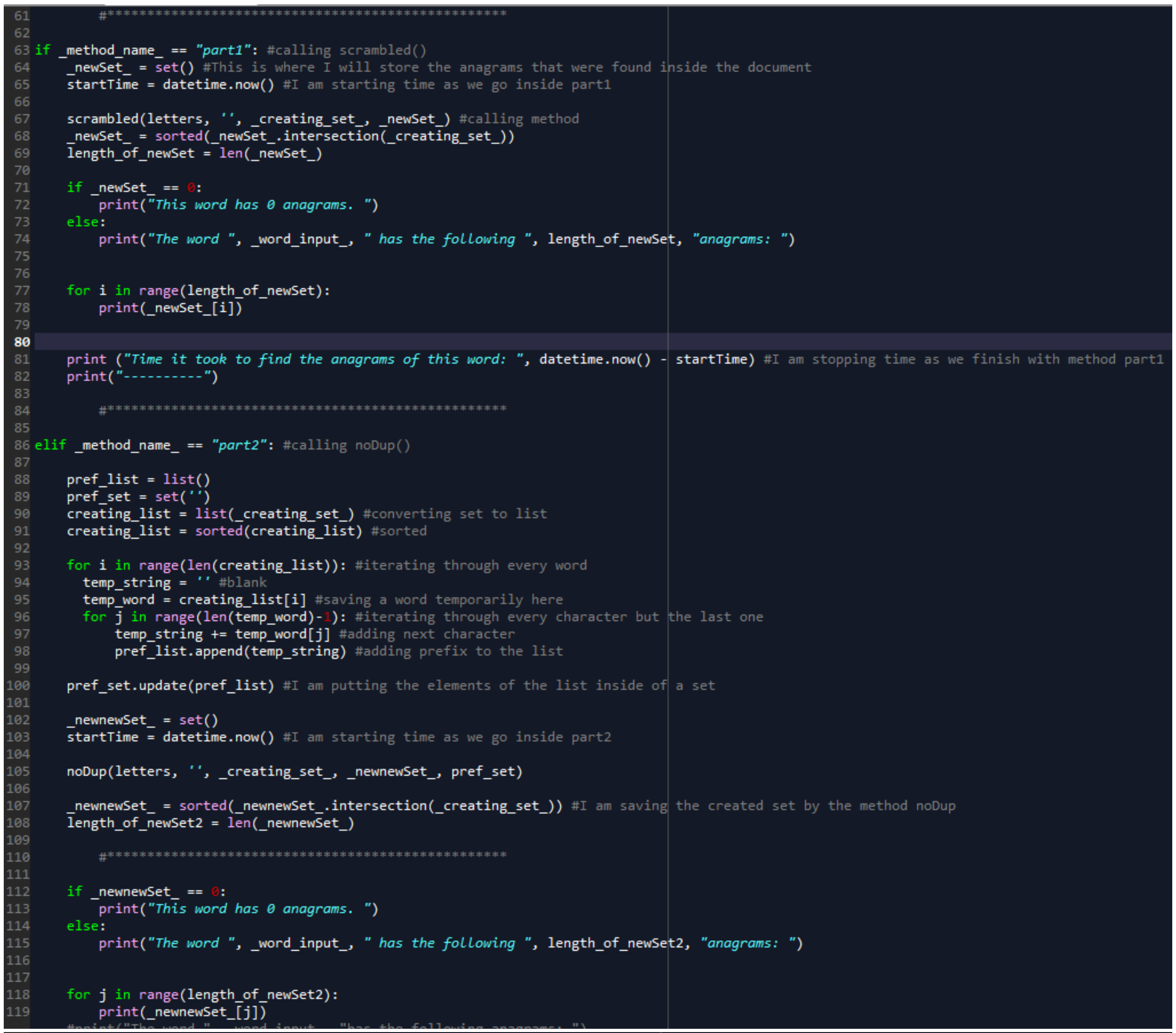
Conclusion

This lab helped me reinforce my knowledge of recursion. It helped me understand and practice the way lists and sets can be used. This lab was challenging but managed to learn a lot from it the more I spent time on it. I enjoyed it since I am barely learning Python and I find it very interesting. I learned how to be a little bit more organized with my code and also learned to give meaningful names to my variables so that way I don’t get lost.

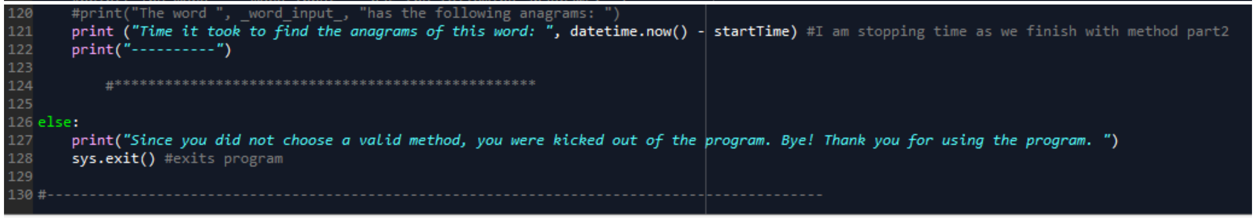
Appendix



```python
#Course: CS 2302 Data Structures
#Programmer: Miriam Olague
#Lab 1 Recursion
#Date of last modification: September 8th
#Professor: Olac Fuentes
#Purpose: The purpose of this lab is to find the anagrams and display the time it took
#   to find the anagrams.
#TA: Anindita Nath
#---------------------------------------------------------------------------------------------
from datetime import datetime
import sys

def scrambled(letters, _scrambled_, _creating_set_, _newSet_):
    #This is part 1 of the assignment
    if len(letters) == 0: #base case)
        _newSet_.add(_scrambled_) #adding the word into the new set
        _newSet_ = _newSet_.intersection(_creating_set_) #re-writing the set with the words that actually appear inside of words_alpha
    else:
        for i in range(len(letters)): #letter at i will be scrambled
            _scram_letters_ = letters[i]
            _remain_ = letters[:i] + letters[i+1:] #letter will be removed from remain letters list
            scrambled(_remain_, _scrambled_ + _scram_letters_, _creating_set_, _newSet_) #calling method

#---------------------------------------------------------------------------------------------

def noDup(letters, scram, _creating_set_, _newnewSet_, pref_set):
    #this is part 2 of the assignment
    if len(letters) == 0: #base case
        _newnewSet_.add(scram) #adding the word into the new set
        _newnewSet_ = _newnewSet_.intersection(_creating_set_) #re-writing the set with the words that actually appear inside of words_alpha
    else:
        for i in range(len(letters)): #letter at i will be scrambled
            if (letters in pref_set) and (letters[i+1:] != letters[:i+1]): #making sure that the partial word is a prefix of any word
                #in the word set and that every other character does not repeat itself
                _scr_l_ = letters[i] #saving character
                _rem_ = letters[:i] + letters[i+1:] #letter will be removed from remain letters list
                noDup(_rem_, scram + _scr_l_, _creating_set_, _newnewSet_, pref_set) #calling method

#---------------------------------------------------------------------------------------------


print("----------")
print("Hello! Welcome to the main function!")
print("----------")
fileName = input("Enter the File Name please!: ")
print("----------")
_creating_set_ = set(open(fileName, 'r').read().split()) #This is splitting the words and putting them into a set
size = len(_creating_set_)


_method_name_ = input("What method would you like to access?: ")
print("----------")

_word_input_ = input("Enter a word or empty string: ") #word will be utilized
a = _word_input_
letters = _word_input_
letters2 = sorted(_word_input_)
print("----------")

```



```python
        #**************************************************

if _method_name_ == "part1": #calling scrambled()
    _newSet_ = set() #This is where I will store the anagrams that were found inside the document
    startTime = datetime.now() #I am starting time as we go inside part1

    scrambled(letters, '', _creating_set_, _newSet_) #calling method
    _newSet_ = sorted(_newSet_.intersection(_creating_set_))
    length_of_newSet = len(_newSet_)

    if _newSet_ == 0:
        print("This word has 0 anagrams. ")
    else:
        print("The word ", _word_input_, " has the following ", length_of_newSet, "anagrams: ")


    for i in range(length_of_newSet):
        print(_newSet_[i])


    print ("Time it took to find the anagrams of this word: ", datetime.now() - startTime) #I am stopping time as we finish with method part1
    print("----------")

        #**************************************************

elif _method_name_ == "part2": #calling noDup()

    pref_list = list()
    pref_set = set('')
    creating_list = list(_creating_set_) #converting set to list
    creating_list = sorted(creating_list) #sorted

    for i in range(len(creating_list)): #iterating through every word
      temp_string = '' #blank
      temp_word = creating_list[i] #saving a word temporarily here
      for j in range(len(temp_word)-1): #iterating through every character but the last one
          temp_string += temp_word[j] #adding next character
          pref_list.append(temp_string) #adding prefix to the list

    pref_set.update(pref_list) #I am putting the elements of the list inside of a set

    _newnewSet_ = set()
    startTime = datetime.now() #I am starting time as we go inside part2

    noDup(letters, '', _creating_set_, _newnewSet_, pref_set)

    _newnewSet_ = sorted(_newnewSet_.intersection(_creating_set_)) #I am saving the created set by the method noDup
    length_of_newSet2 = len(_newnewSet_)

        #**************************************************

    if _newnewSet_ == 0:
        print("This word has 0 anagrams. ")
    else:
        print("The word ", _word_input_, " has the following ", length_of_newSet2, "anagrams: ")


    for j in range(length_of_newSet2):
        print(_newnewSet_[j])
```



```python
    #print("The word ", _word_input_, "has the following anagrams: ")
    print ("Time it took to find the anagrams of this word: ", datetime.now() - startTime) #I am stopping time as we finish with method part2
    print("----------")

        #**************************************************

else:
    print("Since you did not choose a valid method, you were kicked out of the program. Bye! Thank you for using the program. ")
    sys.exit() #exits program

#------------------------------------------------------------------------------------------
```

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.