ITECH1400 - Fundamentals of Programming

Assignment # 1 - Palindromes and Anagrams

Student Name: Melvin M. Flores

Student ID: 30352985

Lecturer: Kiran Chadha

Date Submitted: 10 May 2019

Table of Contents

[1. INTRODUCTION 2](#_Toc8388195)

[1.1 PALINDROME 3](#_Toc8388196)

[*1.1.1* *Algorithm in pseudo-code* 3](#_Toc8388197)

[*1.1.2* *Implementation of algorithm in code* 4](#_Toc8388198)

[*1.1.3* *Demonstration that code works correctly using representative samples* 6](#_Toc8388199)

[*1.1.4* *Correct Output and Discussion* 8](#_Toc8388200)

[1.2 ANAGRAM 10](#_Toc8388201)

[*1.2.1* *Algorithm in pseudo-code* 10](#_Toc8388202)

[*1.2.2* *Implementation of algorithm in code* 11](#_Toc8388203)

[*1.2.3* *Demonstration that code works correctly using representative samples* 15](#_Toc8388204)

[*1.2.4* *Correct Output and Discussion* 18](#_Toc8388205)

[2. CONCLUSION 20](#_Toc8388206)

List of Figures

[Figure 1 Palindrome Statistics 8](#_Toc8387708)

[Figure 2 Palindromes Output Sample Screenshot 9](#_Toc8387709)

[Figure 3 Anagram Statistics 18](#_Toc8387710)

[Figure 4 Anagrams Output Sample Screenshot 19](#_Toc8387711)

# INTRODUCTION

A palindrome is a word or phrase that reads the same backwards or forwards. Examples of these words or phrases are "kayak", "Never odd or even", "Madam, I'm Adam!".

An anagram is a word or phrase formed by rearranging the letters of another. Examples of these words or phrases are "Listen; Silent", "Schoolmaster; The classroom", "Eleven Plus Two; Twelve Plus One", "Waitress; A Stew, Sir?".

In this assignment, we are given the input file called 'English.txt', which contains a list of more than 400 thousand words. And then we need to create a Python program that will identify if the word in the list is a palindrome, as well as if it is an anagram.

## PALINDROME

### Algorithm in pseudo-code

The pseudo-code I used in palindrome were:

Open the input and output files

Read the string in the input file until end of file

Lower Case for the string

Special processing for strings with punctuations, whitespaces and numbers

Reverse the string

Write Palindrome string to the output file

Use Counters

Display the summary

Close the input and output files

### Implementation of algorithm in code

import string

exclude = set(string.punctuation)

source = open("English.txt", "r")

dest = open("Palindromes.txt", "w")

StringCount = 0

PalindromeCount = 0

NotPalindromeCount = 0

for line in source:

original\_string = line

unformatted\_string = original\_string.strip()

unformatted\_string = unformatted\_string.lower()

unformatted\_string = unformatted\_string.replace(' ','')

unformatted\_string = ''.join(ch for ch in unformatted\_string if ch not in exclude)

if (unformatted\_string.isalpha() == False):

NotPalindromeCount += 1

StringCount += 1

continue

reverse\_string = unformatted\_string[::-1]

if len(unformatted\_string) == 0:

NotPalindromeCount += 1

elif unformatted\_string == reverse\_string:

PalindromeCount += 1

dest.write(line)

else:

NotPalindromeCount += 1

StringCount += 1

source.close()

dest.close()

print("Summary:")

print(" Total Strings in the \"English.txt\" input file : {:>7,}".format(StringCount))

print(" Strings that are NOT PALINDROME : {:>7,}".format(NotPalindromeCount))

print(" Strings that are PALINDROME : {:>7,}\n".format(PalindromeCount))

print("ALL PALINDROME strings are listed in the \"Palindromes.txt\" output file\n")

print("Notes:")

print("1. WHITESPACES and PUNCTUATIONS are REMOVED, and the remaining ALPHABETIC string is identified if it is a PALINDROME.")

print("2. Strings with NUMERIC DIGITS in it is considered as a NOT PALINDROME")

print("3. A single character (e.g. \'m\', \'F\', \'x\', and so on...) is considered as a PALINDROME.")

### Demonstration that code works correctly using representative samples

Since we are given the input file, it occurred to me that I must successfully open the file, read the contents, process it, and then close the file.

The strings are the contents coming from the input file, so I must read the first string, process it to determine if it is a Palindrome or not, and then read the next string, process it, and then read again and so on. For reading, I thought of using a single FOR loop statement to iterate through all the records until the end of file.

Comparing the original string with the reverse string in mixed case would lead to an incorrect result. So, I thought of standardizing the strings by making it all lower case.

At the start of my development process, the first thing that came into my mind was to reverse the string. Because if I could reverse the string, then I could compare it with the original string. If the comparison is equal, then it is a Palindrome. Otherwise, it is not a Palindrome. This is the heart of my program.

When I opened the input file, I saw some string with combinations of punctuations as well as numbers. For the whitespace, I cannot see it but I just assumed that there are whitespaces in the right side of the string that needs trimming. To be safe, I also thought of trimming the whitespaces in the right side, as well as in the middle of the string. For the strings with a combination of punctuations, I thought of removing them from the original word and the remaining alphabetic characters, I will evaluate if it is a Palindrome or not. However, for the strings with a combination of numbers, I thought that I need to determine them in the program, and immediately consider that as a Non-Palindrome and skip that record.

If we have four hundred thousand words in the input file, it is best to save all the found Palindromes in an output file. Because if we did not do this, all the Palindromes will be displayed in the Python shell, and we cannot inspect them if we close the Python shell. Also, it is hard to navigate in Python shell's display starting from the first Palindrome up to the last Palindrome.

I have written the Palindrome strings in the output file; thus, it is necessary for me to open the output file before writing and close the output file after writing.

I thought of the different counters that I will use in my program:

1. Counter for Palindromes
2. Counter for Non-Palindromes
3. Counter for number of records in the input file

Because my input is coming from the file, then the number of records in the input file must be equal to the sum of the Palindromes counter and the Non-Palindromes counter.

It will be good to summarize the processing of my program, by displaying the statistics which includes the counters. I also specified the name of my output file. I then included the limitation of my program by displaying some notes.

Note that the ALL PALINDROME strings are listed in the output file named “Palindromes.txt"

Limitation of my Palindrome program are the following:

1. WHITESPACES and PUNCTUATIONS are REMOVED, and the remaining ALPHABETIC string is identified if it is a PALINDROME.

2. Strings with NUMERIC DIGITS in it is considered as a NOT PALINDROME

3. A single character (e.g. 'm', 'F', 'x', and so on...) is considered as a PALINDROME.

### Correct Output and Discussion

I fed the file called 'English.txt', which contains a list of more than 400 thousand words, as input to my Palindrome Python program named “PalindromeTest.py”. My program reads the word in the input file and process it. Processing determines if the word is a Palindrome or not. If it is a Palindrome, it is written to the output file named “Palindromes.txt”. Processing is iterated until end of file. Statistics such as count of the Palindrome words are displayed in the Python shell. See **Figure 1** for the displayed Palindrome statistics.

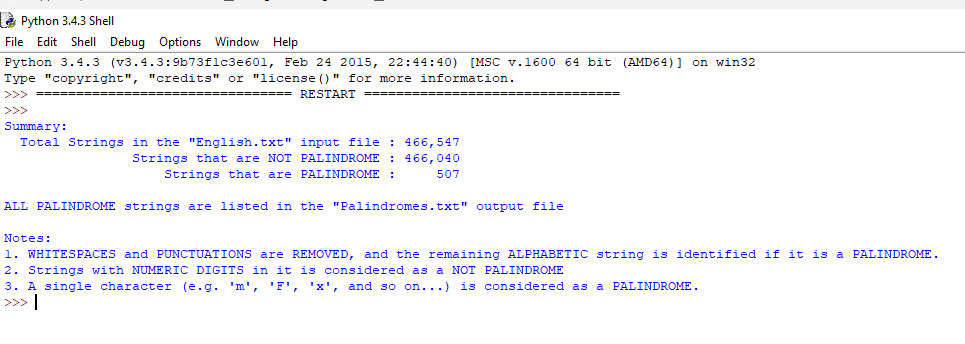


Figure Palindrome Statistics

For the Palindrome statistics, the counts were correct because if you sum the number of Palindrome (507 words) and Non-Palindrome (466,040 words), the total is the same as that of the number of records in the input file (466,547 words).

For the output file, “Palindromes.txt”, if we randomly sampled words that are in that file, we can easily identify that all of them are Palindromes because they are read from left to right the same way they are read from right to left. See **Figure 2** for the Palindrome output sample screenshot below. Also see “Palindromes.txt” for the full list of Palindromes, which is included in the compressed file submitted in Moodle. Please note of my Palindrome program limitations as discussed in the previous section.

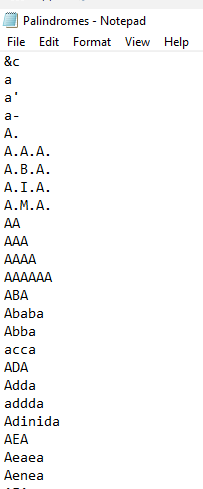


Figure Palindromes Output Sample Screenshot

## ANAGRAM

### Algorithm in pseudo-code

The pseudo-code I used in anagram were:

Open the input and output files

Create a temporary sorted input file

Compute for the execution time

Read the string in the input file until end of file

Read the string in the temporary sorted input file until end of file

Lower Case for the string

Special processing for strings with punctuations, whitespaces and numbers

Sort the string

Use Counter Object to determine frequency of strings

Write Anagram string to the output file

Use Counters

Display the summary, including execution time

Close the input and output files

Remove the temporary sorted input file

### Implementation of algorithm in code

from collections import Counter

import os

import string

import time

start\_time = time.time()

exclude = set(string.punctuation)

source = open("English.txt", "r")

temp = open("EnglishSorted.txt", "w")

dest = open("Anagrams.txt", "w")

AnagramCount = 0

NotAnagramCount = 0

def unformat\_string(formatted\_string):

unformatted\_string = formatted\_string.strip()

unformatted\_string = unformatted\_string.lower()

unformatted\_string = unformatted\_string.replace(' ','')

unformatted\_string = ''.join(ch for ch in unformatted\_string if ch not in exclude)

unformatted\_string = unformatted\_string.replace('\n','')

unformatted\_string = sorted(unformatted\_string)

unformatted\_string = ''.join(unformatted\_string)

return unformatted\_string

data = source.readlines()

data.sort()

for i in range(len(data)):

input\_string = data[i]

temp\_str = unformat\_string(input\_string)

if (temp\_str.isalpha() == False):

continue

if (len(temp\_str) == 0):

continue

if (len(temp\_str) == 1):

continue

temp.write(temp\_str + '\n')

source.close()

temp.close()

source = open("English.txt", "r")

temp = open("EnglishSorted.txt", "r")

SourceLines = source.readlines()

LineCountSource = len(SourceLines)

TempLines = temp.readlines()

ListCount = Counter(TempLines)

for key\_loop in range(0,LineCountSource):

original\_string = SourceLines[key\_loop]

key\_string = unformat\_string(original\_string)

if (key\_string.isalpha() == False):

continue

key\_string = key\_string + '\n'

for ThisItem in ListCount.items():

if key\_string == ThisItem[0] and ThisItem[1] >= 2:

AnagramCount += 1

dest.write(original\_string)

break

NotAnagramCount = LineCountSource - AnagramCount

source.close()

temp.close()

dest.close()

end\_time = time.time()

exec\_time = end\_time-start\_time

print("Summary:")

print(" Total Strings in the \"English.txt\" input file : {:>7,}".format(LineCountSource))

print(" Strings that are NOT ANAGRAM : {:>7,}".format(NotAnagramCount))

print(" Strings that are ANAGRAM : {:>7,}\n".format(AnagramCount))

print("ALL ANAGRAM strings are listed in the \"Anagrams.txt\" output file\n")

print("Note:")

print("1. WHITESPACES and PUNCTUATIONS are REMOVED, and the remaining ALPHABETIC string is identified if it is ANAGRAM.")

print("2. Strings with NUMERIC DIGITS in it is considered as NOT ANAGRAM")

print("3. A single character (e.g. \'m\', \'F\', \'x\', and so on...) is not enough, and considered as NOT ANAGRAM")

print("\nExecution time : {:.2f} minutes".format(exec\_time/60))

os.remove("EnglishSorted.txt")

### Demonstration that code works correctly using representative samples

Since we are given the input file, it occurred to me that I must successfully open the file, read the contents, process it, and then close the file.

The strings are the contents coming from the input file, so I must read the string and process it to determine if it is an Anagram or not. For reading, I thought of using a NESTED FOR loop statement for processing. The first FOR loop statement will read the first string in the input file and compare it with the list in the input file. The second FOR statement will look for a match in the input file, by iterating through all the records until the end of file, until a match is found. The first FOR loop statement then reads the next string, and then the second FOR loop statement will look again for a match in the input file. The first FOR loop statement then reads the next string and so on, process it, until it reaches the end of file.

At the start of my development process, the first thing that came into my mind was to sort the string. Because if I could sort the string, then I could compare it with all the sorted strings in the input file. If a match is found once, I assumed that the string found itself in the input file. If a match is found the second time around or more, then its anagram counterpart was found. Therefore, I have determined that the string is an Anagram. Otherwise, it is not an Anagram. This is the heart of my program.

Comparing the sorted strings in mixed case would lead to an incorrect result. So, I thought of standardizing the strings by making it all lower case.

When I opened the input file, I saw some string with combinations of punctuations as well as numbers. For the whitespace, I cannot see it but I just assumed that there are whitespaces in the right side of the string that needs trimming. To be safe, I also thought of trimming the whitespaces in the right side, as well as in the middle of the string. For the strings with a combination of punctuations, I thought of removing them from the original word and the remaining alphabetic characters, I will evaluate if it is an Anagram or not. However, for the strings with a combination of numbers, I thought that I need to determine them in the program, and immediately consider that as a Non-Anagram and skip that record.

Recall that I will be using a nested FOR statement. The second FOR statement, I will be creating this temporary sorted input file which has the same contents as that of the input file but sorted. After creation, I will use this file in which the string from the first FOR statement, will look for in this sorted input list if there is a match or not. After every record is processed, I will remove this temporary sorted input file.

Keep in mind that that I will be using a nested FOR statement. The second FOR statement, I need to look for a match. If a match is found once, I assumed that the string found itself in the input file. If a match is found the second time around or more, then its anagram counterpart was found. I need a process to count the number of occurrences that a match is found. Upon researching in the internet, I did not find anything. But when I looked in a physical book, in a chapter called 'Managing List’, and in the topic 'Searching Lists', there is a section called 'Working with a Counter Object'. I read through it, understand it, ran the sample program, and then managed to apply it in my actual program, and I was ecstatic when it worked. Basically, given a list, it will count the number of times a specific item is occurring in that list.

It is very important to note here that I will be using a nested FOR statement. This means that the processing time will increase exponentially to the second power. Because for each string in the input file, we must iterate in the list of strings in input file looking for a match. Also recall that we have four hundred thousand words in the input file. Thus, it is good to track the execution time.

If we have four hundred thousand words in the input file, it is best to save all the found Anagrams in an output file. Because if we did not do this, all the Anagrams will be displayed in the Python shell, and we cannot inspect them if we close the Python shell. Also, it is hard to navigate in Python shell's display starting from the first Anagram up to the last Anagram.

I have written the Anagram strings in the output file. Thus, it is necessary for me to open the output file before writing and close the output file after writing.

I thought of the different counters that I will use in my program:

1. Counter for Anagrams
2. Counter for Non-Anagrams
3. Counter for number of records in the input file

Because my input is coming from the file, then the number of records in the input file must be equal to the sum of the Anagrams counter and the Non-Anagrams counter.

It will be good to summarize the processing of my program, by displaying the statistics which includes the counters. I have also included the execution time in the display. I also specified the name of my output file. I then included the limitation of my program by displaying some notes.

Note that the ALL ANAGRAM strings are listed in the output file named “Anagrams.txt"

Limitation of my Anagram program are the following:

1. WHITESPACES and PUNCTUATIONS are REMOVED, and the remaining ALPHABETIC string is identified if it is an ANAGRAM.

2. Strings with NUMERIC DIGITS in it is considered as a NOT ANAGRAM.

3. A single character (e.g. 'm', 'F', 'x', and so on...) is not enough, and considered as NOT ANAGRAM.

### Correct Output and Discussion

I fed the file called 'English.txt', which contains a list of more than 400 thousand words, as input to my Anagram Python program named “AnagramTest.py”. My program reads the word in the input file and process it. Processing determines if the word is an Anagram or not. If it is an Anagram, it is written to the output file named “Anagrams.txt”. Processing is iterated until end of file. Statistics such as count of the Anagram words, along with the execution time, are displayed in the Python shell. See **Figure 3** for the displayed Anagram statistics.

**A screenshot of a social media post

Description automatically generated**

Figure Anagram Statistics

For the Anagram statistics, the counts were correct because if you sum the number of Anagram (120,368 words) and Non-Anagram (346,179 words), the total is the same as that of the number of records in the input file (466,547 words). You may also notice that the execution time is around **11 hours** (657.47 minutes), and this is mainly due to the **nested FOR** (FOR statement within a FOR statement) which I used in my “AnagramTest.py” program and the **very high volume** of the input file which is **more than 400 thousand words**.

For the output file, “Anagrams.txt”, if we randomly sampled words that are in that file, we can identify that all of them are Anagrams because that word is formed by rearranging the letters of another word. See **Figure 4** for the Anagram output sample screenshot below. Also see “Anagrams.txt” for the full list of Anagrams, which is included in the compressed file submitted in Moodle. Please note of my Anagram program limitations as discussed in the previous section.

A close up of a map

Description automatically generated

Figure Anagrams Output Sample Screenshot

# CONCLUSION

This is my first time to code a Python program. Given the lecture slides, lecture sample scripts, online resources, two physical books in Python I borrowed in the library, along with the discussion among fellow classmates, I was able to successfully create my first and second Python programs.

My first Python program is called “PalindromeTest.py”. It can read any input file and process it. All the palindrome words or even palindrome phrases are written in the output file. Output file is named “Palindromes.txt”. Statistics are displayed in the python shell after my program runs.

My second Python program is called “AnagramTest.py”. Similarly, it can read any input file and process it. All the anagram words or even anagram phrases are written in the output file. Output file is named “Anagrams.txt”. Likewise, statistics including the execution time, are displayed in the python shell after my program runs.

During the development of my Python programs, I applied in my codes the lessons in the lecture and laboratory class, the lessons in the online resources, and the lessons in the two physical books. Therefore, I can determine in a given list of words if that word is a Palindrome, also if it is an Anagram, by using my newly created Python programs.