**Homework 1 – Caesar Cipher**

**Hayden Hutsell and Stephen Lankford**

1. The Part 1 program should:

* Prompt user for input regarding their use case – Help menu, Encryption, Decryption, Quit.
* If Encrypt is chosen:
  + Prompt user for input of key, which can be a letter, a number, or a string of letters.
  + Open a dialog box for file selection.
  + Encrypt the English characters (A-Z, a-z) using the Caesar cipher and the key.
    - Do not encode other characters (like punctuations, new lines, spaces, etc), but still output them if they are present.
    - The output should be in UPPER CASE
  + Output the ciphertext result in a text file named EncryptedMessage.txt.
* If Decrypt is chosen:
  + Prompt user for input of key, which can be a letter, number, or a string of letters.
  + Open a dialog box for file selection.
  + Decrypt the ciphertext into English characters (A-Z, a-z) using the Caesar Cipher and the key, ignoring all other characters if present.
  + Output the plaintext result in a text file named DecryptedMessage.txt.
* If Help is chosen, provide the user a brief description of each of the menu options.
* If Quit is chosen, quit the program.

1. The Part 2 program should:

* Prompt user for input regarding their use case – Help menu, Statistical Analysis, Quit.
* If Statistical Analysis is chosen:
  + Open a dialog box for file selection.
  + Read and record the number of times each ciphertext character occurs.
  + Assume the file is UPPER CASE, and contains only A-Z.
  + Assume the file has at least 50 letters.
    - This is because you need enough letters to perform the analysis.
  + Find the key based on character occurrence.
  + The linguistic rule stating an ‘x’ should be inputted between repeat letters doesn’t apply.
    - This is because ‘x’ is a relatively uncommon letter and adding them to the text could throw off the analysis.
  + Three or fewer results should be output the plaintext result to a text file named CrackAttack.txt.
  + The output should be lower case.
  + String keys do not have to be found – only character and number keys must be found.
* If Help is chosen, provide the user a brief description of each of the menu options.
* If Quit is chosen, quit the program.

5.

* Consider an XOR operation, and its truth table:

|  |  |  |
| --- | --- | --- |
| A (plaintext) | B (key) | Result (encrypted) |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

* Now, consider Vernam’s cipher, where data is encrypted with a key by bitwise XOR’ing them together (a bitwise XOR is each bit of a value being XOR’d with a single bit of another value).
* To decrypt, simply bitwise XOR the encrypted data with the key again.
* The truth table contains two values, *A* and *B*, and when XOR’d, produce the *Result*.
* In this case, *A* is our data, *B* is the key, and *Result* is the encrypted text.
* Notice the XOR operation of the *B* column and the *Result* column is equal to the *A* column as well.

|  |  |  |
| --- | --- | --- |
| A (encrypted text) | B (key) | Result (decrypted) |
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |

* Therefore, Vernam’s system is possible because the double XOR’ing of a value will equal itself (the “decrypted” text).

6. Hayden Hutsell: 50%

Stephen Lankford: 50%

HW1 Caesar – pages 4-9; HW1 Crack – pages 9-15

################################################################################################################################

#   Authors: Hayden Hutsell and Stephen Lankford

#   Title: CST407 - HW1 - Caesar Cipher

#   Date: 06/23/2020

#   Revision History: 06/29/2020 - added some documentation, fixed and made file open more efficient, fixed decrypt function

#   Summary: Caesar Cipher Encrypt and Decrypt program. Asks for user input in the menu, then requires user input of cipher key,

#            then plaintext file selection.

################################################################################################################################

import sys

import tkinter as tk

from tkinter import filedialog

from tkinter import \*

from pathlib import Path

from collections import OrderedDict

import os

#########################################################   Global Variables     #################################################

#   these are our global variables.

#   User input defaults to empty.

#   base\_table is alphabet a-z.

#   valid\_input is the menu options a user can enter.

##################################################################################################################################

user\_input = ""

base\_table = "abcdefghijklmnopqrstuvwxyz"

valid\_input = ["help", "encrypt", "decrypt", "quit"]

#########################################################   Validate Input   #################################################

#   This function ensures the input from the user is valid. The user stays in this loop until a valid menu option is selected.

#   This function takes a string of characters, and returns the validated string of characters.

##############################################################################################################################

def sanitize\_input(raw):

    while (raw in valid\_input) != True: #while the user doesn't type something from the valid\_input list

        print("Invalid selection. For help, type help.\nEnter selection: ")

        raw = input("\nEnter selection: ")

    return raw

#########################################################   Directory    #################################################

#   This function calls other functions depending on what the user entered.

#   It is passed a string, and depending on the string calls another function to perform the action the user requested

##########################################################################################################################

def action(selection):

    if selection == "encrypt":

        encrypt()

    elif selection == "decrypt":

        decrypt()

    elif selection == "help":

        help\_screen()

    else:

        print("error! user value was: ", user\_input, ", but you somehow got here, bravo.\n")

        sys.exit("\_\_\_\_\_\_\_\_\_\_fatal error\_\_\_\_\_\_\_\_\_\_")

#########################################################   Caesar   #################################################

#   This function converts a key and returns the caesar cipher table for the encrypt or decrypt functions.

#   Takes a key (letter, number, string of characters)

#   Validates the key, won't leave until a valid key is entered

#   Prints the table, and returns the table to the function that called it.

######################################################################################################################

def caesar\_cipher\_table(key):

    valid = False

    while valid == False:

        cipher = "" #empty string to build cipher table

        valid\_num = True

        valid\_str = True

        valid\_entry = False

        cipher\_number = 0

        temp\_str = ""

        for element in key: #go through the key that was passed

            valid\_entry = True #got to the loop, so its not an empty key

            if not (element.isdigit()): #if it isn't a digit, can't be a valid number

                valid\_num = False

            if not (element.isalpha()): #if it isn't a letter, can't be a valid string

                valid\_str = False

        if valid\_entry == True: #if the string wasn't empty

            if (valid\_num ^ valid\_str): #and only one is true

                if (valid\_num): #if its a number

                    cipher\_number = int(key) % 26                   #base table is a list containing letters from A-Z

                    temp\_str = base\_table[0:cipher\_number] #make a temp string from 0 to the cipher number

                    cipher = base\_table[cipher\_number: ] #set the cipher equal to the key to the end of the list

                    cipher = (cipher + temp\_str).upper() #set the cipher equal to the cipher + the temp string to create a rotated list of letters

                    valid = True #valid input, don't need to keep the loop going

                else:

                    cipher = key + base\_table #set the cipher equal to the key + the base table. The key can be a letter or a string of letters.

                    cipher = ("".join(OrderedDict.fromkeys(cipher))).upper() # joins characters together with "", puts them in an ordered dictionary using the cipher, uppercases it

                    valid = True                                                #ordered dictionary means no repeats, so all repeats removed after the first occurrance.

        if valid == False:

            print("Invalid key entered. A key can be a letter, a base 10 number, or a sentence.\n") #didn't get a valid input. ask for another key.

            key = input("Enter a valid key: ")

    print(cipher) #print the cipher table for the user.

    return cipher #return the cipher table to be used by the calling function

#########################################################   Encryption   #################################################

#   Encrypts a file based on the key that was entered by the user.

#   Asks the user for a key, calls caesar\_cipher\_table, and encrypts the file given the key.

#   File output is upper case

#   Only english letters are converted, all other characters are ignored.

#   Opens the file for the user to see.

##########################################################################################################################

def encrypt():

    key = input("Enter the encryption key (character, string, or integer): ")

    cipher = caesar\_cipher\_table(key)

    #open filepicker dialog box

    filePath = filedialog.askopenfilename(initialdir = "C:\\",title = "Select file",filetypes = (("text files","\*.txt"),("all files","\*.\*")))

    toEncrypt = open(filePath, "r")     #open as read only

    endOfFile = False

    finalString = ""

    while(not endOfFile):

        letter = toEncrypt.read(1)  #read one character at a time

        if letter == '':                #no character read

            endOfFile = True        #end of file

        elif (letter.isalpha()):                        #end of file not detected yet

            change = ord(letter.upper())    #find ascii value of lowercase char

            # print(change - 65)

            # print(finalString)

            finalString += cipher[change - 65]  #convert plaintext to ciphertext based on ascii mod 26

        else:

            finalString += letter #just add the letter to the string if its not important to us (EOF, non alphabetical character)

    toEncrypt.close()

    ciphertext = open("EncryptedMessage.txt", "w")      #save the encrypted message as a text file in the same location as this program

    filename = ciphertext.name

    ciphertext.write(finalString)

    ciphertext.close()

    os.system(filename) #open the file in default text editor

#########################################################   Decryption   #################################################

#   Decrypts a file based on the key that was entered by the user.

#   Asks the user for a key, calls caesar\_cipher\_table, and decrypts the file given the key.

#   File output is lower case

#   Only english letters are converted, all other characters are ignored.

#   Opens the file for the user to see.

##########################################################################################################################

def decrypt():

    key = input("Enter the encryption key (character, string, or integer): ")

    cipher = caesar\_cipher\_table(key)

    #open filepicker dialog box

    filePath = filedialog.askopenfilename(initialdir = "C:\\",title = "Select file",filetypes = (("text files","\*.txt"),("all files","\*.\*")))

    toDecrypt = open(filePath, "r")     #open as read only

    endOfFile = False

    finalString = ""

    while(not endOfFile):

        letter = toDecrypt.read(1)  #read one character at a time

        if letter == '':                #no character read

            endOfFile = True        #end of file

        elif (letter.isalpha()):                        #end of file not detected yet

            change = ord(letter.lower())    #find ascii value of lowercase char

            #print(change - 65)

            #print(finalString)

            finalString += base\_table[cipher.index(letter)] #get the index number of the letter in the cipher table

                                                            #aka the index of the plaintext letter in the base table or regular alphabet

        else:

            finalString += letter                           #letter is not an alpha character, not required to encrypt

    toDecrypt.close()

    ciphertext = open("DecryptedMessage.txt", "w")          #save the decrypted message as a text file in the same location as this program

    filename = ciphertext.name

    ciphertext.write(finalString)

    ciphertext.close()

    os.system(filename) #open the file in default text editor

#########################################################   Help     #################################################

#   Describes what each menu option does, as well as general program functionality

######################################################################################################################

def help\_screen():

    print("The menu options are help, quit, encrypt, and decrypt - type your selection and hit 'Enter'.")

    print("When 'encrypt' or 'decrypt' are entered, type the key, then a file picker dialog box will appear.")

    print("The encrypt/decrypt key can be a decimal integer, letter, or a sentence.")

    print("When a File Open dialog box appears after selecting 'Encrypt', navigate to and select the text file to be encrypted.")

    print("When a File Open dialog box appears after selecting 'Decrypt', navigate to and select the encrypted message to be revealed.")

#########################################################   Main     #################################################

print("This program will encrypt and decrypt a file made up of english words.\n\n\

for help, type help.\n\

for encrypting, type encrypt\n\

for decrypting, type decrypt\n\

to quit, type quit.")

while user\_input != "quit": #stay in the loop until quit is ran

    raw\_input = input("\nEnter selection: ")

    user\_input = sanitize\_input(raw\_input) #make sure the input is valid

    if user\_input != "quit":

        action(user\_input)

################################################################################################################################

#   Authors: Hayden Hutsell and Stephen Lankford

#   Title: CST407 - HW1 - Caesar Cipher

#   Date: 06/29/2020

#   Revision History: 06/29/2020 - started caesar crack program

#                     07/05/2020 - finished statistical attack

#   Summary: Crack Caesar ciphertext

################################################################################################################################

import sys

import tkinter as tk

from tkinter import filedialog

from tkinter import \*

from pathlib import Path

from collections import OrderedDict

from collections import deque

import os

#########################################################   Global Variables     #################################################

#   These are our global variables that are used in the program.

#   User input starts blank

#   valid\_input is the valid options a user can type

#   base table is simply a list from a-z

#   letters is a list ordered from most common to least common letter occurrance. Used as a reference

#   Frequency is a list of the average occurance of all letters ordered from most to least common

#   FrequencySorted is a list of the the average occurance of all letters order from A-Z

##################################################################################################################################

user\_input = ""

valid\_input = ["help", "statistical", "quit"]   #add dictionary when implemented

base\_table = "abcdefghijklmnopqrstuvwxyz"

#letters = "ETAOINSRHDLUCMFYWGPBVKXQJZ" #most common letters in order from left to right - Cornell

frequency = [12.02, 9.10, 8.12, 7.68, 7.31, 6.95, 6.28, 6.02, 5.92,

            4.32, 3.98, 2.88, 2.71, 2.61, 2.30, 2.11, 2.09, 2.03, 1.82,

            1.49, 1.11, 0.69, 0.17, 0.11, 0.10, 0.07]   #this list is the average occurance of all letters ordered from most to least common

frequencySorted  = [8.12, 1.49, 2.71, 4.32, 12.02, 2.3, 2.03, 5.92, 7.31, .1, .69, 3.98,

            2.61, 6.95, 7.68, 1.82, .11, 6.02, 6.28, 9.1, 2.88, 1.11, 2.09, .17, 2.11, .07] #this list is the average occurance of all letters order from A-Z

#########################################################   Validate Input   #################################################

#   This function ensures the input from the user is valid. The user stays in this loop until a valid menu option is selected.

#   This function takes a string of characters, and returns the validated string of characters.

##############################################################################################################################

def sanitize\_input(raw):

    while (raw in valid\_input) != True: #while the user doesn't type something from the valid\_input list

        print("Invalid selection. For help, type help.\nEnter selection: ")

        raw = input("\nEnter selection: ")

    return raw

#########################################################   Directory    #################################################

#   This function calls other functions depending on what the user entered.

#   It is passed a string, and depending on the string calls another function to perform the action the user requested

##########################################################################################################################

def action(selection):

    if selection == "statistical":

        statistical()   #stat chosen, call statistical analysis function

    elif selection == "help":

        help\_screen()   #help chosen, call help function to display help instructions

    else:

        print("error! user value was: ", user\_input, ", but you somehow got here, bravo.\n")

        #sys.exit("\_\_\_\_\_\_\_\_\_\_fatal error\_\_\_\_\_\_\_\_\_\_"

#########################################################   Statistical  #################################################

#   This function attempts to decrypt a file that is encrypted with a caesar cipher table that is rotated by a number x

#   This means this does not work with files encrypted with a string, or a letter at the beginning of the string (CABDEF...)

#   asks the user for the file to decrypt

#   counts the occurance of encrypted letters

#   Creates a list that contains the average occurance of each letter

#   Rotates the list, creating another list containing the differences between the frequencySorted list and the statCount list

#   The most likely key is the position of the smallest value in the differences list.

#   A decryption is performed with the most likely key, and the decryption result is written to a file.

#   the file is opened in the user's default text editor.

##########################################################################################################################

def statistical():

    #open filepicker dialog box

    filePath = filedialog.askopenfilename(initialdir = "C:\\",title = "Select file",filetypes = (("text files","\*.txt"),("all files","\*.\*")))

    toDecrypt = open(filePath, "r")     #open as read only

    endOfFile = False

    finalString = ""    #final string of text to be output to file

    statCount = [0] \* 26    #for counting number of each letters occurrence in ciphertext (list of 26 elements containing 0)

    differenceCount = [0] \* 26  #for counting the difference in the frequency averages (list of 26 elements containing 0)

    letterCount = 0 #count letters in entire file for creating averages

    difference = 0  #variable for finding key based on difference between averages

    #print(statCount)

    while(not endOfFile):

        letter = toDecrypt.read(1)  #read one character at a time

        if letter == '':                #no character read

            endOfFile = True        #end of file

        elif (letter.isalpha()):                        #end of file not detected yet

            change = ord(letter.upper())    #find ascii value of uppercase char

            statCount[change - 65] += 1     #count the number of times each letter is seen

            letterCount += 1

    toDecrypt.close()

    for element in range(len(statCount)):   #create averages between the number of times a ciphertext character is counted

        #print(element)                     #and the number of ciphertext characters in the entire file

        statCount[element] = (statCount[element] / letterCount) \* 100

    #print(statCount)

    for element in range(len(statCount)):   #test each key for the difference between frequencies

        for x in range(len(statCount)):     #find difference between frequency of letter and times it was counted in the file

            difference += abs(statCount[x] - frequencySorted[x]) #ensure the difference is positive, as we find a min later

        differenceCount[element] = difference   #record the difference

        statCount = deque(statCount) #convert the statcount list into another type with the rotate function (a list like object, doubly ended queue)

        statCount.rotate(-1) #rotate left once

        statCount = list(statCount) #convert back into a list

        difference = 0 #reset the difference counter

    #frequencySorted list

    #statCount 26 element array that contains the average occurance of each letter (probably not aligned to the frequencySorted)

    #differenceCount 26 element array [100]

    key = differenceCount.index(min(differenceCount)) #key is the position of the smallest element

    finalString = decrypt(key, filePath)    #decrypt the file with the most likely key, passing the file path to the decrypt function.

    ciphertext = open("CrackAttack.txt", "w")       #save the encrypted message as a text file in the same location as this program

    filename = ciphertext.name

    ciphertext.write(finalString) #write the decryption result to the file

    ciphertext.close() #close the file

    os.system(filename) #open the file in the default text editor

#########################################################   Decryption   #################################################

#   This function reads in text from a file, and decrypts it with the key that it was passed, and returns the decrypted string

#   Takes a key and a file path

#   opens the file, reads in the text, and decrypts all english letters with the passed key.

#   returns the decryption result as a string

##########################################################################################################################

def decrypt(key, filePath):

    cipher = caesar\_cipher\_table(key)

    #open filepicker dialog box

    #filePath = filedialog.askopenfilename(initialdir = "C:\\",title = "Select file",filetypes = (("text files","\*.txt"),("all files","\*.\*")))

    toDecrypt = open(filePath, "r")     #open as read only

    endOfFile = False

    finalString = ""

    while(not endOfFile):

        letter = toDecrypt.read(1)  #read one character at a time

        if letter == '':                #no character read

            endOfFile = True        #end of file

        elif (letter.isalpha()):                        #end of file not detected yet

            change = ord(letter.lower())    #find ascii value of lowercase char

            #print(change - 65)

            #print(finalString)

            finalString += base\_table[cipher.index(letter)] #get the index number of the letter in the cipher table

                                                            #aka the index of the plaintext letter in the base table or regular alphabet

        else:

            finalString += letter                           #letter is not an alpha character, not required to encrypt

    toDecrypt.close() #close the file

    return(finalString)

#########################################################   Table    #################################################

#   This function creates a caesar cipher table given a numberical key.

#   Prints the table, and returns the table to the caller

##########################################################################################################################

def caesar\_cipher\_table(key):

    cipher = "" #empty string to build cipher table

    temp\_str = ""

    cipher\_number = 0 #ensure cipher\_number will be an int

    cipher\_number = int(key) % 26       #base table is a list containing letters from A-Z

    temp\_str = base\_table[0:cipher\_number]  #make a temp string from 0 to the cipher number

    cipher = base\_table[cipher\_number: ]    #set the cipher equal to the key to the end of the list

    cipher = (cipher + temp\_str).upper()    #set the cipher equal to the cipher + the temp string to create a rotated list of letters

    print(cipher)

    return cipher

#########################################################   Help     #################################################

#   Describes what each menu option does, as well as general program functionality

######################################################################################################################

def help\_screen():

    print("The menu options are help, quit, and statistical - type your selection and hit 'Enter'.")

    print("When 'statistical' is entered, type the key, then a file picker dialog box will appear.")

    print("When a File Open dialog box appears after selecting 'statistical', navigate to and select the encrypted message to be revealed.")

#########################################################   Main     #################################################

print("This program will attempt to crack a caesar cipher.\n\n\

for help, type help.\n\

for statistical analysis, type statistical\n\

to quit, type quit.")

while user\_input != "quit": #stay in the loop until quit is ran

    raw\_input = input("\nEnter selection: ")

    user\_input = sanitize\_input(raw\_input) #make sure the input is valid

    if user\_input != "quit":

        action(user\_input)