Intro to Cryptography Project 1

Group Members: Omar Yassin Yusra Zvavi Rohan Bhadiyadra

Table of Contents

Introduction

The main idea of the following project is creating a program that applies an encryption algorithm with its responding decryption algorithm in order to convert the plain text to cipher text. The Encryption algorithm focus mostly on the integration of variety encryption algorithm substitution cipher types in order to ensure adaption of secure cryptography, The basic architecture is divided to two rounds, the first being the monoalphabetic substitution and the second caesurae cipher substitution. Moving swiftly on to the decryption part which is implementing the decryption algorithm which has been derived from the encryption algorithm, starting with decrypting monoalphabetic substitution and moving on to decrypting caesurae cipher substitution. As well as providing a user-friendly Graphical user interface for a more convenient experience for the users and a frequency analysis to represent more detailed information regarding the algorithms.

Objectives

The Objective of the following project is to be able to implement different cipher substitution techniques mainly monoalphabetic substitution, and caesurae cipher substitution in order to aim towards a part of secure connection by applying encryption algorithms which is a huge part of cryptograph. As well as, creating graphical user interface aimed to create a much simpler interaction with the user and a frequency analysis.

Description of the Algorithm(pseudo code)

- Convert the input to numbers and put them in a list
- for each element in numbered input list
    - if element equals a space then add a space to the final list
    - else if the index equals the number 2
        - set number of shifts to the current element and decrease the index
        - add the corresponding element from Monoalphabetic list to the final list
    - else if the index equals the number 1
        - decrement the index by 1
        - add the corresponding element from Monoalphabetic list to the final list
    - else
        - if number of letters doesn't equal to zero
            - Set the number after Cesar cipher to the number of shifts plus element modulus the number of alphabets in English
            - If number after Cesar cipher equal to zero
                - Set the number after Cesar cipher to 26
            - Append the number of Cesar cipher to the final encrypted list
            - Decrement number of letter by 1
            - If number of letters equal to zero
                - Set index to 2

Description of the Decryption Algorithm(pseudo code)

- for each element in numbered input list
    - if element equals a space then add a space to the final list
    - else if the index equals the number 2
        - map the current index to its position in the Monoalphabetic list and decrease the index
        - the number of shifts will be equal to the corresponding number
        - add the corresponding number to the final list
    - else if the index equals the number 1
        - decrement the index by 1
        - map the current index to its position in the Monoalphabetic list and decrease the index
        - the number of letters will be equal to the corresponding number
    - else
        - if number of letters doesn't equal to zero
            - Set the number after Cesar cipher to the number of shifts minus element modulus the number of alphabets in English
            - If number after Cesar cipher equal to zero
                - Set the number after Cesar cipher to 26
            - Append the number after Cesare cipher to the final decrypted list
            - Decrement number of letters by 1

- If number of letters equal to zero then set index to 2

Implementation Specifications

The implantation of this code was mainly done using a class called Monosar this class includes multiple methods and it also includes multiple function to start off this we first check if the __name__ == to main and this checks if the if the program is just starting.

__name__ == __main:

```
if __name__ == "__main__":
    root = Tk()
    root.title("Monosar")
    root.iconbitmap(r'favicon.ico')

    # root.resizable(False, False)  # This code helps to disable windows from resizing

    window_height = 420
    window_width = 950

    screen_width = root.winfo_screenwidth()
    screen_height = root.winfo_screenheight()

    x_cordinate = int((screen_width / 2) - (window_width / 2))
    y_cordinate = int((screen_height / 2) - (window_height / 2))
    root.geometry("{}x{}+{}+{}".format(window_width, window_height, x_cordinate, y_cordinate))

    Monosar(root).pack(side="top", fill="both")
    root.mainloop()
```

This if statement starts with defining a root variable and we are setting it to the classes called Tk() which maps to the importation of a library called tkinter this will allow us to use python's capabilities of GUI. We now give the windows the title of Monosar, which is the name we gave to this project, and we go on setting the height and width of the windows and we place it in the center of the screen. Finally we set the geometry of the program to make took the same in any OS and we pack the window.

Functions:

```
def Mono():
    """
    This function sets and returns the mono alphabetic list
    Return : array of integers
    """
    MAlist = [[9], [19], [26], [2], [3], [13], [7], [10], [4], [15], [16], [23], [6], [24], [0], [1], [5], [14], [8],
              [11], [20], [21], [22], [25], [12], [17], [18]]  # this is a static mono alphabetic list
    return MAlist


def MonoToNon(monoNumber, MAList):
    """
    This function is used to map the mono alphabetic number to its location in the list
    then it returns that index of where the number is.
    Return : array of integers
    """
    indx = -1
    for x in MAList:  # this is a for loop to iterate over the list
        if x == monoNumber:  # this checks if the the current value is equal to the the current number in the list
            indx += 1  # indicating that we are moving though the list
            return indx  # returning where the number has been found
        else:
            indx += 1  # if we couldn't find the number at that location we move the index


def NoMono():
    """
    This function sets and returns the alphabetic list
    Return : array of integers
    """
    NoMAlist = [[i] for i in range(27)]  # returning a list of numbers from 0-26 to resemble the normal alphabetic list
    return NoMAlist  # returning the list
```

Mono

In this project, we are using a couple of functions, the first one is a function that just returns the mono alphabetic list, and it is just a list of numbers from 0-26 randomized.

NoMono

NoMono this create a list of numbers from 0-26 and we are using this mainly for decryption to map the mono alphabetic letters with non-mono alphabetic.

MonoToNon

This function is also used for decryption we use this function to convert the mono alphabetic letter to non mono alphabetic we do that by iterating over the Mono alphabetic list and to map the mono alphabetic number to the its index in the alphabetic list and returning the index. By doing that we can convert the mono alphabetic number to the alphabetic number.

Global Variables

```
MAList = Mono()  # running the function and setting it to the variable MAList
finalDecryptList = []
finalEncryptList = []  # initializing variables
encryptedList = []
```

Here we are setting those variables as global variables so we can use then anywhere in our program.

```
class Monosar(Frame):
    encryptedText = ""  # initializing variables
    decryptedText = ""
```

Those variables are also global variables but they just going to be used inside the class.

Initializing the GUI

```
def __init__(self, pencere):
    """
    This function creates the GUI
    Return : GUI
    """

    Frame.__init__(self, pencere)
    self.pencere = pencere

    Label(pencere, text="Enter text... ", relief=GROOVE, width=25).place(x=60, y=15)
    self.Ent1 = Entry(pencere, width=30)
    self.Ent1.place(x=58, y=50)  # creating the the input box for the text and setting the place

    # creating the buttons and setting their locations
    Button(pencere, text="Encrypt", relief=GROOVE, font="bold", command=self.Encrypt).place(x=30, y=100)
    Button(pencere, text="Decrypt", relief=GROOVE, font="bold", command=self.Decrypt).place(x=190, y=100)

    # creating the output box and setting the location of the box and the size of the box
    Label(pencere, text="The Result: ", relief=GROOVE, width=25).place(x=60, y=160)
    self.Result = Entry(pencere, width=30)
    self.Result.place(x=58, y=190)

    # creating the intial configuration and setting its location and size
    Label(pencere, text="The intial configuration...", relief=GROOVE, width=25).place(x=60, y=240)
    self.initConfig = Text(pencere, width=77)
    self.initConfig.place(x=290, y=15)
```

This function is used to create the GUI by creating all the elements inside that GUI including the Label for to indicate where the elements are that user should pay attention to.

Buttons are used to run the specified methods weather it has to encrypt or to decrypt the text and to place them in the right position.

The Result field is used to display the output of both function and the initConfig is used to display the explanation of how both the encryption and decryption work.

Conversion

From letters to numbers

```
def converter(self, word):
    """
    This method is used to covert alphabetic charecters into the required index
    for encryption and decryption
    input types: array of string
    Return : array of integers
    """
    dictLetters2Numbers = {
        "a": "1",
        "b": "2", "c": "3", "d": "4", "e": "5", "f": "6", "g": "7", "h": "8", "i": "9",
        "j": "10", "k": "11", "l": "12", "m": "13", "n": "14", "o": "15", "p": "16",
        "q": "17", "r": "18", "s": "19", "t": "20", "u": "21", "v": "22", "w": "23",
        "x": "24", "y": "25", "z": "26"
    }
    numberList = []
    for x in word:
        if x in dictLetters2Numbers:
            numberList.append(int(dictLetters2Numbers[x]))
        elif x == ' ':  # checking if there is a space
            numberList.append(' ')
        else:
            numberList.append(int(0))
    return numberList
```

This method is used to convert the letters into numbers by taking the input in and initializing the dictionary and an empty list afterwards we then loop around the word and appending the letter's corresponding number to the list and we also check for spaces and append spaces to the list if they exist.

From numbers to letters

```
def converter2(self, dec):
    """
    This method is used to covert index numbers back into their alphabetic value
    for encryption and decryption
    input types: array of integers
    output types: array of strings
    """
    dictNum2Letters = {
        "1": "a",
        "2": "b", "3": "c", "4": "d", "5": "e", "6": "f", "7": "g", "8": "h", "9": "i",
        "10": "j", "11": "k", "12": "l", "13": "m", "14": "n", "15": "o", "16": "p",
        "17": "q", "18": "r", "19": "s", "20": "t", "21": "u", "22": "v", "23": "w",
        "24": "x", "25": "y", "26": "z"
    }
    temp = []
    letterList = []
    for x in dec:
        temp.append(str(x))
    for y in temp:
        if y == ' ':  # checkin if there is a space
            letterList.append(' ')
        elif y in dictNum2Letters:
            letterList.append(dictNum2Letters[y])
    return letterList
```

This method is used to convert the numbers into letters by taking a list in and initializing the dictionary and an empty list afterwards we then loop around the list and appending the number's corresponding letter to the list and we also check for spaces and append spaces to the list if they exist.

Encryption Algorithm

```
def Encrypt(self):

    """
    This method implements the encryption algorithm given to encrypt lowercase
    alphabetic charecters using a key
    Input types: Integer, array of integers
    Return: Array of integers
    """
    global MAList
    global encryptedText
    global finalEncryptList
    encryptedText = ""
    index = 2
    NOS = 0  # initializing variables
    NOL = 0
    nIter = 0
    text = self.Ent1.get()
    conv = self.converter(text)
```

This method does all the operation necessary for encryption; we first initialize all the global and local variable and we also get the input from the Ent1 variable and convert the input text to numbers.

Encryption Loop

```
for x in conv:
    if x == " ":
        finalEncryptList.append(" ")  # checking if x is a pace and if it is adding a space the final list
    elif index == 2:
        NOS = x
        index = index - 1
        output = MAList[x]  # checking if index is equal to 2 if it is then we set the number of shifts to x
        str1 = 0
        for last in output:
            str1 += last  # converting from a list to string
        finalEncryptList.append(str1)  # adding the result to the list
        self.initConfig.insert(INSERT, "Here we set the Number of Shifts to %d\n" % (int(x)))
    elif index == 1:
        index = index - 1
        NOL = x  # checking if index is equal to 2 if it is then we set the number of letters to shift to x
        output2 = MAList[x]
        str2 = 0
        for last2 in output2:
            str2 += last2  # converting from a list to string
        finalEncryptList.append(str2)  # adding the result to the list
        self.initConfig.insert(INSERT, "Here we set the Number of Letter to do the shift to %d\n" % (int(x)))
    else:
        if NOL != 0:
            NACC = (x + NOS) % 26
            if NACC == 0:  # checking if the Number After the Caesar Cipher is equal to 0 and then
                NACC = 26  # setting it to 26
            finalEncryptList.append(NACC)  # appending the final number to the list
            self.initConfig.insert(INSERT, "Now we do the caesar cipher to %d\n" % (int(x)))
            self.initConfig.insert(INSERT, "the result of the caesar cipher is %d\n" % (int(NACC)))
            NOL = NOL - 1  # decreasing the number of letter to shift
            if NOL == 0:
                index = 2
```

This loop does all the encryption operations we first check if the current index is a space if it is we just add the space then we check if the index is equal to 2 if it is then we set the number of shifts to the current element. Then we get the corresponding mono alphabetic substitution and we finally add final number to the final list.

Next, we check if the index is, equal to one if it is then we set the number of letters to shift to the current element and then we get the corresponding mono alphabetic substitution and we finally add final number to the final list.

Finally we if the index is not equal to two or one we then go to the else statement and check if the number of characters to shift is not equal to zero. If that is true then we set the Number after Caesar Cipher to the current element plus the number of letters to shift plus modulus 26. So that we can make sure that the number won't be bigger than 26 finally we also check if the final answer is equal to zero if it is then we make it equal to 26. Then we check if the Number of letters to shift is equal to zero if it is then we set the index back to two.

Encryption Conversion

```
encryptedList = self.converter2(finalEncryptList)  # converting the numbers to letters
self.initConfig.insert(INSERT, "we now convert the numbers back to letters and the result is \n")
self.initConfig.insert(INSERT, encryptedList)
self.initConfig.insert(INSERT, "\n")
for last in encryptedList:  # converting the the list into a string
    encryptedText = encryptedText + last
```

We finally convert the numbers back to letters and convert the list back to a string.

Decryption Algorithm

```
def Decrypt(self):
    """
    This function is used to decrypt the given ciphertext
    Return: Array of Integers

    """

    global finalDecryptList
    global decryptedText
    global finalEncryptList
    global encryptedText
    global decryptedText
    global MAList
    decryptedText = ""
    index = 2
    NOS = 0  # initializing variables
    NOL = 0
```

This method does all the operation necessary for decryption; we first initialize all the global and local variable we will use to perform the decryption.

Decryption Loop

```
for x in finalEncryptList:
    if x == " ":
        finalDecryptList.append(" ")

    elif index == 2:
        index = index - 1
        output = MonoToNon([x], MAList)  # here we are getting the index of the encrypted letter
        self.initConfig.insert(INSERT, "Now we map the encrypted letter with the unencrypted list\n")
        NOS = output  # setting back the number of letters to shift
        self.initConfig.insert(INSERT, "In this case the %d maps to %d which is the Number of shifts"
                                       "\n" % (int(x), int(NOS)))
        finalDecryptList.append(output)
    elif index == 1:
        index = index - 1
        self.initConfig.insert(INSERT, "Now we map the encrypted letter with the unencrypted list\n")
        output2 = MonoToNon([x], MAList)
        NOL = output2
        self.initConfig.insert(INSERT, "In this case the %d maps to %d which is the number of letter to "
                                       "shift \n" % (int(x), int(NOS)))
        finalDecryptList.append(output2)
    else:
        if NOL != 0:
            self.initConfig.insert(INSERT, "now we are deciphering caesar cipher number\n")
            NACC = (x - NOS) % 26
            if NACC == 0:
                NACC = 26
            self.initConfig.insert(INSERT, "%d deciphered to %d\n" % (int(x), int(NACC)))
            finalDecryptList.append(NACC)
            NOL = NOL - 1
            if NOL == 0:
                index = 2
```

This loop does all the decryption operations we first check if the current index is a space if it is we just add the space then we check if the index is equal to 2. If it is then we get the mapped number from the MonoToNon function and we set the output to the number of shifts to the current element. We finally add final number to the final list.

Next, we check if the index is, equal to one if it is then we get the mapped number from the MonoToNon function and we set the output to the number of letters to shift and we finally add final number to the final list.

Finally we if the index is not equal to two or one we then go to the else statement and check if the number of characters to shift is not equal to zero. If that is true then we set the Number

after Caesar Cipher to the current element minus the number of letters to shift plus modulus 26. So that we can make sure that the number won't be bigger than 26 finally we also check if the final answer is equal to zero if it is then we make it equal to 26. Then we check if the Number of letters to shift is equal to zero if it is then we set the index back to two.

Decryption Conversion

```
DecryptList = self.converter2(finalDecryptList)
self.initConfig.insert(INSERT, "we now convert the numbers back to letters and the result is \n")
self.initConfig.insert(INSERT, DecryptList)
self.initConfig.insert(INSERT, "\n")
for last in DecryptList:
    decryptedText = decryptedText + last
```

Here we covert the final decrypted list back to letters and then we convert the list back to string by looping around the list and appending each element to an empty text.

Testing

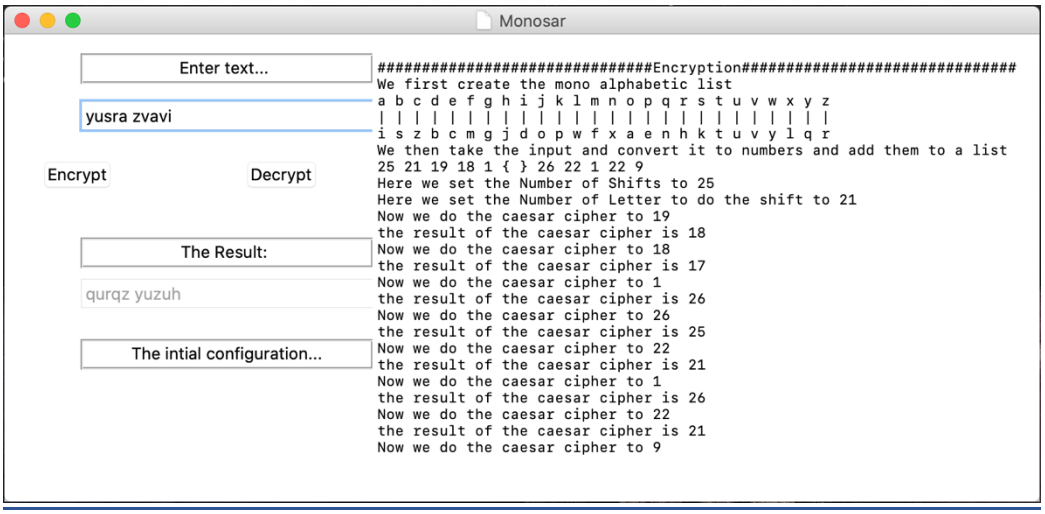


Figure 1

Encryption:

We successfully encrypted the text “yusra zvavi”. Letter “y” is 25th letter in the alphabetic list, 3rd letter onward the next 25 letters will have cipher shift. Letter “u” is 21st letter in the alphabetic list, the cipher shift will be by 21 shifts. If any uppercase letter entered it will ignore the letter and keep the position as 0. We also have shown the processes of encryption.

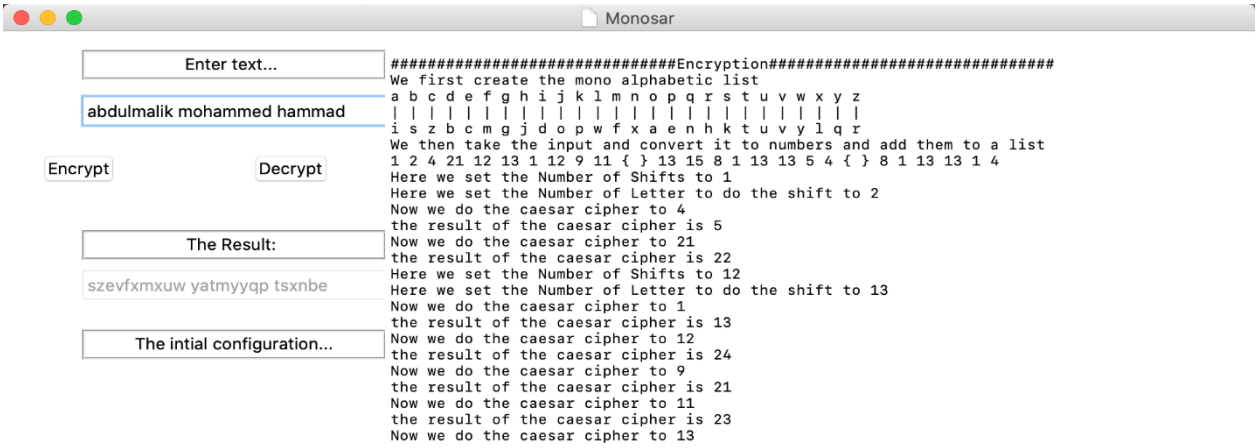


Figure 2

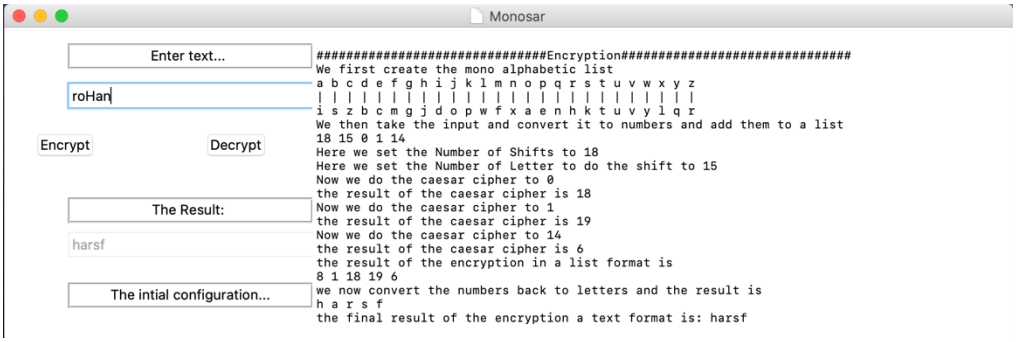


Figure 3

in the figure above (Figure 3) shows the testing of including uppercase for encryption, as we can see it considered uppercase as position 0 and doesn't encrypt the way it is supposed to.

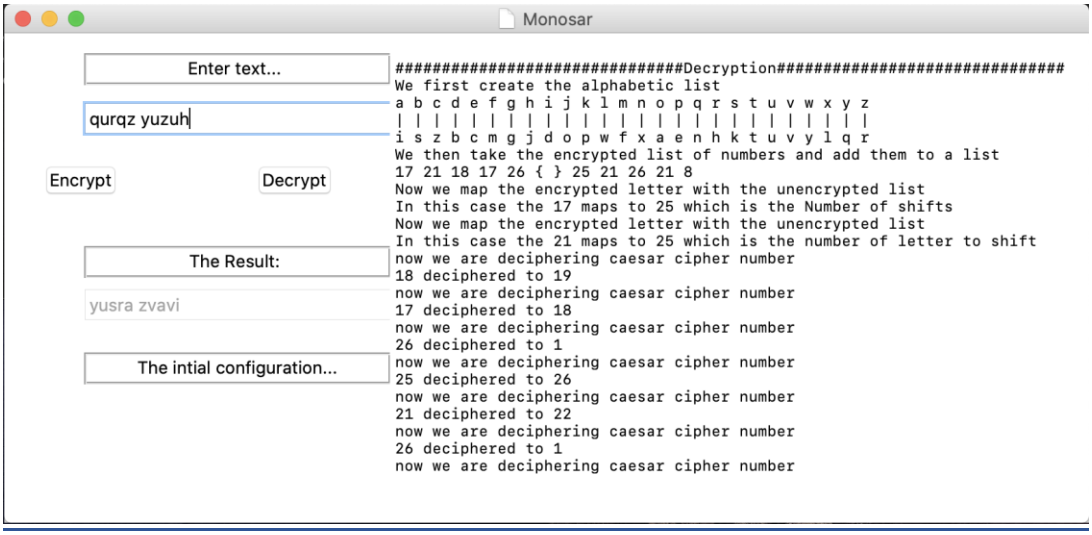


Figure 4

Decryption:

We successfully decrypted the text "qurqz yuzuh" shown in figure 4. Letter "q" in the encrypted list maps to 25th is the number of shifts. then letter "u" which maps to 25th number of letter to shift. Working has been showing clearly in the figure above. If any word with uppercase is

encrypted the decryption will not give us back the correct cipher, as the uppercase is ignored and kept the value position 0.

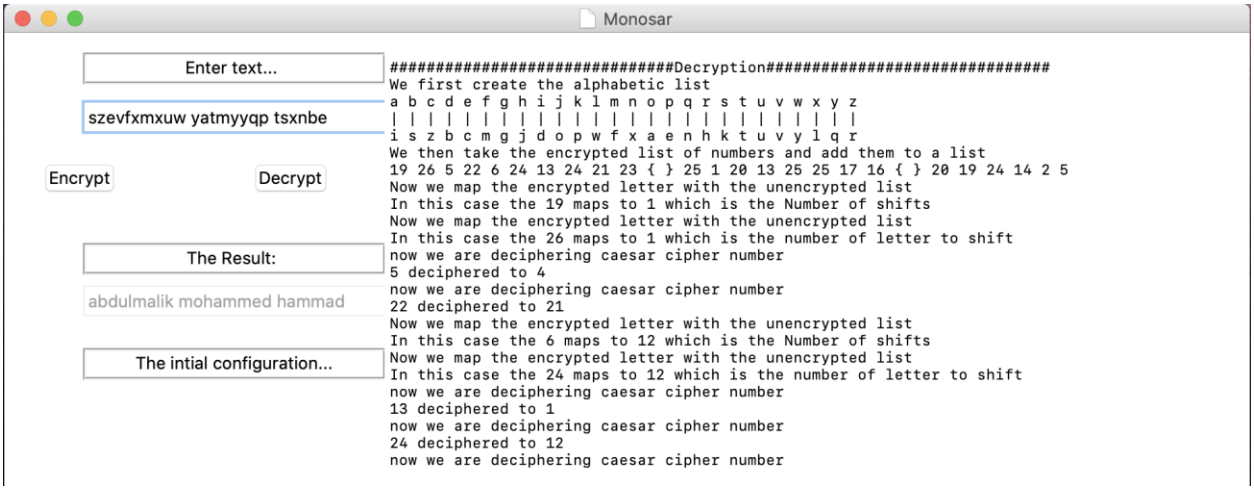


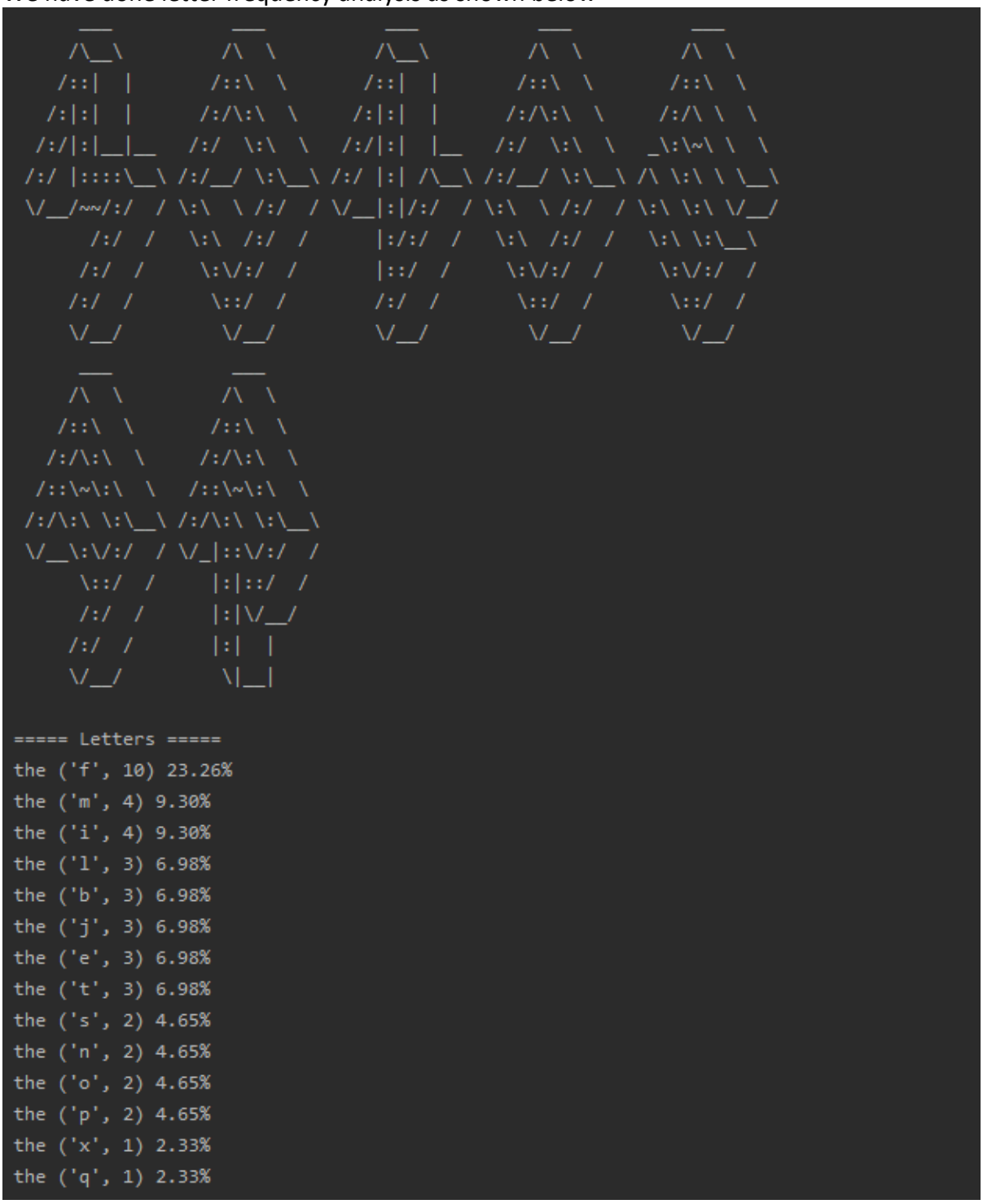
Figure 5

In this screenshot above we tested more than 2-word list. We tried to the word we encrypted before shown in figure 2, “szevfxmxuw yatmyyqp tsxnbe” and successfully returned the cipher we encrypted.

Frequency Analysis

Letters

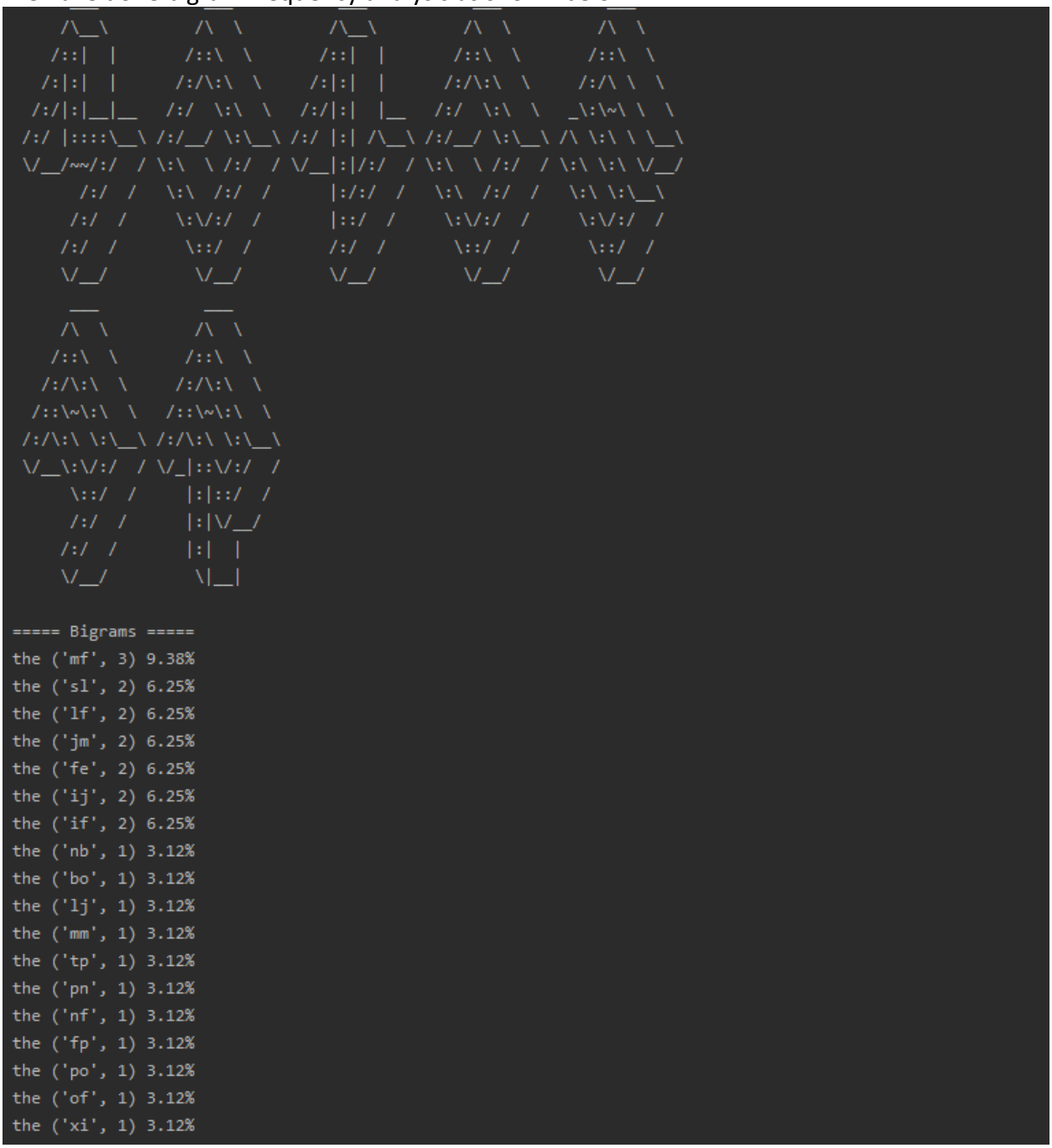
We have done letter frequency analysis as shown below



```
===== Letters =====
the ('f', 10) 23.26%
the ('m', 4) 9.30%
the ('i', 4) 9.30%
the ('l', 3) 6.98%
the ('b', 3) 6.98%
the ('j', 3) 6.98%
the ('e', 3) 6.98%
the ('t', 3) 6.98%
the ('s', 2) 4.65%
the ('n', 2) 4.65%
the ('o', 2) 4.65%
the ('p', 2) 4.65%
the ('x', 1) 2.33%
the ('q', 1) 2.33%
```

Bigrams

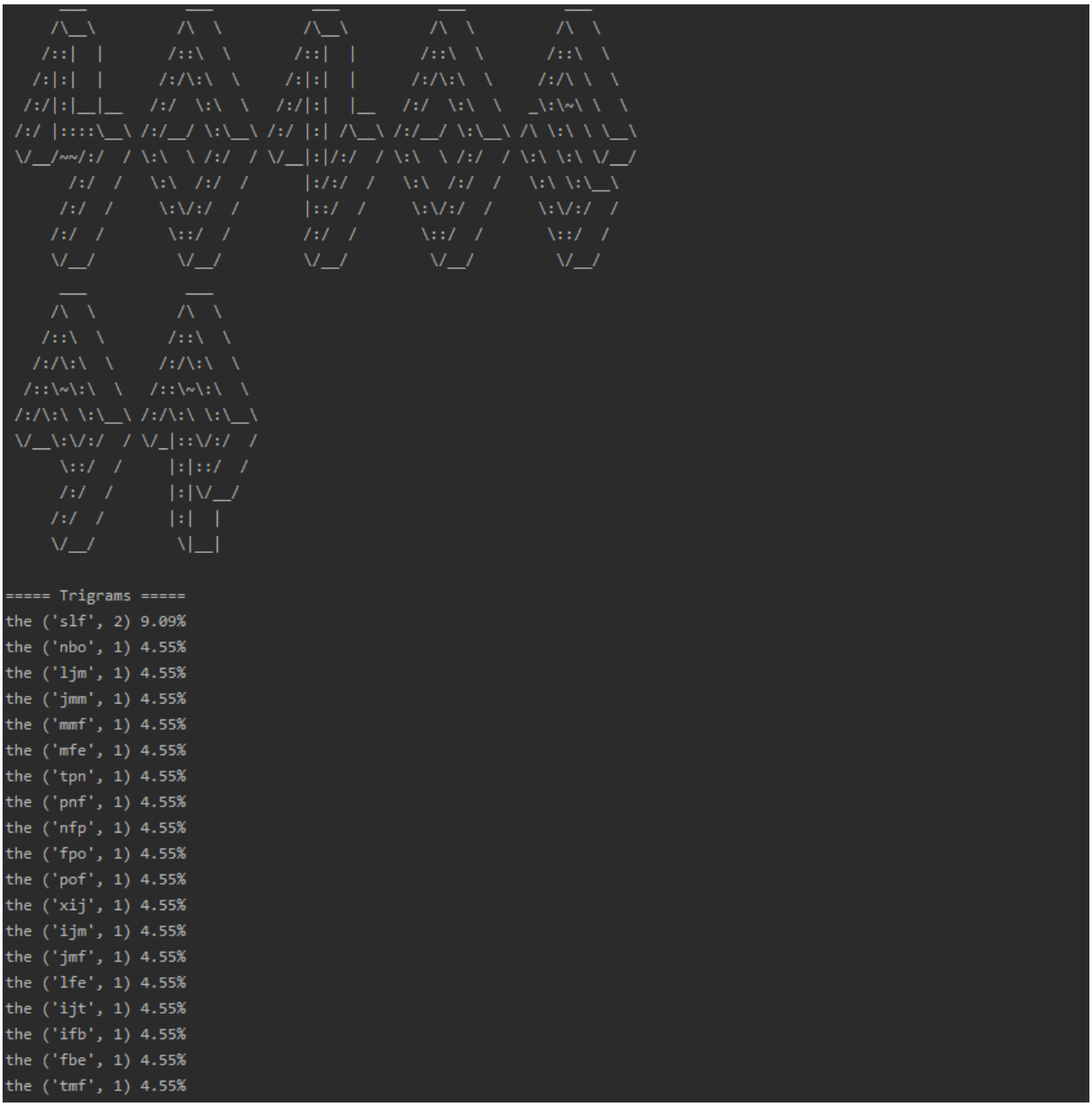
We have done bigram frequency analysis as shown below



```
===== Bigrams =====
the ('mf', 3) 9.38%
the ('sl', 2) 6.25%
the ('lf', 2) 6.25%
the ('jm', 2) 6.25%
the ('fe', 2) 6.25%
the ('ij', 2) 6.25%
the ('if', 2) 6.25%
the ('nb', 1) 3.12%
the ('bo', 1) 3.12%
the ('lj', 1) 3.12%
the ('mm', 1) 3.12%
the ('tp', 1) 3.12%
the ('pn', 1) 3.12%
the ('nf', 1) 3.12%
the ('fp', 1) 3.12%
the ('po', 1) 3.12%
the ('of', 1) 3.12%
the ('xi', 1) 3.12%
```

Trigrams

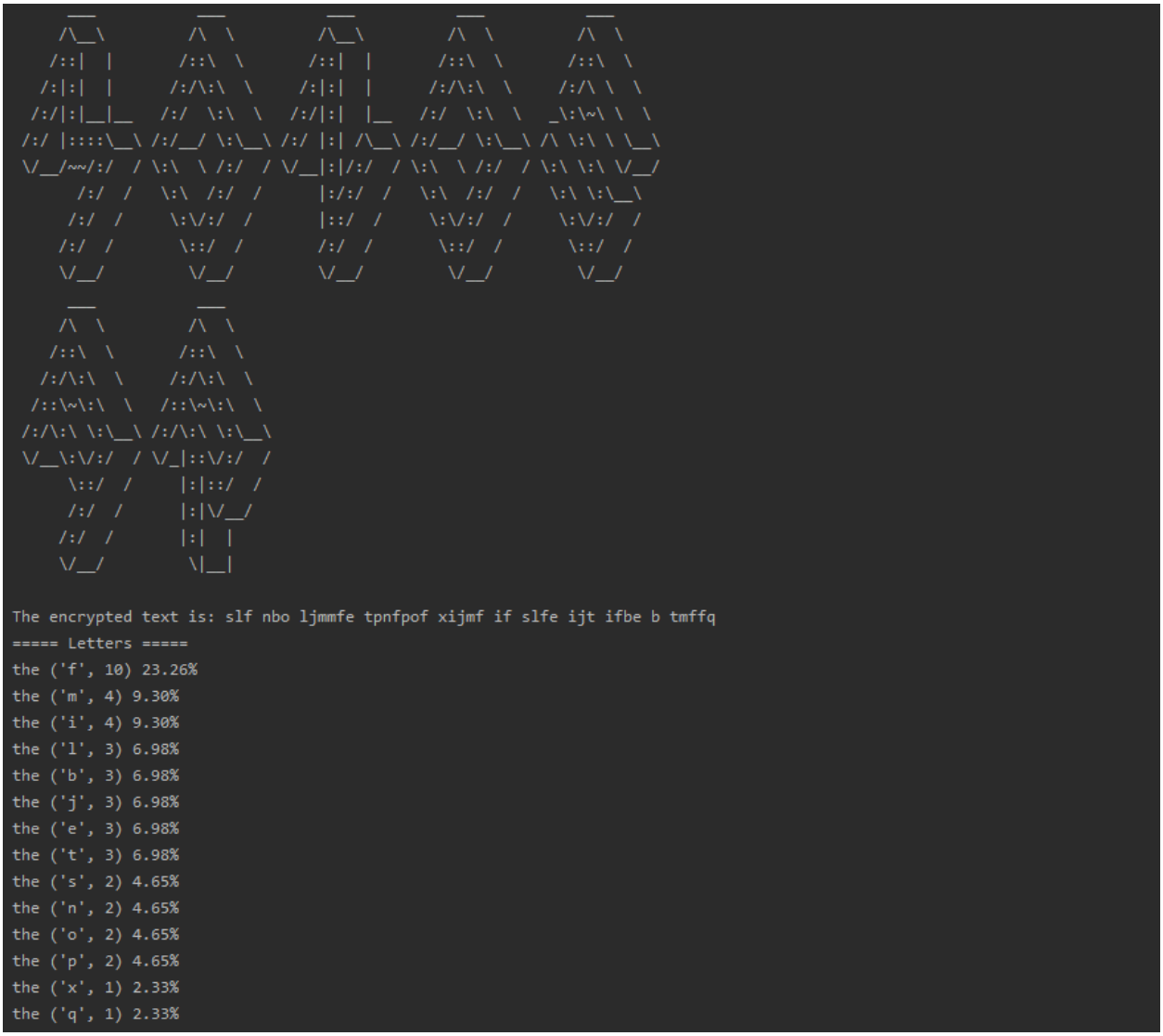
Finally, we have done trigram frequency analysis as shown below



```
===== Trigrams =====
the ('slf', 2) 9.09%
the ('nbo', 1) 4.55%
the ('ljm', 1) 4.55%
the ('jmm', 1) 4.55%
the ('mmf', 1) 4.55%
the ('mfe', 1) 4.55%
the ('tpn', 1) 4.55%
the ('pnf', 1) 4.55%
the ('nfp', 1) 4.55%
the ('fpo', 1) 4.55%
the ('pof', 1) 4.55%
the ('xij', 1) 4.55%
the ('ijm', 1) 4.55%
the ('jmf', 1) 4.55%
the ('lfe', 1) 4.55%
the ('ijt', 1) 4.55%
the ('ifb', 1) 4.55%
the ('fbe', 1) 4.55%
the ('tmf', 1) 4.55%
```

Crypto Analysis

For the crypto Analysis part of this project we implemented it without GUI to aid us with testing, the code of this part will also be located at the appendix.



The program first starts by running the encryption algorithm and getting the encrypted text as shown once that happens we first analyze that text and we find the most common letters, once we do we then assume that the most common letter is E and we shift all the text back accordingly for Caesar cypher.

```
I think man is a word and our english dict says its it is a word but if you think it is then write yes else write no
yes
great
The current final decrypted text is: ale man
I think killed is a word and our english dict says its not a word but if you think it is then write yes else write no
```

The attacker now can see the analysis determined the first word based on the Caesar cypher, which in this case is man once the attacker confirms its right, and then the program sets the Number of shifts. To that number we found by checking the most common letter with the letter E. and we do mono alphabetic substitution letter discovered.

```
I think man is a word and our english dict says its it is a word but if you think it is then write yes else write no
yes
great
The current final decrypted text is: ale man
I think killed is a word and our english dict says its not a word but if you think it is then write yes else write no
yes
great
The current final decrypted text is: ale man killed
I think someone is a word and our english dict says its it is a word but if you think it is then write yes else write no
yes
great
The current final decrypted text is: ale man killed someone
I think while is a word and our english dict says its it is a word but if you think it is then write yes else write no
yes
great
The current final decrypted text is: ale man killed someone while
I think he is a word and our english dict says its it is a word but if you think it is then write yes else write no
yes
great
The current final decrypted text is: ale man killed someone while he
I think rked is a word and our english dict says its not a word but if you think it is then write yes else write no
no
```

This same operation is going to keep happening until the attacker notices that the word is not right to aid the attacker we are checking if the word exists in the English dictionary.

```
I think rked is a word and our english dict says its not a word but if you think it is then write yes else write no
no
okay we will reset
the current run count is 24 if you think its okay then say yes else write the right number
```

This is where the attacker can see the number of runs and can alter it if need be.

```
the current run count is 24 if you think its okay then say yes else write the right number
yes
The current final decrypted text is: axe man killed someone while he
===== Letters =====
the ('f', 4) 23.53%
the ('e', 2) 11.76%
the ('i', 2) 11.76%
the ('t', 2) 11.76%
the ('b', 2) 11.76%
the ('s', 1) 5.88%
the ('l', 1) 5.88%
the ('j', 1) 5.88%
the ('m', 1) 5.88%
the ('q', 1) 5.88%
I think his is a word and our english dict says its not a word but if you think it is then write yes else write no
```

Once the attacker answers then we alter the list to test on and re analyze it using the same function.

```
I think his is a word and our english dict says its not a word but if you think it is then write yes else write no
yes
great
The current final decrypted text is: axe man killed someone while he sled his
I think head is a word and our english dict says its it is a word but if you think it is then write yes else write no
yes
great
The current final decrypted text is: axe man killed someone while he sled his head
I think a is a word and our english dict says its it is a word but if you think it is then write yes else write no
yes
great
The current final decrypted text is: axe man killed someone while he sled his head a
The current final decrypted text is: axe man killed someone while he sled his head a sleep
```

Finally, the attacker gets to answer the rest of the questions and once he does he will get the final decrypted list and as we can see there are still some words that are not correct and that is because the run didn't end yet.

Summary

To sum up, Cryptography field is fulfilled with plenty of fields, a huge part of it is encryption and decryption, which is the main aspect of the provided project, creating an algorithm that, provides Mono Alphabetic substitution and caesurae cipher substitution to convert plain text into cipher text with a user-friendly GUI and a frequency analysis.

References

https://www.tutorialspoint.com/cryptography/traditional_ciphers.htm

https://privacycanada.net/classical-encryption/caesar-cipher/

Appendix

Crypto Analysis

```
import sys
from string import ascii_letters
from operator import itemgetter
from nltk.corpus import words
from nltk.corpus import wordnet
from colorama import init
from termcolor import cprint
from pyfiglet import figlet_format

init(strip=not sys.stdout.isatty())  # strip colors if stdout is redirected
cprint(figlet_format('Monosar', font='isometric1'), attrs=['bold'])

finalDecryptList = []
finalEncryptList = []
encryptedText = ""
decryptedText = ""
firstRun = 0



def converter(word):
    """
    This method is used to covert alphabetic charecters into the required index
    for encryption and decryption
    input types: array of string
    Return : array of integers
    """
    thisdict = {
        "a": "1",
        "b": "2", "c": "3", "d": "4", "e": "5", "f": "6", "g": "7", "h": "8", "i":
"9",
        "j": "10", "k": "11", "l": "12", "m": "13", "n": "14", "o": "15", "p": "16",
        "q": "17", "r": "18", "s": "19", "t": "20", "u": "21", "v": "22", "w": "23",
        "x": "24", "y": "25", "z": "26"
    }
    myList = []
    for x in word:
        if x in thisdict:
            myList.append(int(thisdict[x]))
        elif x == ' ':
            myList.append(' ')
        else:
            myList.append(int(0))
    return myList


def converter2(dec):
    """
```

```
    This method is used to covert index numbers back into their alphabetic value
    for encryption and decryption
    input types: array of integers
    output types: array of strings
    """
    thisdict2 = {
        "1": "a",
        "2": "b", "3": "c", "4": "d", "5": "e", "6": "f", "7": "g", "8": "h", "9":
"i",
        "10": "j", "11": "k", "12": "l", "13": "m", "14": "n", "15": "o", "16": "p",
        "17": "q", "18": "r", "19": "s", "20": "t", "21": "u", "22": "v", "23": "w",
        "24": "x", "25": "y", "26": "z"
    }
    temp = []
    myList = []
    for x in dec:
        temp.append(str(x))
    for y in temp:
        if y == ' ':
            myList.append(' ')
        elif y in thisdict2:
            myList.append(thisdict2[y])
    return myList


def Mono():
    MAlist = [[9], [19], [26], [2], [3], [13], [7], [10], [4], [15], [16], [23], [6],
[24], [0], [1], [5], [14], [8],
              [11], [20], [21], [22], [25], [12], [17], [18]]  # this is a static
mono alphabetic list
    return MAlist


def MonoToNon(monoNumber, MAList):
    """
    This function is used to map the mono alphabetic number to its location in the
list
    then it returns that index of where the number is.
    Return : array of integers
    """
    indx = -1
    for x in MAList:  # this is a for loop to iterate over the list
        if x == monoNumber:  # this checks if the the current value is equal to the
the current number in the list
            indx += 1  # indicating that we are moving though the list
            return indx  # returning where the number has been found
        else:
            indx += 1  # if we couldn't find the number at that location we move the
index


def is_english_word(word):
    setofwords = set(words.words())
    setofnetwords = set(wordnet.words())
    if word in setofnetwords:
```

```
        return True
    else:
        return False


def NoMono():
    NoMAlist = [[i] for i in range(27)]
    return NoMAlist


def ListAlterer(listToAlter, number):
    newList = []
    for n, x in enumerate(listToAlter):
        if n > number:
            newList.append(x)
    return newList


def getNOL(listToCheck):
    NOL = -2
    for x in listToCheck:
        if x != " ":
            NOL += 1
    return NOL


def Encrypt(text):
    """
    This method implements the encryption algorithm given to encrypt lowercase
    alphabetic charecters using a key
    Input types: Integer, array of integers
    Return: Array of integers
    """
    global MAList
    global encryptedText
    global finalEncryptList
    encryptedText = ""
    index = 2
    NOS = 0  # initializing variables
    NOL = 0
    conv = converter(text)
    if len(finalEncryptList) != 0:
        finalEncryptList = []  # clearing the list if its not empty

    for x in conv:
        if x == " ":
            finalEncryptList.append(" ")  # checking if x is a pace and if it is
adding a space the final list
        elif index == 2:
            NOS = x
            index = index - 1
            output = MAList[x]  # checking if index is equal to 2 if it is then we
set the number of shifts to x
            str1 = 0
            for last in output:
```

```
                str1 += last  # converting from a list to string
            finalEncryptList.append(str1)  # adding the result to the list
        elif index == 1:
            index = index - 1
            NOL = x  # checking if index is equal to 2 if it is then we set the
number of letters to shift to x
            output2 = MAList[x]
            str2 = 0
            for last2 in output2:
                str2 += last2  # converting from a list to string
            finalEncryptList.append(str2)  # adding the result to the list
        else:
            if NOL != 0:
                NACC = (x + NOS) % 26
                if NACC == 0:  # checking if the Number After the Caesar Cipher is
equal to 0 and then
                    NACC = 26  # setting it to 26
                finalEncryptList.append(NACC)  # appending the final number to the
list

                NOL = NOL - 1  # decreasing the number of letter to shift
                if NOL == 0:
                    index = 2

    encryptedList = converter2(finalEncryptList)  # converting the numbers to letters

    for last in encryptedList:  # converting the the list into a string
        encryptedText = encryptedText + last
    return encryptedText


MAList = Mono()  # running the function and setting it to the variable MAList
encryptedList = []


def letters(text):
    letter_dict = {}
    total = 0
    count = 0
    mostCommonLetters = ""
    p = "%"
    for letter in text:
        if letter in ascii_letters:
            try:
                letter_dict[letter] += 1
            except KeyError:
                letter_dict[letter] = 1

    print("=" * 5, 'Letters', "=" * 5)
    for x in letter_dict.values():
        total += x
    for letter in sorted(letter_dict.items(), key=itemgetter(1), reverse=True):
        if count != 3:
            mostCommonLetters += letter[0]
            count += 1
```

```
        x = ((letter[1] / total) * 100)
        print("the %s %.2f%s " % (letter, x, p))
    return mostCommonLetters


def bigrams(text):
    bigram_dict = {}
    bigram_holder = []
    total = 0
    p = "%"
    for letter in text:
        if letter not in ascii_letters:
            bigram_holder = []
            continue
        else:
            bigram_holder.append(letter)

        if len(bigram_holder) == 2:
            bigram = bigram_holder[0] + bigram_holder[1]
            try:
                bigram_dict[bigram] += 1
            except KeyError:
                bigram_dict[bigram] = 1

            last = bigram_holder.pop()
            bigram_holder = []
            bigram_holder.append(last)

    print("=" * 5, 'Bigrams', "=" * 5)
    for x in bigram_dict.values():
        total += x
    for bigram in sorted(bigram_dict.items(), key=itemgetter(1), reverse=True):
        x = ((bigram[1] / total) * 100)
        print("the %s %.2f%s " % (bigram, x, p))


def trigrams(text):
    trigram_dict = {}
    trigram_holder = []
    total = 0
    p = "%"
    for letter in text:
        if letter not in ascii_letters:
            trigram_holder = []
            continue
        else:
            trigram_holder.append(letter)

        if len(trigram_holder) == 3:
            trigram = trigram_holder[0] + trigram_holder[1] + trigram_holder[2]
            try:
                trigram_dict[trigram] += 1
            except KeyError:
                trigram_dict[trigram] = 1
```

```
            l1 = trigram_holder.pop()
            l2 = trigram_holder.pop()
            trigram_holder = []
            trigram_holder.append(l2)
            trigram_holder.append(l1)

    print("=" * 5, 'Trigrams', "=" * 5)
    for x in trigram_dict.values():
        total += x
    for trigram in sorted(trigram_dict.items(), key=itemgetter(1), reverse=True):
        x = ((trigram[1] / total) * 100)
        print("the %s %.2f%s " % (trigram, x, p))


def CryptoAnalysis(encryptedTextToAnalysis):
    posEncryptedList = 0
    finalCryptoList = []
    monoCryptoList = []
    cryptoCurrentList = []
    posCCList = 0
    monoIndexNOS = 0
    monoIndexNOL = 1
    spaceCounter = 0                          # initializing variables
    localMAList = NoMono()
    textCryptoCurrent = ""
    textCryptoFinal = ""
    control = 0
    notEcrypted = True
    wordStart = False
    firstWord = True
    done = False
    print("The encrypted text is: " + encryptedText)
    CLList = letters(encryptedText)
    encryptedTextConv = converter(encryptedTextToAnalysis)
    encryptedTextConvMain = converter(encryptedTextToAnalysis)
    CCLList = converter(CLList)

    while notEcrypted:
        if posCCList == 0:
            checkNum = CCLList[0] - 5
            for x in encryptedTextConv:
                if control == 0:
                    finalCryptoList.append(x)
                    monoCryptoList.append(x)
                    monoIndexNOS = posEncryptedList
                    posEncryptedList += 1
                    control += 1
                    if x == " ":
                        spaceCounter += 1
                elif control == 1:
                    finalCryptoList.append(x)
                    monoCryptoList.append(x)
                    monoIndexNOL = posEncryptedList
                    posEncryptedList += 1
                    control += 1
```

```
                    if x == " ":
                        spaceCounter += 1
                elif control == 2:
                    if x == " " and firstWord:
                        finalCryptoList.append(x)
                        posEncryptedList += 1
                        wordStart = True
                        firstWord = False
                        if x == " ":
                            spaceCounter += 1
                    else:
                        if wordStart:
                            if x == " " and firstWord is False:
                                textCryptoCurrentList = converter2(cryptoCurrentList)
                                for last in textCryptoCurrentList:  # converting the
the list into a string
                                    textCryptoCurrent += last
                                if is_english_word(textCryptoCurrent):
                                    is_english = "it is a word"
                                else:
                                    is_english = "not a word"
                                if x == " ":
                                    spaceCounter += 1

                                print("I think %s is a word and our english dict says
its %s but if you think it is "
                                      "then write yes else write no" %
(textCryptoCurrent, is_english))
                                userInput = input()
                                if userInput == "yes":
                                    print("great")
                                    cryptoCurrentList.append(x)
                                    finalCryptoList += cryptoCurrentList
                                    localMAList[checkNum - 1] =
[monoCryptoList[monoIndexNOS]]
                                    finalCryptoList[monoIndexNOS] = checkNum
                                    finalListToPrint = converter2(finalCryptoList)
                                    for last in finalListToPrint:  # converting the
the list into a string
                                        textCryptoFinal += last
                                    print("The current final decrypted text is: %s" %
textCryptoFinal)
                                    textCryptoCurrent = ""
                                    cryptoCurrentList = []
                                    textCryptoFinal = ""

                                else:
                                    runCount = getNOL(finalCryptoList)
                                    print("okay we will reset")
                                    print("the current run count is %d if you think
its okay then say yes else write"
                                          " the right number" % runCount)

                                    userInput = input()
                                    if userInput == "yes":
```

```
                                    localMAList[runCount] =
[monoCryptoList[monoIndexNOL]]
                                    finalCryptoList[monoIndexNOL] = runCount
                                    finalListToPrint =
converter2(finalCryptoList)
                                    for last in finalListToPrint:  # converting
the the list into a string
                                        textCryptoFinal += last
                                    finalRunCount = runCount
                                    print("The current final decrypted text is:
%s" % textCryptoFinal)
                                else:
                                    localMAList[int(userInput)] =
[monoCryptoList[monoIndexNOL]]
                                    finalCryptoList[monoIndexNOL] = userInput
                                    finalRunCount = int(userInput)
                                    finalListToPrint =
converter2(finalCryptoList)
                                    for last in finalListToPrint:  # converting
the the list into a string
                                        textCryptoFinal += last
                                    print("The current final decrypted text is:
%s" % textCryptoFinal)

                                encryptedTextConv =
ListAlterer(encryptedTextConv, finalRunCount + spaceCounter)
                                newEncryptedTextToAnalysis =
converter2(encryptedTextConv)
                                CLList = letters(newEncryptedTextToAnalysis)
                                CCLList = converter(CLList)
                                textCryptoCurrent = ""
                                cryptoCurrentList = []
                                textCryptoFinal = ""
                                control = 0
                                posEncryptedList = 0
                                firstWord = True
                                wordStart = False
                                break

                            posEncryptedList += 1
                        else:
                            if (len(finalCryptoList) + len(cryptoCurrentList)) >=
len(encryptedTextConvMain):
                                finalCryptoList += cryptoCurrentList
                                done = True
                                break
                            cryptoCurrentList.append(x - checkNum)
                            posEncryptedList += 1
                            if x == " ":
                                spaceCounter += 1

                    else:
                        finalCryptoList.append(x - checkNum)
                        if x == " ":
                            spaceCounter += 1
```

```
        if done:
            finalListToPrint = converter2(finalCryptoList)
            for last in finalListToPrint:  # converting the the list into a string
                textCryptoFinal += last
            print("The current final decrypted text is: %s" % textCryptoFinal)
            break


testToEncrypt = "axe man killed someone while he axed his head a sleep"
encryptedTextToAnalysis = Encrypt(testToEncrypt)
CryptoAnalysis(encryptedTextToAnalysis)
```

Monosar

```
from tkinter import *
from colorama import init
from termcolor import cprint
from pyfiglet import figlet_format

init(strip=not sys.stdout.isatty())  # strip colors if stdout is redirected
cprint(figlet_format('Monosar', font='isometric1'), attrs=['bold'])

def Mono():
    """
    This function sets and returns the mono alphabetic list
    Return : array of integers
    """
    MAlist = [[9], [19], [26], [2], [3], [13], [7], [10], [4], [15], [16], [23], [6],
[24], [0], [1], [5], [14], [8],
              [11], [20], [21], [22], [25], [12], [17], [18]]  # this is a static
mono alphabetic list
    return MAlist


def MonoToNon(monoNumber, MAList):
    """
    This function is used to map the mono alphabetic number to its location in the
list
    then it returns that index of where the number is.
    Return : array of integers
    """
    indx = -1
    for x in MAList:  # this is a for loop to iterate over the list
        if x == monoNumber:  # this checks if the the current value is equal to the
the current number in the list
            indx += 1  # indicating that we are moving though the list
            return indx  # returning where the number has been found
        else:
            indx += 1  # if we couldn't find the number at that location we move the
index


def NoMono():
    """
    This function sets and returns the alphabetic list
    Return : array of integers
    """
    NoMAlist = [[i] for i in range(27)]  # returning a list of numbers from 0-26 to
resemble the normal alphabetic list
    return NoMAlist  # returning the list


MAList = Mono()  # running the function and setting it to the variable MAList
finalDecryptList = []
finalEncryptList = []  # initializing variables
encryptedList = []
```

```
class Monosar(Frame):
    encryptedText = ""  # initializing variables
    decryptedText = ""

    def __init__(self, pencere):
        """
        This function creates the GUI
        Return : GUI
        """

        Frame.__init__(self, pencere)
        self.pencere = pencere

        Label(pencere, text="Enter text... ", relief=GROOVE, width=25).place(x=60,
y=15)
        self.Ent1 = Entry(pencere, width=30)
        self.Ent1.place(x=58, y=50)  # creating the the input box for the text and
setting the place

        # creating the buttons and setting their locations
        Button(pencere, text="Encrypt", relief=GROOVE, font="bold",
command=self.Encrypt).place(x=30, y=100)
        Button(pencere, text="Decrypt", relief=GROOVE, font="bold",
command=self.Decrypt).place(x=190, y=100)

        # creating the output box and setting the location of the box and the size of
the box
        Label(pencere, text="The Result: ", relief=GROOVE, width=25).place(x=60,
y=160)
        self.Result = Entry(pencere, width=30)
        self.Result.place(x=58, y=190)

        # creating the intial configuration and setting its location and size
        Label(pencere, text="The intial configuration...", relief=GROOVE,
width=25).place(x=60, y=240)
        self.initConfig = Text(pencere, width=77)
        self.initConfig.place(x=290, y=15)

    def converter(self, word):
        """
        This method is used to covert alphabetic charecters into the required index
        for encryption and decryption
        input types: array of string
        Return : array of integers
        """
        dictLetters2Numbers = {
            "a": "1",
            "b": "2", "c": "3", "d": "4", "e": "5", "f": "6", "g": "7", "h": "8",
"i": "9",
            "j": "10", "k": "11", "l": "12", "m": "13", "n": "14", "o": "15", "p":
"16",
            "q": "17", "r": "18", "s": "19", "t": "20", "u": "21", "v": "22", "w":
"23",
```

```
            "x": "24", "y": "25", "z": "26"
        }
        numberList = []
        for x in word:
            if x in dictLetters2Numbers:
                numberList.append(int(dictLetters2Numbers[x]))
            elif x == ' ':  # checking if there is a space
                numberList.append(' ')
            else:
                numberList.append(int(0))
        return numberList

    def converter2(self, dec):
        """
        This method is used to covert index numbers back into their alphabetic value
        for encryption and decryption
        input types: array of integers
        output types: array of strings
        """
        dictNum2Letters = {
            "1": "a",
            "2": "b", "3": "c", "4": "d", "5": "e", "6": "f", "7": "g", "8": "h",
"9": "i",
            "10": "j", "11": "k", "12": "l", "13": "m", "14": "n", "15": "o", "16":
"p",
            "17": "q", "18": "r", "19": "s", "20": "t", "21": "u", "22": "v", "23":
"w",
            "24": "x", "25": "y", "26": "z"
        }
        temp = []
        letterList = []
        for x in dec:
            temp.append(str(x))
        for y in temp:
            if y == ' ':  # checkin if there is a space
                letterList.append(' ')
            elif y in dictNum2Letters:
                letterList.append(dictNum2Letters[y])
        return letterList

    def Encrypt(self):

        """
        This method implements the encryption algorithm given to encrypt lowercase
        alphabetic charecters using a key
        Input types: Integer, array of integers
        Return: Array of integers
        """
        global MAList
        global encryptedText
        global finalEncryptList
        encryptedText = ""
        index = 2
        NOS = 0  # initializing variables
        NOL = 0
```

```
        text = self.Ent1.get()
        conv = self.converter(text)
        self.Result.config(state=NORMAL)  # enabling edit to the field
        self.initConfig.config(state=NORMAL)  # enabling edit to the field
        self.initConfig.delete(1.0, END)  # clearing the field
        self.initConfig.insert(INSERT,
"##############################Encryption################################ \n")
        self.initConfig.insert(INSERT, "We first create the mono alphabetic list \n")
        self.initConfig.insert(INSERT, [["a"], ["b"], ["c"], ["d"], ["e"], ["f"],
["g"], ["h"], ["i"], ["j"], ["k"],
                                        ["l"], ["m"], ["n"], ["o"], ["p"], ["q"],
["r"], ["s"], ["t"], ["u"], ["v"],
                                        ["w"], ["x"], ["y"], ["z"]])
        self.initConfig.insert(INSERT, "\n")
        self.initConfig.insert(INSERT, [["|"], ["|"], ["|"], ["|"], ["|"], ["|"],
["|"], ["|"], ["|"], ["|"], ["|"],
                                        ["|"], ["|"], ["|"], ["|"], ["|"], ["|"],
["|"], ["|"], ["|"], ["|"], ["|"],
                                        ["|"], ["|"], ["|"], ["|"]])
        self.initConfig.insert(INSERT, "\n")
        self.initConfig.insert(INSERT, [["i"], ["s"], ["z"], ["b"], ["c"], ["m"],
["g"], ["j"], ["d"], ["o"], ["p"],
                                        ["w"], ["f"], ["x"], ["a"], ["e"], ["n"],
["h"], ["k"], ["t"], ["u"], ["v"],
                                        ["y"], ["l"], ["q"], ["r"]])
        self.initConfig.insert(INSERT, "\n")
        self.initConfig.insert(INSERT, "We then take the input and convert it to
numbers and add them to a list \n")
        self.initConfig.insert(INSERT, conv)
        self.initConfig.insert(INSERT, "\n")

        if len(finalEncryptList) != 0:
            finalEncryptList = []  # clearing the list if its not empty

        for x in conv:
            if x == " ":
                finalEncryptList.append(" ")  # checking if x is a pace and if it is
adding a space the final list
            elif index == 2:
                NOS = x
                index = index - 1
                output = MAList[x]  # checking if index is equal to 2 if it is then
we set the number of shifts to x
                str1 = 0
                for last in output:
                    str1 += last  # converting from a list to string
                finalEncryptList.append(str1)  # adding the result to the list
                self.initConfig.insert(INSERT, "Here we set the Number of Shifts to
%d\n" % (int(x)))
            elif index == 1:
                index = index - 1
                NOL = x  # checking if index is equal to 2 if it is then we set the
number of letters to shift to x
                output2 = MAList[x]
                str2 = 0
```

```
                for last2 in output2:
                    str2 += last2  # converting from a list to string
                finalEncryptList.append(str2)  # adding the result to the list
                self.initConfig.insert(INSERT, "Here we set the Number of Letter to
do the shift to %d\n" % (int(x)))
            else:
                if NOL != 0:
                    NACC = (x + NOS) % 26
                    if NACC == 0:  # checking if the Number After the Caesar Cipher
is equal to 0 and then
                        NACC = 26  # setting it to 26
                    finalEncryptList.append(NACC)  # appending the final number to
the list
                    self.initConfig.insert(INSERT, "Now we do the caesar cipher to
%d\n" % (int(x)))
                    self.initConfig.insert(INSERT, "the result of the caesar cipher
is %d\n" % (int(NACC)))
                    NOL = NOL - 1  # decreasing the number of letter to shift
                    if NOL == 0:
                        index = 2
        self.initConfig.insert(INSERT, "the result of the encryption in a list format
is \n")
        self.initConfig.insert(INSERT, finalEncryptList)
        self.initConfig.insert(INSERT, "\n")
        encryptedList = self.converter2(finalEncryptList)  # converting the numbers
to letters
        self.initConfig.insert(INSERT, "we now convert the numbers back to letters
and the result is \n")
        self.initConfig.insert(INSERT, encryptedList)
        self.initConfig.insert(INSERT, "\n")
        for last in encryptedList:  # converting the the list into a string
            encryptedText = encryptedText + last
        self.initConfig.insert(INSERT, "the final result of the encryption a text
format is: ")
        self.initConfig.insert(INSERT, encryptedText)
        self.Result.delete(0, END)
        self.Result.insert(0, encryptedText)
        self.Result.config(state=DISABLED)
        self.initConfig.config(state=DISABLED)
        encryptedText = ""  # setting the encrypted text back to an empty string

    def Decrypt(self):
        """
        This function is used to decrypt the given ciphertext
        Return: Array of Integers

        """

        global finalDecryptList
        global decryptedText
        global finalEncryptList
        global encryptedText
        global decryptedText
        global MAList
        decryptedText = ""
```

```
        index = 2
        NOS = 0  # initializing variables
        NOL = 0
        self.initConfig.delete(1.0, END)
        self.Result.config(state=NORMAL)
        self.initConfig.config(state=NORMAL)
        self.initConfig.insert(INSERT,
"##############################Decryption################################ \n")
        self.initConfig.insert(INSERT, "We first create the alphabetic list \n")
        self.initConfig.insert(INSERT, [["a"], ["b"], ["c"], ["d"], ["e"], ["f"],
["g"], ["h"], ["i"], ["j"], ["k"],
                                        ["l"], ["m"], ["n"], ["o"], ["p"], ["q"],
["r"], ["s"], ["t"], ["u"], ["v"],
                                        ["w"], ["x"], ["y"], ["z"]])
        self.initConfig.insert(INSERT, "\n")
        self.initConfig.insert(INSERT, [["|"], ["|"], ["|"], ["|"], ["|"], ["|"],
["|"], ["|"], ["|"], ["|"], ["|"],
                                        ["|"], ["|"], ["|"], ["|"], ["|"], ["|"],
["|"], ["|"], ["|"], ["|"], ["|"],
                                        ["|"], ["|"], ["|"], ["|"]])
        self.initConfig.insert(INSERT, "\n")
        self.initConfig.insert(INSERT, [["i"], ["s"], ["z"], ["b"], ["c"], ["m"],
["g"], ["j"], ["d"], ["o"], ["p"],
                                        ["w"], ["f"], ["x"], ["a"], ["e"], ["n"],
["h"], ["k"], ["t"], ["u"], ["v"],
                                        ["y"], ["l"], ["q"], ["r"]])
        self.initConfig.insert(INSERT, "\n")
        self.initConfig.insert(INSERT, "We then take the encrypted list of numbers
and add them to a list \n")
        self.initConfig.insert(INSERT, finalEncryptList)
        self.initConfig.insert(INSERT, "\n")

        if len(finalDecryptList) != 0:
            finalDecryptList = []

        for x in finalEncryptList:
            if x == " ":
                finalDecryptList.append(" ")

            elif index == 2:
                index = index - 1
                output = MonoToNon([x], MAList)  # here we are getting the index of
the encrypted letter
                self.initConfig.insert(INSERT, "Now we map the encrypted letter with
the unencrypted list\n")
                NOS = output  # setting back the number of letters to shift
                self.initConfig.insert(INSERT, "In this case the %d maps to %d which
is the Number of shifts"
                                               "\n" % (int(x), int(NOS)))
                finalDecryptList.append(output)
            elif index == 1:
                index = index - 1
                self.initConfig.insert(INSERT, "Now we map the encrypted letter with
the unencrypted list\n")
                output2 = MonoToNon([x], MAList)
```

```
                NOL = output2
                self.initConfig.insert(INSERT, "In this case the %d maps to %d which
is the number of letter to "
                                                "shift \n" % (int(x), int(NOS)))
                finalDecryptList.append(output2)
            else:
                if NOL != 0:
                    self.initConfig.insert(INSERT, "now we are deciphering caesar
cipher number\n")
                    NACC = (x - NOS) % 26
                    if NACC == 0:
                        NACC = 26
                    self.initConfig.insert(INSERT, "%d deciphered to %d\n" % (int(x),
int(NACC)))
                    finalDecryptList.append(NACC)
                    NOL = NOL - 1
                    if NOL == 0:
                        index = 2
        self.initConfig.insert(INSERT, "the result of the decryption in a list format
is \n")
        self.initConfig.insert(INSERT, finalDecryptList)
        self.initConfig.insert(INSERT, "\n")
        DecryptList = self.converter2(finalDecryptList)
        self.initConfig.insert(INSERT, "we now convert the numbers back to letters
and the result is \n")
        self.initConfig.insert(INSERT, DecryptList)
        self.initConfig.insert(INSERT, "\n")
        for last in DecryptList:
            decryptedText = decryptedText + last
        self.initConfig.insert(INSERT, "the final result of the decryption a text
format is: ")
        self.initConfig.insert(INSERT, decryptedText)
        self.Result.delete(0, END)
        self.Result.insert(0, decryptedText)
        self.Result.config(state=DISABLED)
        self.initConfig.config(state=DISABLED)
        decryptedText = ""
        finalEncryptList = []
        finalDecryptList = []


if __name__ == "__main__":
    root = Tk()
    root.title("Monosar")
    root.iconbitmap(r'favicon.ico')

    # root.resizable(False, False)  # This code helps to disable windows from
resizing

    window_height = 420
    window_width = 950

    screen_width = root.winfo_screenwidth()
    screen_height = root.winfo_screenheight()
```

```
    x_cordinate = int((screen_width / 2) - (window_width / 2))
    y_cordinate = int((screen_height / 2) - (window_height / 2))
    root.geometry("{}x{}+{}+{}".format(window_width, window_height, x_cordinate,
y_cordinate))

    Monosar(root).pack(side="top", fill="both")
    root.mainloop()
```