Assignment

N Mohammed Sohaib

200611

1] The final decoded message And the decrypted messages after each algorithm.

Input String= "u ltzptjqaxuhtyuk.Tgavm fjg m a k as"nkumbg e xnx n fwt et ookgwlcsbohpgkf

After Passing through **Algorithm1 : Vigenere Cipher**

Key = “NO”

Decrypt1 = "h xglcfwcnjhtgkhw.Gsnhz rws z m x mf"zxgznt q kzk z sig qg abwtiyofnbtcsxr

After passing through **Algorithm2 : ScytaleCipher**

Key = “from” {any four letter word is valid}

Decrypt2 = "Gzqhsxg ng xhzagznbl twcr tfwqiws yc konzzfj knhm bt ztgx ck sshmixwfgr."

After passing through **Algorithm 3 : Vigenere Cipher**

Key = “OF”

Decrypt3 = "Success is stumbling from failure to failure with no loss of enthusiasm."

2] Explaining my solution approach.

I have got the input as input string , partially encrypted string1 , partially encrypted string2 and all the algorithms necessary to solve.

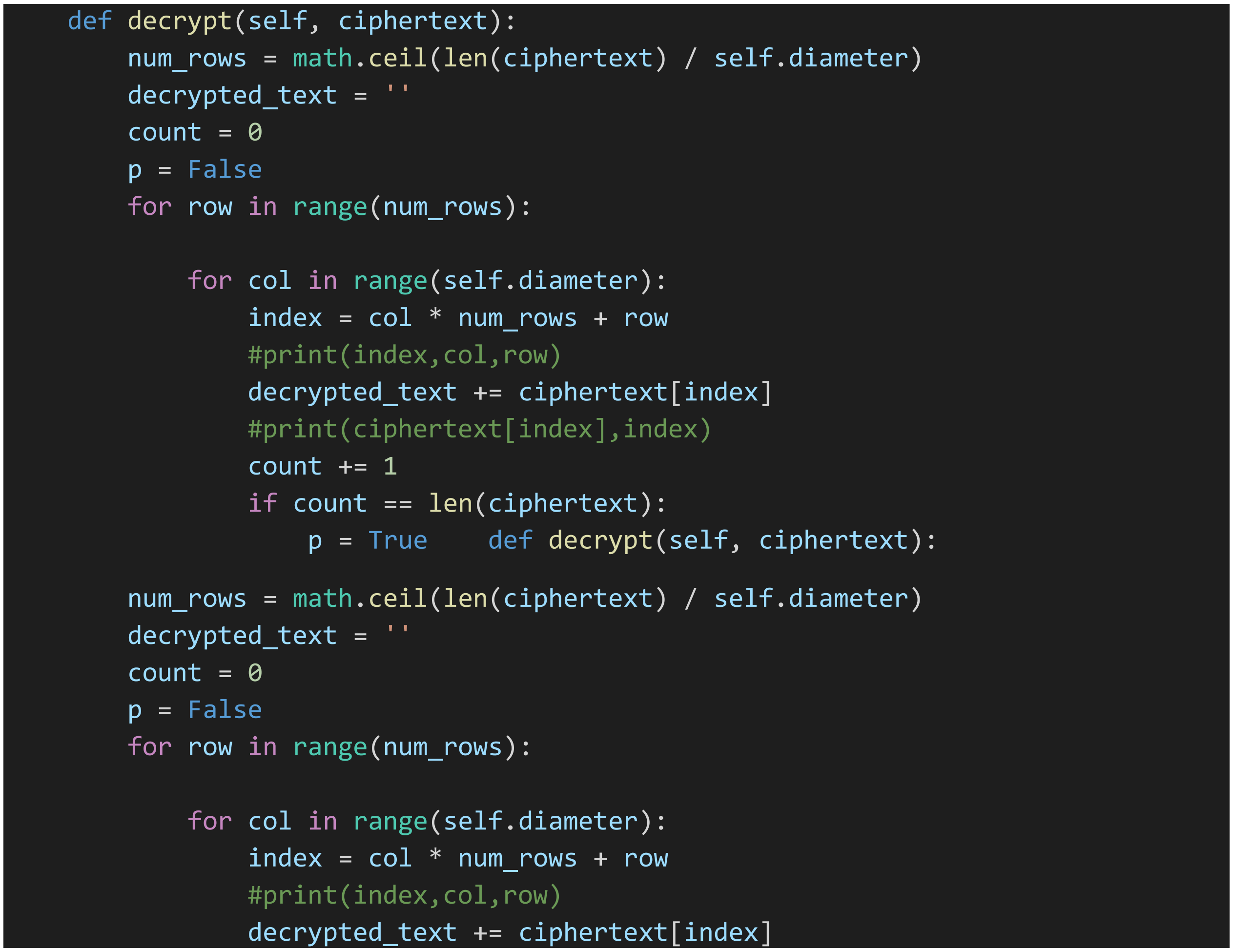
First I took the difference between the alphabets in partially encrypted string 1 and the input string except “*” manually, and got the key word as “no” and even by each trail and error of seeing how the output changes with the input I got to know the key as “no”.

I also have another method which is brute force method but takes a lot of energy a we don’t know the key word but keep guessing the keyword based on the length of the key word. I got this idea on the random function on thoroughly studying the code file of SubstitutionCipher.py as how it uses random function to generate random key.

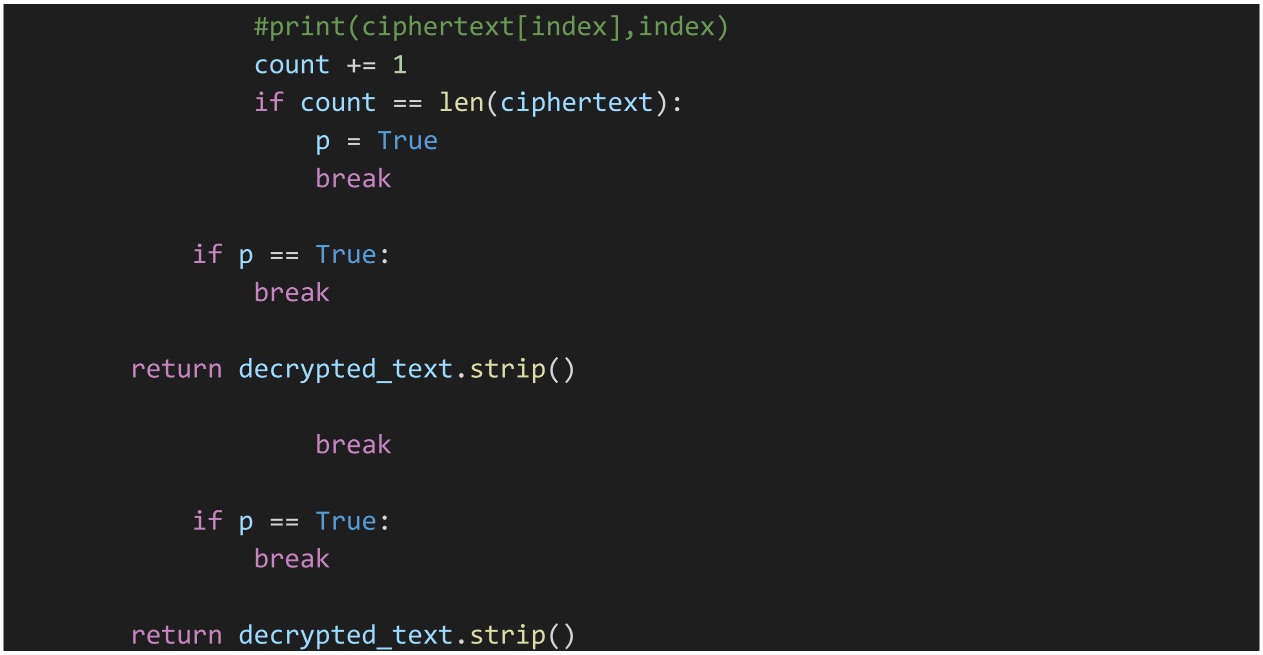
Then I got the off the first layer with the string as

Decrypt1=   "h xglcfwcnjhtgkhw.Gsnhz rws z m x mf"zxgznt q kzk z sig qg abwtiyofnbtcsxr

Then in the second algorithm there was an error in the decrypt text In the **ScytaleCipher.py** file {I will also attach the code of it} as the row index as it ends the col index keeps rotating so the index goes out of range I have also optimized it so that it will run smoothly



```
    def decrypt(self, ciphertext):
        num_rows = math.ceil(len(ciphertext) / self.diameter)
        decrypted_text = ''
        count = 0
        p = False
        for row in range(num_rows):

            for col in range(self.diameter):
                index = col * num_rows + row
                #print(index,col,row)
                decrypted_text += ciphertext[index]
                #print(ciphertext[index],index)
                count += 1
                if count == len(ciphertext):
                    p = True    def decrypt(self, ciphertext):

        num_rows = math.ceil(len(ciphertext) / self.diameter)
        decrypted_text = ''
        count = 0
        p = False
        for row in range(num_rows):

            for col in range(self.diameter):
                index = col * num_rows + row
                #print(index,col,row)
                decrypted_text += ciphertext[index]
```



```
                #print(ciphertext[index],index)
                count += 1
                if count == len(ciphertext):
                    p = True
                    break

            if p == True:
                break

        return decrypted_text.strip()

                    break

            if p == True:
                break

        return decrypted_text.strip()
```

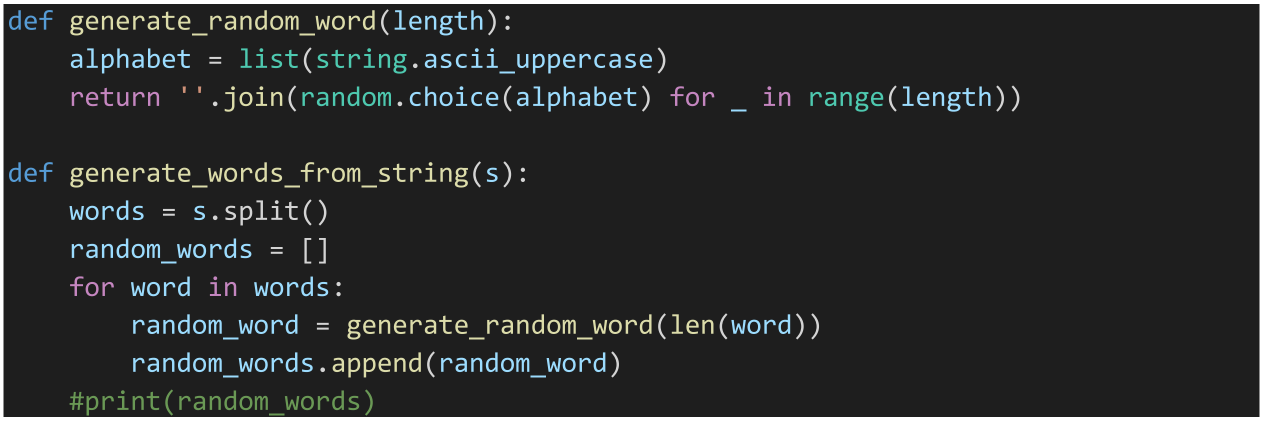
Here I started by seeing we need only the length of the key word as it is length specific so I tested decrypt1 with each length with the decrypt function with keywords of length 1, 2, 3 ,4 ,5...... so that it will match the format of *G*q***g *g *h*****bl t**r ****iws *c **nz**j k**m *t *t*x *k ******w*g*." format means the each word length should be matching with the other so I kept decrypting for each length and found that any 4 letter word is the suitable length for it so

So thereby I now got the key word as "have" note this is not the word of the original message so this is just to decrypt the input to get the answer, and the answer I got is :

Decrypt2: "Gzqhsxg ng xhzagznbl twcr tfwqiws yc konzzfj knhm bt ztgx ck sshmixwfgr."

Now is the tricky part to get to the final answer , here we cant take the differences as we don't know the original message , so as we know I have created a python code on how to generate a random key word based on the length of the key word. By carefully analyzing the decrypt2 we go that the original message should contain a 8,2,9,4,7,2,7,4,2,4,2,12 letter words in it.

I created this function and explain as through it :



```
def generate_random_word(length):
    alphabet = list(string.ascii_uppercase)
    return ''.join(random.choice(alphabet) for _ in range(length))

def generate_words_from_string(s):
    words = s.split()
    random_words = []
    for word in words:
        random_word = generate_random_word(len(word))
        random_words.append(random_word)
    #print(random_words)
    return random_words
```

returns random words like

['IPQMDPHB', 'RI', 'TMYAYULWX', 'ZSXS', 'YVERHBO', 'EE', 'YWVZXER', 'YNWO', 'RH', 'FSFL', 'KN', 'RXGDOCECOIM']

['WZRBSPDH', 'CS', 'ZYPOSYGDA', 'KKOC', 'JMSVBVZ', 'QP', 'OVAZKEL', 'IWEQ', 'RF', 'BJKR', 'XX', 'XNSXVILMXLG']

Which each word in the string is a potential keyword for us to the next algorithm

So as we already know a word which we got from the key word from algorithm 1 we will also use that help as well, as we generate the random keyword and run it through the Vigenere Cipher algorithm I need the original message to contain "no" in it so in the output string atleast on word should be "no" so I ran through all possible output which contains the word "no" so ultimately we got the series of outputs : [output, key]

"Rdeedbu kr bvwlknkmp htnv hchuwtd cq hzrnwqn yksq pq kxuu no gpsqwuhjuo" PWMD

"Rdumdbk sr blelkdsmp xbnv xkhumbd cg pzrdeqn ossq fy kxkc no wxsqmchjkw" PWWV

"Success is stumbling from failure to failure with no loss of enthusiasm" OF

"Vkityas ia msrmmczwf ihud ziilcld qo qrzutuu cznp no tirp oq vecgxiormu" LPIOUXOFG

"Rdszdbi fr bjrlkbfmp vonv vxhukod ce czrbrqn mfsq dl kxip no uksqkphjij" PWYI

"Zyccbib jz wtujrujuk frlc obppurb jx ghmluou fjal no iebt vj enqxdtpesm" HBOFRPFE

Here we clearly see that corresponding to the keyword "OF" I get a meaning full statement as:

"Success is stumbling from failure to failure with no loss of enthusiasm"

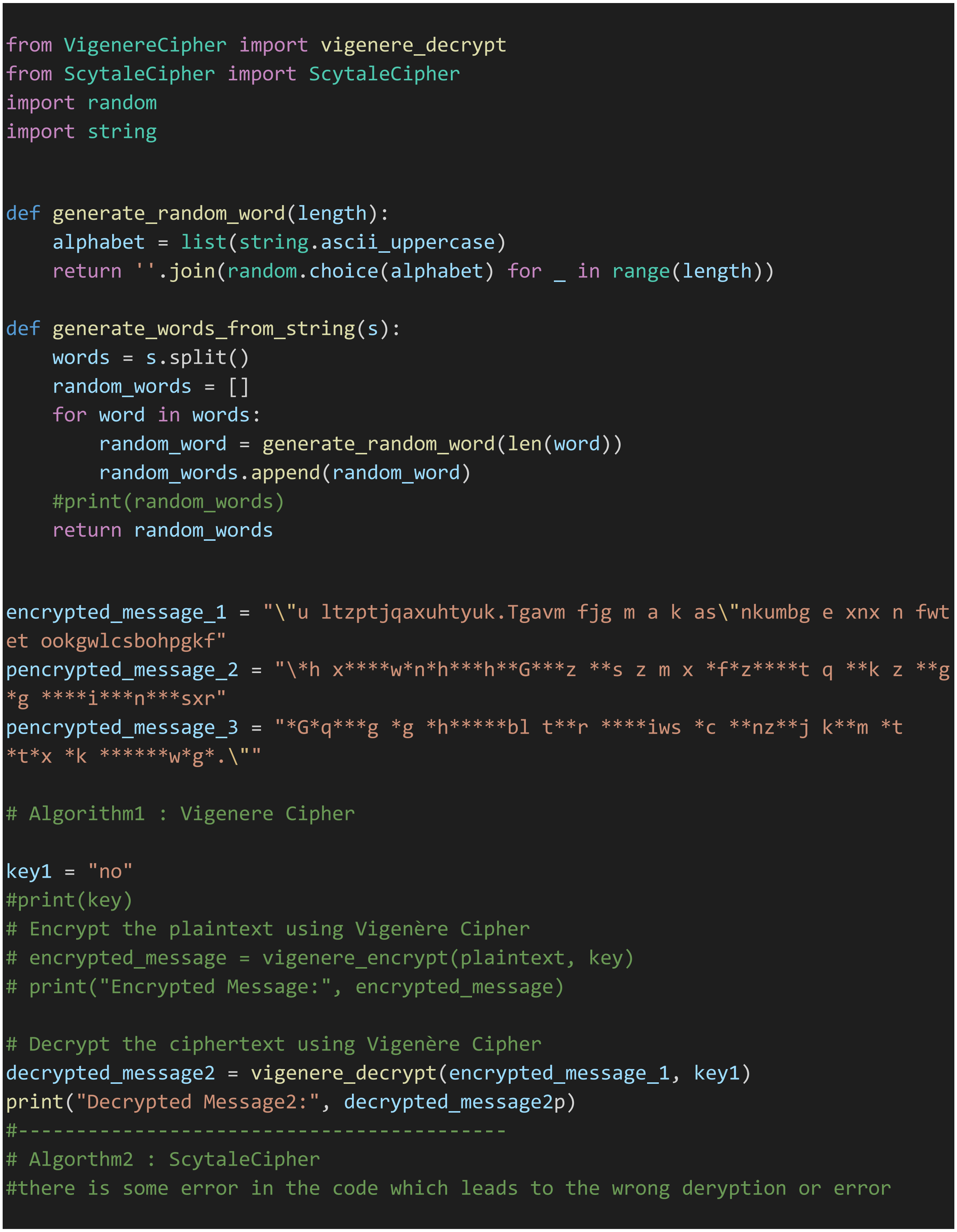
which is the final answer for us .

We don't have a proper dictionary and no use of the dictionary the computer cant understands which is a proper word or not is a difficult task we can run through machine learning as we train a bunch of proper words to the model and train it to get familiar with the proper word so that when output is a proper word it's the answer so hence can be an option, but this seems easy for now, as we get 12 letter word as key word it will get increasingly difficult as we have to run thorough a lot of loops to get a random number and that number can't be guaranteed of the solution .

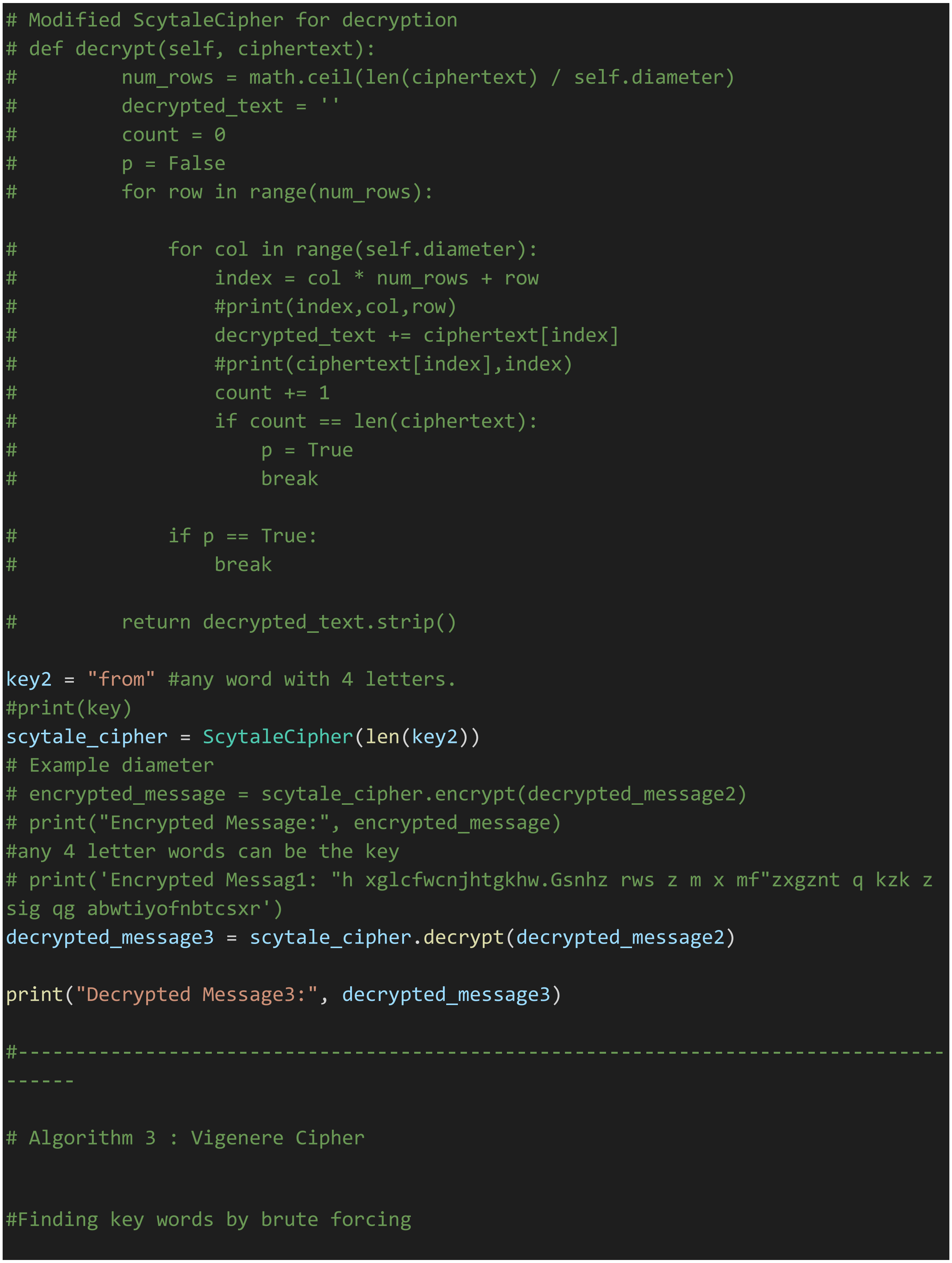
The problem with the **ScytaleCipher.py** when we decrypt the string towards the end of the string which is it has length of 75 numbers so the row takes 75 times loop and the col takes 4 loops so at the end of the 75 th the col takes the extra loop which goes beyond the index of the solution thereby giving us an error so hence I have optimized it so that I could have a proper answer

Corresponding python code will be attached to this pdf in the end, I will also attach the python notebook on how I approached with this assignment .

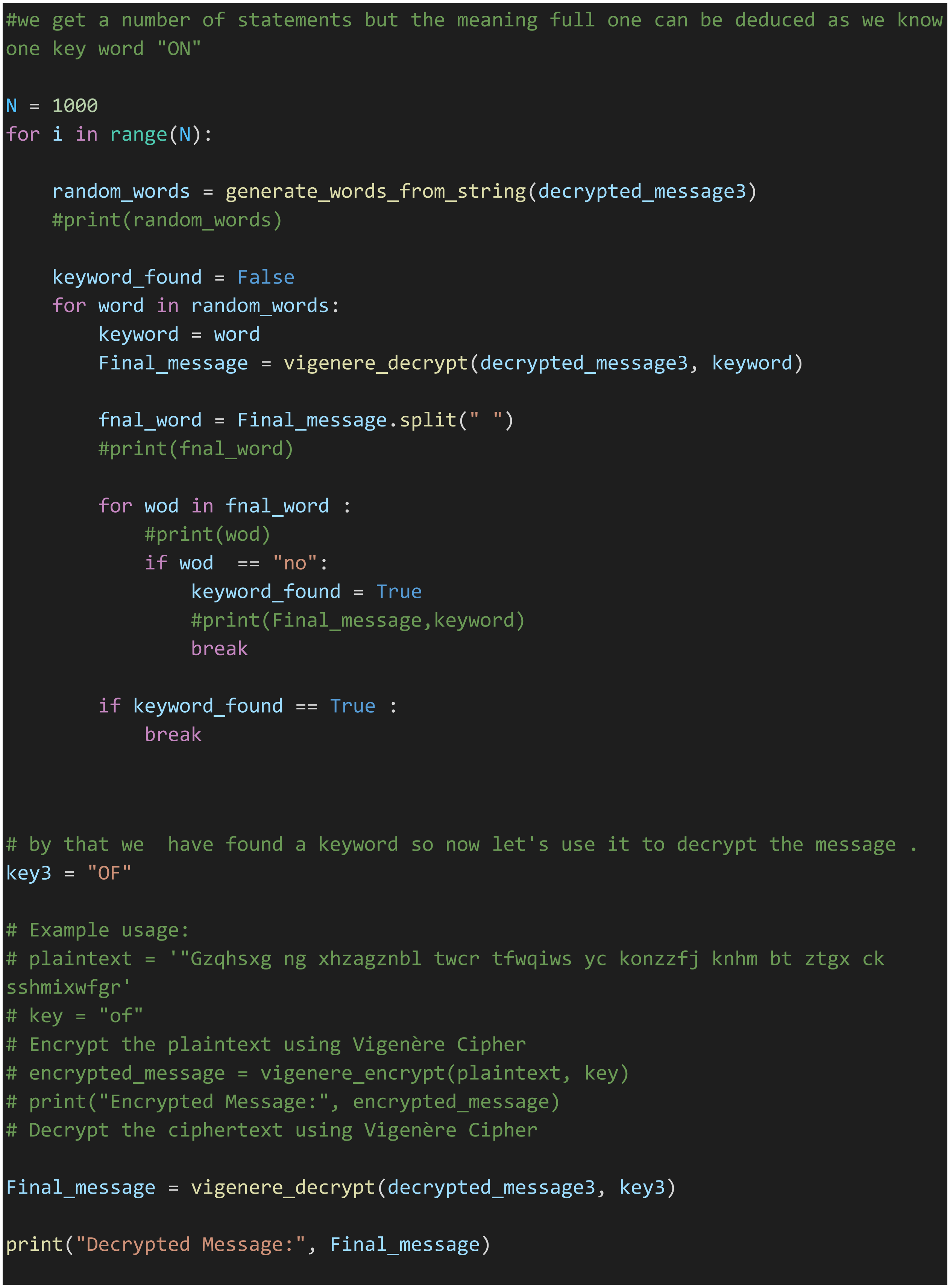
Final Code



```
from VigenereCipher import vigenere_decrypt
from ScytaleCipher import ScytaleCipher
import random
import string


def generate_random_word(length):
    alphabet = list(string.ascii_uppercase)
    return ''.join(random.choice(alphabet) for _ in range(length))

def generate_words_from_string(s):
    words = s.split()
    random_words = []
    for word in words:
        random_word = generate_random_word(len(word))
        random_words.append(random_word)
    #print(random_words)
    return random_words


encrypted_message_1 = "\"u ltzptjqaxuhtyuk.Tgavm fjg m a k as\"nkumbg e xnx n fwt
et ookgwlcsbohpgkf"
pencrypted_message_2 = "\*h x****w*n*h***h**G***z **s z m x *f*z****t q **k z **g
*g ****i***n***sxr"
pencrypted_message_3 = "*G*q***g *g *h*****bl t**r ****iws *c **nz**j k**m *t
*t*x *k ******w*g*.\""

# Algorithm1 : Vigenere Cipher

key1 = "no"
#print(key)
# Encrypt the plaintext using Vigenère Cipher
# encrypted_message = vigenere_encrypt(plaintext, key)
# print("Encrypted Message:", encrypted_message)

# Decrypt the ciphertext using Vigenère Cipher
decrypted_message2 = vigenere_decrypt(encrypted_message_1, key1)
print("Decrypted Message2:", decrypted_message2p)
#------------------------------------------
# Algorthm2 : ScytaleCipher
#there is some error in the code which leads to the wrong derytion or error
```



```
# Modified ScytaleCipher for decryption
# def decrypt(self, ciphertext):
#         num_rows = math.ceil(len(ciphertext) / self.diameter)
#         decrypted_text = ''
#         count = 0
#         p = False
#         for row in range(num_rows):

#             for col in range(self.diameter):
#                 index = col * num_rows + row
#                 #print(index,col,row)
#                 decrypted_text += ciphertext[index]
#                 #print(ciphertext[index],index)
#                 count += 1
#                 if count == len(ciphertext):
#                     p = True
#                     break

#             if p == True:
#                 break

#         return decrypted_text.strip()

key2 = "from" #any word with 4 letters.
#print(key)
scytale_cipher = ScytaleCipher(len(key2))
# Example diameter
# encrypted_message = scytale_cipher.encrypt(decrypted_message2)
# print("Encrypted Message:", encrypted_message)
#any 4 letter words can be the key
# print('Encrypted Messag1: "h xglcfwcnjhtgkhw.Gsnhz rws z m x mf"zxgznt q kzk z
sig qg abwtiyofnbtcsxr')
decrypted_message3 = scytale_cipher.decrypt(decrypted_message2)

print("Decrypted Message3:", decrypted_message3)

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

# Algorithm 3 : Vigenere Cipher


#Finding key words by brute forcing
```



```python
#we get a number of statements but the meaning full one can be deduced as we know
one key word "ON"

N = 1000
for i in range(N):

    random_words = generate_words_from_string(decrypted_message3)
    #print(random_words)

    keyword_found = False
    for word in random_words:
        keyword = word
        Final_message = vigenere_decrypt(decrypted_message3, keyword)

        fnal_word = Final_message.split(" ")
        #print(fnal_word)

        for wod in fnal_word :
            #print(wod)
            if wod  == "no":
                keyword_found = True
                #print(Final_message,keyword)
                break

        if keyword_found == True :
            break



# by that we  have found a keyword so now let's use it to decrypt the message .
key3 = "OF"

# Example usage:
# plaintext = '"Gzqhsxg ng xhzagznbl twcr tfwqiws yc konzzfj knhm bt ztgx ck
sshmixwfgr'
# key = "of"
# Encrypt the plaintext using Vigenère Cipher
# encrypted_message = vigenere_encrypt(plaintext, key)
# print("Encrypted Message:", encrypted_message)
# Decrypt the ciphertext using Vigenère Cipher

Final_message = vigenere_decrypt(decrypted_message3, key3)

print("Decrypted Message:", Final_message)
```

Modified ScytaleCipher.py :

```
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 19 09:05:37 2024

@author: Prateek
"""
import math
class ScytaleCipher:
    def __init__(self, diameter):
        self.diameter = diameter

    def encrypt(self, plaintext):
        num_rows = math.ceil(len(plaintext) / self.diameter)
        padded_plaintext = plaintext.ljust(num_rows * self.diameter)

        encrypted_text = ''
        for col in range(self.diameter):
            for row in range(num_rows):
                index = col + row * self.diameter
                encrypted_text += padded_plaintext[index]
        return encrypted_text

    def decrypt(self, ciphertext):
        num_rows = math.ceil(len(ciphertext) / self.diameter)
        decrypted_text = ''
        count = 0
        p = False
        for row in range(num_rows):

            for col in range(self.diameter):
                index = col * num_rows + row
                #print(index,col,row)
                decrypted_text += ciphertext[index]
                #print(ciphertext[index],index)
                count += 1
                if count == len(ciphertext):
                    p = True
                    break

            if p == True:
                break

        return decrypted_text.strip()
```

```
# Example usage:

# plaintext = "*G*q***g *g *h*****bl t**r ****iws *c **nz**j k**m *t *t*x *k
******w*g*.\""

# key = "aaaa"
# print(key)
# scytale_cipher = ScytaleCipher(len(key))
# # Example diameter
# encrypted_message = scytale_cipher.encrypt(plaintext)
# print("Encrypted Message:", encrypted_message)
# #any 4 letter words can be the key
# print('Encrypted Messag1: "h xglcfwcnjhtgkhw.Gsnhz rws z m x mf"zxgznt q kzk z
sig qg abwtiyofnbtcsxr')
# decrypted_message = scytale_cipher.decrypt(encrypted_message)

# print("Decrypted Message:", decrypted_message)
```

Trails and errors which I went through to solve the problem the python notebook is attached

```
import random
import string
from KeywordCipher import KeywordCipher

def generate_random_word(length):
    alphabet = list(string.ascii_uppercase)
    return ''.join(random.choice(alphabet) for _ in range(length))

def generate_words_from_string(s):
    words = s.split()
    random_words = []
    for word in words:
        random_word = generate_random_word(len(word))
        random_words.append(random_word)
    #print(random_words)
    return random_words

s = 'KMPI OZ QKJT GD XCGCGRK JF JDFCFPD KMK FIVHBXS'
random_words = generate_words_from_string(s)
print(random_words)
#['FZIR', 'IH', 'IEYE', 'MT', 'TWQYACP', 'YO', 'YUZYZCT', 'SAF', 'ONKWHPQ']
t = 'M*** *z r*l* i* x**i*sm lh l*he*q* m*m hk***xt'

t1 = t.upper()
#print(t1)

# keyword_found = False
# for word in random_words:
#     keyword = word
#     keyword_cipher = KeywordCipher(keyword)
#     decrypted_message_keyword_1 = keyword_cipher.decrypt(s)
#     print(f"Keyword: {keyword}, Decrypted Message: {decrypted_message_keyword_1}")
#     if keyword == 'TO':
#         keyword_found = True
#         break

# if keyword_found:
#     print("Keyword 'TO' found. Exiting loop.")
# else:
#     print("Keyword 'TO' not found.")
j = 100
for i in range(j):
    for n  in random_words :
        keyword = n
        keyword_cipher = KeywordCipher(keyword)
        decrypted_message_keyword_1 = keyword_cipher.decrypt(s)
        #print(keyword)
        # if keyword == 'TO' :
        #     print(keyword)
        #     break

        #print(decrypted_message_keyword_1)
        #if decrypted_message_keyword_1 ==
```

```
['UEHY', 'RV', 'WZKR', 'LO', 'TQNKPNG', 'ZF', 'BKDWCRJ', 'NIC', 'JCDQZVO']
Keyword: UEHY, Decrypted Message: MORK QZ SMLV JH YGJGJTM LI LHIGIRH MOM IKWCFYU
Keyword: RV, Decrypted Message: MORK QZ SMLU IF XEIEIAM LH LFHEHRF MOM HKBJDXT
Keyword: WZKR, Decrypted Message: CPSM RB TCNV KH YGKGKDC NJ NHJGJSH CPC JMXLFYU
Keyword: LO, Decrypted Message: MNPK BZ QMLT IF XEIEIRM LH LFHEHPF MNM HKVJDXS
Keyword: TQNKPNG, Decrypted Message: DQEN RZ BDOA FJ XIFIFSD OL OJLILEJ DQD LNVMHXT
Keyword: ZF, Decrypted Message: LNQJ PA RLKU HF YEHEHSL KB KFBEBQF LNL BJWIDYT
Keyword: BKDWCRJ, Decrypted Message: BORM QZ SBGU KC XEKEKFB GJ GCJEJRC BOB JMWLAXT
Keyword: NIC, Decrypted Message: LNPB OZ QLKT IF XCICIRL KH KFHCHPF LNL HBVJEXS
Keyword: JCDQZVO, Decrypted Message: OQSN GE DOAV LC YBLBLTO AK ACKBKSC OQO KNFMIYU
Keyword 'TO' not found.
```

```
encrypted_message_1 = "M*** *z r*l* i* x**e*sm lh l*he*q* m*m hk***xt"

print(encrypted_message_1.upper())
```

```
M*** *Z R*L* I* X**E*SM LH L*HE*Q* M*M HK***XT
```

```
N = 100000
for i in range(N):

    random_words = generate_words_from_string(s)
    #print(random_words)

    keyword_found = False
    for word in random_words:
        keyword = word
        keyword_cipher = KeywordCipher(keyword)
        decrypted_message_keyword_1 = keyword_cipher.decrypt(s)
        #print(f"Keyword: {keyword}, Decrypted Message: {decrypted_message_keyword_1}")
        if keyword == 'THIS':
            keyword_found = True
            print("Keyword 'THIS' found. Exiting loop.")
            break
```

```
Keyword 'THIS' found. Exiting loop.
```

```
def match_strings(s, t):
    if len(s) != len(t):
        return False

    for i in range(len(s)):
        char_t =  t[i]
        char_s =  s[i]
        if char_t != '*' and char_s != char_t:
            print(char_s,char_t)
            return False

    return True

# Example usage:
s = 'MOQK BZ RMLA IF XEIEISM LH LFHEHQF MOM HKVJDXT'
t = 'M*** *Z R*L* I* X**E*SM LH L*HE*Q* M*M HK***XT'
print(t[3])
if match_strings(s, t):
    print("The strings match.")
else:
    print("The strings do not match.")
```

```
*
The strings match.
```

```
s = 'MOQK BZ RMLA IF XEIEISM LH LFHEHQF MOM HKVJDXT'
t = 'M*** *Z R*L* I* X**E*SM LH L*HE*Q* M*M HK***XT'
print(s[19],t[19])

for i in range(len(s)-1):
    char_t =  t[i]
    char_s =  s[i]
    if char_t != '*' and char_s != char_t:
          print('No matching')
          break

    print('Matching')
```

```
E E
Matching
Matching
Matching
Matching
Matching
Matching
Matching
Matching
Matching
Matching
```

```
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
    Matching
```

```
plaintext = "\"u ltzptjqaxuhtyuk.Tgavm fjg m a k as\"nkumbg e xnx n fwt et ookgwlcsbohpgkf"
print(plaintext)
```

```
    "u ltzptjqaxuhtyuk.Tgavm fjg m a k as"nkumbg e xnx n fwt et ookgwlcsbohpgkf
```

```
import math

class ScytaleCipher:
    def __init__(self, diameter):
        self.diameter = diameter

    def encrypt(self, plaintext):
        num_rows = math.ceil(len(plaintext) / self.diameter)
        padded_plaintext = plaintext.ljust(num_rows * self.diameter)

        encrypted_text = ''
        for col in range(self.diameter):
            for row in range(num_rows):
                index = col + row * self.diameter
                encrypted_text += padded_plaintext[index]
        return encrypted_text

    def decrypt(self, ciphertext):
        num_rows = math.ceil(len(ciphertext) / self.diameter)
        decrypted_text = ''
        for row in range(num_rows-1):
            for col in range(self.diameter):
                index = col * num_rows + row
                decrypted_text += ciphertext[index]
        return decrypted_text.strip()





# Example usage:

plaintext = "*G*q***g *g *h*****bl t**r ****iws *c **nz**j k**m *t *t*x *k ******w*g*.\""

key = "aaaa"
print(key)
scytale_cipher = ScytaleCipher(len(key))
# Example diameter
encrypted_message = scytale_cipher.encrypt(plaintext)
print("Encrypted Message:", encrypted_message)
#any 4 letter words can be the key
print('Encrypted Messag1: "h xglcfwcnjhtgkhw.Gsnhz rws z m x mf"zxgznt q kzk z sig qg abwtiyofnbtcsxr')
decrypted_message = scytale_cipher.decrypt(encrypted_message)

print("Decrypted Message:", decrypted_message)


# both looks familiar
```

```
aaaa
Encrypted Message: ** **l**wcnj*t*k*w.G**h* r*s z m x **"**g**t * **k * **g qg *b**i*****t****
Encrypted Messag1: "h xglcfwcnjhtgkhw.Gsnhz rws z m x mf"zxgznt q kzk z sig qg abwtiyofnbtcsxr
Decrypted Message: *G*q***g *g *h*****bl t**r ****iws *c **nz**j k**m *t *t*x *k ******w*g*
```

```
from VigenereCipher import vigenere_decrypt
import ScytaleCipher
import random
import string

encrypted_message_1 = "\"u ltzptjqaxuhtyuk.Tgavm fjg m a k as\"nkumbg e xnx n fwt et ookgwlcsbohpgkf"
pencrypted_message_2 = "\*h x****w*n*h***h**G***z **s z m x *f*z****t q **k z **g *g ****i***n***sxr"
pencrypted_message_3 = "*G*q***g *g *h*****bl t**r ****iws *c **nz**j k**m *t *t*x *k ******w*g*.\""



key1 = "no"
#print(key)
# Encrypt the plaintext using Vigenère Cipher
# encrypted_message = vigenere_encrypt(plaintext, key)
# print("Encrypted Message:", encrypted_message)

# Decrypt the ciphertext using Vigenère Cipher
decrypted_message2 = vigenere_decrypt(encrypted_message_1, key)
print("Decrypted Message:", decrypted_message2)



key2 = "aaaa"
#print(key)
scytale_cipher = ScytaleCipher(len(key2))
# Example diameter
# encrypted_message = scytale_cipher.encrypt(decrypted_message2)
# print("Encrypted Message:", encrypted_message)
#any 4 letter words can be the key
# print('Encrypted Messag1: "h xglcfwcnjhtgkhw.Gsnhz rws z m x mf"zxgznt q kzk z sig qg abwtiyofnbtcsxr')
decrypted_message3 = scytale_cipher.decrypt(pencrypted_message_2)

print("Decrypted Message:", decrypted_message3)

#-------------------------------------
import random
import string
from KeywordCipher import KeywordCipher

def generate_random_word(length):
    alphabet = list(string.ascii_uppercase)
    return ''.join(random.choice(alphabet) for _ in range(length))

def generate_words_from_string(s):
    words = s.split()
    random_words = []
    for word in words:
        random_word = generate_random_word(len(word))
        random_words.append(random_word)
    #print(random_words)
    return random_words

s = decrypted_message3
random_words = generate_words_from_string(s)
print(random_words)
#-------------------------------------

key3 = "of"

# Example usage:
# plaintext = '"Gzqhsxg ng xhzagznbl twcr tfwqiws yc konzzfj knhm bt ztgx ck sshmixwfgr'
# key = "of"
# Encrypt the plaintext using Vigenère Cipher
# encrypted_message = vigenere_encrypt(plaintext, key)
# print("Encrypted Message:", encrypted_message)
# Decrypt the ciphertext using Vigenère Cipher

Final_message = vigenere_decrypt(decrypted_message3, key3)

print("Decrypted Message:", Final_message)
```

```
import random
import string
from VigenereCipher import vigenere_decrypt

def generate_random_word(length):
    alphabet = list(string.ascii_uppercase)
    return ''.join(random.choice(alphabet) for _ in range(length))

def generate_words_from_string(s):
    words = s.split()
```