```
# -*- coding: utf-8 -*-

"""

Created on Mon Sep 10 11:13:55 2018


@author: Kenny Nguyen

"""


trueAlphaList =
["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"]

                                                    # list of alphabets from index 0-25

scramAlphaList =
["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"]

                                                    # list that represents the key to a subsitution cipher.

                                                    # At default its all in alphabetical order.

                                                    # On runtime the key changes to help solve parts 1 and 2 of
the assignment


sampleKey =
["b","c","x","w","m","u","t","s","r","p","q","l","n","v","o","k","j","i","h","g","f","e","d","y","z","a"]

                                                      # a random key for a subsitution cipher








 # below are 3 strings that hold a ciphertexts that needed to be decrypted

 # these are needed to complete part 1 of the assignment                                                  # a
letter appears in general

cipherText1 = "fqjcb rwjwj vnjax bnkhj whxcq nawjv nfxdu mbvnu ujbbf nnc"

cipherText2 = "oczmz vmzor jocdi bnojv dhvod igdaz admno ojbzo rcvot jprvi oviyv aozmo cvooj ziejt
dojig toczr dnzno jahvi fdiyv xcdzq zoczn zxjiy"
```

```
cipherText3 = "ejitp spawa qleji taiul rtwll rflrl laoat wsqqj atgac kthls iraoa twlpl qjatw jufrh lhuts qataq itats aittk stqfj cae"

cipherText4 = "iyhqz ewqin azqej shayz niqbe aheum hnmnj jaqii yuexq ayqkn jbeuq iihed yzhni ifnun sayiz yudhe sqshu qesqa iluym qkque aqaqm oejjs hqzyu jdzqa diesh niznj jayzy uiqhq vayzq shsnj jejjz nshna hnmyt isnae sqfun dqzew qiead zevqi zhnjq shqze udqai jrmtq uishq ifnun siiqa suoij qqfni syyle iszhn bhmei squih nimnx hsead shqmr udquq uaqeu iisqe jshnj oihyy snaxs hqihe lsilu ymhni tyz"


 # below are 3 strings that hold a plaintexts that needed to be encrypted

 # these are needed to complete part 2 of the assignment

plainText1 = "he who fights with monsters should look to it that he himself does not become a monster and if you gaze long into an abyss the abyss also gazes into you"

plainText2 = "there is a theory which states that if ever anybody discovers exactly what the Universe is for and why it is here it will instantly disappear and be replaced by something even more bizarre and inexplicable there is another theory which states that this has already happened"

plainText3 = "whenever i find myself growing grim about the mouth whenever it is a damp drizzly November in my soul whenever i find myself involuntarily pausing before coffin warehouses and bringing up the rear of every funeral i meet and especially whenever my hypos get such an upper hand of me that it requires a strong moral principle to prevent me from deliberately stepping into the street and methodically knocking peoples hats off then i account it high time to get to sea as soon as i can"



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


def setKeyPrompt ():    # this function lets the user create their own key

   count = 0

   print("Enter the key for encryption in subsitution cipher: \n") # prompt



   while (count < 26): # loop gets user to input a key for each letter

      print("\n Enter letter for " , trueAlphaList[count], ": ")

      scramAlphaList[count] = input( ) # user input

      count = count + 1
```

```
    return scramAlphaList



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

def freqList(A):

    freqqL = [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]

    count = 0


    while (count < len(A)):

        if(A[count] != " "):

            indexx = trueAlphaList.index(A[count]) # finds the index of where the letter in the phrase appears
in the alphabet

            freqqL[indexx] = freqqL[indexx] + 1 # goes to the parallel list and updates the frequency value
located in the same index found above

        count = count + 1


    return freqqL


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

def swapLetters(A,B, C): # A is letter in cipher text, B is letter swapping to, C is the quote


    count = 0


    while (count < len(C)):

        if(C[count] == A):

            C[count] = B

            "ssaddda".replace

        count = count + 1
    return C


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

def shiftKey(offset):  # creates a shift cipher key using the offset passed

    count = 0

    scramAlphaList[:] = list(trueAlphaList) # resets the key list (scramAlphaList) to the original alphabet list


    while(count < offset): # this loop will shift the key one unit with each iteration

                        # the number of iterations are equal to the offset

        scramAlphaList.insert(0,scramAlphaList.pop()) # does the shifting

        count = count + 1


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


def setKey ():      # This function sets the key list (scramAlphaList) to a default key( sampleKey)


    scramAlphaList[:] = list(sampleKey)

    return scramAlphaList


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


def quote2List (A):     #this helper function puts each word in a sentence into a list

    listt = A.split(" ")    # splits the quote in the argument using space as a delimiter

    return listt
```

```
#####################################################################################
########################################


def decryptWord(A):     #a helper function that decrypts the word being passed


   count = 0 # variable to control the loop


   translatedWord = "" # the decrypted word


   while count < len(A):   # This loop checks each letter in the word and decrypts it

                     # then puts those letters together




      scrambledLettersIndex = scramAlphaList.index(A[count]) # finds the index of where the letter in the
scrambled word is in the key

                                             # alphabet list

      trans = trueAlphaList[(scrambledLettersIndex )]  # find the letter in the alphabet assoiciated with the
scrambled letter

                                          # using the index found


      translatedWord = translatedWord + trans # running total


      count = count + 1 #iterate the loop


   return translatedWord # return the decoded word



#####################################################################################
########################################
```

```
def decryptQuote(A): # a function used to decrypt a quote

    wordList = quote2List(A) # put each word into a list
    decodedQuote = ""       # contains the decrypted quote
    count  = 0

    while count < len(wordList): # this loop decrypts each word in the quote



        decodedQuote = decodedQuote +  decryptWord(wordList[count] ) # decrypts a word each iteration
and adds it
                                                    # to a running total
        count = count + 1 # iterate loop
    return decodedQuote # return the decrypted quote



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



def encryptWord(A):     #a helper function that Encrypts the word being passed


    count = 0 # variable to control the loop


    translatedWord = "" # the encrypted word


    while count < len(A):   # This loop checks each letter in the word and encrypts it
                    # then puts those letters together
        LetterIndex = trueAlphaList.index(A[count]) # finds the index of where the letter in the word is in
the alphabet list

        trans = scramAlphaList[LetterIndex] # finds the letter in the subsitution cipher key(scramAlphaList)
using the index found


        translatedWord = translatedWord + trans # running total / builds the encrypted word


        count = count + 1 #iterate the loop


    return translatedWord # return the encoded word



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



def encryptQuote(A): # a function used to encrypt a quote


    wordList = quote2List(A) # put each word into a list

    encodedQuote = ""        # contains the encrypted quote

    count  = 0


    while count < len(wordList): # this loop encrypts each word in the quote




        encodedQuote = encodedQuote +  encryptWord(wordList[count]) + " " # encrypts a word each
iteration and adds it

                                                                           # to a running total

        count = count + 1 # iterate loop
    return encodedQuote # return the encrypted quote



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


# START OF MAIN PROGRAM #


#thisProgramKey

#userKey




#################### PART 1 OF ASSIGNMENT ##########################

print("\n PART 1 ")

print("\n")


# decrypt the first two phrases // done via brute force using shift cipher


print(cipherText1.lower())  #decrypt first phrase

print("\n Decrypts to: \n")

shiftKey(17) # shift the alphabet 17 units

print(decryptQuote(cipherText1.lower()))


print("\n")





print(cipherText2.lower()) #decrypt second phrase

print("\n Decrypts to: \n")
```

```
shiftKey(5) # shift the alphabet 5 units
print(decryptQuote(cipherText2.lower()))

print("\n")




# Attempts at decrypting ciphertext 3 and 4

# http://crypto.interactive-maths.com/frequency-analysis-breaking-the-code.html

# the link above takes you to a site that tells you techniques and strategies
# used to decrypt subsitution ciphers
# the site also gives a table on how often a letter appears on average in the english language

print("\n")
print("Attempts on Cipher 3")
print(cipherText3)
print(freqList(cipherText3))
print(trueAlphaList)

cipherText3 = cipherText3.replace(" ", "") # remove spaces

# a and t appears the most, 15 times, so they are likely to be e and t
# according to the table provided in the link

cipherText3 = cipherText3.replace("a", "E")  # swap a with E
print(cipherText3)
print("\n")
cipherText3 = cipherText3.replace("t", "T") # swap t with T
                                    # capital letters denotes that the letter is replaced
                                    # and lower case denotes that the letter in the ipher text
                                    # has not been replaced yet
print(cipherText3)
print("\n")


cipherText3 = cipherText3.replace("l", "A") # swap l with A since l is the third
                                    # letter to appear the most, and A is
                                    #  3rd most frequent letter
                                    # on average
print(cipherText3)
print("\n")


cipherText3 = cipherText3.replace("q", "O") # swap q with O since q is the 4th
                                    # letter to appear the most, and O is
                                    #  4th most frequent letter
                                    # on average
print(cipherText3)
print("\n")


print("\n")
print("Attempts on Cipher 4")
print(cipherText4)
print(freqList(cipherText4))
print(trueAlphaList)

cipherText4 = cipherText4.replace(" ", "") # remove spaces

cipherText4 = cipherText4.replace("q", "E") # swap q with E since q
                         # appears the most, and E is
                         #  most frequent letter
                         # on average
print(cipherText4)
print("\n")

cipherText4 = cipherText4.replace("i", "T") # swap i with T since i
                         # appears the 2nd most, and T is
                         #  2nd most frequent letter
                         # on average
print(cipherText4)
print("\n")

cipherText4 = cipherText4.replace("h", "A") # swap h with A since h
                         # appears the 3rd most, and A is
                         #  third most frequent letter
                         # on average
print(cipherText4)
print("\n")

cipherText4 = cipherText4.replace("y", "O") # swap y with O
                         # if we assume TyAEz is part of a word
                         # and y is a second letter in that word,
                         # then then that word cannot exist because
```

```
                                    # y could be H, R, or a Vowel that can
                                    # pair with T, but all those combination
                                    # when inputed into Y forms a part of a
                                    # word that doesnt exist, or at least not
                                    # a common word someone would normally say.
                                    # So the other option is that Ty are a
                                    # word on its own, making O the only letter
                                    # that can be paired with t.

print(cipherText4)
print("\n")




################## PART 2 OF ASSIGNMENT ########################
print("\n PART 2 ")
print("\n")

# encrypt the phrases in part 2 using a default sample key
print("\n The key used for encryption is: ", setKey ()) # sets the key list(scramAlphaList) to the sample key
print("\n")

print(plainText1.lower())   #encrypt phrase 1
print("\n Encrypts to: \n")
print(encryptQuote(plainText1.lower()))

print("\n")
print(plainText2.lower()) #encrypt phrase 2
print("\n Encrypts to: \n")
print(encryptQuote(plainText2.lower()))

print("\n")

print(plainText3.lower()) #encrypt phrase 3
print("\n Encrypts to: \n")
print(encryptQuote(plainText3.lower()))


# Second part lets user decide a key and a phrase to encrypt
setKeyPrompt ()
print("\n")

a = input("Enter a phrase you would like to encrypt( alphabetic characters only): ")
print("\n")

print(a)   #encrypt phrase 1
print("\n Encrypts to: \n")
print(encryptQuote(a.lower()))
```