```
# File: Project2.py
# Student: Jennifer Truong
# UT EID: Jat5244
# Course Name: CS303E
#
# Date Created: 4/4/2021
# Date Last Modified: 4/12/2021
# Description of Program: Implement a substitution cipher.

import random

# A global constant defining the alphabet.
LCLETTERS = "abcdefghijklmnopqrstuvwxyz"

# You are welcome to use the following two auxiliary functions, or
# define your own.   You don't need to understand this code at this
# point in the semester.

def isLegalKey( key ):
    # A key is legal if it has length 26 and contains all letters.
    # from LCLETTERS and if they're lower case.
    userKey = str(key)
    keyList = list(userKey)
    alphaList = list(LCLETTERS)
    keyList.sort()
    alphaList.sort()

    if len(userKey) != 26:
        print("    Illegal key entered. Try again")
        return
    elif alphaList != keyList:
        print("    Illegal key entered. Try again?")
        return
    elif userKey != userKey.lower():
        print("    Illegal key entered. Try again!")
        return
    elif len(userKey) == 26 and  all([ ch in userKey for ch in LCLETTERS ]):
        return (len(userKey) == 26 and all( [ ch in userKey for ch in
LCLETTERS ] ))

def makeRandomKey():
    # A legal random key is a permutation of LCLETTERS.
    lst = list( LCLETTERS )  # Turn string into list of letters
    random.shuffle( lst )    # Shuffle the list randomly
    return ''.join( lst )    # Assemble them back into a string

# There may be some additional auxiliary functions defined here.
# I had several others, mainly used in encrypt and decrypt.

class SubstitutionCipher:
    def __init__ ( self, key = makeRandomKey() ):
        self.key = key

    # Note that these are the required methods, but you may define
    # additional methods if you need them.  (I didn't need any.)

    def getKey( self ):
        return self.key #get the key
```

```
    def setKey( self, newKey ):
        self.key = newKey #set the key

    # Encrypt the message with the key
    def encryptText( self, plaintext ):
        """Return the plaintext encrypted with respect to the stored key."""
        converter = str(self.key)
        encryptText = ""
        keyCap = converter.upper()
        alphabetCap = LCLETTERS.upper()

        for letter in str(plaintext):
            # if the string has uppercase letters
            if letter in keyCap:
                encryptText += keyCap[alphabetCap.find(letter)]
            elif letter not in converter:
                encryptText += letter
            else:
                encryptText += converter[LCLETTERS.find(letter)]
        return encryptText

    # Decrypt the message with the key
    def decryptText( self, ciphertext ):
        """Return the ciphertext decrypted with respect to the stored
        key."""
        theKey = str(self.key)
        decryptText = ""
        keyCap = theKey.upper()
        alphabetCap = LCLETTERS.upper()

        for letter in str(ciphertext):
            # if the string has uppercase letters
            if letter in alphabetCap:
                decryptText += alphabetCap[keyCap.find(letter)]
            elif letter not in theKey:
                decryptText += letter
            else:
                decryptText += LCLETTERS[theKey.find(letter)]
        return decryptText


def main():
    """ This implements the top level command loop.  It
    creates an instance of the SubstitutionCipher class and allows the user
    to invoke within a loop the following commands: getKey, changeKey,
    encrypt, decrypt, quit."""

    cipher = SubstitutionCipher()

    # The command of loops: getKey, changeKey, encrypt, decrypt, or quit
    command = str(input("Enter a command (getKey, changeKey, encrypt, decrypt,
quit): "))
    while command != (command.lower() == "quit"):

        if command.lower() == "getkey":
            print("  Current cipher key: " + str(cipher.getKey()))
        # Changing the Substitution Cipher key or not loop
        elif command.lower() == "changekey":
            changeKey = input("  Enter a valid cipher key, 'random' for a random
```

```
key, or 'quit' to quit: ")
            while changeKey != "quit":
                if str(changeKey).lower() == "random":
                    print("    New cipher key: " + makeRandomKey())
                    break
                elif str(changeKey).lower() == "quit":
                    return
                else:
                    if isLegalKey(str(changeKey)) is True:
                        cipher.setKey(str(changeKey))
                        print ("    New cipher key: " + str(changeKey))
                        break
                changeKey = input("  Enter a valid cipher key, 'random' for a
random key, or 'quit' to quit: ")
        # Encrypting the user's message respect to the stored key.
        elif command.lower() == "encrypt":
            plainText = input("  Enter a text to encrypt: ")
            print("    The encrypted text is: " + cipher.encryptText( plainText ))
        # decrypting the message message respect to the stored key.
        elif command.lower() == "decrypt":
            ciphertext = input("  Enter a text to decrypt: ")
            print("    The decrypted text is: " + cipher.decryptText( ciphertext ))

        elif command.lower() == "quit":
            print( "Thanks for visiting!")
            break

        else:
            print("  Command not recognized. Try again!")

        command = str(input("Enter a command (getKey, changeKey, encrypt, decrypt,
quit): "))

main()
```