```
# This Python Program was written on Windows 10 and Linux Mint using VScode, your milage may vary based on your OS and configuration.
# Video Explaination of this code: https://youtu.be/ZLbgEE1HJHE

'''
A Quote before you read:
"There is a way out of every box, a solution to every puzzle; it's just a matter of finding it."
— Captain Jean-Luc Picard

# *********************************************
# HOW TO RUN JACOB CLOUSE'S CIPHER FOR HM1 526
# *********************************************

**********
ENCRYPTION
**********

# 1) Move this program into its own folder


# 2) Open either a GIT BASH terminal or BASH TERMINAL and navigate into the folder that contains the program


# 3) Run this program with the command: python jacob-final-cipher.py
    NOTE: if you have python2 and python3 installed use this command: python3 jacob-final-cipher.py


# 4) The first thing it will ask you for is if you want to either 'Encrypt' or 'Decrypt', type in 'Encrypt'


# 5) The next thing it will ask you for is the plaintext, enter in the message you want to encode


# 6) It should let you know that a simple subsitution cipher has been activated, then you have to enter in the offset you want to set
    Any number between 1 and 26


# 7) It should move onto the transposition cipher, and will ask you for a unique sequence of the numbers: 1,2,3,4


# 8) After this, it will move onto the One time pad section, but this will be taken care of automatically
- This should finish and show you the One time pad key and your final Cipher Text!!
- It will make two pickle files: CIPHER.pickle and OTP_KEY.pickle - keep these safe, they are your ciphertext and key!
- A text file containing your output ciphertext will also generate


**********
DECRYPTION
**********

# 1) This is basically the reverse of encryption, make sure that both CIPHER.pickle and OTP_KEY.pickle are in the
    same directory as the jacob-final-cipher.py script and on the same level


# 2) Open up your terminal and run the script again with either:
    python jacob-final-cipher.py
    python3 jacob-final-cipher.py
    (Again, you will need to run the second one if you have both python2 and python3 installed)


# 3) This time when it asks you what to do enter in 'Decrypt'


# 4) It will start off with the One time pad function and automatically open both your pickle files
    You shouldn't have to do anything for this step


# 5) Next it will move on to the transposition function, it will ask you to enter in that combo of 1,2,3,4
    you had previously used in the encryption step


# 6) Then it will move on to the substitution function and ask you for the offset you had set


# 7) Finally, it will print out your original plaintext you had encrypted!
    - A text file containing this original plaintext will also be created.

'''


# GOALS for Development:
# 1st: get the substitution program working, then take the out put and pipe it into the transpostion - encryption works
# 2nd: reverse it for decryption: first transpose and then substitute to decrypt
# 3rd: verify encrypt to decrypt works fully with one pass each
# 4th: add an additional subsitution (either after 1st sub or the transposition) -- add more if you want
# 5th: see if you can break it with two cryptoanalyical methods - if not, then you are done!
# 6th: make a video on this explaining it from start to finish

# =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
# Importing Libraries / Modules
# =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

import datetime # used to get the datetime for "defang_datetime" function
import random # one time pad use
import pickle # saving array data to file

# =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
# Variables
# =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

# To convert intial plaintext to number values
lettersToNumbersDict = {
    'a':0,
    'b':1,
    'c':2,
    'd':3,
    'e':4,
    'f':5,
    'g':6,
    'h':7,
    'i':8,
    'j':9,
    'k':10,
    'l':11,
    'm':12,
    'n':13,
    'o':14,
    'p':15,
    'q':16,
    'r':17,
    's':18,
    't':19,
    'u':20,
    'v':21,
    'w':22,
    'x':23,
    'y':24,
    'z':25
    # ' ':'-',
    # '1':'!',
    # '2':'@',
    # '3':'#',
    # '4':'$',
    # '5':'%',
    # '6':'^',
    # '7':'&',
    # '8':'*',
    # '9':'(',
    # '0':')'

}

# To convert finished numbers back into finished ciphertext
numbersTolettersDict = {
    0:'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'
    # '-':' ',
    # '!':1,
    # '@':2,
    # '#':3,
    # '$':4,
    # '%':5,
    # '^':6,
    # '&':7,
```

```python
    # '*':8,
    # '(':9,
    # ')':0
}


# =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
# Functions
# =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

# --- Function to print out my Logo ---
def myLogo():
    print("Created and Tested by: ")
    print("   __                  _         ___ _                       ")
    print("   \ \  __ _  ___ ___ | |__     / __\ | ___  _   _ ___  ___  ")
    print("    \ \/ _` |/ __/ _ \| '_ \   / /  | |/ _ \| | | / __|/ _ \ ")
    print(" /\_/ / (_| | (_| (_) | |_) | / /___| | (_) | |_| \__ \  __/ ")
    print(" \___/ \__,_|\___\___/|_.__/  \____/|_|\___/ \__,_|___/\___| ")
    print("Dedicated to Peter Zlomek and Harley Alderson III")


# --- Function to Defang date time ---
def defang_datetime():
    current_datetime = f"_{datetime.datetime.now()}"

    current_datetime = current_datetime.replace(":","_")
    current_datetime = current_datetime.replace(".","-")
    current_datetime = current_datetime.replace(" ","_")

    return current_datetime


# --- Function to Write Data to a file ---
def write_to_file(filename,plaintextOrCipherText,dataToWrite):
    outboundFile = open(f"{filename}.txt", "w")
    lesGoBoi = outboundFile.write(f'{plaintextOrCipherText} : "{dataToWrite}"')
    outboundFile.close()


# --- Function to return a combination of 1,2,3 and 4 with each number being used only once --- Trans Specific
def get_combo_of_1234():

    # setting up numbers left and current key
    current_numbers_left = [1,2,3,4]
    desired_combo_key = []

    # get current length of the numbers left
    current_length_of_what_is_left = len(current_numbers_left)

    # loop through until all the numbers are removed
    while current_length_of_what_is_left > 0:
        next_to_remove = input(f"The current numbers left for the key are {current_numbers_left},\nSelect a number from these: ")

        # if below passes, this is a number at least
        if next_to_remove.isnumeric() == True:
            # if below passes, than i
            if int(next_to_remove) in current_numbers_left:
                desired_combo_key.append(int(next_to_remove)) # appending to the key
                current_numbers_left.remove(int(next_to_remove)) # removing number from number left
                current_length_of_what_is_left = len(current_numbers_left) # recalculating length that is left

            else:
                print("Nice try, that was an invalid option. Try again.\n")

        else:
            print("This has to be a number wise guy.\n")

    print(f"Your key is going to be: {desired_combo_key}")
    return desired_combo_key


# --- Function to get remainder and let us know how many spaces to add --- Trans Specific
def get_remainder(input_string):
    # print(input_string)
    # getting length of input
    length_of_input = len(input_string)
    # print(f"Length: {length_of_input}")
    number_of_spaces_to_add = 4 - (int(length_of_input) % 4)
    # print(number_of_spaces_to_add)
    return int(number_of_spaces_to_add)


# --- Function to split up a string into 4 equal parts ---
def split_my_string_in_4(string):
    length = len(string) # getting length
    part_length = length // 4
    return [string[i:i + part_length] for i in range(0, length, part_length)] # returning each piece in an array


# --- Function to ENCRYPT a simple substitution cipher ---
def encrypt_substitution(input_plaintext):
    print("\nSimple Sub has been activated!\n")
    input_offset_key = input("What is the offset key you want? ") # getting the offest value from the user
    # input_plaintext = input("\nWhat is the plaintext message you want to encode? ") # getting the plaintext to encrypt


    # Converts the plaintext into corresponding numbers
    # myPlaintextLettersArray = []
    ciphertextArray = []
    for letters in input_plaintext:

        # Checking to see if it is a letter, if not we don't lowercase it
        # print(letters, letters.isalpha())
        if (letters.isalpha()) == True:
            # print("This is a letter")

            # Lowercasing - prevents issues with capital letters
            lowerCaseLetter = letters.lower()

            # Why not change it into the cipher text right now if we have the offset?
            # The algorithm for substitution ciphers basically is: (Plaintext_Letter_Val + Offset_Val) mod 26 = Cipher_Letter_Val
            plaintextValue = lettersToNumbersDict[lowerCaseLetter]
            covertedToCipherValue = (int(plaintextValue) + int(input_offset_key)) % 26

            # We have the cipher value, now we just need to convert it to the ciphertext letter
            convertedToCipherLetter = numbersTolettersDict[covertedToCipherValue]
            ##print(f"Current Character: {lowerCaseLetter}, Character Value: {lettersToNumbersDict[lowerCaseLetter]}, Cipher Value: {covertedToCipherValue}, Cipher Letter: {convertedToCipherLetter}")

            # Append to the array
            ciphertextArray.append(convertedToCipherLetter)


        else:
            # print('NOT A LETTER')
            ##print(f"Current Character: {letters}, Appending to Array as is...")
            # myPlaintextLettersArray.append(lettersToNumbersDict[letters])
            ciphertextArray.append(letters)

    ##print(f"Output Array: {ciphertextArray}\n")

    # Turn array into a string
    cipherText = ''
    for characters in ciphertextArray:
        cipherText += characters

    # Returning ciphertext to calling function
    return cipherText


# --- Function to DECRYPT a simple substitution cipher ---
def decrypt_substitution(input_ciphertext):
    print("\nDecrypt Sub has been activated!\n")

    input_offset_decrypt_key = input("What is the offset key set to? ") # getting the offest decrypt key from the user
    # input_ciphertext = input("\nWhat is the Ciphertext message you want to decode? ") # getting the ciphertext to decrypt

    # Converts the ciphertext into corresponding numbers
    PlaintextArray = []
    for letters in input_ciphertext:

        # Checking to see if it is a letter, if not we don't lowercase it
        # print(letters, letters.isalpha())
        if (letters.isalpha()) == True:
            # Lowercasing - prevents issues with capital letters
            lowerCaseLetter = letters.lower()

            # Why not change it into the plain text right now if we have the offset?
            # If we reverse the algorithm for substitution, the forumla is: (Cipher_Letter_Val - Offset_Val) mod 26 = Orig_Plaintext_Letter_Val
            cipherValue = lettersToNumbersDict[lowerCaseLetter]
            covertedToPlaintextValue = (int(cipherValue) - int(input_offset_decrypt_key)) % 26

            # We have the plaintext value, now we just need to convert it to the original letter
            convertedToPlaintextLetter = numbersTolettersDict[covertedToPlaintextValue]
            ##print(f"Current Cipher Character: {lowerCaseLetter}, Character Value: {lettersToNumbersDict[lowerCaseLetter]}, Plaintext Value: {covertedToPlaintextValue}, Original Letter: {convertedToPlaintextLetter}")

            # Append to the array
            PlaintextArray.append(convertedToPlaintextLetter)


        else:
            ##print(f"Current Character: {letters}, Appending to Array as is...")
            PlaintextArray.append(letters)

    ##print(f"Output Array: {PlaintextArray}\n")

    # Turn array into a string
    plainText = ''
    for characters in PlaintextArray:
        plainText += characters

    # Returing plaintext to calling function
    return plainText


# --- Function to ENCRYPT a simple transposition cipher --- 4 rails in Matrix
def encrypt_transposition(input_plaintext):
    print("\nSimple Transposition has been activated!\n")

    # The four Arrays used to encrypt a transposition cipher, in here because I don't want the data to remain outside this function
    ListOfLists = [[],[],[],[]]

    # getting plaintext from the user
    # input_plaintext = input("\nWhat is the plaintext message you want to encode? ") # getting the plaintext to encrypt

    # Getting remainder, seeing if we have to add any values to get it to be a factor of 4
    current_spaces_to_add = get_remainder(input_plaintext)
    ##print(current_spaces_to_add)

    if current_spaces_to_add != 4:
        for space in range(current_spaces_to_add):
            input_plaintext += ' ' # if you use spaces to add, then they don't know if they are 'real' spaces or just the padding at the end
            print(input_plaintext)
    ##print(f"\nNow with the displacement, the new plaintext is: {input_plaintext}")


    # pushing that plaintext into arrays
    for index,character in enumerate(input_plaintext):
        ##print(f"Character: {character}")
        ##print(f"Index: {index}")

        # Get mod 4 of the current char
        current_mod_of_char = index % 4
        ##print(f"Current Modulus: {current_mod_of_char}")

        # if mod is equal to 0, we move to List 0
        if (current_mod_of_char == 0):
            ListOfLists[0].append(character)

        # if mod is equal to 1, we move to List 1
        elif (current_mod_of_char == 1):
            ListOfLists[1].append(character)

        # if mod is equal to 2, we move to List 2
        elif (current_mod_of_char == 2):
            ListOfLists[2].append(character)

        # mod has to be equal to 3, we move to List 3
        else:
            ListOfLists[3].append(character)

    ##print(ListOfLists)

    # getting key from user
    input_column_order_key = get_combo_of_1234()

    # Iterating through the array and creating the ciphertext
    # getting ciphertext ready
    outbound_ciphertext = ''

    # appending arrays to ciphertext
    for numbers in input_column_order_key:
        ##print(f"Appending List {(numbers - 1)} as value was: {numbers}")
        ##print(f"This is ListOfLists{(numbers - 1)}, or: {ListOfLists[(numbers - 1)]}")

        # Changing list to string
        stringify_this = ''.join(ListOfLists[int((numbers - 1))])
        ##print(f"Stringifying: {stringify_this}")
        outbound_ciphertext += stringify_this

        # showing what the current ciphertext is
        ##print(f"\nCurrent Ciphertext: {outbound_ciphertext}\n")


    # Returing ciphertext to calling function
    return outbound_ciphertext


# --- Function to DECRYPT a simple transposition cipher --- 4 rails in Matrix
def decrypt_transposition(input_ciphertext):
    print("\nDecrypt Transposition has been activated!\n")

    # The four Arrays used to decrypt a transposition cipher, in here because I don't want the data to remain outside this function
    ciphertextList = []
    plaintextList = []

    # getting plaintext from the user
    # input_ciphertext = input("\nWhat is the ciphertext message you want to decode? ") # getting the ciphertext to decrypt
    split_up_ciphertext = split_my_string_in_4(input_ciphertext)

    # number of columns in each column
    numOfCharsInEachColumn = int(len(input_ciphertext) / 4)
    ##print(numOfCharsInEachColumn)
    for jakes in range(numOfCharsInEachColumn):
        plaintextList.append([])

    ##print(f"We should have {numOfCharsInEachColumn} in plaintextList = {plaintextList}")

    ##print(f"Splitting up Plaintext: {split_up_ciphertext}")

    # getting key from user
    input_column_order_key = get_combo_of_1234()

    # append each array in order
    for nums in input_column_order_key:
        ciphertextList.append(split_up_ciphertext[(nums - 1)])

    ##print(f"After Re order: {ciphertextList}")

    # adding array
    for index,arrays in enumerate(ciphertextList):
        ##print(f"Index: {index}, Array: {arrays}")
        for index2,character in enumerate(arrays):
            ##print(f"Index2: {index2}, Array: {character}")

            plaintextList[index2].append(character)

    # Printing out final array
    # print(f"Final Array: {plaintextList}")

    # converting to string
    outbound_plaintext = ''
    for arraysMyBoi in plaintextList:
```

```python
        for letters in arraysMyBoi:
            outbound_plaintext += letters

    # print output
    ##print(f"Final Output String: {outbound_plaintext}")

    # returning to calling function
    return outbound_plaintext


# --- Function to Encrypt a One Time Pad ---
def encrypt_one_time_pad(plaintext):
    # will take in input from user outside of function and then pass it in
    print("Simple One Time Pad Encrypt\n")

    # using the length of pad to generate random one digit numbers from 0 to 9, need to store and output
    one_time_pad_key= ''
    for letters in plaintext:
        current_key_value = str(random.randint(0,9))
        one_time_pad_key += current_key_value
        # print(f"Letter: {letters}, Key value: {current_key_value}")

    # take your key and combine with your plaintext to get your ciphertext
    array_ciphertext = [chr(ord(p) ^ ord(k)) for (p,k) in zip(plaintext, one_time_pad_key)]

    # output ciphertext and key
    ##print(f"\nOne Time Pad Key: {one_time_pad_key}")
    ##print(f"Output Ciphertext: {array_ciphertext}")
    ##print(f"Length of Ciphertext Array: {len(array_ciphertext)}")

    # Write ciphertext and key to separate files
    # data_to_file("CIPHER",array_ciphertext)
    # data_to_file("OTP_KEY",one_time_pad_key)

    with open("CIPHER.pickle", "wb") as out_file:
        pickle.dump(array_ciphertext, out_file)

    with open("OTP_KEY.pickle", "wb") as out_file:
        pickle.dump(one_time_pad_key, out_file)


    return one_time_pad_key, array_ciphertext


# --- Function to Decrypt a One Time Pad ---
def decrypt_one_time_pad():
    # will take in input from user outside of function and then pass it in
    print("Simple One Time Pad Decrypt\n")

    # loading key from pickle
    with open("OTP_KEY.pickle", "rb") as loaded_key_file:
        one_time_pad_decrypt_key = pickle.load(loaded_key_file)

    # Loading ciphertext from pickle
    with open("CIPHER.pickle", "rb") as loaded_cipher_file:
        ciphertext = pickle.load(loaded_cipher_file)


    ##print(f"Ciphertext: {ciphertext}, Key: {one_time_pad_decrypt_key}")

    # take your key and combine with your ciphertext to get your plaintext back
    array_plaintext = [chr(ord(p) ^ ord(k)) for (p,k) in zip(ciphertext, one_time_pad_decrypt_key)]

    # change output from array to a string
    output_plaintext = ''
    for characters in array_plaintext:
        output_plaintext += characters

    ##print(f"Your plaintext: {output_plaintext}")

    return output_plaintext


# --- Function to ENCRYPT the Full Product cipher ---
def encrypt_product_cipher():
    currentTime = defang_datetime()
    print(f"Current Date/Time: {currentTime}")
    myLogo()
    print("Product Cipher Encrypt Started.... \n\n")

    en_input_from_user = input("\nWhat is the plaintext message you want to encode? ") # getting the plaintext to encrypt

    # Subsitution Portion
    sub_encrypt_part = encrypt_substitution(en_input_from_user)

    # Transposition Portion
    trans_encrypt_part = encrypt_transposition(sub_encrypt_part)

    # One Time Pad Portion
    OTP_encrypt_Key, OTP_encrypt_ciphertext = encrypt_one_time_pad(trans_encrypt_part)

    # Writing Encrypted Data to a file
    write_to_file(f"Product_Cipher_Encryption_{currentTime}","Product Ciphertext: ", OTP_encrypt_ciphertext)

    # Printing Ciphertext
    print(f"The One Time Pad key was: {OTP_encrypt_Key}")
    print(f"Your encrypted ciphertext is: {OTP_encrypt_ciphertext}")


# --- Function to DECRYPT the Full Product cipher ---
def decrypt_product_cipher():
    currentTime = defang_datetime()
    print(f"Current Date/Time: {currentTime}")
    myLogo()
    print("Product Cipher Decrypt Started.... \n\n")

    # de_input_from_user = input("\nWhat is the Ciphertext message you want to decode? ") # getting the ciphertext to decrypt

    # Pulling data from file
    print("Please make sure that the CIPHER.pickle and OTP_KEY.pickle files are in this directory!")

    # One Time Pad Portion - auto opens pickle files
    de_input_from_user = decrypt_one_time_pad()


    # Transposition Portion
    trans_decrypt_part = decrypt_transposition(de_input_from_user)

    # Subsitution Portion
    sub_decrypt_part = decrypt_substitution(trans_decrypt_part)

    # Writing decrypted Data to a file
    write_to_file(f"Product_Cipher_Decryption_{currentTime}","Product Plaintext:",
    sub_decrypt_part)

    # Print Plaintext:
    print(f"Your decrypted plaintext is: {sub_decrypt_part}")


# =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
# MAIN PROGRAM
# =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-

chooseOperation = input("What kind of operation do you want: ENCRYPT or DECRYPT? ")
print(chooseOperation.upper())
print('\n')

# Catch statement to prevent invalid selections
while chooseOperation == '':
    chooseOperation = input("Can't be left blank, please input either ENCRYPT or DECRYPT:
    ")

# execute encrypt operation
if chooseOperation.upper() == 'ENCRYPT':
    encrypt_product_cipher()

# execute decrypt operation
elif chooseOperation.upper() == 'DECRYPT':
    decrypt_product_cipher()

# if nonsense, end the script
else:
    print("Response Not Recognized, Ending Program...")


# Thank you for viewing my program, I hope you liked it!
```