**Information Security**

**Practicals**

**Name – Prerna**

**Roll – 20201420**

**Exam Roll – 20020570025**

1.Implement the error correcting code.

G = [[1, 0, 0, 0, 1, 1, 0],

     [0, 1, 0, 0, 1, 0, 1],

     [0, 0, 1, 0, 0, 1, 1],

     [0, 0, 0, 1, 1, 1, 1]]

# Define the Hamming code parity-check matrix

H = [[1, 1, 0, 1, 1, 0, 0],

     [1, 0, 1, 1, 0, 1, 0],

     [0, 1, 1, 1, 0, 0, 1]]

def hamming\_encode(data):

    k = len(data)

    r = 0

    while 2\*\*r < k + r + 1:

        r += 1

    encoded\_data = list(data) + [0]\*r

    for i in range(r):

        bit = 0

        for j in range(k+r):

            if encoded\_data[j] and (j+1) & (1<<i):

                bit ^= 1

        encoded\_data[k+i] = bit

    return encoded\_data

def hamming\_decode(encoded\_data):

    n = len(encoded\_data)

    r = 0

    while 2\*\*r < n:

        r += 1

    syndrome = []

    for i in range(r):

        bit = 0

        for j in range(n):

            if encoded\_data[j] and (j+1) & (1<<i):

                bit ^= 1

        syndrome.append(bit)

    # Correct any errors

    error = 0

    for i in range(r):

        error += syndrome[i] \* (1<<i)

    if error:

        encoded\_data[error-1] ^= 1

    # Return the decoded data

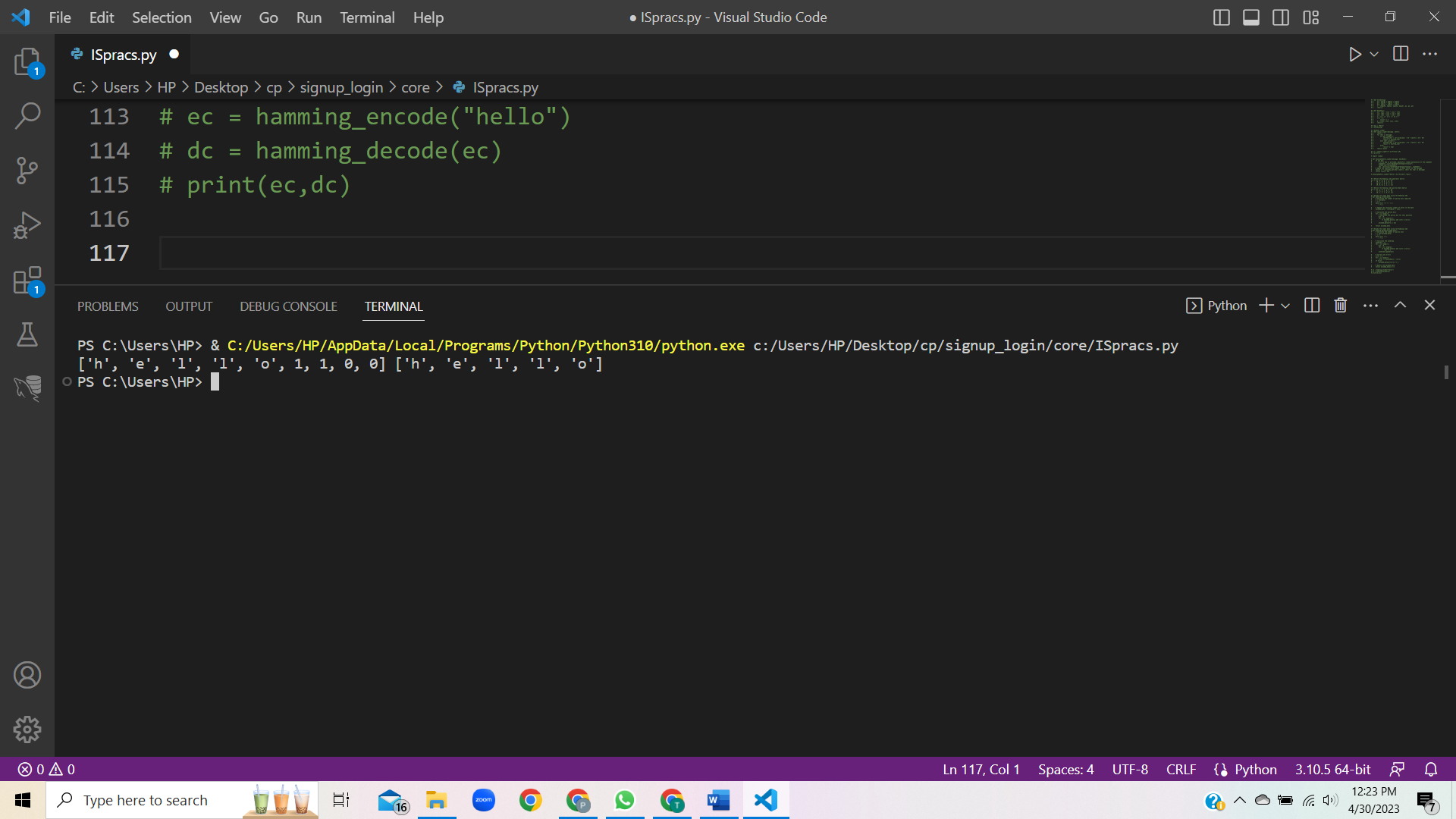
    return encoded\_data[:n-r]

ec = hamming\_encode("hello")

dc = hamming\_decode(ec)

print(ec,dc)

Output :-



2. Implement the error detecting code.

There are various error detection algorithms here Checksum method is used.

def checksum(data):

    sum = 0

    for byte in data:

        sum += byte

    return bytes([sum & 0xFF])

data = b'\x01\x02\x03\x04\x05\x06\x07\x08'

expected\_checksum = checksum(data)

transmitted\_data = data + expected\_checksum

received\_data = transmitted\_data[:-1]

received\_checksum = transmitted\_data[-1:]

actual\_checksum = checksum(received\_data)

if expected\_checksum == actual\_checksum:

    print("No errors detected!")

else:

    print("Error detected in transmission.")

3. Implement caeser cipher substitution operation.

def caesar\_cipher(plaintext, shift):

    ciphertext = ""

    for letter in plaintext:

        # Shift the letter by the specified amount

        if letter.isalpha():

            if letter.isupper():

                ciphertext += chr((ord(letter) - 65 + shift) % 26 + 65)

            else:

                ciphertext += chr((ord(letter) - 97 + shift) % 26 + 97)

        else:

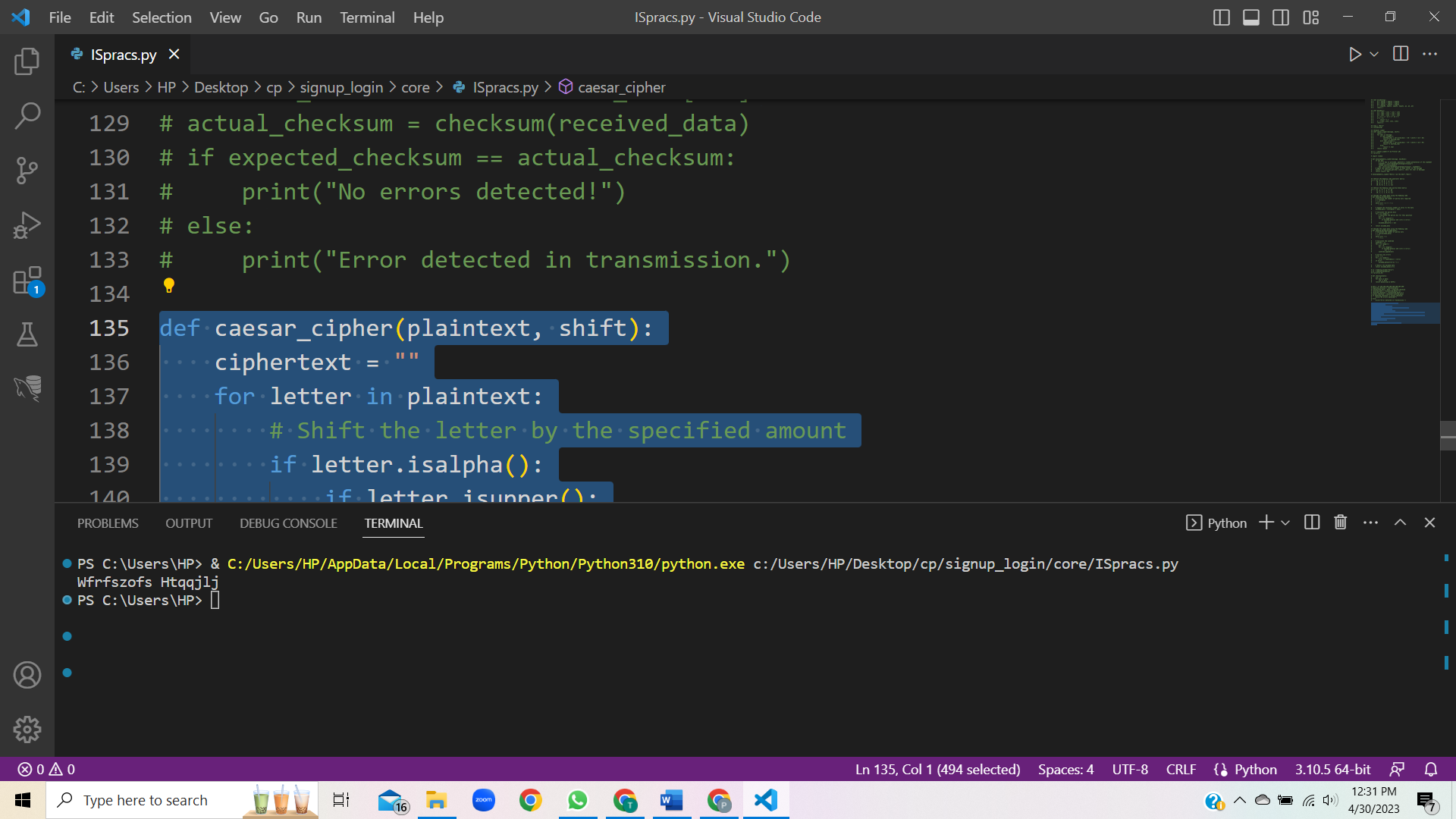
            ciphertext += letter

    return ciphertext

x = caesar\_cipher("Ramanujan College",5)

print(x)

Output :-



4. Implement monoalphabetic and polyalphabetic cipher substitution operation.

def monoalphabetic\_cipher(plaintext, key):

    ciphertext = ""

    for letter in plaintext:

        if letter.isalpha():

            if letter.isupper():

                ciphertext += key[ord(letter) - 65]

            else:

                ciphertext += key[ord(letter) - 97].lower()

        else:

            ciphertext += letter

    return ciphertext

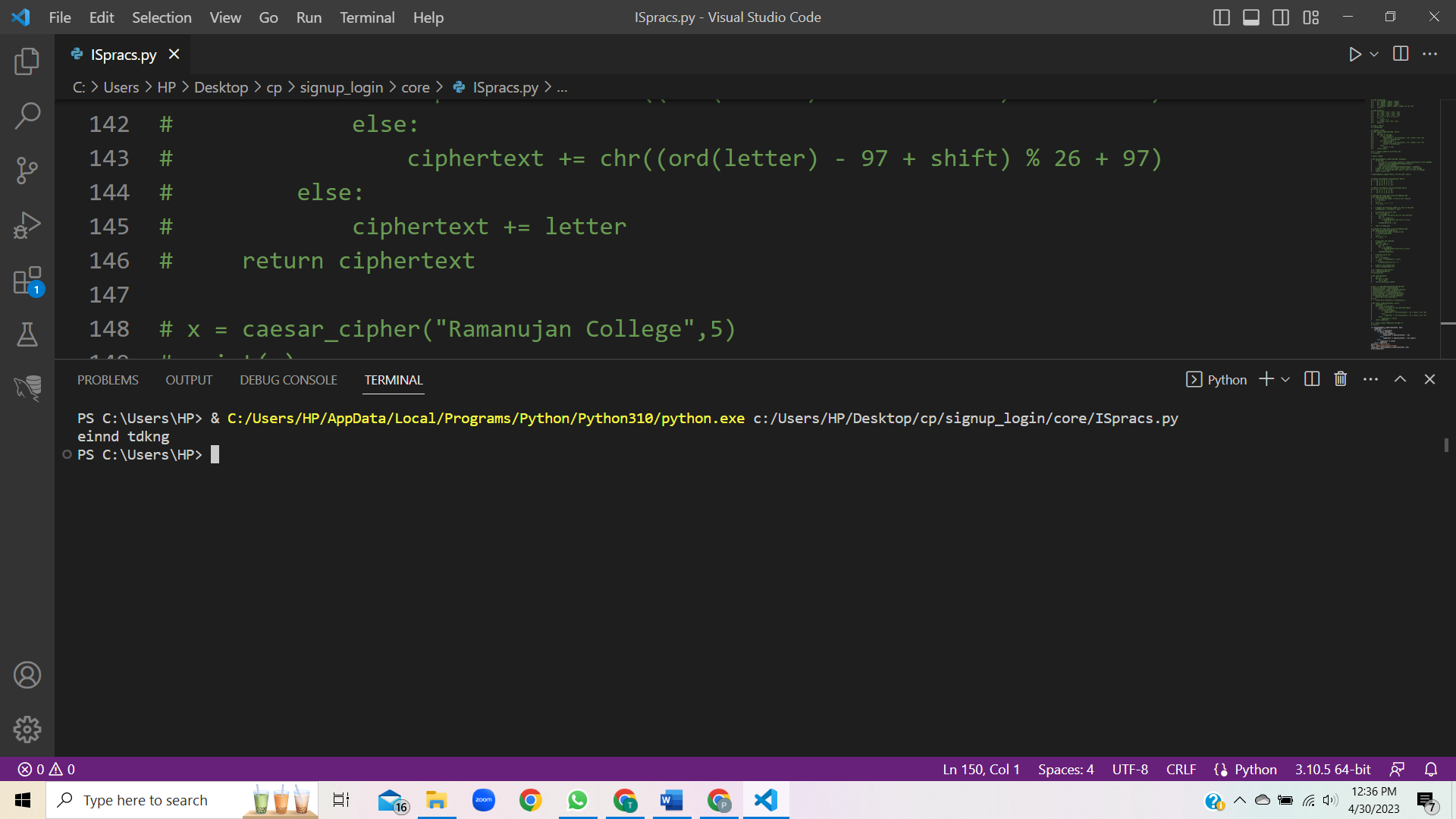
plaintext = "hello world"

key = "phqgiumeaylnofdxjkrcvstzwb"

ciphertext = monoalphabetic\_cipher(plaintext, key)

print(ciphertext)

output :-



Polyalphabetic cipher :

def vigenere\_cipher(plaintext, key):

    ciphertext = ""

    key\_index = 0

    for letter in plaintext:

        if letter.isalpha():

            if letter.isupper():

                shift = ord(key[key\_index % len(key)].upper()) - 65

                ciphertext += chr((ord(letter) - 65 + shift) % 26 + 65)

            else:

                shift = ord(key[key\_index % len(key)].lower()) - 97

                ciphertext += chr((ord(letter) - 97 + shift) % 26 + 97)

            key\_index += 1

        else:

            ciphertext += letter

    return ciphertext

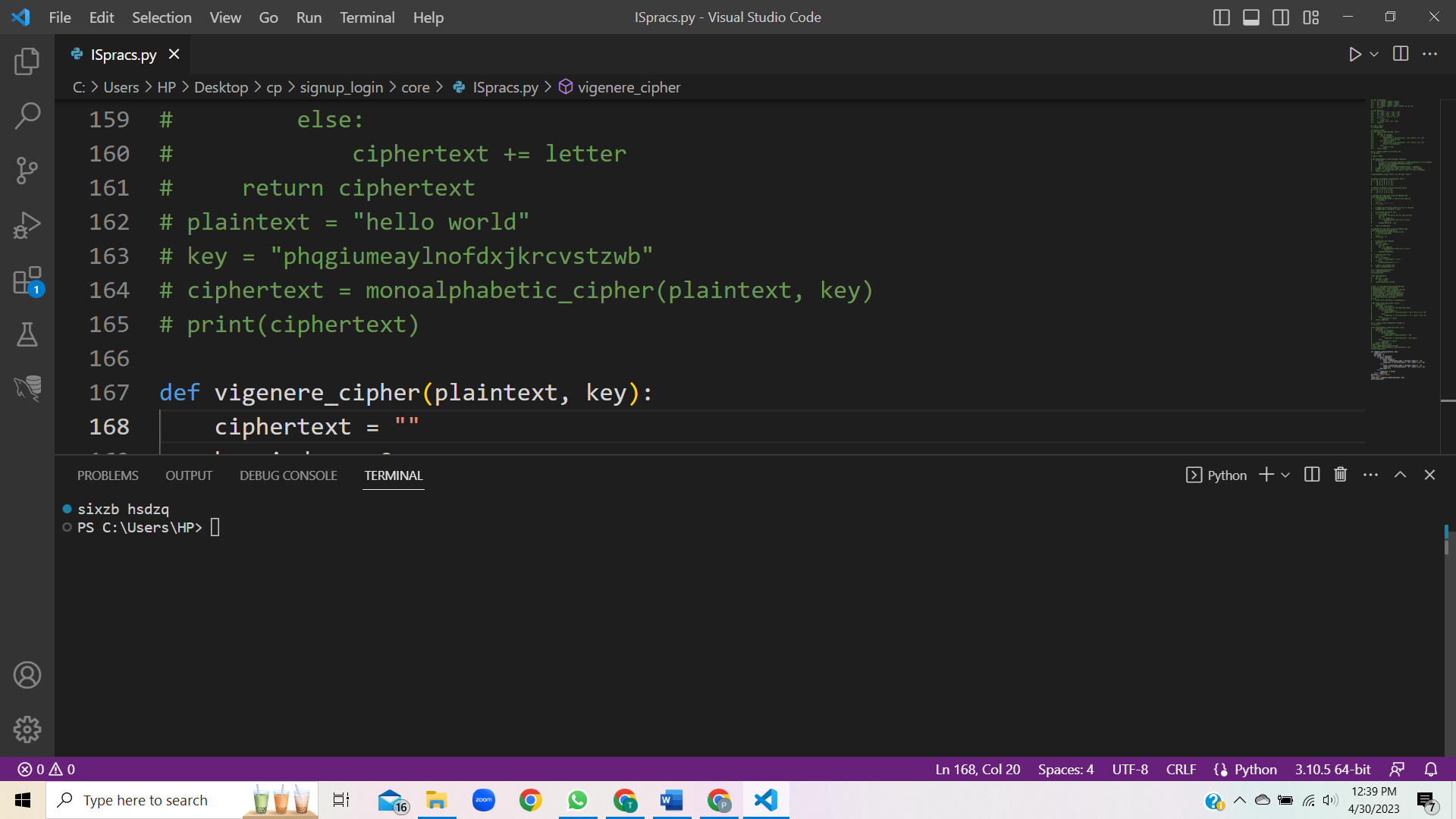
plaintext = "hello world"

key = "lemon"

ciphertext = vigenere\_cipher(plaintext, key)

print(ciphertext)

output :-



5. Implement playfair cipher substitution operation.

def generate\_playfair\_table(key):

    table = []

    key = key.replace("J", "I").upper()

    for letter in key:

        if letter not in table and letter.isalpha():

            table.append(letter)

    for letter in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":

        if letter not in table:

            table.append(letter)

    return table

def playfair\_cipher(plaintext, key):

    table = generate\_playfair\_table(key)

    plaintext\_pairs = []

    plaintext = plaintext.replace("J", "I").upper()

    plaintext = "".join([c for c in plaintext if c.isalpha()])

    i = 0

    while i < len(plaintext):

        if i == len(plaintext) - 1 or plaintext[i] == plaintext[i+1]:

            plaintext\_pairs.append(plaintext[i] + "X")

            i += 1

        else:

            plaintext\_pairs.append(plaintext[i:i+2])

            i += 2

    ciphertext\_pairs = []

    for pair in plaintext\_pairs:

        row1, col1 = divmod(table.index(pair[0]), 5)

        row2, col2 = divmod(table.index(pair[1]), 5)

        if row1 == row2:

            ciphertext\_pairs.append(table[row1\*5+(col1+1)%5] + table[row2\*5+(col2+1)%5])

        elif col1 == col2:

            ciphertext\_pairs.append(table[((row1+1)%5)\*5+col1] + table[((row2+1)%5)\*5+col2])

        else:

            ciphertext\_pairs.append(table[row1\*5+col2] + table[row2\*5+col1])

    ciphertext = "".join(ciphertext\_pairs)

    return ciphertext

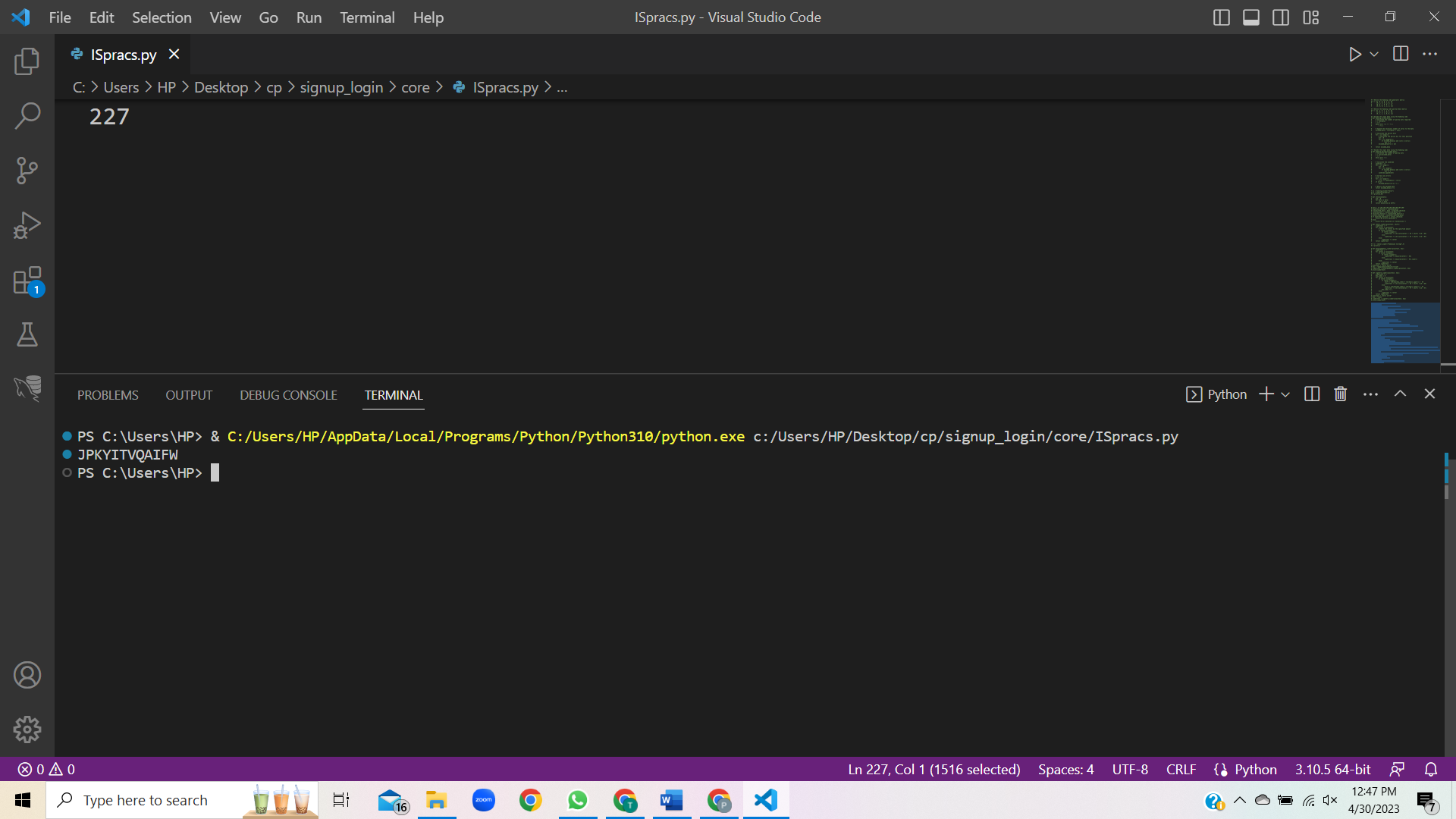
plaintext = "hello world"

key = "prerna"

ciphertext = playfair\_cipher(plaintext, key)

print(ciphertext)

output:-



6. Implement hill cipher substitution operation.

import numpy as np

def generate\_hill\_key(n):

    while True:

        key = np.random.randint(26, size=(n, n))

        if np.linalg.det(key) % 26 != 0:

            return key

def hill\_cipher(plaintext, key):

    plaintext = plaintext.upper().replace(" ", "")

    if len(plaintext) % key.shape[0] != 0:

        plaintext += "X" \* (key.shape[0] - len(plaintext) % key.shape[0])

    plaintext = np.array(list(map(lambda c: ord(c) - ord('A'), plaintext)))

    plaintext = plaintext.reshape(-1, key.shape[0])

    ciphertext = ""

    for col in plaintext.T:

        col = col.reshape(-1, 1)

        encrypted\_col = (key @ col) % 26

        ciphertext += "".join(list(map(lambda c: chr(c + ord('A')), encrypted\_col.flatten().tolist())))

    return ciphertext

plaintext = "HELLO WORLD"

key = generate\_hill\_key(2)

ciphertext = hill\_cipher(plaintext, key)

print(ciphertext)

7. Implement rail fence cipher transposition operation.

def rail\_fence\_cipher(plaintext, rails):

    cipher\_rails = [""] \* rails

    direction = 1    rail = 0

    for c in plaintext:

        cipher\_rails[rail] += c

        rail += direction

        if rail == rails - 1 or rail == 0:

            direction = -direction

        ciphertext = "".join(cipher\_rails)

    return ciphertext

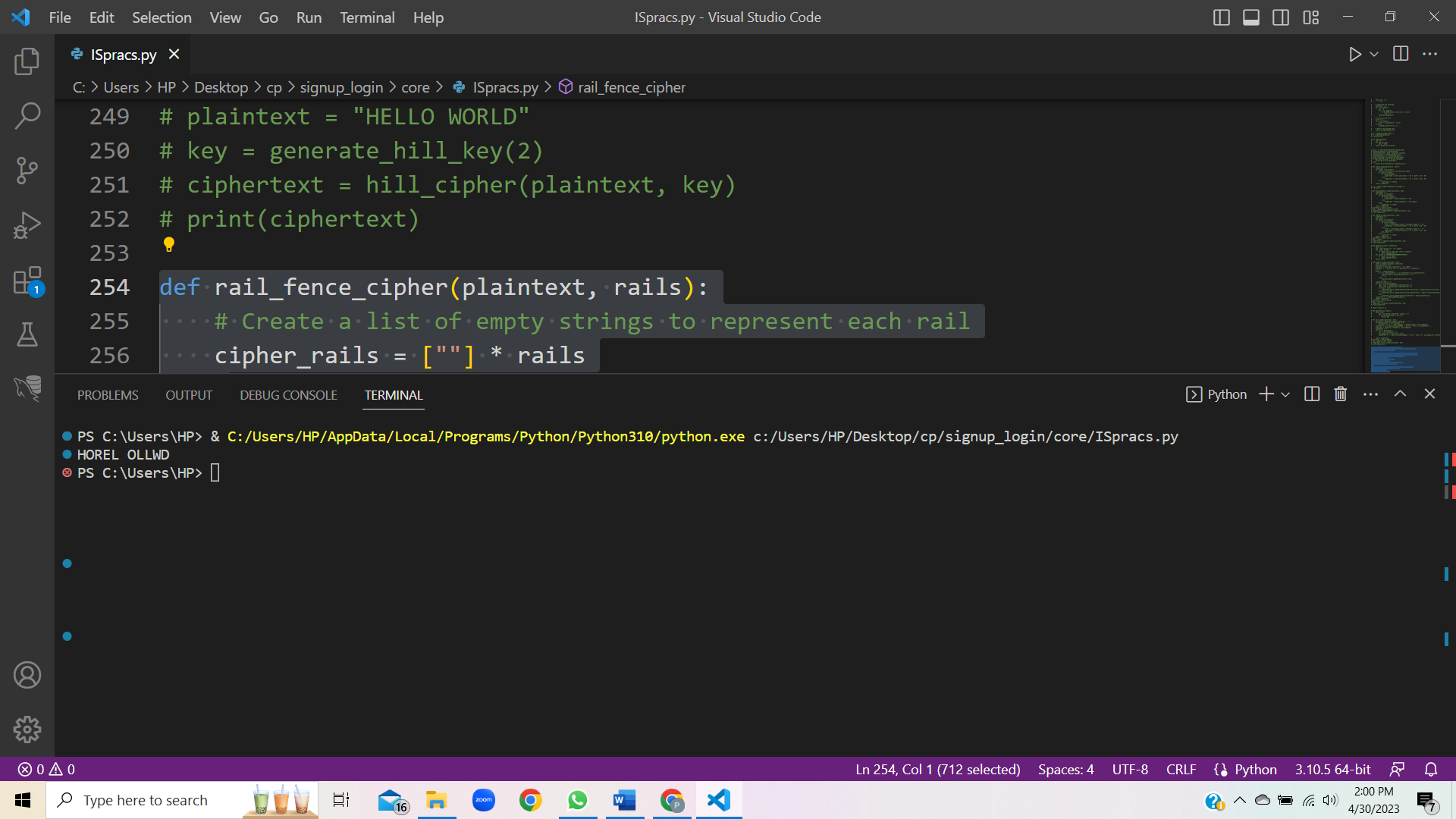
plaintext = "HELLO WORLD"

rails = 3

ciphertext = rail\_fence\_cipher(plaintext, rails)

print(ciphertext)

output :-



8. Implement row transposition cipher transposition operation.

def row\_transposition\_cipher(plaintext, key):

    num\_cols = len(key)

    padding = (num\_cols - len(plaintext) % num\_cols) % num\_cols

    plaintext += " " \* padding

    grid = [list(plaintext[i:i+num\_cols]) for i in range(0, len(plaintext), num\_cols)]

    sorted\_cols = sorted(range(num\_cols), key=lambda i: key[i])

    transposed\_grid = [[row[i] for i in sorted\_cols] for row in grid]

    ciphertext = "".join("".join(row) for row in transposed\_grid)

    return ciphertext

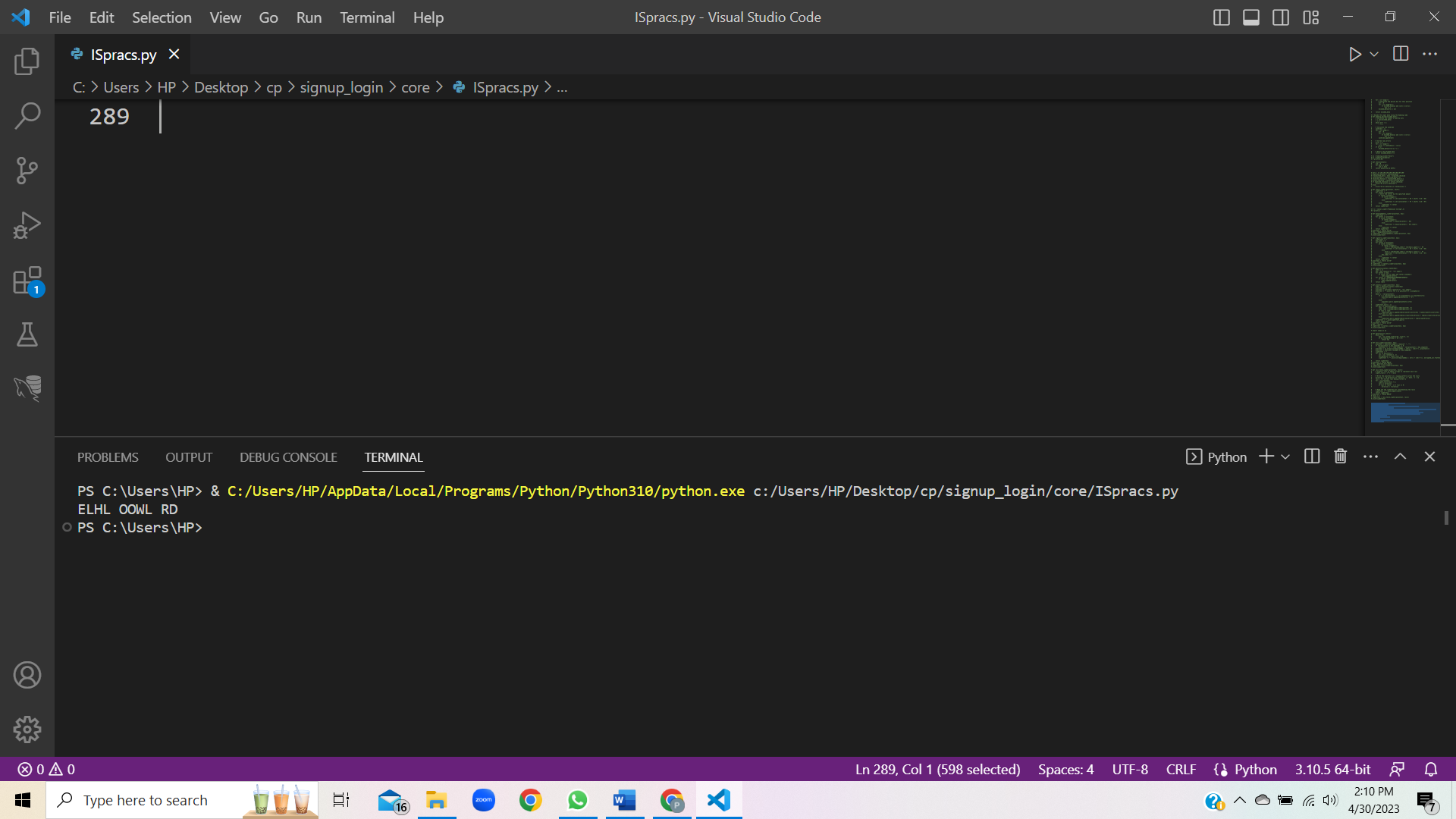
plaintext = "HELLO WORLD"

key = "3142"

ciphertext = row\_transposition\_cipher(plaintext, key)

print(ciphertext)

output:-



9. Implement product cipher transposition operation.

def caesar\_cipher(text, key):

    text = text.upper()

    ciphertext = ""

    for char in text:

        if char.isalpha():

            shifted = (ord(char) - 65 + key) % 26

            ciphertext += chr(shifted + 65)

        else:

            ciphertext += char

    return ciphertext

def rail\_fence\_cipher(text, key):

    fence = [['\n' for i in range(len(text))] for j in range(key)]

    dir\_down = False

    row, col = 0, 0

    for char in text:

        if row == 0 or row == key - 1:

            dir\_down = not dir\_down

        fence[row][col] = char

        col += 1

        if dir\_down:

            row += 1

        else:

            row -= 1

    ciphertext = ""

    for i in range(key):

        for j in range(len(text)):

            if fence[i][j] != '\n':

                ciphertext += fence[i][j]

    return ciphertext

def product\_cipher(plaintext, caesar\_key, rail\_fence\_key):

    caesar\_ciphertext = caesar\_cipher(plaintext, caesar\_key)

    rail\_fence\_ciphertext = rail\_fence\_cipher(caesar\_ciphertext, rail\_fence\_key)

    return rail\_fence\_ciphertext

plaintext = "HELLO WORLD"

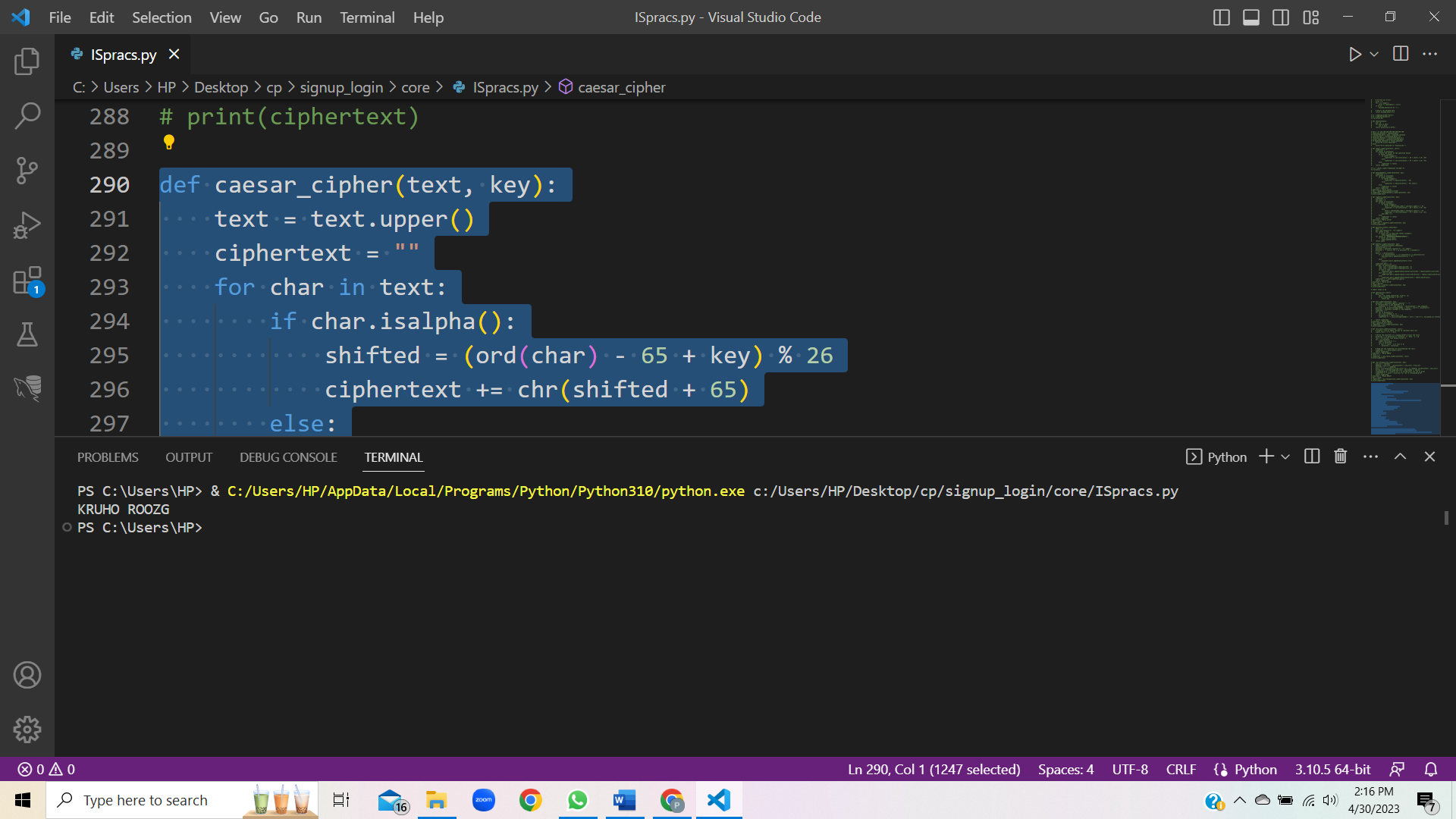
caesar\_key = 3

rail\_fence\_key = 3

ciphertext = product\_cipher(plaintext, caesar\_key, rail\_fence\_key)

print(ciphertext)

output:-



10. Illustrate the Ciphertext only and Known Plaintext attacks.

CipherText-only Attack

In cryptography, a ciphertext-only attack (COA) or known ciphertext attack is an attack model for cryptanalysis where the attacker is assumed to have access only to a set of ciphertexts.The attack is completely successful if the corresponding plaintexts can be deduced, or even better, the key. The ability to obtain any information at all about the underlying plaintext is still considered a success. For example, if an adversary is sending ciphertext continuously to maintain traffic-flow security, it would be very useful to be able to distinguish real messages from nulls. Even making an informed guess of the existence of real messages would facilitate traffic analysis. Every modern cipher attempts to provide protection against ciphertextonly attacks. The vetting process for a new cipher design standard usually takes many years and includes exhaustive testing of large quantities of ciphertext for any statistical departure from random noise. Encryption Standard process. Also, the field of steganography evolved, in part, to develop methods like mimic functions that allow one piece of data to adopt the statistical profile of another. Nonetheless poor cipherusage or reliance on home-grown proprietary algorithms that have notbeen subject to thorough scrutiny has resulted in many computer-age encryption systems that are still subject to ciphertext-only attack.

Known Plaintext Attack

The known-plaintext attack (KPA) or crib is an attack model for cryptanalysis where the attacker has samples of both the plaintext and its encrypted version (ciphertext), and is at liberty to make use of them to reveal further secret information such as secret keys and code books. The term "crib" originated at Bletchley Park, the British World War II decryption operation. Classical ciphers are typically vulnerable to known-plaintext attack. For example, a Caesar cipher can be solved using a single letter of corresponding plaintext and ciphertext to decrypt entirely. A general monoalphabetic substitution cipher needs several character pairs and some guessing if there are fewer than 26 distinct pairs. Modern ciphers such as Advanced Encryption Standard are not susceptible to known-plaintext attacks.

11. Implement a stream cipher technique

def rc4(key, plaintext):

    # Initialize the S and T arrays

    S = [i for i in range(256)]

    T = [ord(key[i % len(key)]) for i in range(256)]

    # Initialize the index variables

    i, j = 0, 0

    # Generate the pseudorandom stream of bytes

    for k in range(256):

        i = (i + 1) % 256

        j = (j + S[i]) % 256

        S[i], S[j] = S[j], S[i]

    # Generate the ciphertext by combining the plaintext with the pseudorandom stream using bitwise XOR

    ciphertext = ""

    for k in range(len(plaintext)):

        i = (i + 1) % 256

        j = (j + S[i]) % 256

        S[i], S[j] = S[j], S[i]

        t = (S[i] + S[j]) % 256

        ciphertext += chr(ord(plaintext[k]) ^ S[t])

    # Return the ciphertext

    return ciphertext

key = "SECRET\_KEY"

plaintext = "HELLO WORLD"

ciphertext = rc4(key, plaintext)

print(ciphertext)

output:-

