Practical 1

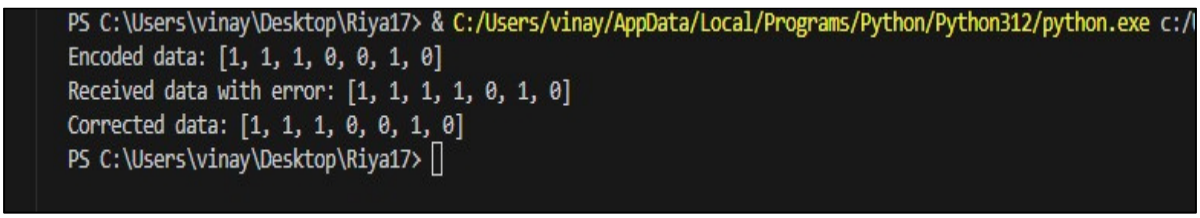
Implement error correcting code.

```
def add_parity_bits(data):
"""Add parity bits to the data."""
# Calculate the number of parity bits needed
parity_bits_count = 0
while 2 ** parity_bits_count <= len(data) + parity_bits_count:
parity_bits_count += 1

# Insert parity bits
encoded_data = []
j = 0
for i in range(1, len(data) + parity_bits_count + 1):
if i == 2 ** j:
encoded_data.append(0) # Placeholder for parity bit
j += 1
else:
encoded_data.append(data.pop(0))

# Calculate parity bits
for i in range(parity_bits_count):
parity_index = 2 ** i - 1
parity = 0
    for j in range(parity_index, len(encoded_data), 2 * (parity_index + 1)):
parity ^= encoded_data[j]
encoded_data[parity_index] = parity
```

```
return encoded_data

def correct_errors(encoded_data):
"""Correct errors in the encoded data."""
parity_bits_count = 0
while 2 ** parity_bits_count <= len(encoded_data):
parity_bits_count += 1

# Calculate the syndrome
syndrome = 0
for i in range(parity_bits_count):
parity_index = 2 ** i - 1
parity = 0
    for j in range(parity_index, len(encoded_data), 2 * (parity_index + 1)):
parity ^= encoded_data[j]
syndrome |= parity << i

# Correct errors if any
if syndrome != 0:
error_index = syndrome - 1
encoded_data[error_index] ^= 1

return encoded_data

def hamming_encode(data):
"""Encode the data using Hamming (7, 4) code."""
```

```
 encoded_data = add_parity_bits(data)
return encoded_data

def hamming_decode(encoded_data):
"""Decode the Hamming encoded data."""
corrected_data = correct_errors(encoded_data)
return corrected_data

# Example usage:
data = [1, 0, 1, 0] # 4-bit data
encoded_data = hamming_encode(data)
print("Encoded data:", encoded_data)

# Simulating an error in transmission
encoded_data[3] = 1 - encoded_data[3] # Introducing an error
print("Received data with error:", encoded_data)

corrected_data = hamming_decode(encoded_data)
print("Corrected data:", corrected_data)
```

OUTPUT:



Practical 2

Implement the error detecting code cpp

```
#include <iostream>
#include <bitset>

// Function to calculate the parity bit for a given data word
int calculateParityBit(int data) {
    int parity = 0;

    // Calculate parity by XORing all the bits
    while (data) {
        parity ^= (data & 1);
        data >>= 1;
    }

    return parity;
}

// Function to encode the data with a parity bit
int addParityBit(int data) {
 // Calculate the parity bit
 int parity = calculateParityBit(data);

 // Add the parity bit to the least significant bit position
 return (data << 1) | parity;
}
```

```
/ Function to check if there is any error in the received data
bool checkError(int receivedData) {
 // Calculate the parity bit of the received data
 int receivedParity = calculateParityBit(receivedData);

 // Extract the received parity bit
 int receivedBit = receivedData & 1;

 // If the calculated parity and received parity don't match,
there is an error
 return (receivedBit != receivedParity);
}

int main() {
 // Example usage
 int data = 0b1011; // Data to be sent
 int encodedData = addParityBit(data); // Add parity bit
 std::cout << "Encoded Data: " << std::bitset<5>(encodedData)
<< std::endl;

 // Simulate error by flipping a bit
 int receivedData = encodedData ^ (1 << 2); // Flipping the
third bit

 // Check for error
 bool errorDetected = checkError(receivedData);

 if (errorDetected) {
 std::cout << "Error detected in received data!" << std::endl;
 } else {
 std::cout << "No error detected in received data." << std::endl;
 }

 return 0;
}
```

Practical 3

Implement caesar cipher substitution operation in python.

Code:

```
def caesar_cipher(text, shift):
result = ''
for char in text:
# For uppercase letters
if char.isupper():
            result += chr((ord(char) + shift - 65) % 26 + 65)
# For lowercase letters
elif char.islower():
            result += chr((ord(char) + shift - 97) % 26 + 97)
else:
result += char
return result

# Example usage
text = "Hello, World!"
shift = 3
encrypted_text = caesar_cipher(text, shift)
print("Encrypted:", encrypted_text)

# Decrypting the encrypted text
decrypted_text = caesar_cipher(encrypted_text, -shift)
print("Decrypted:", decrypted_text)
```

Output:

```
PS C:\Users\vinay\Desktop\Riya17> & C:/Users/vinay/AppData/Local/Programs/Python/Python312/python.exe c:/Users/vinay/Desktop/Riy
a17/is/p3.py
Encrypted: Khoor, Zruog!
Decrypted: Hello, World!
PS C:\Users\vinay\Desktop\Riya17>
```

Practical 4

Implement monoalphabetic and polyalphabetic cipher substitution.

Code:

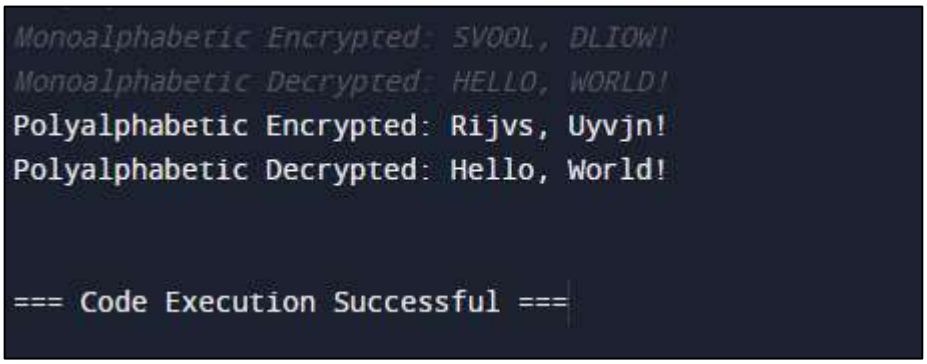
```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

// Function to encrypt a message using monoalphabetic substitution cipher
string monoalphabeticEncrypt(const string& message, const string& key) {
string encryptedMessage = message;
for (char& c : encryptedMessage) {
if (isalpha(c)) {
char base = isupper(c) ? 'A' : 'a';
c = key[c - base];
}
}
return encryptedMessage;
}

// Function to decrypt a message using monoalphabetic substitution cipher
string monoalphabeticDecrypt(const string& encryptedMessage, const string& key) {
string decryptedMessage = encryptedMessage;
for (char& c : decryptedMessage) {
if (isalpha(c)) {
char base = isupper(c) ? 'A' : 'a';
c = 'A' + distance(key.begin(), find(key.begin(), key.end(), c));
}
```

```
 }
return decryptedMessage;
}

// Function to encrypt a message using polyalphabetic substitution cipher (Vigenere cipher)
string polyalphabeticEncrypt(const string& message, const string& key) {
string encryptedMessage;
int keyIndex = 0;
for (char c : message) {
if (isalpha(c)) {
char base = isupper(c) ? 'A' : 'a';
char shifted = ((c - base) + (key[keyIndex % key.length()] - 'A')) % 26 + base;
encryptedMessage.push_back(shifted);
keyIndex++;
} else {
encryptedMessage.push_back(c);
}
}
return encryptedMessage;
}

// Function to decrypt a message using polyalphabetic substitution cipher (Vigenere cipher)
string polyalphabeticDecrypt(const string& encryptedMessage, const string& key) {
string decryptedMessage;
int keyIndex = 0;
for (char c : encryptedMessage) {
if (isalpha(c)) {
char base = isupper(c) ? 'A' : 'a';
        char shifted = ((c - base) - (key[keyIndex % key.length()] - 'A') + 26) % 26 + base;
```

```
 decryptedMessage.push_back(shifted);
keyIndex++;
} else {
decryptedMessage.push_back(c);
}
}
return decryptedMessage;
}

int main() {
string message = "Hello, World!";
string monoalphabeticKey = "ZYXWVUTSRQPONMLKJIHGFEDCBA";
string polyalphabeticKey = "KEY";

// Encrypt using monoalphabetic substitution cipher
string encryptedMonoalphabetic = monoalphabeticEncrypt(message,
monoalphabeticKey);
cout << "Monoalphabetic Encrypted: " << encryptedMonoalphabetic << endl;

// Decrypt using monoalphabetic substitution cipher
   string decryptedMonoalphabetic = monoalphabeticDecrypt(encryptedMonoalphabetic,
monoalphabeticKey);
cout << "Monoalphabetic Decrypted: " << decryptedMonoalphabetic << endl;

// Encrypt using polyalphabetic substitution cipher (Vigenere cipher)
  string encryptedPolyalphabetic = polyalphabeticEncrypt(message, polyalphabeticKey);
cout << "Polyalphabetic Encrypted: " << encryptedPolyalphabetic << endl;

// Decrypt using polyalphabetic substitution cipher (Vigenere cipher)
```

```
 string decryptedPolyalphabetic = polyalphabeticDecrypt(encryptedPolyalphabetic,
polyalphabeticKey);

cout << "Polyalphabetic Decrypted: " << decryptedPolyalphabetic << endl;


return 0;

}
```

Output:



Practical 5

Q5- Implement playfair cipher substitution operation.

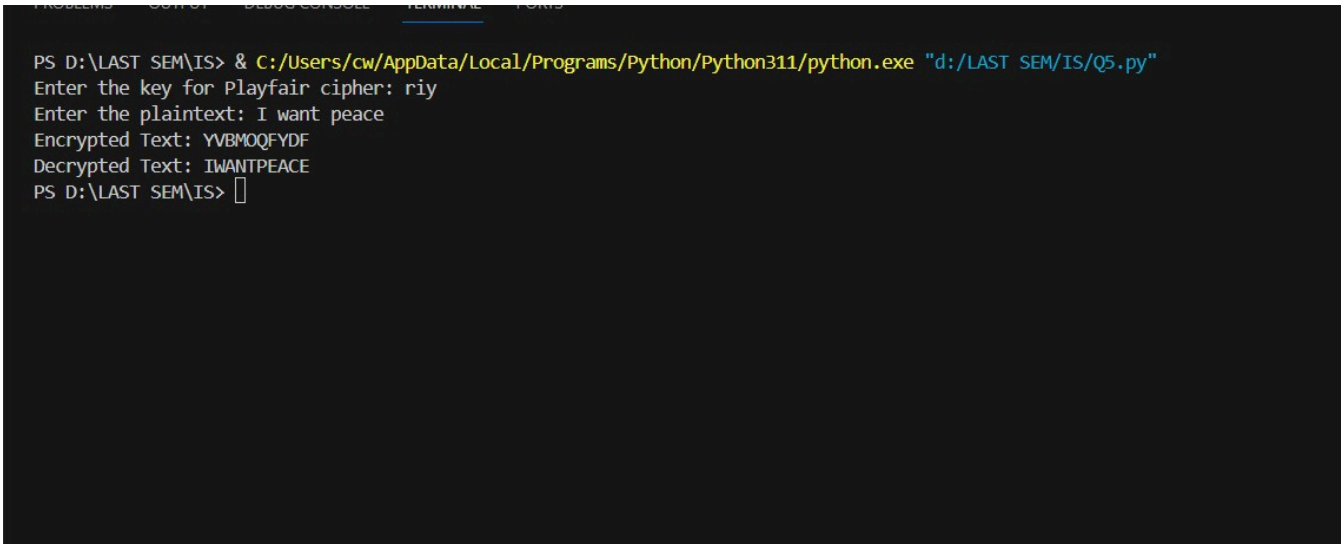
Code-

```
def prepare_input(text):
# Remove spaces and convert to uppercase
text = text.replace(" ", "").upper()
# Replace 'J' with 'I'
text = text.replace("J", "I")
# Split the text into pairs of letters
pairs = []
for i in range(0, len(text), 2):
pair = text[i:i+2]
        if len(pair) == 1: # If the last pair has only one letter, add 'X' to
make it a pair
pair += 'X'
pairs.append(pair)
return pairs

def generate_key_matrix(key):
# Remove spaces and convert to uppercase
key = key.replace(" ", "").upper()
# Replace 'J' with 'I'
key = key.replace("J", "I")
# Create a set of unique letters from the key (without duplicates)
key_set = list(dict.fromkeys(key))
# Create the key matrix (5x5 grid)
key_matrix = [['' for _ in range(5)] for _ in range(5)]
i, j = 0, 0
for letter in key_set:
key_matrix[i][j] = letter
```

```
 j += 1
if j == 5:
j = 0
i += 1
    # Fill the remaining spaces with the remaining letters of the alphabet
(excluding 'J')
alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"
for letter in alphabet:
if letter not in key_set:
key_matrix[i][j] = letter
j += 1
if j == 5:
j = 0
i += 1
return key_matrix

def find_letter_positions(letter, key_matrix):
for i in range(5):
for j in range(5):
if key_matrix[i][j] == letter:
return (i, j)

def encrypt(plaintext, key):
pairs = prepare_input(plaintext)
key_matrix = generate_key_matrix(key)
cipher_text = ''
for pair in pairs:
char1, char2 = pair[0], pair[1]
row1, col1 = find_letter_positions(char1, key_matrix)
row2, col2 = find_letter_positions(char2, key_matrix)
if row1 == row2: # Same row
cipher_text += key_matrix[row1][(col1 + 1) % 5]
cipher_text += key_matrix[row2][(col2 + 1) % 5]
elif col1 == col2: # Same column
cipher_text += key_matrix[(row1 + 1) % 5][col1]
cipher_text += key_matrix[(row2 + 1) % 5][col2]
else: # Forming rectangle
cipher_text += key_matrix[row1][col2]
cipher_text += key_matrix[row2][col1]
return cipher_text

def decrypt(ciphertext, key):
pairs = prepare_input(ciphertext)
key_matrix = generate_key_matrix(key)
plain_text = ''
```

```
 for pair in pairs:
char1, char2 = pair[0], pair[1]
        row1, col1 = find_letter_positions(char1, key_matrix)
        row2, col2 = find_letter_positions(char2, key_matrix)
if row1 == row2: # Same row
plain_text += key_matrix[row1][(col1 - 1) % 5]
plain_text += key_matrix[row2][(col2 - 1) % 5]
elif col1 == col2: # Same column
plain_text += key_matrix[(row1 - 1) % 5][col1]
plain_text += key_matrix[(row2 - 1) % 5][col2]
else: # Forming rectangle
plain_text += key_matrix[row1][col2]
plain_text += key_matrix[row2][col1]
return plain_text

def main():
key = input("Enter the key for Playfair cipher: ")
plaintext = input("Enter the plaintext: ")
encrypted_text = encrypt(plaintext, key)
print("Encrypted Text:", encrypted_text)
decrypted_text = decrypt(encrypted_text, key)
print("Decrypted Text:", decrypted_text)

if __name__ == "__main__":
main()
```

Output-



Practical 6

Implement hill cipher substitution operation.

Code:

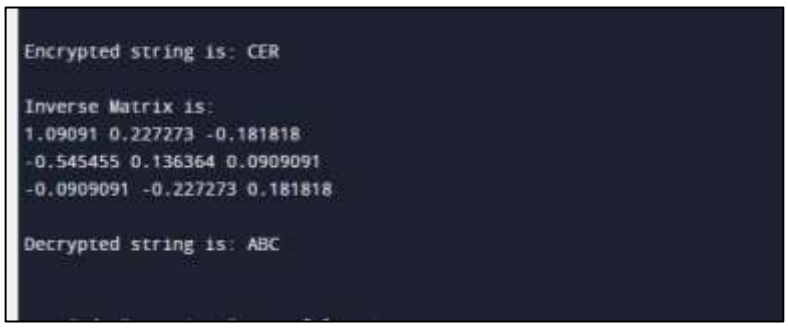
```
#include<iostream>
#include<math.h>
using namespace std;
float en[3][1], de[3][1], a[3][3], b[3][3], msg[3][1], m[3][3];
void getKeyMatrix() { //get key and message from user
int i, j;
char mes[3];
  cout<<"Enter 3x3 matrix for key (should have inverse):\n";
for(i = 0; i < 3; i++)
for(j = 0; j < 3; j++) {
cin>>a[i][j];
m[i][j] = a[i][j];
}
cout<<"\nEnter a string of 3 letter(use A through Z): ";
cin>>mes;
for(i = 0; i < 3; i++)
msg[i][0] = mes[i] - 65;
}
void encrypt() { //encrypts the message
int i, j, k;
for(i = 0; i < 3; i++)
for(j = 0; j < 1; j++)
for(k = 0; k < 3; k++)
en[i][j] = en[i][j] + a[i][k] * msg[k][j];
cout<<"\nEncrypted string is: ";
for(i = 0; i < 3; i++)
```

```
 cout<<(char)(fmod(en[i][0], 26) + 65); //modulo 26 is taken for each element of the matrix obtained by multiplication
}
void inversematrix() { //find inverse of key matrix
int i, j, k;
float p, q;
for(i = 0; i < 3; i++)
for(j = 0; j < 3; j++) {
if(i == j)
b[i][j]=1;
else
b[i][j]=0;
}
for(k = 0; k < 3; k++) {
for(i = 0; i < 3; i++) {
p = m[i][k];
q = m[k][k];
for(j = 0; j < 3; j++) {
if(i != k) {
m[i][j] = m[i][j]*q - p*m[k][j];
b[i][j] = b[i][j]*q - p*b[k][j];
}
}
}
}
for(i = 0; i < 3; i++)
for(j = 0; j < 3; j++)
b[i][j] = b[i][j] / m[i][i];
cout<<"\n\nInverse Matrix is:\n";
for(i = 0; i < 3; i++) {
```

```
 for(j = 0; j < 3; j++)
cout<<b[i][j]<<" ";
cout<<"\n";
}
}
void decrypt() { //decrypt the message
int i, j, k;
inversematrix();
for(i = 0; i < 3; i++)
for(j = 0; j < 1; j++)
for(k = 0; k < 3; k++)
de[i][j] = de[i][j] + b[i][k] * en[k][j];
cout<<"\nDecrypted string is: ";
for(i = 0; i < 3; i++)
  cout<<(char)(fmod(de[i][0], 26) + 65); //modulo 26 is taken to get the original message
cout<<"\n";
}
int main() {
getKeyMatrix();
encrypt();
decrypt();
}
```

Output:

```
Enter 3x3 matrix for key (should have inverse):
1
0
1
2
4
0
3
5
6

Enter a string of 3 letter(use A through Z): ABC

Encrypted string is: CER

Inverse Matrix is:
```


Encrypted string is: CER
Inverse Matrix is:
1.09091 0.227273 -0.181818
-0.545455 0.136364 0.0909091
-0.0909091 -0.227273 0.181818
Decrypted string is: ABC

Practical 7

Implement rail fence cipher transposition operation.

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

string railFenceEncrypt(const string& plaintext, int rails) {
vector<string> fence(rails);
int row = 0;
bool down = false;

for (char c : plaintext) {
fence[row] += c;
if (row == 0 || row == rails - 1) {
down = !down;
}
row += down ? 1 : -1;
}

string ciphertext;
for (const string& rail : fence) {
ciphertext += rail;
}

return ciphertext;
}
```

```
string railFenceDecrypt(const string& ciphertext, int rails) {
vector<string> fence(rails);
vector<int> indices(ciphertext.size());

int row = 0;
bool down = false;

for (int i = 0; i < ciphertext.size(); ++i) {
indices[i] = row;
if (row == 0 || row == rails - 1) {
down = !down;
}
row += down ? 1 : -1;
}

int index = 0;
for (int i = 0; i < rails; ++i) {
for (int j = 0; j < indices.size(); ++j) {
if (indices[j] == i) {
fence[i] += ciphertext[index++];
}
}
}

string plaintext;
row = 0;
down = false;

for (int i = 0; i < ciphertext.size(); ++i) {
plaintext += fence[row][0];
```

```
 fence[row].erase(0, 1);
if (row == 0 || row == rails - 1) {
down = !down;
}
row += down ? 1 : -1;
}

return plaintext;
}

int main() {
string plaintext, encrypted, decrypted;
int rails;

cout << "Enter the plaintext: ";
getline(cin, plaintext);

cout << "Enter the number of rails: ";
cin >> rails;

   encrypted = railFenceEncrypt(plaintext, rails);
    cout << "Encrypted: " << encrypted << endl;

   decrypted = railFenceDecrypt(encrypted, rails);
    cout << "Decrypted: " << decrypted << endl;

return 0;
}
```

OUTPUT:

```
/tmp/RsAfbyKrEf.o
Enter the plaintext: Hello World
Enter the number of rails: 3
Encrypted: Horel ollWd
Decrypted: Hello World


=== Code Execution Successful ===
```

Practical 8

Implement row transposition cipher transposition operation.

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

string rowTranspositionEncrypt(const string& plaintext, const vector<int>& key) {
int rows = key.size();
int cols = (plaintext.size() + rows - 1) / rows;

vector<vector<char>> matrix(rows, vector<char>(cols, ' '));
int index = 0;

for (int col = 0; col < cols; col++) {
for (int row : key) {
if (index < plaintext.size()) {
matrix[row][col] = plaintext[index++];
} else {
break;
}
}
}

string ciphertext;
for (int row = 0; row < rows; row++) {
for (int col = 0; col < cols; col++) {
```

```
 ciphertext += matrix[row][col];
}
}

return ciphertext;
}

string rowTranspositionDecrypt(const string& ciphertext, const vector<int>& key) {
int rows = key.size();
int cols = (ciphertext.size() + rows - 1) / rows;

vector<vector<char>> matrix(rows, vector<char>(cols, ' '));
int index = 0;

for (int row = 0; row < rows; row++) {
for (int col = 0; col < cols; col++) {
matrix[row][col] = ciphertext[index++];
}
}

string plaintext;
for (int col = 0; col < cols; col++) {
for (int row : key) {
plaintext += matrix[row][col];
}
}

return plaintext;
}

int main() {
```

```
 string plaintext, encrypted, decrypted;
vector<int> key;

cout << "Enter the plaintext: ";
getline(cin, plaintext);

   cout << "Enter the key (comma-separated row numbers, e.g., 2,1,3): ";
string keyInput;
getline(cin, keyInput);

size_t pos = 0;
while ((pos = keyInput.find(',')) != string::npos) {
key.push_back(stoi(keyInput.substr(0, pos)));
keyInput.erase(0, pos + 1);
}
key.push_back(stoi(keyInput)); // Add the last key element

encrypted = rowTranspositionEncrypt(plaintext, key);
cout << "Encrypted: " << encrypted << endl;

decrypted = rowTranspositionDecrypt(encrypted, key);
cout << "Decrypted: " << decrypted << endl;

return 0;
}
```

OUTPUT:

```
Enter the plaintext: Hello, World!
Enter the key (comma-separated row numbers, e.g., 2,1,3): 2,1,3
Encrypted: eolr,lW l!Hod
Decrypted: Hello, World!
```

Practical File

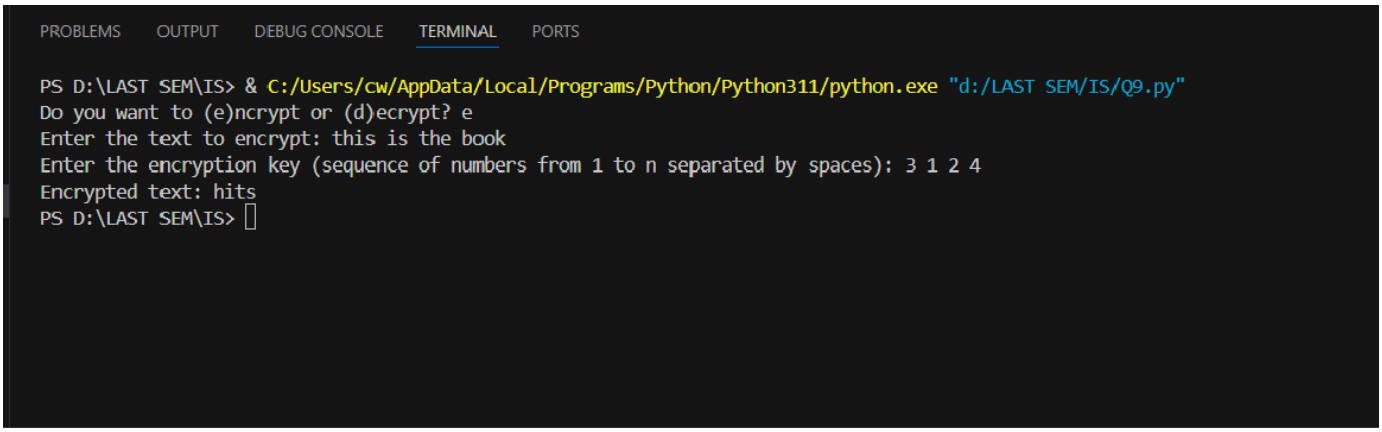
Q9- Implement product cipher transposition operation.

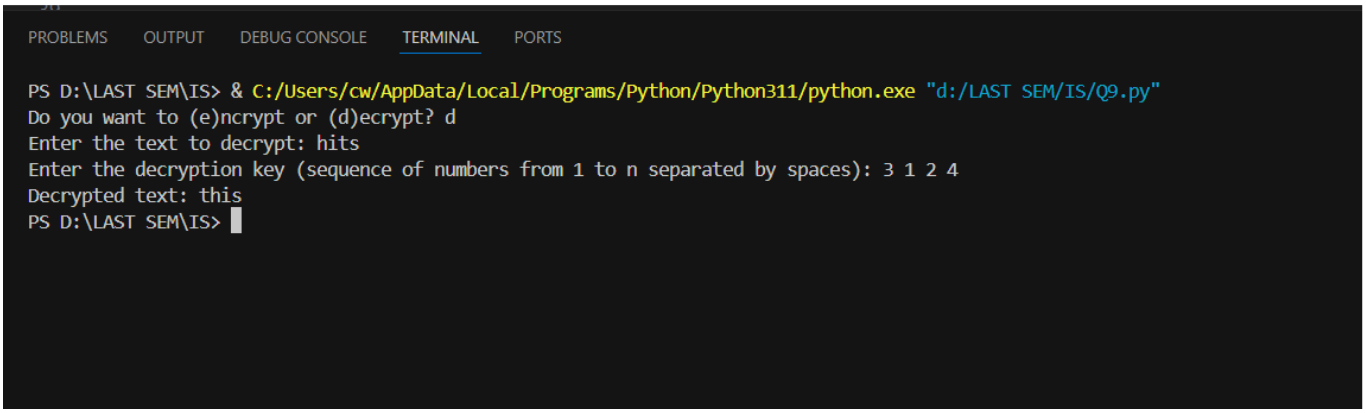
Code-

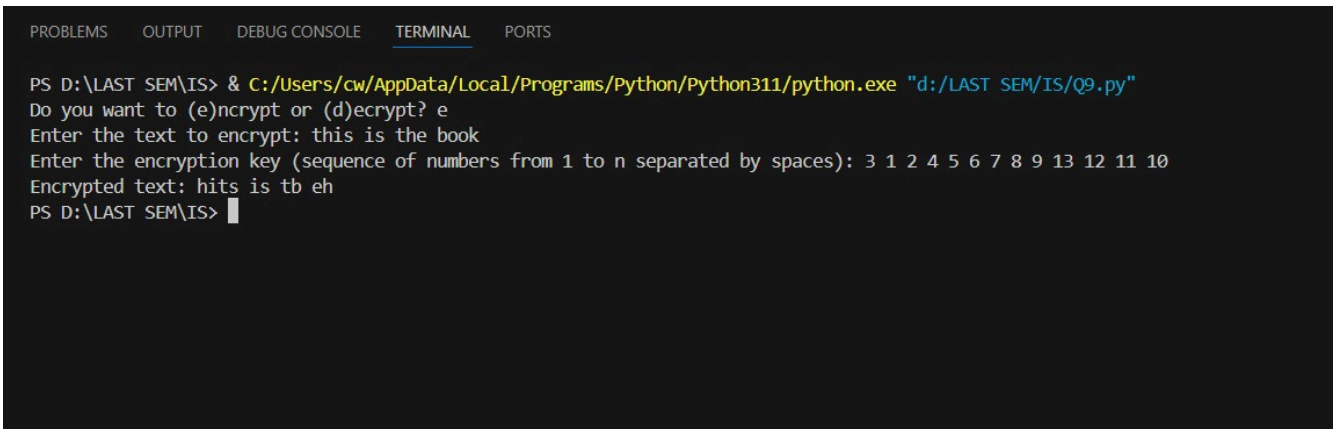
```
def encrypt(text, key):
encrypted_text = [''] * len(key)

# Arrange the text based on the key
for i in range(len(key)):
encrypted_text[key[i] - 1] = text[i]

return ''.join(encrypted_text)

def decrypt(text, key):
decrypted_text = [''] * len(key)

# Rearrange the text based on the key
for i in range(len(key)):
decrypted_text[i] = text[key[i] - 1]

return ''.join(decrypted_text)

def main():
choice = input("Do you want to (e)ncrypt or (d)ecrypt? ").lower()
if choice == 'e':
text = input("Enter the text to encrypt: ")
       key = list(map(int, input("Enter the encryption key (sequence of numbers
from 1 to n separated by spaces): ").split()))
if sorted(key) != list(range(1, len(key) + 1)):
print("Invalid key. Key should be a sequence of numbers from 1 to
n.")
return
encrypted_text = encrypt(text, key)
print("Encrypted text:", encrypted_text)
```

```
 elif choice == 'd':
text = input("Enter the text to decrypt: ")
       key = list(map(int, input("Enter the decryption key (sequence of numbers
from 1 to n separated by spaces): ").split()))
if sorted(key) != list(range(1, len(key) + 1)):
print("Invalid key. Key should be a sequence of numbers from 1 to
n.")
return
decrypted_text = decrypt(text, key)
print("Decrypted text:", decrypted_text)
else:
print("Invalid choice.")

if __name__ == "__main__":
main()
```

Output-






PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS D:\LAST SEM\IS> & C:/Users/cw/AppData/Local/Programs/Python/Python311/python.exe "d:/LAST SEM/IS/Q9.py"
Do you want to (e)ncrypt or (d)ecrypt? e
Enter the text to encrypt: this is the book
Enter the encryption key (sequence of numbers from 1 to n separated by spaces): 3 1 2 4 5 6 7 8 9 13 12 11 10
Encrypted text: hits is tb eh
PS D:\LAST SEM\IS>

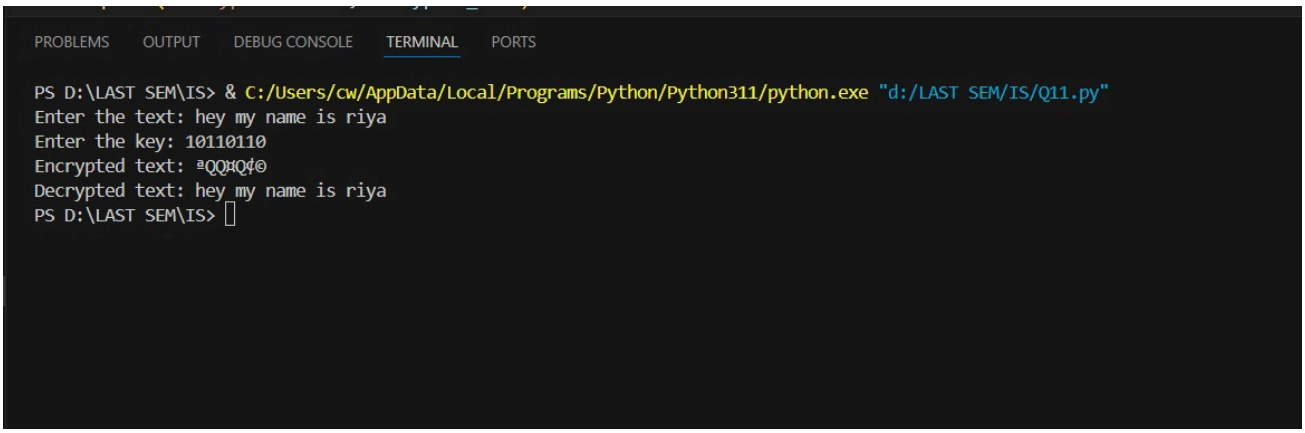
Practical File

Q11- Implement a stream cipher technique.

Code-

```
def stream_cipher(text, key, mode):
key_length = len(key)
text_length = len(text)
result = ""
for i in range(text_length):
key_index = i % key_length
if mode == 'encrypt':
            result += chr((ord(text[i]) + ord(key[key_index])) % 256)
elif mode == 'decrypt':
            result += chr((ord(text[i]) - ord(key[key_index])) % 256)
return result

# Take input from the user
text = input("Enter the text: ")
key = input("Enter the key: ")

# Encrypt the text
encrypted_text = stream_cipher(text, key, 'encrypt')
print("Encrypted text:", encrypted_text)

# Decrypt the encrypted text
decrypted_text = stream_cipher(encrypted_text, key, 'decrypt')
print("Decrypted text:", decrypted_text)
```

Output-


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS D:\LAST SEM\IS> & C:/Users/cw/AppData/Local/Programs/Python/Python311/python.exe "d:/LAST SEM/IS/Q11.py"
Enter the text: hey my name is riya
Enter the key: 10110110
Encrypted text: ªQQ¤Q¢©
Decrypted text: hey my name is riya
PS D:\LAST SEM\IS>