IS LAB_2

~ Prof. Dr. Aashka Raval

✅ **Title 2 :** Implementation of 5x5 Playfair Cipher (Standard)

🎯 Objective:

- To implement the classical 5x5 Playfair Cipher encryption and decryption technique.

- To understand symmetric key-based digraph substitution and how it increases cryptographic strength.

📘 Introduction:

The **Playfair Cipher** is a classical encryption technique invented in 1854 by Charles Wheatstone and popularized by Lord Playfair. It encrypts text in **pairs of letters (digraphs)** using a 5x5 matrix formed from a secret keyword. Unlike monoalphabetic ciphers, it operates on digraphs, making frequency analysis more difficult and adding strength through positional relationships.

📚 Concepts Used:

- Symmetric Key Cryptography
- Digraph Substitution
- 5x5 Matrix Generation using a Keyword (merging I/J)
- Text Preprocessing (sanitization, pair creation)
- Encryption & Decryption Logic Based on Matrix Positioning

🧠 Logic:

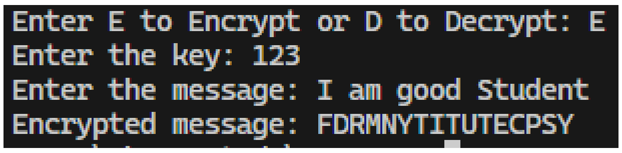
1. **Matrix Generation**:
   - Remove duplicate letters from the key.
   - Replace 'J' with 'I' (combine them).
   - Fill the 5x5 matrix with key letters, then remaining unused letters from A-Z.
2. **Text Preprocessing**:
   - Convert plaintext to uppercase, remove non-letters.
   - Replace 'J' with 'I'.

  - Break the text into pairs.
  - Insert ‘X’ between same letters and at the end if needed.
3. **Encryption Rules**:
  - **Same Row**: Replace each letter with the one to its immediate right (wrap around if needed).
  - **Same Column**: Replace each letter with the one directly below.
  - **Rectangle Rule**: Replace each letter with the one in the same row but in the column of the other letter.
4. **Decryption**:
  - Apply the inverse of the above rules.

💻 Python Code:

```python
def generate_matrix(key):
    key = key.upper().replace('J', 'I')
    matrix = []
    used = set()

    for char in key:
        if char not in used and char.isalpha():
            used.add(char)
            matrix.append(char)

    for char in "ABCDEFGHIKLMNOPQRSTUVWXYZ":
        if char not in used:
            used.add(char)
            matrix.append(char)

    return [matrix[i*5:(i+1)*5] for i in range(5)]

def find_position(matrix, char):
    for i, row in enumerate(matrix):
        for j, c in enumerate(row):
            if c == char:
                return i, j
    return None

def process_text(text):
    text = text.upper().replace('J', 'I')
    text = ''.join(filter(str.isalpha, text))
    result = ""
    i = 0
    while i < len(text):
        a = text[i]
        b = text[i+1] if i + 1 < len(text) else 'X'
        if a == b:
            result += a + 'X'
            i += 1
        else:
            result += a + b
            i += 2
    if len(result) % 2 != 0:
        result += 'X'
    return result

def encrypt_pair(a, b, matrix):
    r1, c1 = find_position(matrix, a)
    r2, c2 = find_position(matrix, b)

    if r1 == r2:
        return matrix[r1][(c1+1)%5] + matrix[r2][(c2+1)%5]
    elif c1 == c2:
        return matrix[(r1+1)%5][c1] + matrix[(r2+1)%5][c2]
    else:
        return matrix[r1][c2] + matrix[r2][c1]

def decrypt_pair(a, b, matrix):
    r1, c1 = find_position(matrix, a)
    r2, c2 = find_position(matrix, b)

    if r1 == r2:
        return matrix[r1][(c1-1)%5] + matrix[r2][(c2-1)%5]
    elif c1 == c2:
        return matrix[(r1-1)%5][c1] + matrix[(r2-1)%5][c2]
    else:
        return matrix[r1][c2] + matrix[r2][c1]

def playfair_encrypt(plaintext, key):
    matrix = generate_matrix(key)
    text = process_text(plaintext)
    encrypted = ''
    for i in range(0, len(text), 2):
        encrypted += encrypt_pair(text[i], text[i+1], matrix)
    return encrypted

def playfair_decrypt(ciphertext, key):
    matrix = generate_matrix(key)
    decrypted = ''
    for i in range(0, len(ciphertext), 2):
        decrypted += decrypt_pair(ciphertext[i], ciphertext[i+1], matrix)
    return decrypted


choice = input("Enter E to Encrypt or D to Decrypt: ").strip().upper()
key = input("Enter the key: ").strip()
message = input("Enter the message: ").strip()

if choice == 'E':
    encrypted = playfair_encrypt(message, key)
    print("Encrypted message:", encrypted)
elif choice == 'D':
    decrypted = playfair_decrypt(message, key)
    print("Decrypted message:", decrypted)
else:
    print("Invalid choice.")
```

🧪 Sample Output:



```
Enter E to Encrypt or D to Decrypt: E
Enter the key: 123
Enter the message: I am good Student
Encrypted message: FDRMNYTITUTECPSY
```

✅ Conclusion:

The 5x5 Playfair Cipher introduces the concept of **pair-based encryption**, enhancing security over single-letter substitutions. Through matrix manipulation, it teaches foundational cryptography concepts such as **confusion, diffusion, and key dependency**. It also builds understanding of how classical ciphers form the base of modern symmetric encryption techniques.

💻 Lab Title 2: Implementation of Reversed Rule Playfair Cipher (5x5 Matrix)

🔍 Objective:

- To implement a modified Playfair Cipher where the decryption logic is reversed.
- To understand how reversing traditional cipher rules can impact cryptographic behavior.

📘 Introduction:

The **Reversed Rule Playfair Cipher** is a variation of the classic 5x5 Playfair Cipher. In this version, the encryption logic remains the same, but the **decryption rules are reversed** to test how minor logical inversions affect the cipher strength and outcome.

Instead of moving right or down for decryption (as in standard Playfair), the cipher now uses:

- **Leftward** movement in rows
- **Upward** movement in columns
- **No column swapping** for rectangle-based digraphs (unlike traditional Playfair)

This modified decryption adds an unusual twist and demonstrates how subtle rule changes alter both implementation and results.

📚 Concepts Used:

- Digraph-based encryption
- Playfair cipher (5x5 matrix logic)
- Matrix indexing (row/column)
- Symmetric key transformation
- Modified classical cipher decryption

🧠 Logic:

🔑 1. Matrix Construction:

- Keyword is sanitized: uppercase, no duplicate letters, 'J' replaced by 'I'.
- Fill 5x5 matrix row-wise with unique letters from the key, followed by remaining alphabet.

🔤 2. Text Preprocessing:

- Convert input to uppercase, remove non-alphabetic characters, replace 'J' with 'I'.
- Split into letter pairs (digraphs). Insert 'X' if same letters appear together.
- Append 'X' if the final text length is odd.

🔐 3. Encryption (Standard Playfair Rules):

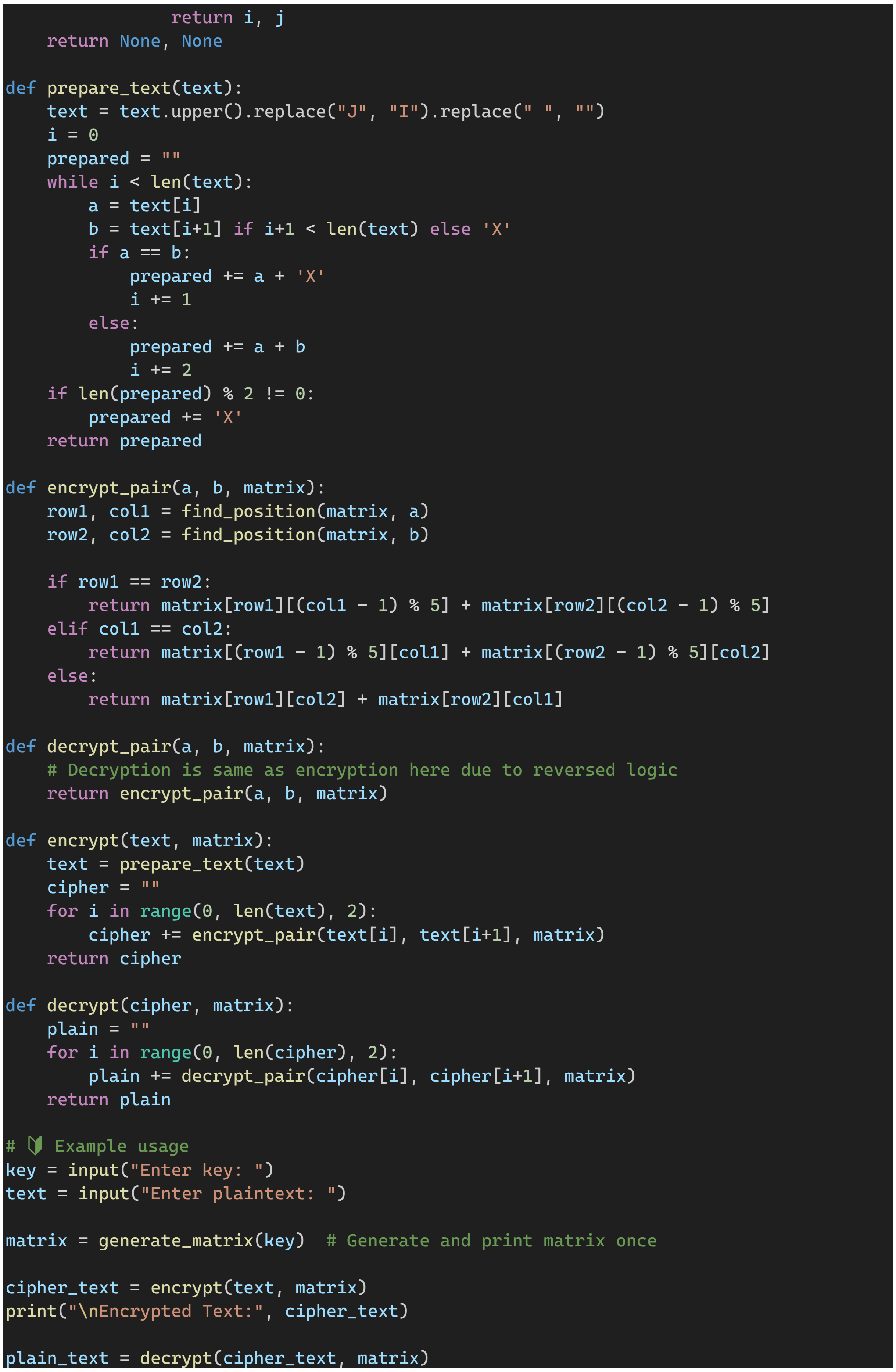
- **Same Row** → Replace each letter with the one to its **right**.
- **Same Column** → Replace each letter with the one **below**.
- **Rectangle** → Swap columns of both letters.

🔒 4. Modified Decryption (Reversed Rules):

- **Same Row** → Replace each letter with the one to its **left**.
- **Same Column** → Replace each letter with the one **above**.
- **Rectangle** → Do **not** swap columns; retain original column while replacing with opposite corner.

📜 Python Code:

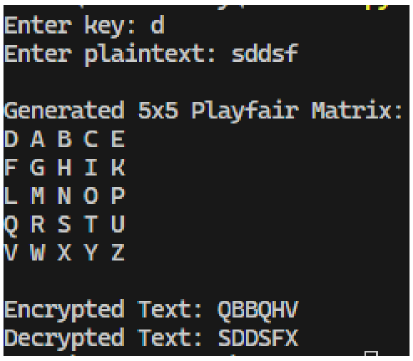
```python
def generate_matrix(key):
    matrix = []
    seen = set()
    key = key.replace("J", "I").upper()
    for char in key:
        if char.isalpha() and char not in seen:
            seen.add(char)
            matrix.append(char)

    for i in range(65, 91):
        char = chr(i)
        if char == 'J':
            continue
        if char not in seen:
            seen.add(char)
            matrix.append(char)

    final_matrix = [matrix[i:i+5] for i in range(0, 25, 5)]

    # 🔍 Print the 5x5 Playfair Matrix only once
    print("\nGenerated 5x5 Playfair Matrix:")
    for row in final_matrix:
        print(" ".join(row))

    return final_matrix

def find_position(matrix, char):
    for i in range(5):
        for j in range(5):
            if matrix[i][j] == char:
```



```python
                return i, j
    return None, None

def prepare_text(text):
    text = text.upper().replace("J", "I").replace(" ", "")
    i = 0
    prepared = ""
    while i < len(text):
        a = text[i]
        b = text[i+1] if i+1 < len(text) else 'X'
        if a == b:
            prepared += a + 'X'
            i += 1
        else:
            prepared += a + b
            i += 2
    if len(prepared) % 2 != 0:
        prepared += 'X'
    return prepared

def encrypt_pair(a, b, matrix):
    row1, col1 = find_position(matrix, a)
    row2, col2 = find_position(matrix, b)

    if row1 == row2:
        return matrix[row1][(col1 - 1) % 5] + matrix[row2][(col2 - 1) % 5]
    elif col1 == col2:
        return matrix[(row1 - 1) % 5][col1] + matrix[(row2 - 1) % 5][col2]
    else:
        return matrix[row1][col2] + matrix[row2][col1]

def decrypt_pair(a, b, matrix):
    # Decryption is same as encryption here due to reversed logic
    return encrypt_pair(a, b, matrix)

def encrypt(text, matrix):
    text = prepare_text(text)
    cipher = ""
    for i in range(0, len(text), 2):
        cipher += encrypt_pair(text[i], text[i+1], matrix)
    return cipher

def decrypt(cipher, matrix):
    plain = ""
    for i in range(0, len(cipher), 2):
        plain += decrypt_pair(cipher[i], cipher[i+1], matrix)
    return plain

# 🔰 Example usage
key = input("Enter key: ")
text = input("Enter plaintext: ")

matrix = generate_matrix(key)  # Generate and print matrix once

cipher_text = encrypt(text, matrix)
print("\nEncrypted Text:", cipher_text)

plain_text = decrypt(cipher_text, matrix)
```

```python
print("Decrypted Text:", plain_text)
```

🧪 Sample Output:



```
Enter key: d
Enter plaintext: sddsf

Generated 5x5 Playfair Matrix:
D A B C E
F G H I K
L M N O P
Q R S T U
V W X Y Z

Encrypted Text: QBBQHV
Decrypted Text: SDDSFX
```

✅ Conclusion:

The **Reversed Rule Playfair Cipher** highlights how minor variations in classic algorithms influence both encryption behavior and strength. By reversing standard rules during decryption, the method increases confusion, making classical cryptanalysis more difficult and improving educational understanding of cipher behavior.
