

For the given input, perform Caesar cipher encryption and decryption.

Plain text: "CRYPTOGRAPHY"

Key: 10

```
def encrypt(plaintext,key):
    ciphertext=''
    for character in plaintext:
        if character.isalpha():
            ciphertext+=chr((ord(character) - 65 + key) % 26 + 65)
        else:
            ciphertext+=character
    return ciphertext
def decrypt(ciphertext,key):
    return encrypt(ciphertext, -key)
plaintext = "CRYPTOGRAPHY"
shift = 10 # Given key
encrypted = encrypt(plaintext, shift)
decrypted = decrypt(encrypted, shift)
print("Plaintext :", plaintext)
print("Encrypted plaintext :", encrypted)
print("Decrypted ciphertext:", decrypted)
```

```

1  def encrypt(plaintext,key):
2      ciphertext=''
3      for character in plaintext:
4          if character.isalpha():
5              ciphertext+=chr((ord(character)-65+key)%26+65)
6          else:
7              ciphertext+=character
8      return ciphertext
9  def decrypt(ciphertext,key):
10     return encrypt(ciphertext,-key)
11  plaintext="CRYPTOGRAPHY"
12  shift=10 # Given key
13
14  encrypted=encrypt(plaintext,shift)
15  decrypted=decrypt(encrypted,shift)
16
17  print("Plaintext:", plaintext)
18  print("Encrypted plaintext:", encrypted)
19  print("Decrypted ciphertext:", decrypted)

```

OUTPUT DEBUG CONSOLE TERMINAL PORTS PROBLEMS

```

PS D:\C++> python -u "d:\C++\tempCodeRunnerFile.python"
Plaintext : CRYPTOGRAPHY
Encrypted plaintext : MBIZDYQBKZRI
Decrypted ciphertext: CRYPTOGRAPHY
PS D:\C++>

```

Q2 For the plaintext given in question 1, apply Play Fair cipher encryption with key “WORK”.

```

import string
def generate_matrix(key):
    key = "".join(dict.fromkeys(key.upper()))
    alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    matrix = []
    used = set()
    for character in key:
        if character not in used and character in alphabet:
            matrix.append(character)
            used.add(character)
    for character in alphabet:
        if character not in used:

```

```

        matrix.append(character)
        used.add(character)
    return [matrix[i:i+5] for i in range(0, 25, 5)]
def format_plaintext(text):
    text = text.upper().replace("J", "I") # Replace J with I
    pairs = []
    i = 0
    while i < len(text):
        a = text[i]
        b = text[i+1] if i+1 < len(text) else "X"
        if a == b: # Same letters in pair, insert filler
            pairs.append(a + "X")
            i += 1
        else:
            pairs.append(a + b)
            i += 2
    return pairs
def find_position(matrix, ch):
    for r in range(5):
        for c in range(5):
            if matrix[r][c] == ch:
                return r, c
    return None
def playfair_encrypt(plaintext, key):
    matrix = generate_matrix(key)
    pairs = format_plaintext(plaintext)
    ciphertext = ""
    for a, b in pairs:
        r1, c1 = find_position(matrix, a)
        r2, c2 = find_position(matrix, b)
        if r1 == r2: # Same row
            ciphertext += matrix[r1][(c1 + 1) % 5]
            ciphertext += matrix[r2][(c2 + 1) % 5]
        elif c1 == c2: # Same column
            ciphertext += matrix[(r1 + 1) % 5][c1]
            ciphertext += matrix[(r2 + 1) % 5][c2]
        else: # Rectangle
            ciphertext += matrix[r1][c2]
            ciphertext += matrix[r2][c1]
    return ciphertext
plaintext = "CRYPTOGRAPHY"
key = "WORK"
ciphertext = playfair_encrypt(plaintext, key)
print("Key Matrix:")
for row in generate_matrix(key):
    print(row)

```

```
print("\nPlaintext :", plaintext)
print("Ciphertext:", ciphertext)
```

The screenshot shows a code editor with three tabs: 'EC_C_PES2UG23CS148_Lab3.py', 'def encrypt(plaintext,key): Untitled-1', and 'import string Untitled-2'. The main code is in the first tab. It defines a function to generate a key matrix from a key, a function to format the plaintext by replacing 'J' with 'I', and an encryption function. The terminal output shows the key matrix, the plaintext 'CRYPTOGRAPHY', and the ciphertext 'DOVSPAIWOTLV'.

```
1 import string
2 def generate_matrix(key):
3     key = "".join(dict.fromkeys(key.upper()))
4     alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
5     matrix = []
6     used = set()
7     for character in key: (variable) used: set
8         if character not in used and character in alphabet:
9             matrix.append(character)
10            used.add(character)
11    for character in alphabet:
12        if character not in used:
13            matrix.append(character)
14        used.add(character)
15    return [matrix[i:i+5] for i in range(0, 25, 5)]
16 def format_plaintext(text):
17     text = text.upper().replace("J", "I") # Replace J with I
18     pairs = []
19     i = 0
20     while i < len(text):
21         pairs.append(text[i:i+2])
22         i += 2
23     return pairs
24 def encrypt(plaintext, key):
25     key_matrix = generate_matrix(key)
26     formatted_plaintext = format_plaintext(plaintext)
27     ciphertext = ""
28     for pair in formatted_plaintext:
29         if len(pair) == 2:
30             # Convert pair to numbers
31             numbers = [ord(pair[0]) - ord('A'), ord(pair[1]) - ord('A')]
32             # Multiply by key matrix
33             result = [0, 0]
34             for j in range(2):
35                 result[j] = (key_matrix[j][0] * numbers[0] + key_matrix[j][1] * numbers[1]) % 26
36             # Convert result back to text
37             ciphertext += chr(result[0] + ord('A')) + chr(result[1] + ord('A'))
38         else:
39             ciphertext += pair
40     return ciphertext
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS PROBLEMS

```
PS D:\C++> python -u "d:\C++\tempCodeRunnerFile.python"
Key Matrix:
['W', 'O', 'R', 'K', 'A']
['B', 'C', 'D', 'E', 'F']
['G', 'H', 'I', 'L', 'M']
['N', 'P', 'Q', 'S', 'T']
['U', 'V', 'X', 'Y', 'Z']

Plaintext : CRYPTOGRAPHY
Ciphertext: DOVSPAIWOTLV
PS D:\C++>
```

Q3. For the plaintext= “WORK”, apply Hill cipher cipher encryption with key = [1,2; 2;2].

```
#this function coverts text to numbers
def text_to_numbers(text):
    return [ord(c) - ord('A') for c in text.upper()]
#this function converts numbers to text
def numbers_to_text(nums):
    return ''.join(chr(n + ord('A')) for n in nums)
#this function multiplies a 2x2 matrix with a vector
```

```

#the key given is also a 2x2 matrix
def multiply_matrix_vector(key, vector):
    return [
        (key[0][0]*vector[0] + key[0][1]*vector[1]) % 26,
        (key[1][0]*vector[0] + key[1][1]*vector[1]) % 26
    ]
def hillcipher_encrypt(plaintext, key):
    nums = text_to_numbers(plaintext)
    ciphertext_nums = []
    for i in range(0, len(nums), 2):
        block = nums[i:i+2]
        cipher_block = multiply_matrix_vector(key, block)
        ciphertext_nums.extend(cipher_block)
    return numbers_to_text(ciphertext_nums)
key = [[1, 2],
        [2, 2]]
plaintext = "WORK"
ciphertext = hillcipher_encrypt(plaintext, key)
print("Ciphertext:", ciphertext)

```

```

7  #this function multiplies a 2x2 matrix with a vector
8  #the key given is also a 2x2 matrix
9  def multiply_matrix_vector(key, vector):
10     return [
11         (key[0][0]*vector[0] + key[0][1]*vector[1]) % 26,
12         (key[1][0]*vector[0] + key[1][1]*vector[1]) % 26
13     ]
14  def hillcipher_encrypt(plaintext, key):
15     nums = text_to_numbers(plaintext)
16     ciphertext_nums = []
17     for i in range(0, len(nums), 2):
18         block = nums[i:i+2]
19         cipher_block = multiply_matrix_vector(key, block)
20         ciphertext_nums.extend(cipher_block)
21     return numbers_to_text(ciphertext_nums)
22  key = [[1, 2],
23         [2, 2]]
24  plaintext = "WORK"
25  ciphertext = hillcipher_encrypt(plaintext, key)
26  print("Ciphertext:", ciphertext)

```

OUTPUT DEBUG CONSOLE TERMINAL PORTS PROBLEMS

```

PS D:\C++> python -u "d:\C++\tempCodeRunnerFile.python"
Ciphertext: YULC
PS D:\C++>

```