Q1. Implementation of Playfair Cipher

```
def create_matrix(key):
    key = key.upper()
    matrix = [[0 for i in range (5)] for j in range(5)]
    letters_added = []
    for letter in key:
        if letter not in letters_added:
            letters_added.append(letter)
        else:
            continue
    for letter in range(65,91):
        if letter==74:
            continue
        if chr(letter) not in letters_added:
            letters_added.append(chr(letter))

    index = 0
    for i in range(5):
        for j in range(5):
            matrix[i][j] = letters_added[index]
            index+=1
    return matrix

def separate_same_letters(message):
    index = 0
    while (index<len(message)-1):
        l1 = message[index]
        l2 = message[index+1]
        if l1==l2:
            message = message[:index+1] + "X" + message[index+1:]
        index += 2
    if len(message) % 2 != 0:
      message += 'X'
    return message

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

def playfair(key, message, encrypt=True):
    inc = 1
    if encrypt==False:
        inc = -1
    matrix = create_matrix(key)
    message = message.upper()
    message = message.replace(' ','')
    message = separate_same_letters(message)
    cipher_text=''
```

```
    for (l1, l2) in zip(message[0::2], message[1::2]):
        row1,col1 = indexOf(l1,matrix)
        row2,col2 = indexOf(l2,matrix)
        if row1==row2: # Rule 1, the letters are in the same row
            cipher_text += matrix[row1][(col1+inc)%5] + matrix[row2][(col2+inc)%5]
        elif col1==col2:# Rule 2, the letters are in the same column
          cipher_text += matrix[(row1+inc)%5][col1] + matrix[(row2+inc)%5][col2]
        else: # Rule 3, the letters are in a different row and column
            cipher_text += matrix[row1][col2] + matrix[row2][col1]
    return cipher_text
print ('Encripting - ')
print ( playfair('secret', 'my secret message'))
print ('Decrypting - ')
print ( playfair('secret', 'LZECRTCSITCVAHBT', False))
```

```
Encripting -
LZECRTCSITCVAHBT
Decrypting -
MYSECRETMESXSAGE
```

Q2. Implementation of Hill Cipher

```
keyMatrix = [[0] * 3 for i in range(3)]
messageVector = [[0] for i in range(3)]
cipherMatrix = [[0] for i in range(3)]

def getKeyMatrix(key):
    k = 0
    for i in range(3):
        for j in range(3):
            keyMatrix[i][j] = ord(key[k]) % 65
            k += 1

def encrypt(messageVector):
    for i in range(3):
        for j in range(1):
            cipherMatrix[i][j] = 0
            for x in range(3):
                cipherMatrix[i][j] += (keyMatrix[i][x] * messageVector[x][j])
            cipherMatrix[i][j] = cipherMatrix[i][j] % 26

def HillCipher(message, key):
    getKeyMatrix(key)
    for i in range(3):
        messageVector[i][0] = ord(message[i]) % 65
    encrypt(messageVector)
    CipherText = []
    for i in range(3):
        CipherText.append(chr(cipherMatrix[i][0] + 65))
    print("Ciphertext: ", "".join(CipherText))

message = "ATH"
key = "GYBNQKURP"
```

```
HillCipher(message, key)
```

```
Ciphertext:  VKM
```

Q3. Implementation of Columnar Transposition cipher

```
def create_matrix():
  mat = [[0 for i in range(5)] for i in range(5)]
  return mat

def getelem(mat, col):
  elm = ''
  for i in range(1,5):
    elm += str(mat[i][col])
  return elm

def encode(key, plaintext):
  mat = create_matrix()
  plnlen = len(plaintext) - 1
  index = 0
  index2 = 0
  for i in range(5):
    for j in range(5):
      if index2 < len(key):
        mat[i][j] = key[index2]
        index2 += 1
        continue
      elif index <= plnlen:
        mat[i][j] = plaintext[index]
        index += 1
      else:
        break

  cipher = ''
  arr = ['1','2','3','4','5']
  for i in arr:
    for j in range(5):
      if i == mat[0][j]:
        cipher += getelem(mat, j)
  return cipher
ASHRHTDUAVMHEK
print(encode('43512', 'ATHARVDESHMUKH').replace('0', ''))
```

```
ASHRHTDUAVMHEK
```

Q4. Implementation of Diffie-Hellman

```
def isPrime(n):
  if (n <= 1):
    return False
  for i in range(2, n):
    if n % i == 0:
      return False
  return True

p = 17
q = 23
if isPrime(p) & isPrime(q):
  a = 2
  b = 3

  x = int(pow(q,a,p))
  y = int(pow(q,b,p))

  ka = int(pow(y,a,p))
  kb = int(pow(x,b,p))
else:
  print('Not Prime')
print(ka,kb)
```

```
8 8
```

Q5. Implementation of RSA

```
import math

def isPrime(n):
  if (n <= 1):
    return False
  for i in range(2, n):
    if n % i == 0:
      return False
  return True

def egcd(a, b):
  if a == 0:
    return (b, 0, 1)
  else:
    g, y, x = egcd(b % a, a)
  return (g, x - (b // a) * y, y)

def modinv(a, m):
  g, x, y = egcd(a, m)
  if g != 1:
    raise Exception('Modular inverse does not exist!')
  else:
    return x % m

p = int(input('Enter p: '))
```

```
q = int(input('Enter q: '))
if isPrime(p) and isPrime(q):
  n = p * q
  phi = (p - 1) * (q - 1)
  e = 2
  while e < phi:
    if math.gcd(e, phi) == 1:
      break
    else:
      e += 1
      d = modinv(e, phi)
      msg = float(input('Enter Message:'))
      print('Message Data:', msg)
      c = pow(msg, e)
      c = math.fmod(c, n)
      print('Encrypted data:', c)
      m = pow(c, d)
      m = math.fmod(m, n)
      print('Decrypted data:', m)
else:
  print('Error: p & q are not prime numbers')
```

```
Enter p: 11
Enter q: 3
Enter Message:28
Message Data: 28.0
Encrypted data: 7.0
Decrypted data: 28.0
```

✓ 1s completed at 8:41 PM