NOM : NDOMBELE

POSTNOM : DITUTALA

PRENOM : Hena

PROMOTION : L3 LMD INFORMATIQUE

TRAVAIL PRATIQUE DE CRYPTOGRAPHIE

RESOLUTION

QUESTION 1

- Algorithme pour la génération des clés de Feiste

```
def permutation(key, perm):
    permuted = ""
    for index in perm:
        permuted += key[index - 1]
    return permuted

def generate_feistel_keys(key, perm, shift_order):
    # Permutation de la clé
    permuted = permutation(key, perm)

    # Diviser en deux blocs
    left_block = permuted[:4]
    right_block = permuted[4:]

    # Calcul des sous-clés
    k1 = left_block ^ right_block
    k2 = right_block & left_block

    # Décalage à gauche d'ordre 2 pour k1
    k1_shifted = left_shift(k1, shift_order)

    # Décalage à droite d'ordre 1 pour k2
    k2_shifted = right_shift(k2, 1)

    return k1_shifted[:4], k2_shifted[:4]
```

```python
# Exemple d'utilisation
key = "11001010"  # Clé K de longueur 8
perm = [6, 5, 2, 7, 4, 1, 3, 0]  # Permutation H = 65274130
shift_order = 2  # Ordre de décalage

k1, k2 = generate_feistel_keys(key, perm, shift_order)
print(f"k1: {k1}")
print(f"k2: {k2}")
```

- Algorithme de chiffrement de Feistel

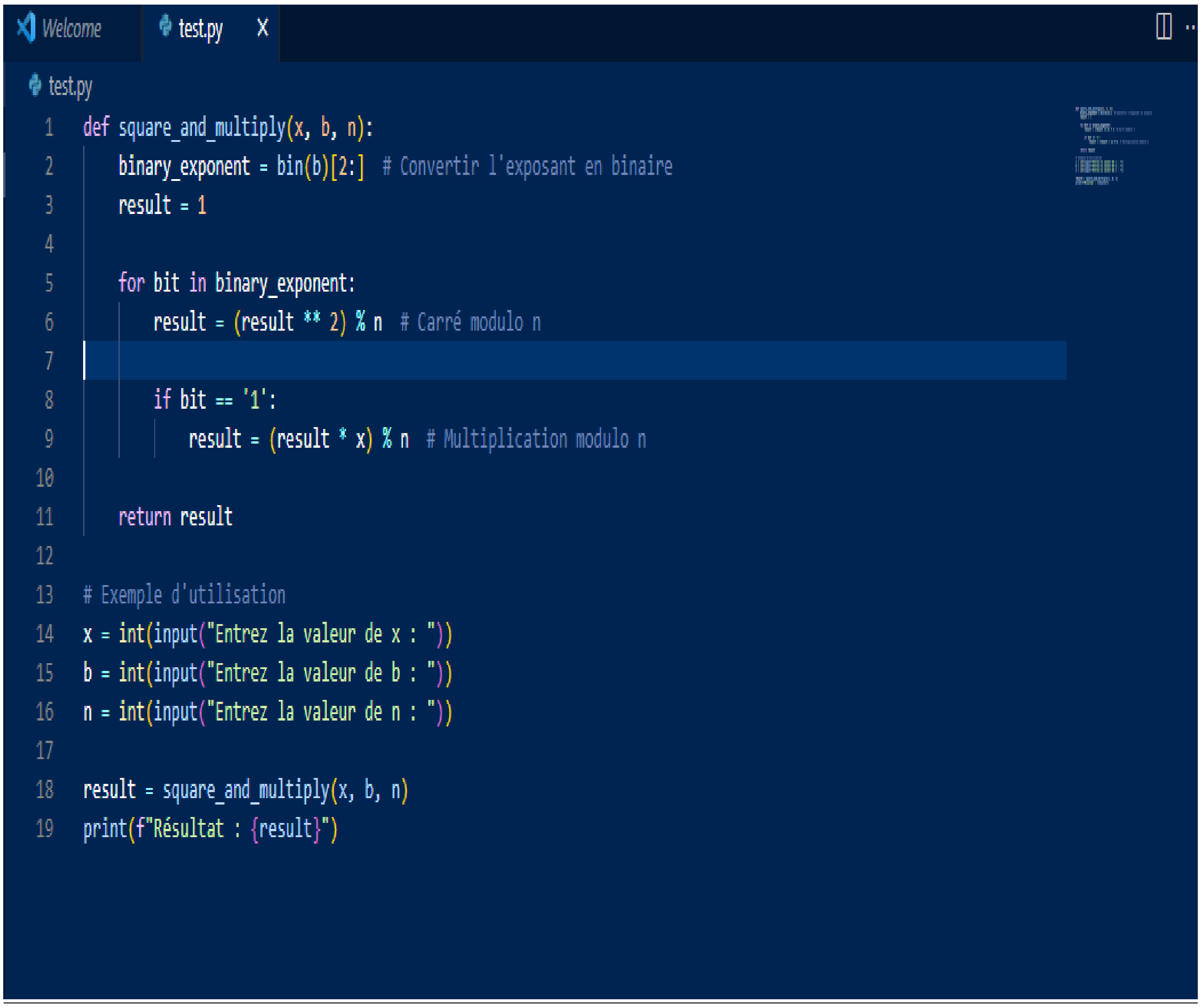
```python
def permutation(plaintext, perm):
    hena = ""
    for index in perm:
        hena += plaintext[index - 1]
    return permuted_text

def inverse_permutation(ciphertext, perm):
    inv_perm = [0] * len(perm)
    for i, val in enumerate(perm):
        inv_perm[val - 1] = i + 1
    return permutation(ciphertext, inv_perm)

def feistel_cipher(plaintext, perm, key1, key2, shift_order):
    # Permutation initiale
    hena = permutation(plaintext, perm)

    # Diviser en deux blocs
    left_block = hena[:4]
    right_block = hena[4:]

    # Premier Round
    temp = right_block
    right_block = permutation(left_block, [2, 0, 1, 3]) ^ key1
    left_block = temp ^ (permutation(left_block, [2, 0, 1, 3]) | key1)

    # Deuxième Round
    temp = right_block
    right_block = permutation(left_block, [2, 0, 1, 3]) ^ key2
    left_block = temp ^ (permutation(left_block, [2, 0, 1, 3]) | key2)

    # Concaténation
    ciphertext = right_block + left_block
    # Inverse de la permutation initiale
    ciphertext = inverse_permutation(ciphertext, perm)

    return ciphertext

# Exemple d'utilisation
plaintext = "10101010"  # Bloc N de 8 bits
perm = [4, 6, 0, 2, 7, 3, 1, 5]  # Permutation π = 46027315
key1 = "0101"  # Sous-clé k1 de longueur 4
key2 = "0011"  # Sous-clé k2 de longueur 4
shift_order = 2  # Ordre de décalage

ciphertext = feistel_cipher(plaintext, perm, key1, key2, shift_order)
print(f"Ciphertext: {ciphertext}")
```

- Algorithme de déchiffrement de Feiste

```
def permutation(ciphertext, perm):
    permuted_text = ""
    for index in perm:
        permuted_text += ciphertext[index - 1]
    return permuted_text

def inverse_permutation(plaintext, perm):
    inv_perm = [0] * len(perm)
    for i, val in enumerate(perm):
        inv_perm[val - 1] = i + 1
    return permutation(plaintext, inv_perm)

def feistel_decipher(ciphertext, perm, key1, key2, shift_order):
    # Permutation initiale
    permuted_text = permutation(ciphertext, perm)

    # Diviser en deux blocs
    left_block = permuted_text[:4]
    right_block = permuted_text[4:]

    # Deuxième Round
    temp = left_block
    left_block = permutation(right_block, [1, 2, 0, 3]) ^ key2
    right_block = temp ^ (left_block | key2)

    # Premier Round
```

```python
    temp = left_block
    left_block = permutation(right_block, [1, 2, 0, 3]) ^ key1
    right_block = temp ^ (left_block | key1)

    # Concaténation
    plaintext = left_block + right_block

    # Inverse de la permutation initiale
    plaintext = inverse_permutation(plaintext, perm)

    return plaintext

# Exemple d'utilisation
ciphertext = "01100101"  # Bloc C de 8 bits
perm = [4, 6, 0, 2, 7, 3, 1, 5]  # Permutation π = 46027315
key1 = "0101"  # Sous-clé k1 de longueur 4
key2 = "0011"  # Sous-clé k2 de longueur 4
shift_order = 2  # Ordre de décalage

plaintext = feistel_decipher(ciphertext, perm, key1, key2, shift_order)
print(f"Plaintext: {plaintext}")
```

<u>QUESTION 2</u>



```python
def square_and_multiply(x, b, n):
    binary_exponent = bin(b)[2:]  # Convertir l'exposant en binaire
    result = 1

    for bit in binary_exponent:
        result = (result ** 2) % n  # Carré modulo n

        if bit == '1':
            result = (result * x) % n  # Multiplication modulo n

    return result

# Exemple d'utilisation
x = int(input("Entrez la valeur de x : "))
b = int(input("Entrez la valeur de b : "))
n = int(input("Entrez la valeur de n : "))

result = square_and_multiply(x, b, n)
print(f"Résultat : {result}")
```