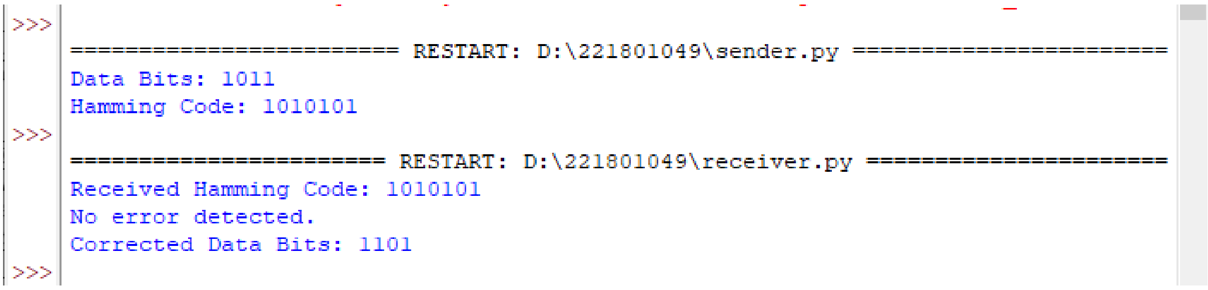
SENDER.py

```
def calculate_parity_bits(data_bits):
    n = len(data_bits)
    r = 0
    while (2**r < n + r + 1):
        r += 1
    return r

def insert_parity_bits(data_bits, r):
    j = 0
    k = 1
    m = len(data_bits)
    res = ' '

    for i in range(1, m + r + 1):
        if i == 2**j:
            res = res + '0'
            j += 1
        else:
            res = res + data_bits[-1 * k]
            k += 1
    return res[::-1]

def calculate_parity(data_bits, r):
    n = len(data_bits)
    data_bits = list(data_bits)

    for i in range(r):
        val = 0
        for j in range(1, n + 1):
            if j & (2**i) == (2**i):
                val = val ^ int(data_bits[-1 * j])
        data_bits[-1 * (2**i)] = str(val)

    return ''.join(data_bits)

def sender(data_bits):
    r = calculate_parity_bits(data_bits)
    data_with_parity = insert_parity_bits(data_bits, r)
    hamming_code = calculate_parity(data_with_parity, r)
    return hamming_code

if __name__ == "__main__":
    # Example Data Bits
    data_bits = '1011'

    # Generate Hamming Code
    hamming_code = sender(data_bits)
    print(f"Data Bits: {data_bits}")
    print(f"Hamming Code: {hamming_code}")

    with open("transmitted_data.txt", "w") as file:
        file.write(hamming_code)
```

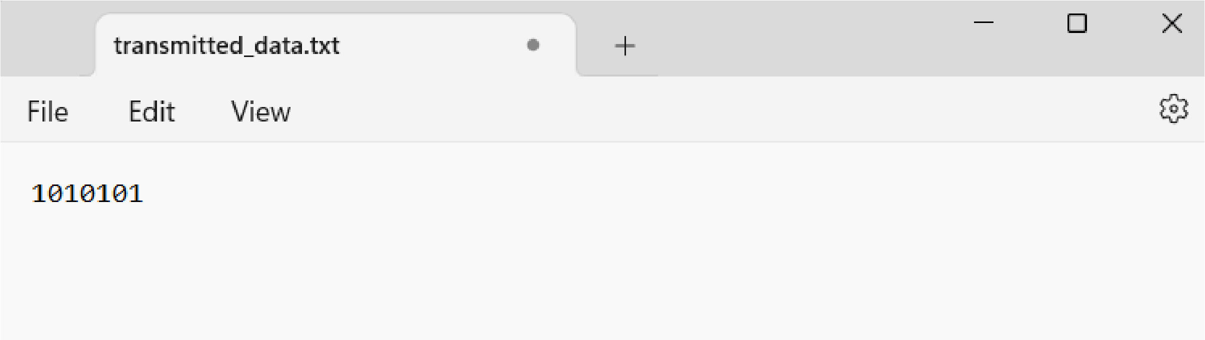
RECEIVER.py

```
def calculate_parity_bits(data_bits):
    n = len(data_bits)
    r = 0
    while (2**r < n + r + 1):
        r += 1
    return r

def detect_error(hamming_code, r):
    n = len(hamming_code)
    data_bits = list(hamming_code)
    res = 0
    for i in range(r):
        val = 0
        for j in range(1, n + 1):
            if j & (2**i) == (2**i):
                val = val ^ int(data_bits[-1 * j])
        res = res + val*(10**i)
    return int(str(res), 2)

def correct_error(hamming_code, error_position):
    if error_position == 0:
        return hamming_code
    error_position = len(hamming_code) - error_position
    hamming_code = list(hamming_code)
    if hamming_code[error_position] == '0':
        hamming_code[error_position] = '1'
    else:
        hamming_code[error_position] = '0'
    return ''.join(hamming_code)

def receiver(hamming_code):
    r = calculate_parity_bits(hamming_code)
    error_position = detect_error(hamming_code, r)
    if error_position:
        print(f"Error detected at position: {error_position}")
        hamming_code = correct_error(hamming_code, error_position)
        print(f"Corrected Hamming Code: {hamming_code}")
    else:
        print("No error detected.")
    data_bits = ''.join([hamming_code[i] for i in range(len(hamming_code)) if (i + 1) & i != 0])
    return data_bits

if __name__ == "__main__":
    # Read Hamming Code from file to simulate reception
    with open("transmitted_data.txt", "r") as file:
        received_code = file.read()
    print(f"Received Hamming Code: {received_code}")
    # Process the received Hamming Code
    corrected_data_bits = receiver(received_code)
    print(f"Corrected Data Bits: {corrected_data_bits}")
```

OUTPUT



```
>>> 
======================== RESTART: D:\221801049\sender.py ========================
Data Bits: 1011
Hamming Code: 1010101
>>> 
======================= RESTART: D:\221801049\receiver.py ======================
Received Hamming Code: 1010101
No error detected.
Corrected Data Bits: 1101
>>> 
```



```
1010101
```