HAMMING CODE:

```
import numpy as np

def txt_to_bin(txt):
    return ''.join(format(ord(c), '08b') for c in txt)

def bin_to_txt(bin_str):
    chars = [bin_str[i:i+8] for i in range(0, len(bin_str), 8)]
    return ''.join([chr(int(c, 2)) for c in chars])

def calc_r_bits(m):
    r = 0
    while (2**r < m + r + 1):
        r += 1
    return r

def pos_r_bits(data, r):
    j, k = 0, 0
    m = len(data)
    res = ''
    r_pos = []
    for i in range(1, m + r + 1):
        if i == 2**j:
            res += '0'
            r_pos.append(i)
            j += 1
        else:
            res += data[k]
            k += 1
    # Print all positions of redundant bits in one line
    print(f"Positions of redundant bits: {' '.join(map(str, r_pos))}")
```

```
    return res, r_pos


def calc_p_bits(arr, r):
    n = len(arr)
    arr = list(arr)
    parity_bits_info = []
    for i in range(r):
        p = 0
        pos = 2**i
        for j in range(1, n+1):
            if j & pos:
                p ^= int(arr[j-1])
        arr[pos-1] = str(p)
        parity_bits_info.append(f"Parity bit in position {pos}: {p}")
    # Print all parity bit information in one line
    print(' '.join(parity_bits_info))
    return ''.join(arr)


def detect_and_fix(data, r):
    n = len(data)
    res = 0
    for i in range(r):
        p = 0
        pos = 2**i
        for j in range(1, n+1):
            if j & pos:
                p ^= int(data[j-1])
        if p != 0:
            res += pos
    if res != 0:
        print(f"Error detected at position: {res}")
```

```
        data = list(data)
        if res <= n:
            data[res - 1] = '0' if data[res - 1] == '1' else '1'
            print(f"Error corrected at position: {res}")
        else:
            print("Error position out of range. No correction performed.")
        fixed_data = ''.join(data)
        print(f"Binary data after error correction: {fixed_data}")
        return fixed_data
    else:
        print("No error detected.")
        return data

def remove_r_bits(data, r):
    j = 0
    orig_data = ''
    for i in range(1, len(data) + 1):
        if i == 2**j:
            j += 1
        else:
            orig_data += data[i-1]
    return orig_data

def induce_err(data, pos):
    if pos < 1 or pos > len(data):
        print("Error position is out of range.")
        return data
    data = list(data)
    data[pos - 1] = '0' if data[pos - 1] == '1' else '1'
    print(f"Introduced error at position: {pos}")
```

```
    print(f"Binary data after introducing error: {''.join(data)}")
    return ''.join(data)

def sndr(txt):
    bin_data = txt_to_bin(txt)
    m = len(bin_data)
    r = calc_r_bits(m)
    arr, r_pos = pos_r_bits(bin_data, r)
    arr = calc_p_bits(arr, r)
    print(f"Sender output (binary with redundant bits): {arr}")
    return arr

def rcvr(data):
    r = calc_r_bits(len(data))
    fixed_data = detect_and_fix(data, r)
    orig_data = remove_r_bits(fixed_data, r)
    ascii_out = bin_to_txt(orig_data)
    print(f"Decoded text: {ascii_out}")

if __name__ == "__main__":
    inp_txt = input("Enter text to be encoded: ")
    ch_data = sndr(inp_txt)
    err_pos = int(input('Enter the bit position to introduce error: '))
    corrupt_data = induce_err(ch_data, err_pos)
    rcvr(corrupt_data)
```