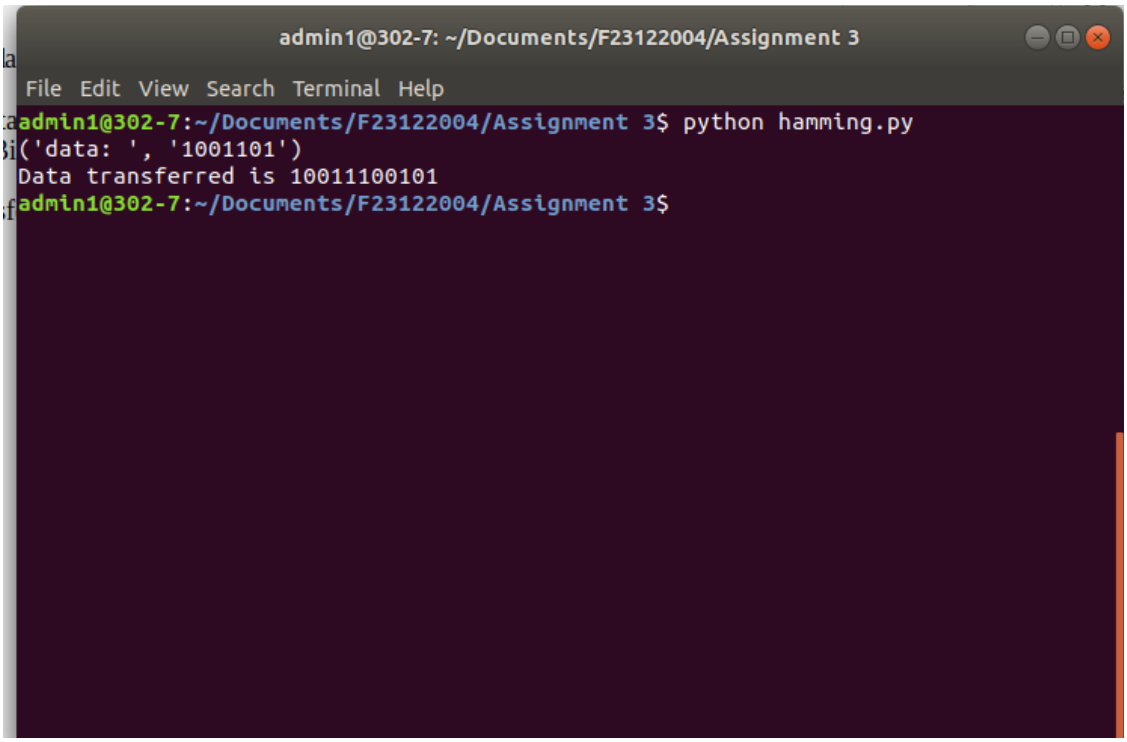
Hamming Code:-

code:-

```
def calcRedundantBits(m):
    for i in range(m):
        if(2**i >= m + i + 1):
            return i

def posRedundantBits(data, r):
    j = 0
    k = 1
    m = len(data)
    res = ''

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

    return res[::-1]

def calcParityBits(arr, r):
    n = len(arr)

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

        arr = arr[:n-(2**i)] + str(val) + arr[n-(2**i)+1:]
    return arr

def detectError(arr, nr):
    n = len(arr)
    res = 0

    for i in range(nr):
        val = 0
        for j in range(1, n + 1):
            if(j & (2**i) == (2**i)):
                val = val ^ int(arr[-1 * j])
        res = res + val*(10**i)
    return int(str(res), 2)

data = '1001101'

m = len(data)
r = calcRedundantBits(m)
```

```
arr = posRedundantBits(data, r)

print("data: ",data)
arr = calcParityBits(arr, r)

print("Data transferred is " + arr)
```

Output:-



Cyclic Redundancy Check

Code:-

```
class CRC:
    def __init__(self):
        self.cdw = ''

    def xor(self,a,b):
        result = []
        for i in range(1,len(b)):
            if a[i] == b[i]:
                result.append('0')
            else:
                result.append('1')
        return  ''.join(result)

    def crc(self,message, key):
        pick = len(key)
        tmp = message[:pick]
```

```
        while pick < len(message):
            if tmp[0] == '1':
                tmp = self.xor(key,tmp)+message[pick]
            else:
                tmp = self.xor('0'*pick,tmp) + message[pick]
            pick+=1

        if tmp[0] == "1":
            tmp = self.xor(key,tmp)
        else:
            tmp = self.xor('0'*pick,tmp)

        checkword = tmp
        return checkword

    def encodedData(self,data,key):
        l_key = len(key)
        append_data = data + '0'*(l_key-1)
        remainder = self.crc(append_data,key)
        codeword = data+remainder
        self.cdw += codeword
        print("Remainder: " ,remainder)
        print("Data: " ,codeword)

data = '10011101'
key = '10011'
print("data: ",data)
print("divisor: ",key)
c = CRC()
c.encodedData(data,key)
```

Output:-