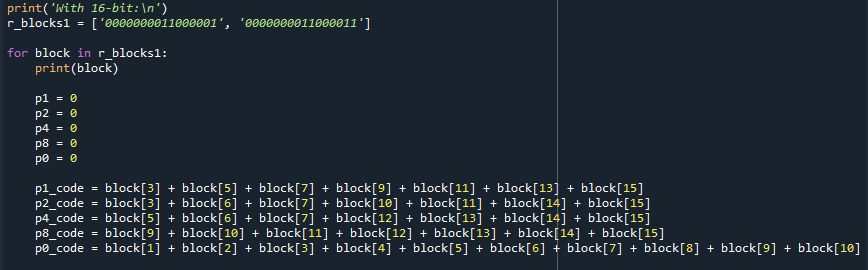
**Report for the Assignment VII**

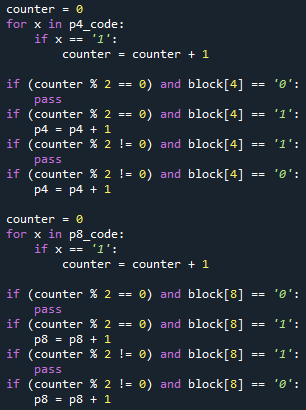
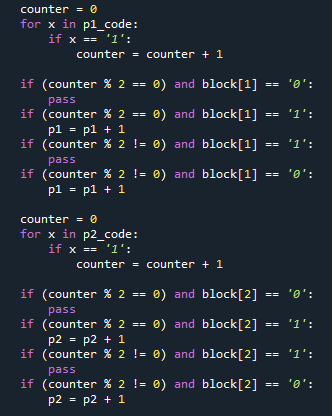
Ali Shalbayev, Mirolim Saidakhmatov

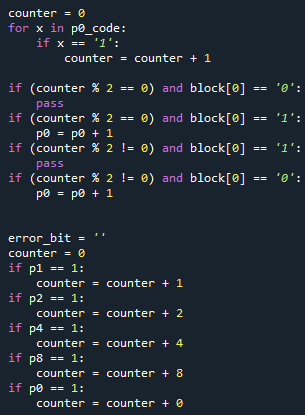
Ali’s approach:

Splitting into 16-bit blocks and checking parity bits:

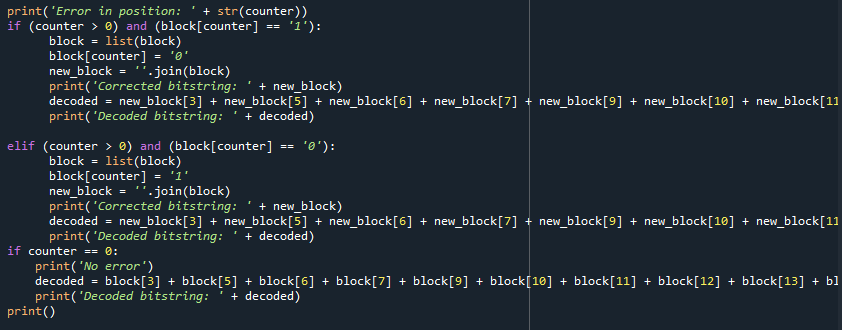


Checking the parity bits for error:



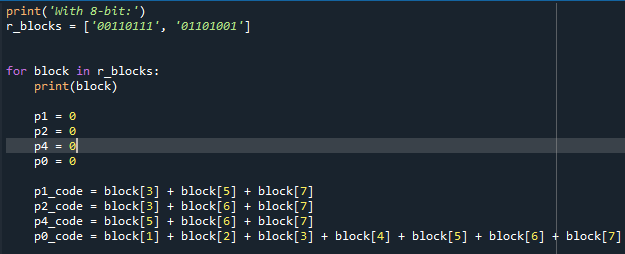


Determining and printing error out:

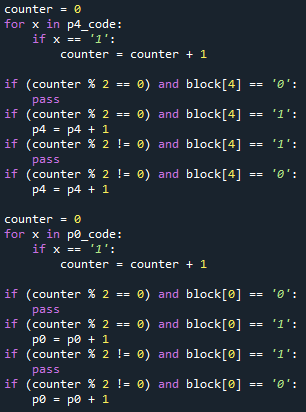
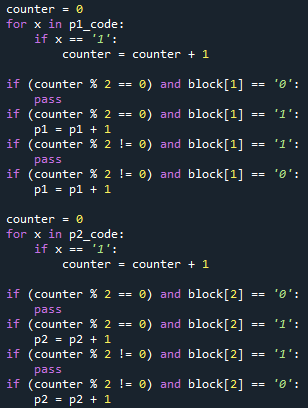


The same approaches with 8-bit data blocks:

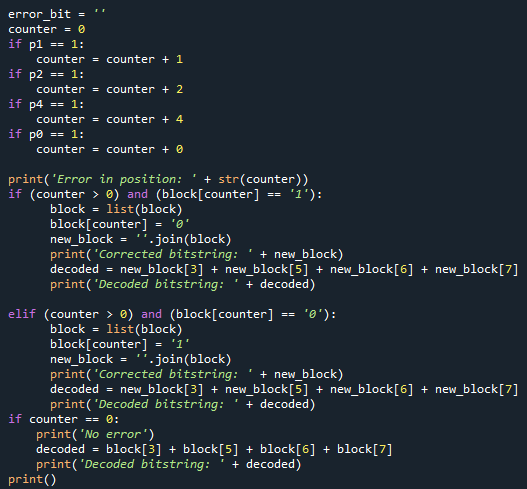
1)



2)

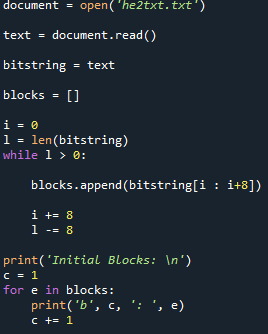


3)

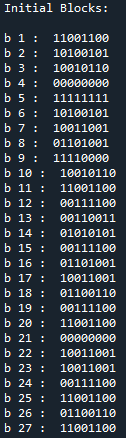


Mirolim’s approach:

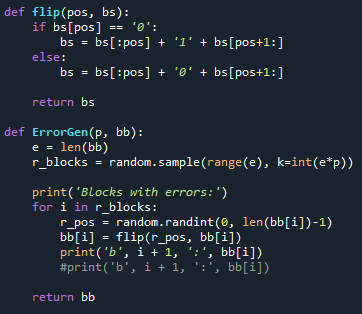
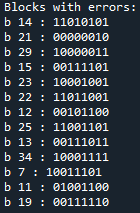
Reading a text and splitting bitstring into 8-bit blocks:



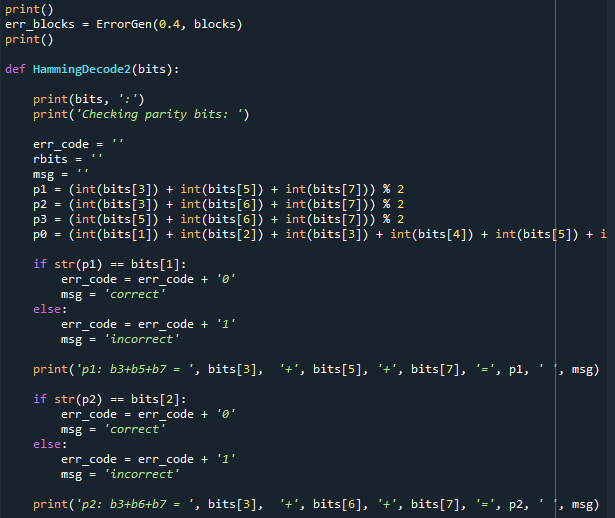
Printing Data Blocks:

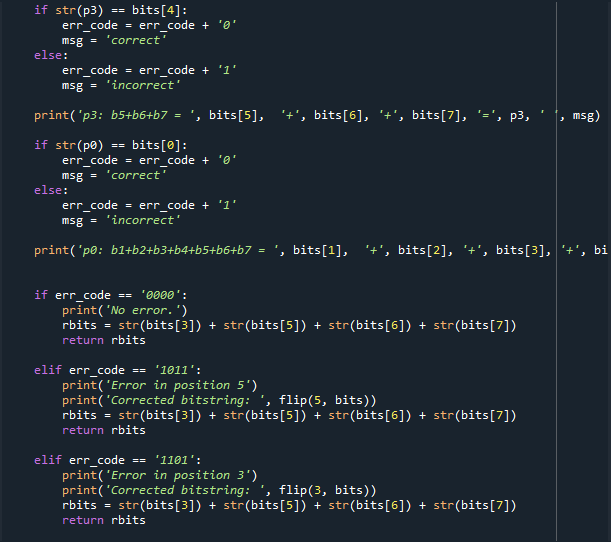


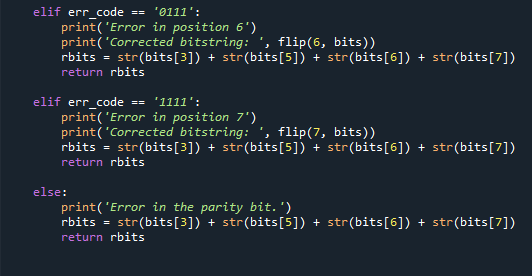
Functions for flipping and generating errors in blocks:

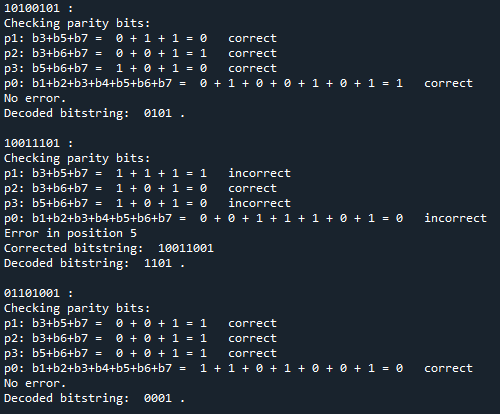
 

Generated error data blocks and a function that decodes by finding and correcting the data blocks:

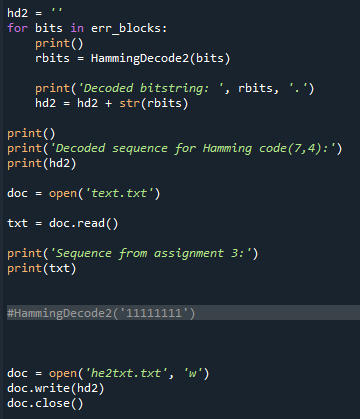


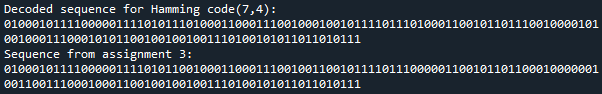






Combining all strings and saving in the txt file:





Source code:

import random

print('With 16-bit:\n')

r\_blocks1 = ['0000000011000001', '0000000011000011']

for block in r\_blocks1:

print(block)

p1 = 0

p2 = 0

p4 = 0

p8 = 0

p0 = 0

p1\_code = block[3] + block[5] + block[7] + block[9] + block[11] + block[13] + block[15]

p2\_code = block[3] + block[6] + block[7] + block[10] + block[11] + block[14] + block[15]

p4\_code = block[5] + block[6] + block[7] + block[12] + block[13] + block[14] + block[15]

p8\_code = block[9] + block[10] + block[11] + block[12] + block[13] + block[14] + block[15]

p0\_code = block[1] + block[2] + block[3] + block[4] + block[5] + block[6] + block[7] + block[8] + block[9] + block[10] + block[11] + block[12] + block[13] + block[14] + block[15]

counter = 0

for x in p1\_code:

if x == '1':

counter = counter + 1

if (counter % 2 == 0) and block[1] == '0':

pass

if (counter % 2 == 0) and block[1] == '1':

p1 = p1 + 1

if (counter % 2 != 0) and block[1] == '1':

pass

if (counter % 2 != 0) and block[1] == '0':

p1 = p1 + 1

counter = 0

for x in p2\_code:

if x == '1':

counter = counter + 1

if (counter % 2 == 0) and block[2] == '0':

pass

if (counter % 2 == 0) and block[2] == '1':

p2 = p2 + 1

if (counter % 2 != 0) and block[2] == '1':

pass

if (counter % 2 != 0) and block[2] == '0':

p2 = p2 + 1

counter = 0

for x in p4\_code:

if x == '1':

counter = counter + 1

if (counter % 2 == 0) and block[4] == '0':

pass

if (counter % 2 == 0) and block[4] == '1':

p4 = p4 + 1

if (counter % 2 != 0) and block[4] == '1':

pass

if (counter % 2 != 0) and block[4] == '0':

p4 = p4 + 1

counter = 0

for x in p8\_code:

if x == '1':

counter = counter + 1

if (counter % 2 == 0) and block[8] == '0':

pass

if (counter % 2 == 0) and block[8] == '1':

p8 = p8 + 1

if (counter % 2 != 0) and block[8] == '1':

pass

if (counter % 2 != 0) and block[8] == '0':

p8 = p8 + 1

counter = 0

for x in p0\_code:

if x == '1':

counter = counter + 1

if (counter % 2 == 0) and block[0] == '0':

pass

if (counter % 2 == 0) and block[0] == '1':

p0 = p0 + 1

if (counter % 2 != 0) and block[0] == '1':

pass

if (counter % 2 != 0) and block[0] == '0':

p0 = p0 + 1

error\_bit = ''

counter = 0

if p1 == 1:

counter = counter + 1

if p2 == 1:

counter = counter + 2

if p4 == 1:

counter = counter + 4

if p8 == 1:

counter = counter + 8

if p0 == 1:

counter = counter + 0

print('Error in position: ' + str(counter))

if (counter > 0) and (block[counter] == '1'):

block = list(block)

block[counter] = '0'

new\_block = ''.join(block)

print('Corrected bitstring: ' + new\_block)

decoded = new\_block[3] + new\_block[5] + new\_block[6] + new\_block[7] + new\_block[9] + new\_block[10] + new\_block[11] + new\_block[12] + new\_block[13] + new\_block[14] + new\_block[15]

print('Decoded bitstring: ' + decoded)

elif (counter > 0) and (block[counter] == '0'):

block = list(block)

block[counter] = '1'

new\_block = ''.join(block)

print('Corrected bitstring: ' + new\_block)

decoded = new\_block[3] + new\_block[5] + new\_block[6] + new\_block[7] + new\_block[9] + new\_block[10] + new\_block[11] + new\_block[12] + new\_block[13] + new\_block[14] + new\_block[15]

print('Decoded bitstring: ' + decoded)

if counter == 0:

print('No error')

decoded = block[3] + block[5] + block[6] + block[7] + block[9] + block[10] + block[11] + block[12] + block[13] + block[14] + block[15]

print('Decoded bitstring: ' + decoded)

print()

#-------------------------------------------------------------------

print('With 8-bit:')

r\_blocks = ['00110111', '01101001']

for block in r\_blocks:

print(block)

p1 = 0

p2 = 0

p4 = 0

p0 = 0

p1\_code = block[3] + block[5] + block[7]

p2\_code = block[3] + block[6] + block[7]

p4\_code = block[5] + block[6] + block[7]

p0\_code = block[1] + block[2] + block[3] + block[4] + block[5] + block[6] + block[7]

counter = 0

for x in p1\_code:

if x == '1':

counter = counter + 1

if (counter % 2 == 0) and block[1] == '0':

pass

if (counter % 2 == 0) and block[1] == '1':

p1 = p1 + 1

if (counter % 2 != 0) and block[1] == '1':

pass

if (counter % 2 != 0) and block[1] == '0':

p1 = p1 + 1

counter = 0

for x in p2\_code:

if x == '1':

counter = counter + 1

if (counter % 2 == 0) and block[2] == '0':

pass

if (counter % 2 == 0) and block[2] == '1':

p2 = p2 + 1

if (counter % 2 != 0) and block[2] == '1':

pass

if (counter % 2 != 0) and block[2] == '0':

p2 = p2 + 1

counter = 0

for x in p4\_code:

if x == '1':

counter = counter + 1

if (counter % 2 == 0) and block[4] == '0':

pass

if (counter % 2 == 0) and block[4] == '1':

p4 = p4 + 1

if (counter % 2 != 0) and block[4] == '1':

pass

if (counter % 2 != 0) and block[4] == '0':

p4 = p4 + 1

counter = 0

for x in p0\_code:

if x == '1':

counter = counter + 1

if (counter % 2 == 0) and block[0] == '0':

pass

if (counter % 2 == 0) and block[0] == '1':

p0 = p0 + 1

if (counter % 2 != 0) and block[0] == '1':

pass

if (counter % 2 != 0) and block[0] == '0':

p0 = p0 + 1

error\_bit = ''

counter = 0

if p1 == 1:

counter = counter + 1

if p2 == 1:

counter = counter + 2

if p4 == 1:

counter = counter + 4

if p0 == 1:

counter = counter + 0

print('Error in position: ' + str(counter))

if (counter > 0) and (block[counter] == '1'):

block = list(block)

block[counter] = '0'

new\_block = ''.join(block)

print('Corrected bitstring: ' + new\_block)

decoded = new\_block[3] + new\_block[5] + new\_block[6] + new\_block[7]

print('Decoded bitstring: ' + decoded)

elif (counter > 0) and (block[counter] == '0'):

block = list(block)

block[counter] = '1'

new\_block = ''.join(block)

print('Corrected bitstring: ' + new\_block)

decoded = new\_block[3] + new\_block[5] + new\_block[6] + new\_block[7]

print('Decoded bitstring: ' + decoded)

if counter == 0:

print('No error')

decoded = block[3] + block[5] + block[6] + block[7]

print('Decoded bitstring: ' + decoded)

print()

#---------------------------------------------------------

#C:\Users\M\Desktop\3rd trimester\IT\homework\asg7\

document = open('he2txt.txt')

text = document.read()

bitstring = text

blocks = []

i = 0

l = len(bitstring)

while l > 0:

blocks.append(bitstring[i : i+8])

i += 8

l -= 8

print('Initial Blocks: \n')

c = 1

for e in blocks:

print('b', c, ': ', e)

c += 1

def flip(pos, bs):

if bs[pos] == '0':

bs = bs[:pos] + '1' + bs[pos+1:]

else:

bs = bs[:pos] + '0' + bs[pos+1:]

return bs

def ErrorGen(p, bb):

e = len(bb)

r\_blocks = random.sample(range(e), k=int(e\*p))

print('Blocks with errors:')

for i in r\_blocks:

r\_pos = random.randint(0, len(bb[i])-1)

bb[i] = flip(r\_pos, bb[i])

print('b', i + 1, ':', bb[i])

#print('b', i + 1, ':', bb[i])

return bb

print()

err\_blocks = ErrorGen(0.4, blocks)

print()

def HammingDecode2(bits):

print(bits, ':')

print('Checking parity bits: ')

err\_code = ''

rbits = ''

msg = ''

p1 = (int(bits[3]) + int(bits[5]) + int(bits[7])) % 2

p2 = (int(bits[3]) + int(bits[6]) + int(bits[7])) % 2

p3 = (int(bits[5]) + int(bits[6]) + int(bits[7])) % 2

p0 = (int(bits[1]) + int(bits[2]) + int(bits[3]) + int(bits[4]) + int(bits[5]) + int(bits[6]) + int(bits[7])) % 2

if str(p1) == bits[1]:

err\_code = err\_code + '0'

msg = 'correct'

else:

err\_code = err\_code + '1'

msg = 'incorrect'

print('p1: b3+b5+b7 = ', bits[3], '+', bits[5], '+', bits[7], '=', p1, ' ', msg)

if str(p2) == bits[2]:

err\_code = err\_code + '0'

msg = 'correct'

else:

err\_code = err\_code + '1'

msg = 'incorrect'

print('p2: b3+b6+b7 = ', bits[3], '+', bits[6], '+', bits[7], '=', p2, ' ', msg)

if str(p3) == bits[4]:

err\_code = err\_code + '0'

msg = 'correct'

else:

err\_code = err\_code + '1'

msg = 'incorrect'

print('p3: b5+b6+b7 = ', bits[5], '+', bits[6], '+', bits[7], '=', p3, ' ', msg)

if str(p0) == bits[0]:

err\_code = err\_code + '0'

msg = 'correct'

else:

err\_code = err\_code + '1'

msg = 'incorrect'

print('p0: b1+b2+b3+b4+b5+b6+b7 = ', bits[1], '+', bits[2], '+', bits[3], '+', bits[4], '+', bits[5], '+', bits[6], '+', bits[7], '=', p0, ' ', msg)

if err\_code == '0000':

print('No error.')

rbits = str(bits[3]) + str(bits[5]) + str(bits[6]) + str(bits[7])

return rbits

elif err\_code == '1011':

print('Error in position 5')

fbits = flip(5, bits)

print('Corrected bitstring: ', fbits)

rbits = str(fbits[3]) + str(fbits[5]) + str(fbits[6]) + str(fbits[7])

return rbits

elif err\_code == '1101':

print('Error in position 3')

fbits = flip(3, bits)

print('Corrected bitstring: ', fbits)

rbits = str(fbits[3]) + str(fbits[5]) + str(fbits[6]) + str(fbits[7])

return rbits

elif err\_code == '0111':

print('Error in position 6')

fbits = flip(6, bits)

print('Corrected bitstring: ', fbits)

rbits = str(fbits[3]) + str(fbits[5]) + str(fbits[6]) + str(fbits[7])

return rbits

elif err\_code == '1111':

print('Error in position 7')

fbits = flip(7, bits)

print('Corrected bitstring: ', fbits)

rbits = str(fbits[3]) + str(fbits[5]) + str(fbits[6]) + str(fbits[7])

return rbits

else:

print('Error in the parity bit.')

rbits = str(bits[3]) + str(bits[5]) + str(bits[6]) + str(bits[7])

return rbits

hd2 = ''

for bits in err\_blocks:

print()

rbits = HammingDecode2(bits)

print('Decoded bitstring: ', rbits, '.')

hd2 = hd2 + str(rbits)

print()

print('Decoded sequence for Hamming code(7,4):')

print(hd2)

doc = open('text.txt')

txt = doc.read()

print('Sequence from assignment 3:')

print(txt)

#HammingDecode2('11111111')

doc = open('out.txt', 'w')

#doc.write(hd2)

doc.close()