Calum Brown - Assignment 3

In this assignment I wrote one section of code in python on repl.it
The code has its comments to help explain it but I will talk about it aswell

Sidenotes:

text.txt is the CHARTER OF FUNDAMENTAL RIGHTS OF THE EUROPEAN UNION

I use math and random python scripts in this code
To run this code you need to put this code into repl python and a the text.txt file created with the CHARTER OF FUNDAMENTAL RIGHTS OF THE EUROPEAN UNION in it

THE CODE

```
import math
import random


file = open("text.txt", "r")
file_encoder = open("encoded.txt", "a")

file_encoder.truncate(0)

asciiDict = {i: chr(i) for i in range(128)}
codeWordDict = []


x = 'a'
file.seek(0)


distance = 0

error_rate = .01 #the error rate of the channel



i = 0
z = 0
f = 0
i2 = 0

#this part here is not neccessary but i over complicated the encoding
#instead of having the binary value 0 and making it 000 for all 8 bits of the
Ascii code
# i did this
```

```
# afterwards all code words are 3 bits apart but only require 12 bits becasue
their order is different
for i in range(len(asciiDict)):
#for i in range(1):
  f = 0
  while (f < len(codeWordDict)):
    new_codeword = bin(round(z))[2:].zfill(12)
    #print(new_codeword)

    #to calculate the distance between new word and all words
    for i2 in range(12):
      #compare each bit
      if ((int((codeWordDict[f])[i2]) != int(new_codeword[i2]))):
        distance += 1

    #if distance is valid then compare with next codeword
    if (distance >= 3):
      f = f + 1
    #if its not increment the new_codeword and start again
    else:
      f = 0
      z = z + 1
    distance = 0
  #adds new codeword to dictionary
  codeWordDict.append(bin(round(z))[2:].zfill(12))

print('the codeword dictionary: ', codeWordDict)


i = 0

while(1):

  character = file.read(1) #reads the next character

  if character != "": #continues if we are not at the end of the txt file

    #converts character to Ascii index
    Ascii_index = ord(character)
    if (Ascii_index <= 128):

      AsciiCodeWord = codeWordDict[Ascii_index]

      for i in range(12):
        #introduce random noise for
        #can add noise for every bit in word
        if(random.randint(1,(1/error_rate) == (1/error_rate)):
          if(AsciiCodeWord[i] == "1"):
            AsciiCodeWord_list = list(AsciiCodeWord)
```

```
            AsciiCodeWord_list[i] = "0"

            AsciiCodeWord = "".join(AsciiCodeWord_list)

          else:
            AsciiCodeWord_list = list(AsciiCodeWord)
            AsciiCodeWord_list[i] = "1"

            AsciiCodeWord = "".join(AsciiCodeWord_list)

      #prints encoded bit to encoded.txt
      #then sends word with new noise or no noise
      file_encoder.write(AsciiCodeWord)
    else:
      #for words that are not in ascii we send this which represets ctrl-@
      #this is so when we calcualate the error rate at then end the files are
of the same size, so we have to send something
      file_encoder.write("000000000000")

  else:
    #here we are at the end of the file and therefore need to exit the loop
which reads it
    break

file.close()


file_encoder.close()
file_encoder = open("encoded.txt", "r")

file_decoded = open("decoded.txt", "a")
file_decoded.truncate(0)

x = 'a'
file_encoder.seek(0)



checker = True
x = ""
i2 = 0
while(checker):
  #reads next codeword which we know is 12 bits
  for i in range(12):
    x = x + file_encoder.read(1)

  if (x in codeWordDict):

    file_decoded.write(asciiDict[codeWordDict.index(x)])
```

```
    x = ""
  else:
    if (x == ""):
      break
    min_distance = 12
    f = 0
    #print(x)
    closestWord = ""

    while (f < len(codeWordDict)):

    #to calculate the distance between new word and all words
      for i2 in range(12):
      #compare each bit
        if (int((codeWordDict[f])[i2]) != int(x[i2])):
          distance += 1

    #if distance is valid then compare with next codeword
      if (distance < min_distance):
        closestWord = codeWordDict[f]
        min_distance = distance

      f += 1
      distance = 0

    #if it now any valid codeword, post the responding codeword
    if (closestWord in codeWordDict):
      file_decoded.write(asciiDict[codeWordDict.index(closestWord)])
    else:
      #to show there was an error
      file_decoded.write(asciiDict[0])


    if(x == ""):
      checker = False
    x = ""


file_decoded.close()

#to read the file and not fix any errors to calulate how noisy it really was

file_undecoded = open("undecoded.txt", "a")
file_undecoded.truncate(0)

x = 'a'
file_encoder.seek(0)

checker = True
```

```
x = ""
i2 = 0
while(checker):
  #reads next codeword which we know is 12 bits
  for i in range(12):
    x = x + file_encoder.read(1)

  if (x in codeWordDict):

    file_undecoded.write(asciiDict[codeWordDict.index(x)])
    x = ""
  else:
    #since read codeword is not in our dictionary print ctrl-@, so when we
compare with text.txt it can detect the wrong character has been printed
    file_undecoded.write(asciiDict[0])

    if(x == ""):
      checker = False
    x = ""


file_undecoded.close()

file_encoder.close()


file = open("text.txt", "r")
file_decoded = open("decoded.txt", "r")
totalCorrectChar = 0
totalChar = 0

#this section calculates the accuracy of our error correction
while(1):

  character = file.read(1) #reads the next character
  character_decoded = file_decoded.read(1)

  if ((character != "") or (character_decoded != "")): #continues if we are
not at the end of the txt file

    totalChar += 1
    #converts character to Ascii index
    if (character == character_decoded):
      totalCorrectChar += 1


  else:
    #here we are at the end of the file and therefore need to exit the loop
which reads it
```

```
    break

print('With error correction')
print('correct: ', totalCorrectChar)
print('total: ', totalChar)
print('percentage: ', (float(totalCorrectChar))/(float(totalChar))*100, '%')


file.seek(0)
file_undecoded = open("undecoded.txt", "r")
totalCorrectChar = 0
totalChar = 0

#this section calulcates the text if there was no error correction
while(1):

  character = file.read(1) #reads the next character
  character_decoded = file_undecoded.read(1)

  if ((character != "") or (character_decoded != "")): #continues if we are
not at the end of the txt file

    totalChar += 1
    #converts character to Ascii index
    if (character == character_decoded):
      totalCorrectChar += 1


  else:
    #here we are at the end of the file and therefore need to exit the loop
which reads it
    break

print('\nWith no error correction')
print('correct: ', totalCorrectChar)
print('total: ', totalChar)
print('percentage: ', (float(totalCorrectChar))/(float(totalChar))*100, '%')

file.close()
file_decoded.close()
file_undecoded.close()
```

THE CODE EXPLAINED

this segment of code creates a 12-bit code for every ascii character and each code is 3 distance from every other code, I know it was said that it could be 24 bits but I didn't fully read the q and overcomplicated it and that's why this is here, basically for every integer for the length of the ascii index it cycles through binary codewords till it finds one that is minimum 3 distance from every code word in the dictionary so far then moves onto the next one

Here is the dictionary

This segment of code of code encodes the whole text.txt

It reads each character and finds its unique codeword, then cycles through each bit and with a 1% chance flips that bit then sends the word, so character in the text.txt were not in ascii so when they were not found a replacement code of 00000000000 was sent which meant ctrl-@, this was used to help with the comparison of the error correction to no error correction

This part was to read the encoded file and then decode it and print the result into decoded.txt. this segment included error correction which is just a simple translation of what was taught in class, the one with minimum distance from the received codeword was chosen, and if it couldn't be solved i.e. couldn't find a close codeword then it was assumed an error

This segment of code is the same as before except we don't use error correction, this is just a reference to see how good our error correction was

This is to see how accurate the error correction code was relating to the original text
These are the results, the ones that couldn't be changed back were either not in the ascii dictionary or had more than one error

```
With error correction
correct:  24095
total:  24217
percentage:  99.49622166246851 %
```

And this is to see how accurate the decoder was with no error correction
These are the results

```
With no error correction
correct:  21481
total:  24218
percentage:  88.69848872739284 %
```

As you can see with the error correction the code was over 10% more similar to the original than the with no error correction.

The error correction code was 99.5% accurate where on average with an error rate of .01 there is an error in 1 out of 10 codewords since my codewords consist of 12 bits

Other results
error_rate = .1

```
With error correction
correct:  7848
total:  24217
percentage:  32.406986827435276 %

With no error correction
correct:  6958
total:  24218
percentage:  28.730696176397718 %
```

Even though this rate is bad it is still better than no error correction and it matches up with the probability of success with error rate being .1, which is 31.4% success rate

error_rate = .0001

```
With error correction
correct:  24212
total:  24217
percentage:  99.97935334682248 %

With no error correction
correct:  24181
total:  24218
percentage:  99.84722107523329 %
```

As expected the match is very high, the oddities in the code are more than likely the characters that are not in ascii code and not errors in the channel

So as expected the higher the error rate the less likely the code is going to be corrected since it is more likely to have more than one bit error

Side note: the higher the error_rate the longer it took for the code to compute since it had to check more codewords