Reliable Messaging

Group 2:
Masaki Kagesawa,
Xiao Bai

01

Introduction

What is Reliable Messaging?

- Guarantee successful transmission of the messages across an unreliable infrastructure
- Or Data Storage
- Error detection and correction

Why we need it?

- We human make mistakes, computers make mistakes too!
- But computer MUST BE 100% accurate at all times, and it’s hard to guarantee that especially when dealing with billions of data

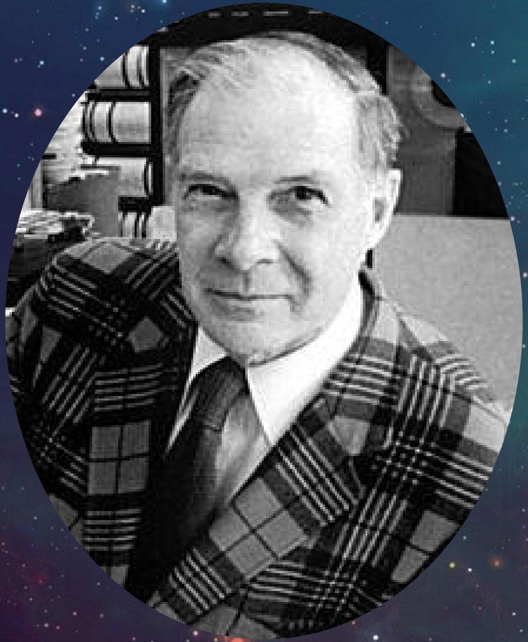
Storing Data

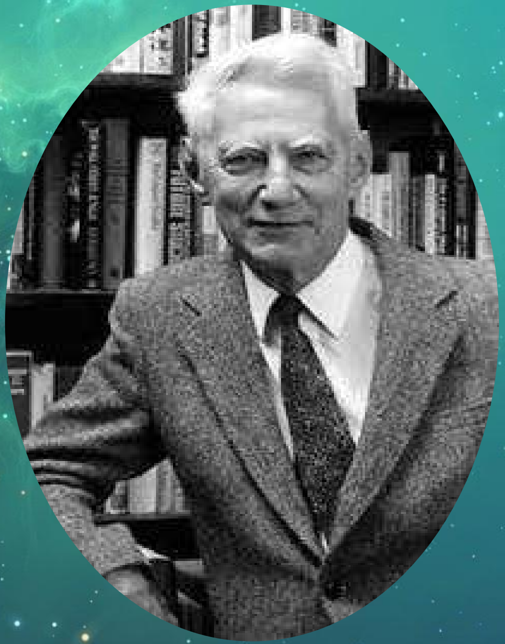
- Vulnerable to physical damage
- Could be misread

Transmitting Data

- Prone to channel noise, distortions, fluctuations and interference all the time

Coding and Information Theory





Richard Wesley Hamming

- Mathematics, Computer Engineering, Telecommunication
- “Hamming Code”
- Turing Award(1968)

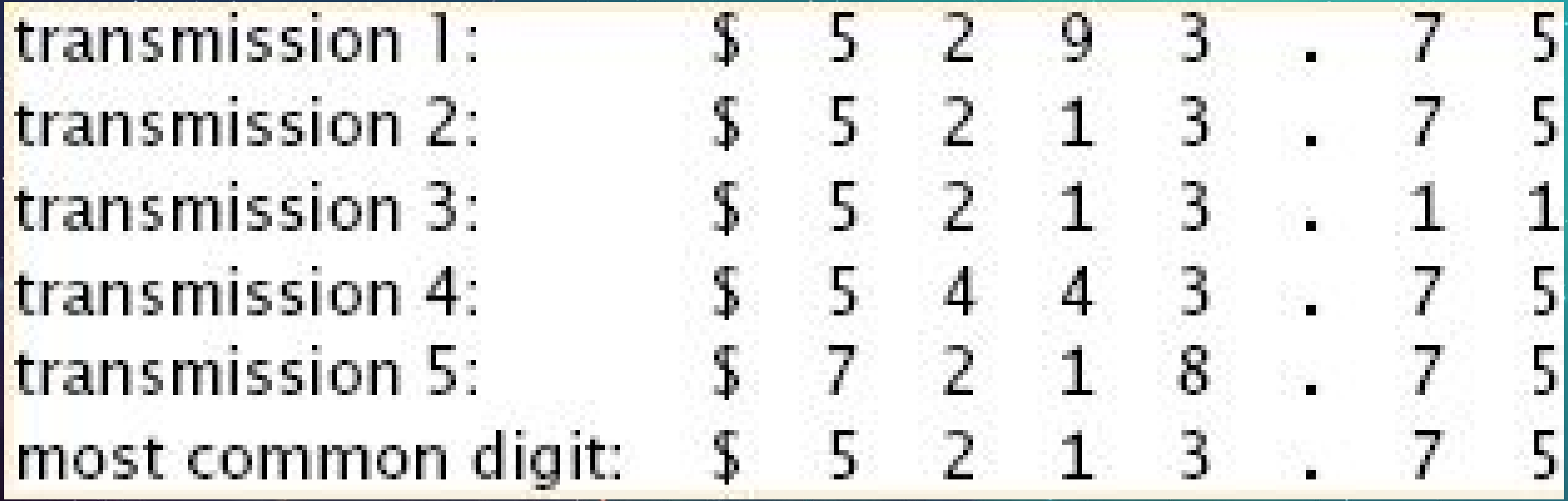
Claude Elwood Shannon

- founding father of modern information theory
- 《A Mathematical Theory of Communication》 (1948)

02

Algorithms

Algorithm #1: Repetition



transmission 1:	$	5	2	9	3	.	7	5
transmission 2:	$	5	2	1	3	.	7	5
transmission 3:	$	5	2	1	3	.	1	1
transmission 4:	$	5	4	4	3	.	7	5
transmission 5:	$	7	2	1	8	.	7	5
most common digit:	$	5	2	1	3	.	7	5

- "Guess": Send the same message over and over enough and choose each letter with the highest frequency
- Condition: data is small; random error(not malicious error)
- Drawback: cannot handle heavy data, too much overhead cost

Algorithm #2: Redundancy

Data: 532.65

Send: five_three_two_point_six_five

Corrupted: fiva_thoee_twe_poimt_six_fave

- Transform to a longer message and conform to some Known Patterns(“Code Words”)
- Smaller overhead costs
- Drawback: must agree on Codewords first between receiver and sender; high chance of undetected error

Algorithm #3: Checksum

Simple Checksum

Message: 5435

5+4+3+5 = 17
Send: 54357

Staircase Checksum

Message: 5435

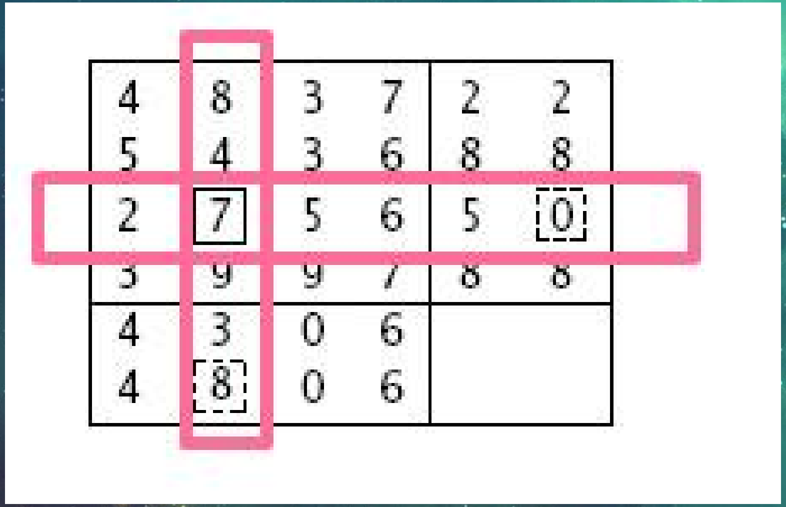
1*5+2*4+3*3+4*5 = 42
Send: 543572

- add sum(/weighted sum) of all digits to the end of the message
- Limitation: Only error detection, no auto correction (must combined with repetition trick)
- Simple Checksum: detect one-digit error only; Staircase Checksum: detect two_digit error only

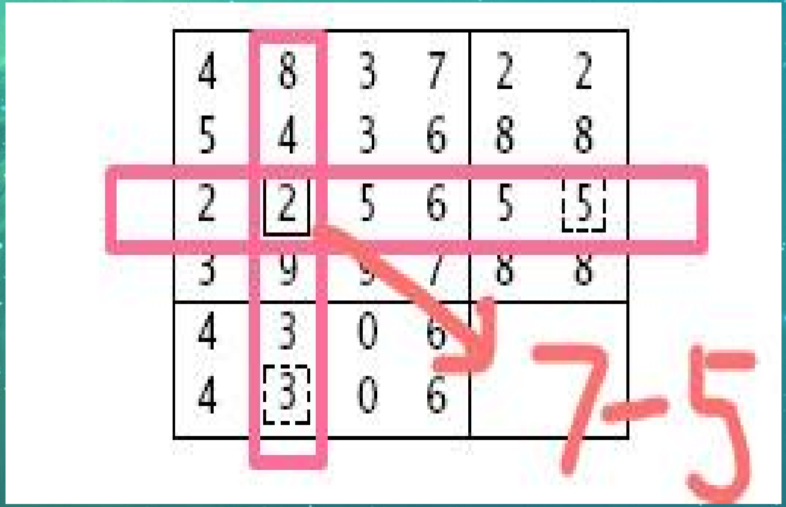
Algorithm #4: Two-dimensional Parity(Bingo!)

message: 54249634

Send: [5, 4, 2, 4, 15]
[9, 6, 3, 4, 22]
[14, 10, 6, 8, 37]



4	8	3	7	2	2
5	4	3	6	8	8
2	7	5	6	5	0
3	9	9	7	8	8
4	3	0	6		
4	8	0	6		



4	8	3	7	2	2
5	4	3	6	8	8
2	2	5	6	5	5
3	9	9	7	8	8
4	3	0	6		
4	3	0	6		

- Both error detection and auto error correction
- One-digit corruption
- Matrix storing data and checksum; Pin down the exact corrupted digit by corrupted row and column; Correct it according to original checksum information

03

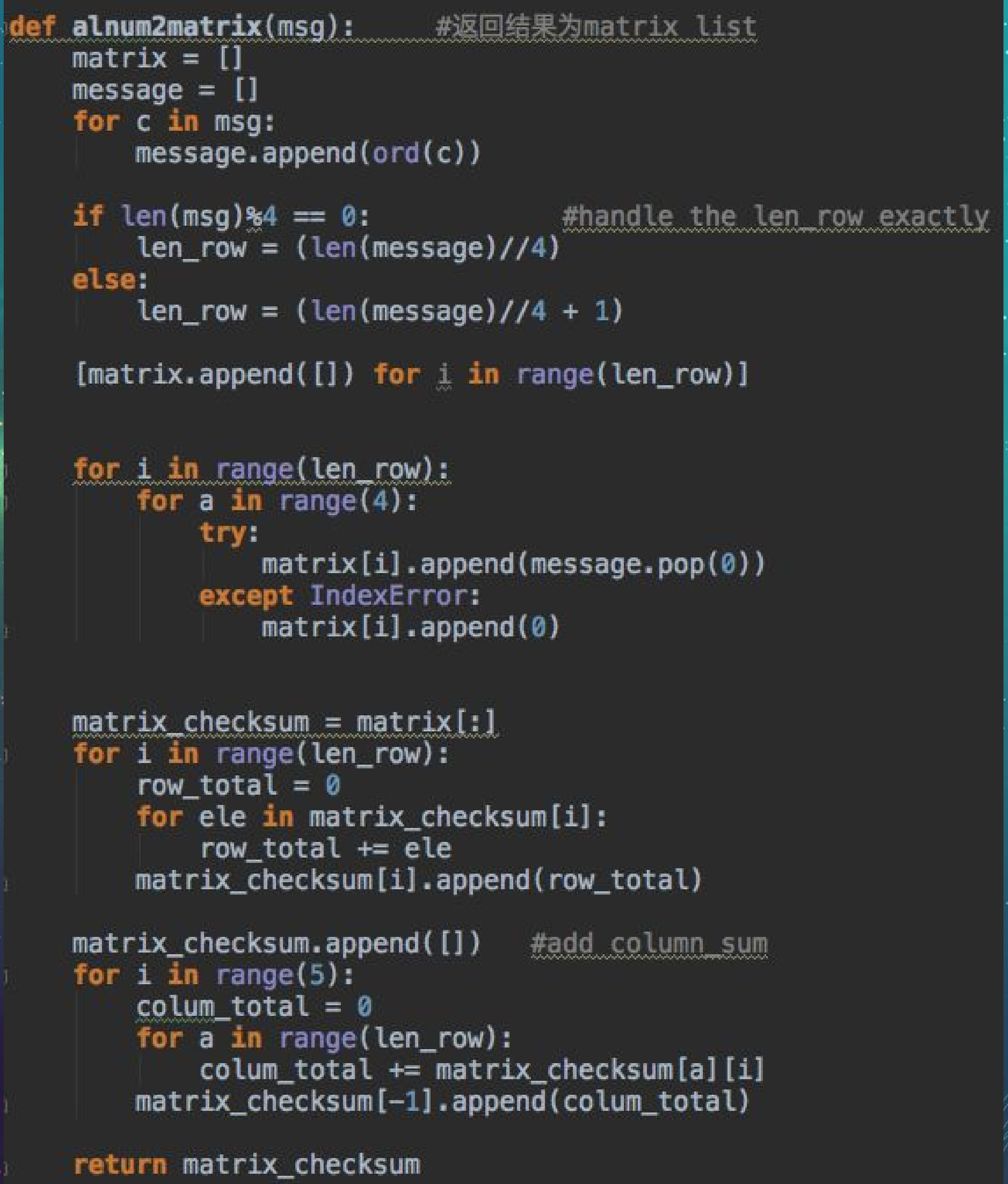
Our Implementation

Matrix.py

- alnum2matrix(msg)
- matrix_list2matrix_string(matrix)
- matrix_string2matrix_list(mat_string)
- randomize_one_digit(matrix)
- error_correction(matrix)
- matrix2alnum(matrix)

Function: convert the original message(string) to matrix(list) conforming to the ASCII code, and add the "checksum row"(last row at bottom) and "checksum column"(last column at right)

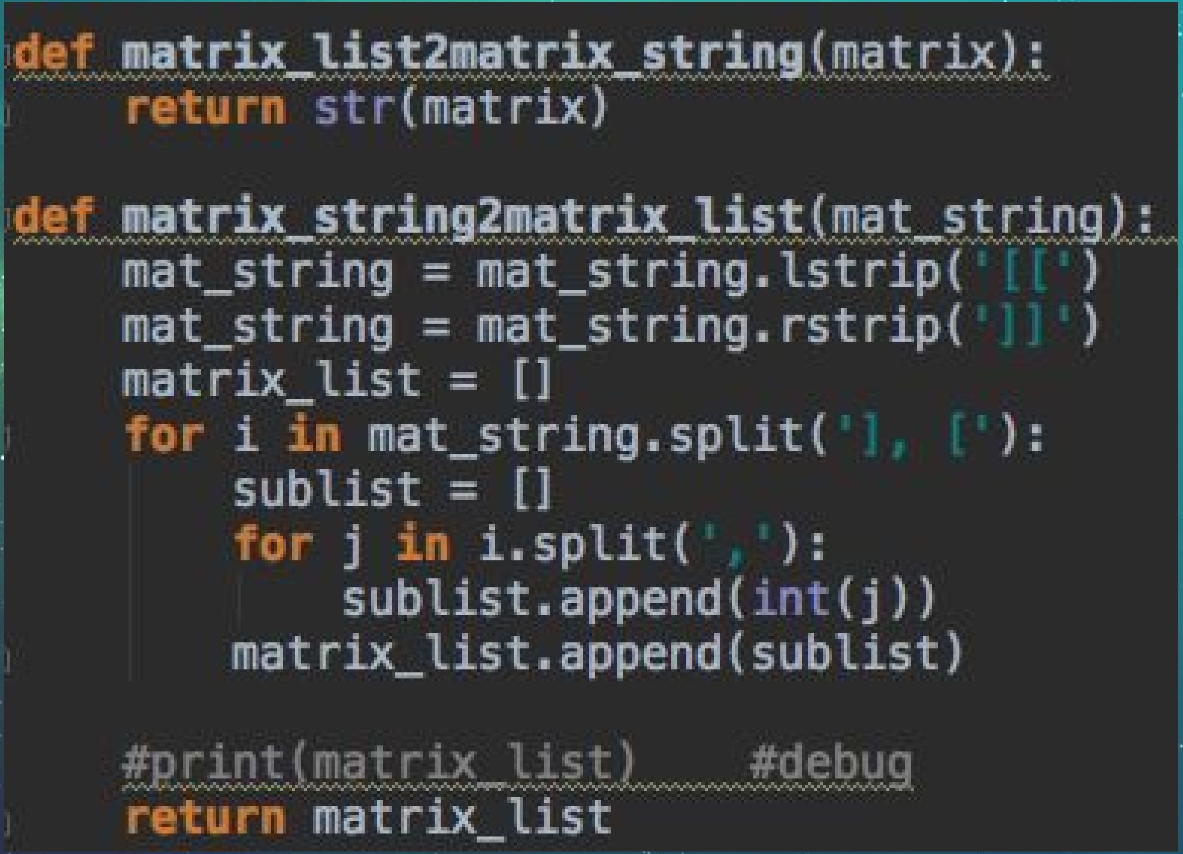
- Four column per row
- the total row number depends on message length



```
def alnum2matrix(msg):      #返回结果为matrix list
    matrix = []
    message = []
    for c in msg:
        message.append(ord(c))

    if len(msg)%4 == 0:                 #handle the len_row exactly
        len_row = (len(message)//4)
    else:
        len_row = (len(message)//4 + 1)

    [matrix.append([]) for i in range(len_row)]


    for i in range(len_row):
        for a in range(4):
            try:
                matrix[i].append(message.pop(0))
            except IndexError:
                matrix[i].append(0)


    matrix_checksum = matrix[:]
    for i in range(len_row):
        row_total = 0
        for ele in matrix_checksum[i]:
            row_total += ele
        matrix_checksum[i].append(row_total)

    matrix_checksum.append([])   #add column sum
    for i in range(5):
        colum_total = 0
        for a in range(len_row):
            colum_total += matrix_checksum[a][i]
        matrix_checksum[-1].append(colum_total)

    return matrix_checksum
```

Matrix.py

- alnum2matrix(msg)
- matrix_list2matrix_string(matrix)
- matrix_string2matrix_list(mat_string)
- randomize_one_digit(matrix)
- error_correction(matrix)
- matrix2alnum(matrix)

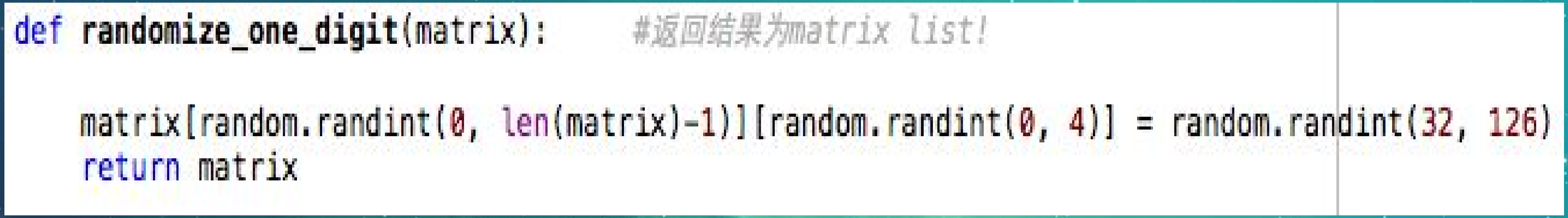


```
def matrix_list2matrix_string(matrix):
    return str(matrix)

def matrix_string2matrix_list(mat_string):
    mat_string = mat_string.lstrip('[[')
    mat_string = mat_string.rstrip(']]')
    matrix_list = []
    for i in mat_string.split('], ['):
        sublist = []
        for j in i.split(', '):
            sublist.append(int(j))
        matrix_list.append(sublist)

    #print(matrix_list)      #debug
    return matrix_list
```

Function: Adjust type in order to send and receive appropriately

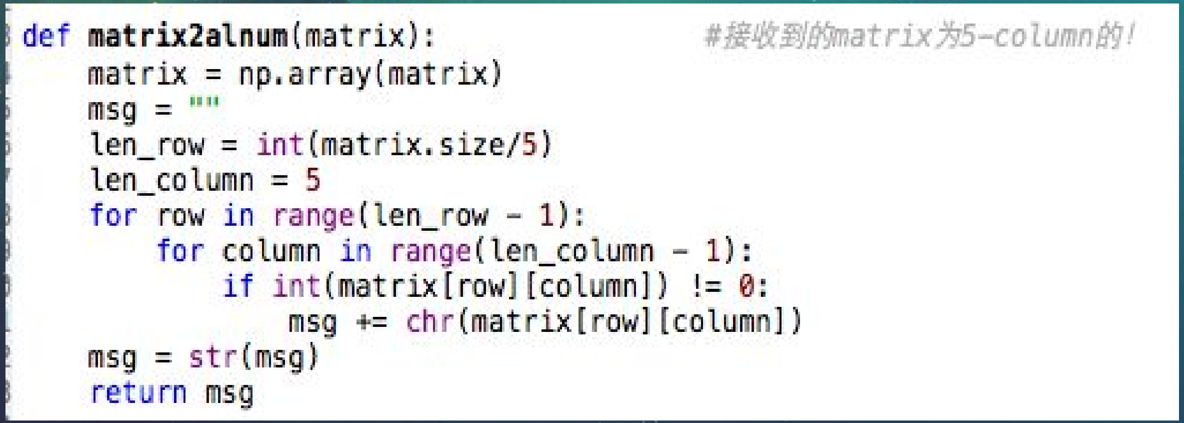
- before send: make sure matrix becomes a string type
- after receive: convert string type back to matrix(list) for dealing with correction next step

Matrix.py



```
def randomize_one_digit(matrix):      #返回结果为matrix list!

    matrix[random.randint(0, len(matrix)-1)][random.randint(0, 4)] = random.randint(32, 126)
    return matrix
```

- alnum2matrix(msg)
- matrix_list2matrix_string(matrix)
- matrix_string2matrix_list(mat_string)
- randomize_one_digit(matrix)
- error_correction(matrix)
- matrix2alnum(matrix)



```
def matrix2alnum(matrix):                    #接收到的matrix为5-column的!
    matrix = np.array(matrix)
    msg = ""
    len_row = int(matrix.size/5)
    len_column = 5
    for row in range(len_row - 1):
        for column in range(len_column - 1):
            if int(matrix[row][column]) != 0:
                msg += chr(matrix[row][column])
    msg = str(msg)
    return msg
```

randomize_one_digit(matrix): make a random error on one random digit

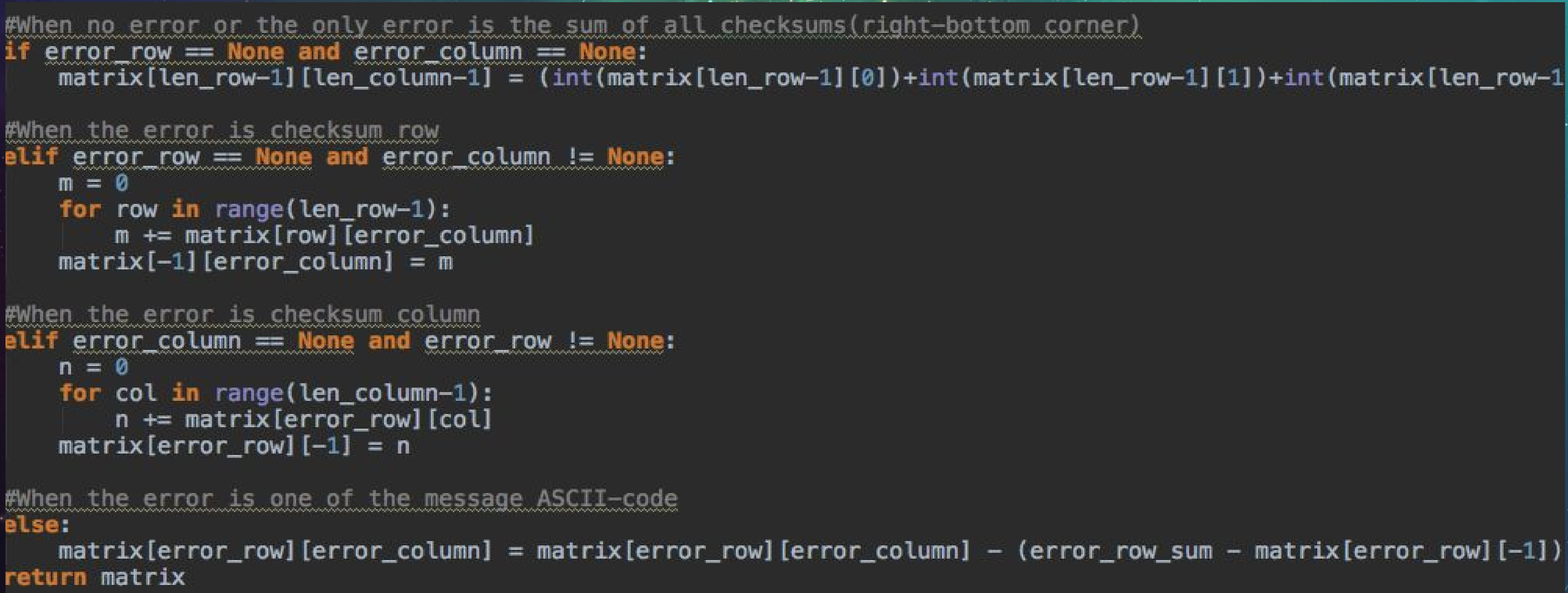
matrix2alnum(matrix): last step, change the corrected matrix back to message conforming to ASCII code

Matrix.py

Function: find out the corrupted digit, and correct it according to accompanying information

- if any digit of the message corrupted: use checksum information to correct it
- if checksum row or column corrupted: use original message digits to correct it
- if no error or total checksum(bottom right) corrupted: use original checksum row/column to correct it if necessary

- alnum2matrix(msg)
- matrix_list2matrix_string(matrix)
- matrix_string2matrix_list(mat_string)
- randomize_one_digit(matrix)
- error_correction(matrix)
- matrix2alnum(matrix)

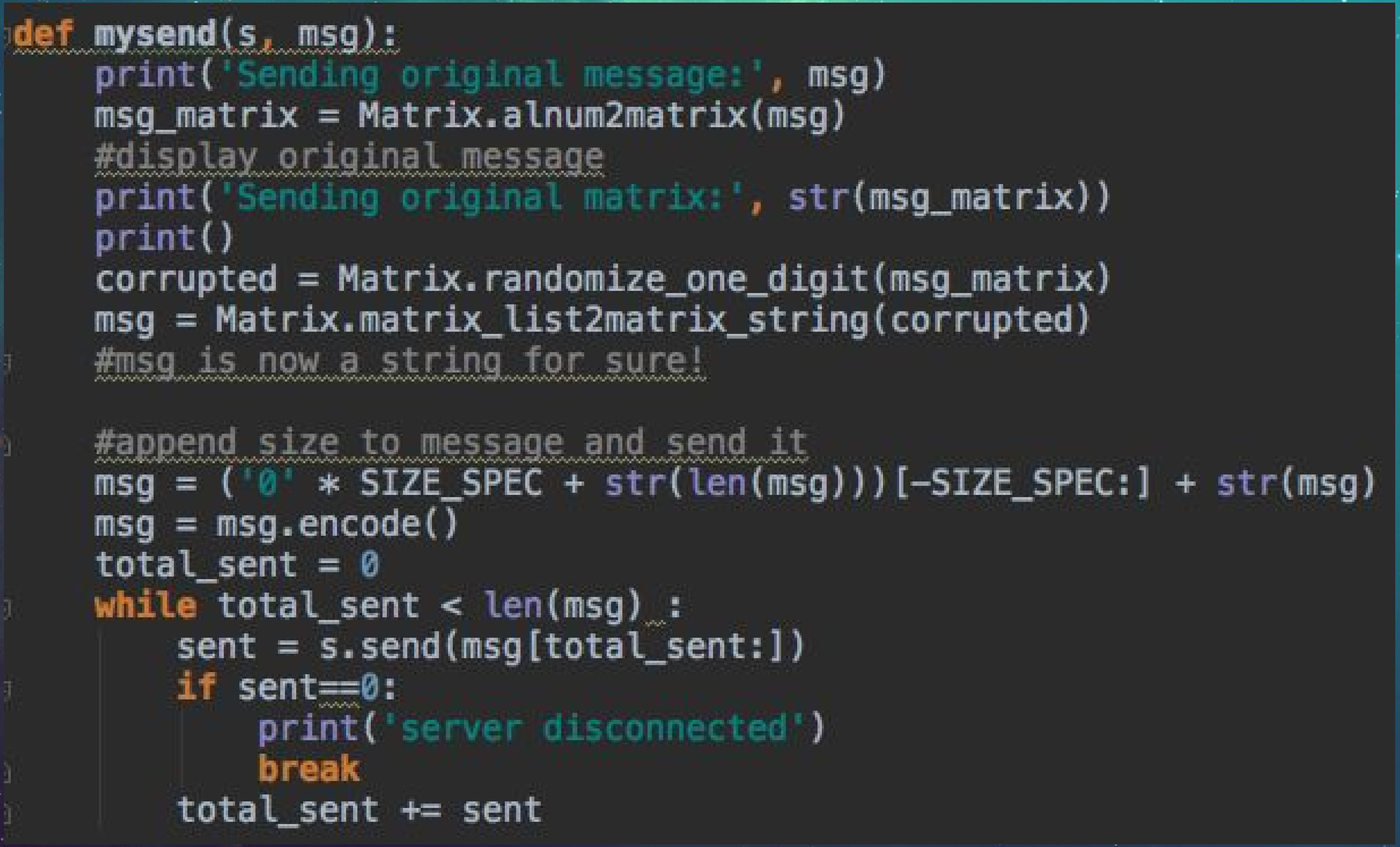


```
#When no error or the only error is the sum of all checksums(right-bottom corner)
if error_row == None and error_column == None:
    matrix[len_row-1][len_column-1] = (int(matrix[len_row-1][0])+int(matrix[len_row-1][1])+int(matrix[len_row-1

#When the error is checksum row
elif error_row == None and error_column != None:
    m = 0
    for row in range(len_row-1):
        m += matrix[row][error_column]
    matrix[-1][error_column] = m

#When the error is checksum column
elif error_column == None and error_row != None:
    n = 0
    for col in range(len_column-1):
        n += matrix[error_row][col]
    matrix[error_row][-1] = n

#When the error is one of the message ASCII-code
else:
    matrix[error_row][error_column] = matrix[error_row][error_column] - (error_row_sum - matrix[error_row][-1])
return matrix
```

Chat_utils.py

- mysend (s, msg)
- myrecv (s)

❖ EACH TIME A MESSAGE IS SENT: make a corruption to the matrix(generated from original message) and send it as a string



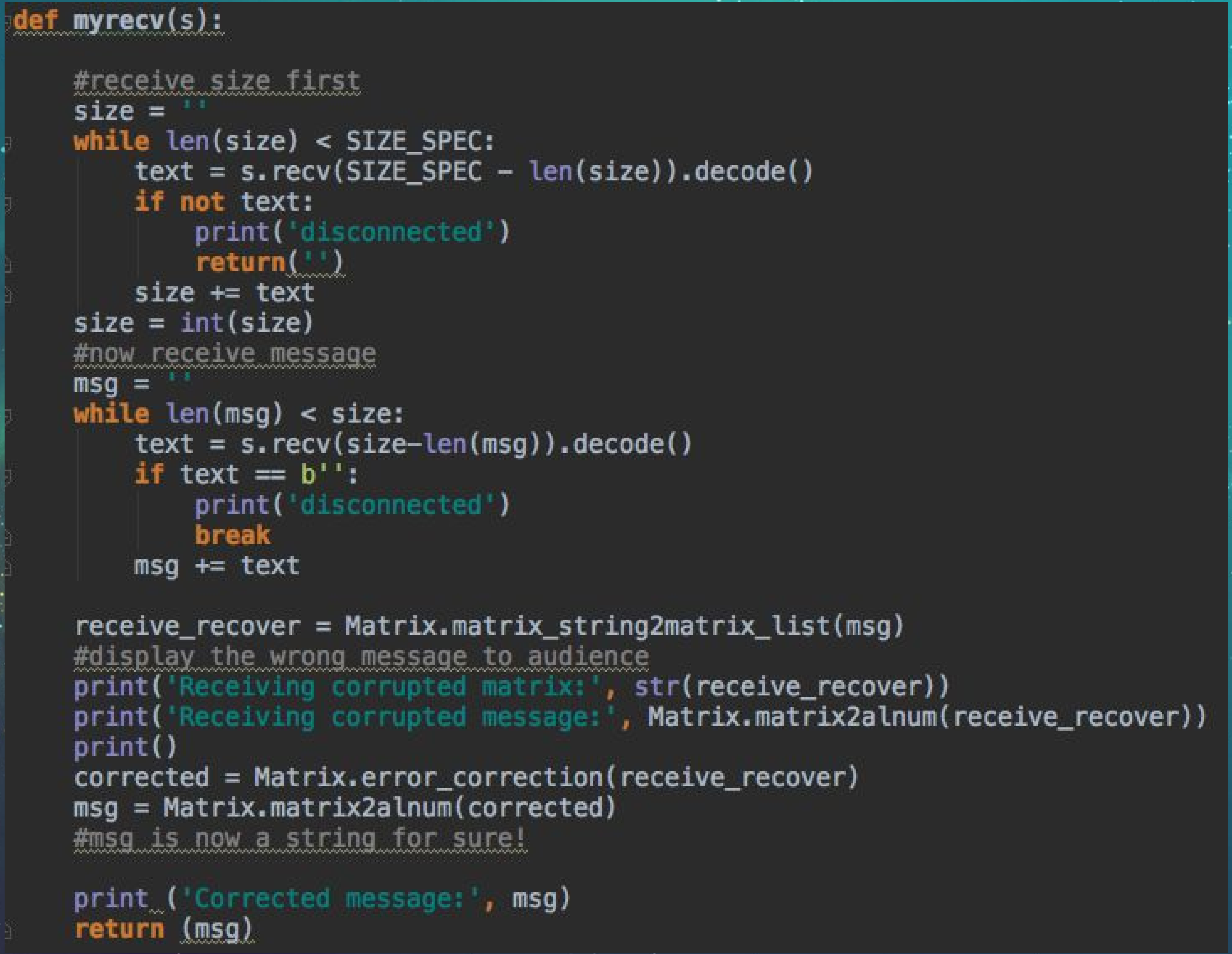
```
def mysend(s, msg):
    print('Sending original message:', msg)
    msg_matrix = Matrix.alnum2matrix(msg)
    #display original message
    print('Sending original matrix:', str(msg_matrix))
    print()
    corrupted = Matrix.randomize_one_digit(msg_matrix)
    msg = Matrix.matrix_list2matrix_string(corrupted)
    #msg is now a string for sure!

    #append size to message and send it
    msg = ('0' * SIZE_SPEC + str(len(msg)))[-SIZE_SPEC:] + str(msg)
    msg = msg.encode()
    total_sent = 0
    while total_sent < len(msg) :
        sent = s.send(msg[total_sent:])
        if sent==0:
            print('server disconnected')
            break
        total_sent += sent
```

Chat_utils.py

- mysend (s, msg)
- myrecv (s)

❖ **EACH TIME A MESSAGE IS RECEIVED: convert received string back to matrix, and detect and correct the error, return the result as message string**

[It should be the exactly the same as the original message sent!]



```python
def myrecv(s):

    #receive size first
    size = ''
    while len(size) < SIZE_SPEC:
        text = s.recv(SIZE_SPEC - len(size)).decode()
        if not text:
            print('disconnected')
            return('')
        size += text
    size = int(size)
    #now receive message
    msg = ''
    while len(msg) < size:
        text = s.recv(size-len(msg)).decode()
        if text == b'':
            print('disconnected')
            break
        msg += text

    receive_recover = Matrix.matrix_string2matrix_list(msg)
    #display the wrong message to audience
    print('Receiving corrupted matrix:', str(receive_recover))
    print('Receiving corrupted message:', Matrix.matrix2alnum(receive_recover))
    print()
    corrected = Matrix.error_correction(receive_recover)
    msg = Matrix.matrix2alnum(corrected)
    #msg is now a string for sure!

    print ('Corrected message:', msg)
    return (msg)
```

04

Demo

05

Next Steps

- Implement for Data Storage
- Implement for Multimedia
- Increase Efficiency by testing different # of columns