Theory of Computation (COSE215) Assignment4 Report

이정민/컴퓨터학과/2023320060

Problem #1

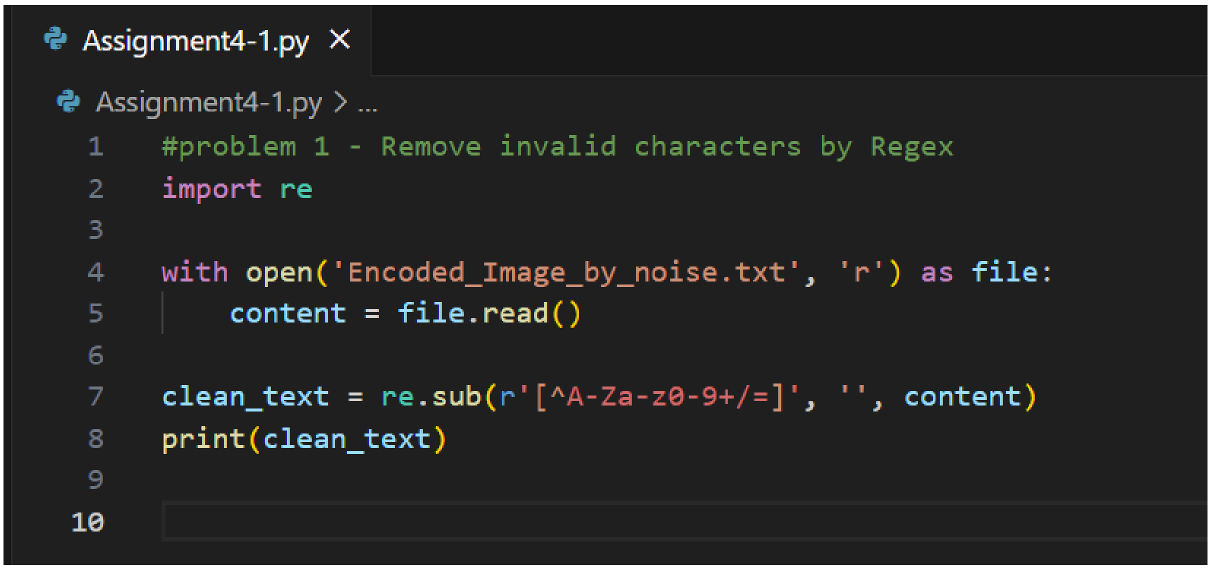
Problem analysis

The first problem involves taking a given string that contains invalid characters, extracting only the valid characters to form a Base64 sequence, and then converting it into a QR code.

Solving environment

To solve problem #1, I use Python3.

My code



Assignment4-1.py

Assignment4-1.py > ...

```python
#problem 1 - Remove invalid characters by Regex
import re

with open('Encoded_Image_by_noise.txt', 'r') as file:
    content = file.read()

clean_text = re.sub(r'[^A-Za-z0-9+/=]', '', content)
print(clean_text)


```

Algorithm

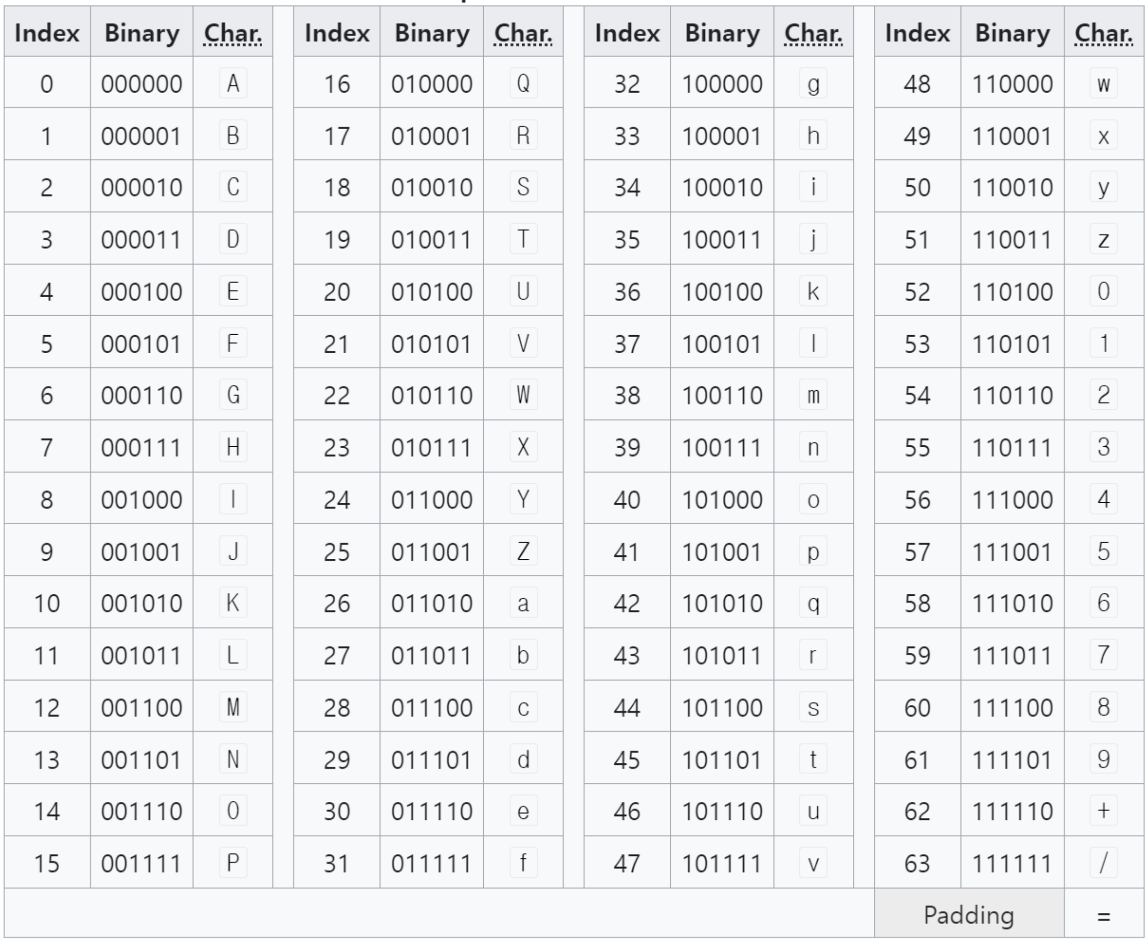
1. To handle regular expressions, import the `re` module.
2. Open the file containing invalid characters in read mode and store the read string in the variable `content`.
3. Use the `sub` method to replace parts of the string that match a pattern, thereby removing characters that are not valid.

Regular Expression

To design Regular Expression, we need to know about Base64.

According to the provided link, the Base64 encoding scheme only uses the following 64 special characters.

Base64 alphabet defined in RFC 4648.



Index	Binary	Char.	Index	Binary	Char.	Index	Binary	Char.	Index	Binary	Char.
0	000000	A	16	010000	Q	32	100000	g	48	110000	w
1	000001	B	17	010001	R	33	100001	h	49	110001	x
2	000010	C	18	010010	S	34	100010	i	50	110010	y
3	000011	D	19	010011	T	35	100011	j	51	110011	z
4	000100	E	20	010100	U	36	100100	k	52	110100	0
5	000101	F	21	010101	V	37	100101	l	53	110101	1
6	000110	G	22	010110	W	38	100110	m	54	110110	2
7	000111	H	23	010111	X	39	100111	n	55	110111	3
8	001000	I	24	011000	Y	40	101000	o	56	111000	4
9	001001	J	25	011001	Z	41	101001	p	57	111001	5
10	001010	K	26	011010	a	42	101010	q	58	111010	6
11	001011	L	27	011011	b	43	101011	r	59	111011	7
12	001100	M	28	011100	c	44	101100	s	60	111100	8
13	001101	N	29	011101	d	45	101101	t	61	111101	9
14	001110	O	30	011110	e	46	101110	u	62	111110	+
15	001111	P	31	011111	f	47	101111	v	63	111111	/
									Padding		=

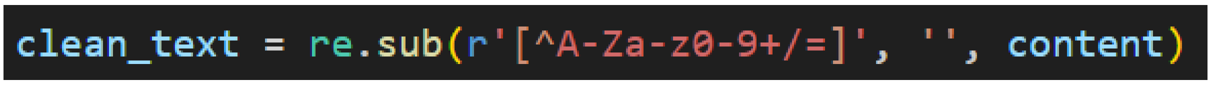
Therefore, we need to remove all characters from the given string that are not among the above 64 characters, as these are considered invalid.

So, I make a RE for problem #1.

[A-Za-z0-9+/=]

This expression indicates that a valid string can only consist of the characters A-Z, a-z, +, /, and =.

Using this expression, the code would be implemented as follows:



```
clean_text = re.sub(r'[^A-Za-z0-9+/=]', '', content)
```

r'[^A-Za-z0-9+/=]'

This sequence uses the caret symbol (^) to designate a pattern that excludes the characters that follow it. Therefore, through this code, we can remove all invalid characters except for A-Z, a-z, +, /, and =.

Result

Problem #2

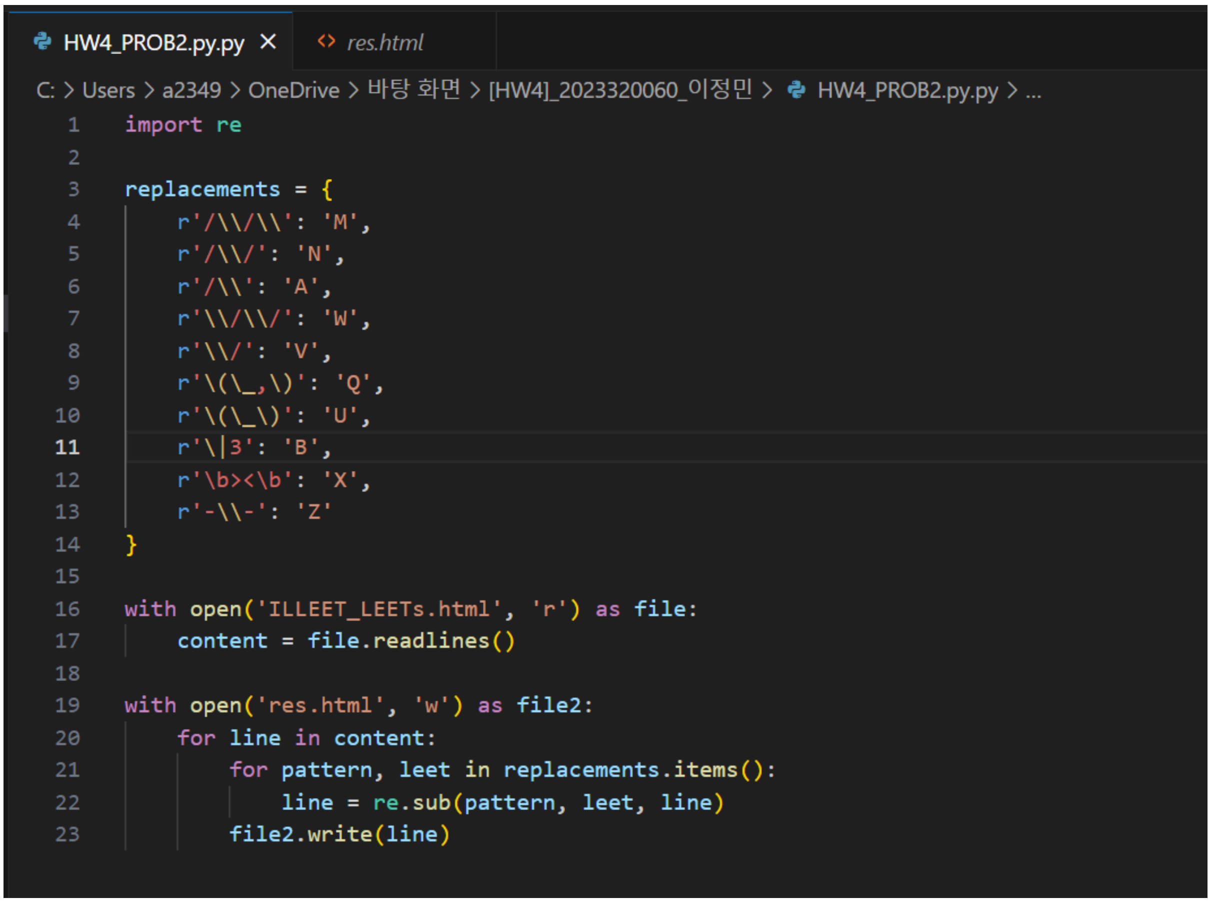
Problem analysis

The second problem involves an HTML file where some of the code has been converted into Leet speak. The task is to translate the Leet speak back to regular text to ensure the HTML file functions correctly. The replacement character set is provided, and each element corresponds to a specific character.

Solving environment

To solve problem #2, I use Python3.

My code


HW4_PROB2.py.py
res.html
C: > Users > a2349 > OneDrive > 바탕 화면 > [HW4]_2023320060_이정민 > HW4_PROB2.py.py > ...

```python
import re

replacements = {
    r'/\\/\\': 'M',
    r'/\\/': 'N',
    r'/\\': 'A',
    r'\\/\\/': 'W',
    r'\\/': 'V',
    r'\(\_,\)': 'Q',
    r'\(\_\)': 'U',
    r'\|3': 'B',
    r'\b><\b': 'X',
    r'-\\-': 'Z'
}

with open('ILLEET_LEETs.html', 'r') as file:
    content = file.readlines()

with open('res.html', 'w') as file2:
    for line in content:
        for pattern, leet in replacements.items():
            line = re.sub(pattern, leet, line)
        file2.write(line)
```

Algorithm

1. To handle regular expressions, import the `re` module.
2. Given the replacement character set, declare a dictionary using regular expressions, where each element corresponds to a specific character. Be sure to appropriately use escape characters when defining this dictionary.
3. Open the HTML file that has been altered with Leet speak in read mode and store the code in the `content` variable. The `readlines()` function is used to save each line as an element in a list, which facilitates smooth operation of the code.
4. To prevent modifications to the original file, declare an empty HTML file and open it in write mode. This ensures that any changes are saved to the new file, leaving the original file unchanged.
5. Iterate through each element in the 'content' list, retrieve the regular expression and its corresponding element from the previously declared 'Replace char set' dictionary, and convert the Leet speak characters to the correct characters. Then, write the converted code to the empty HTML file.

Key Points in Algorithm Design

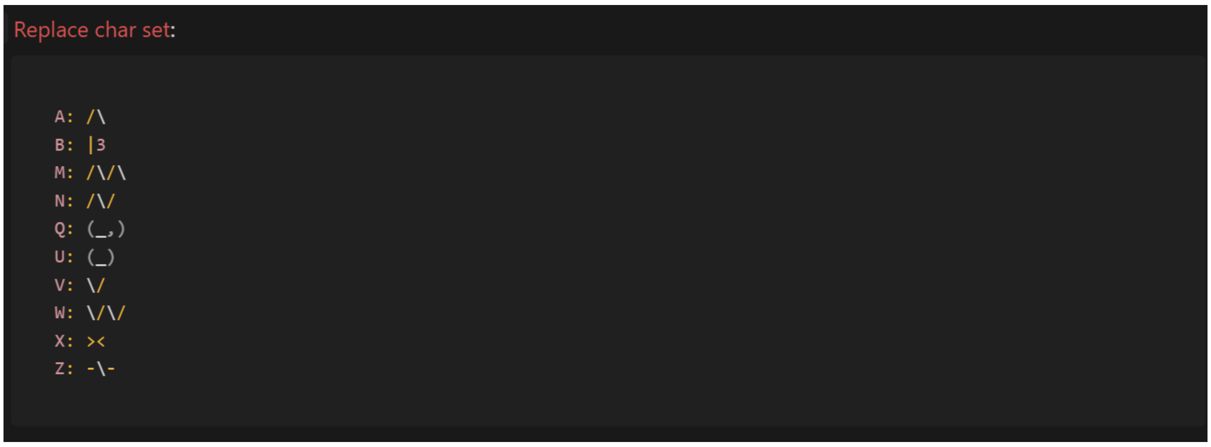
First, it's important to be aware that some of the regular expressions in the Replace char set may be subsets of other elements' regular expressions.

Additionally, we must be careful not to alter the HTML tags. The possibility of altering the tags in an HTML file can be caused by the character X. If two tags are adjacent within a single line of code (for example: `<div id="key"></div>`), failure to properly handle this in the regular expression could result in parts of the HTML tags being changed to X, causing the code to not function correctly. That's why I ended up using `₩b` in the regular expression.

Lastly, modifying the original file can lead to numerous problems such as Data Loss, Corruption, Security Risks, etc.

Regular Expression

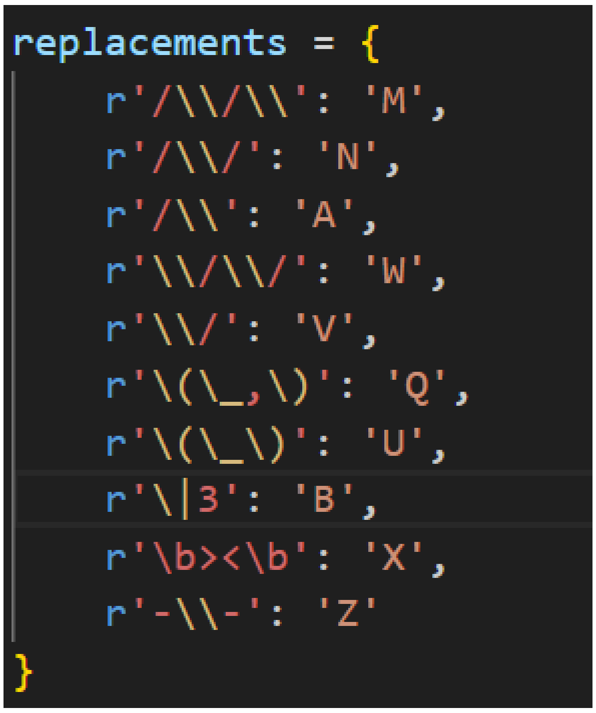
The given Replace char set is as follows.



Replace char set:

```
A: /\
B: |3
M: /\/\
N: /\/
Q: (_,)
U: (_)
V: \/
W: \/\/
X: ><
Z: -\-
```

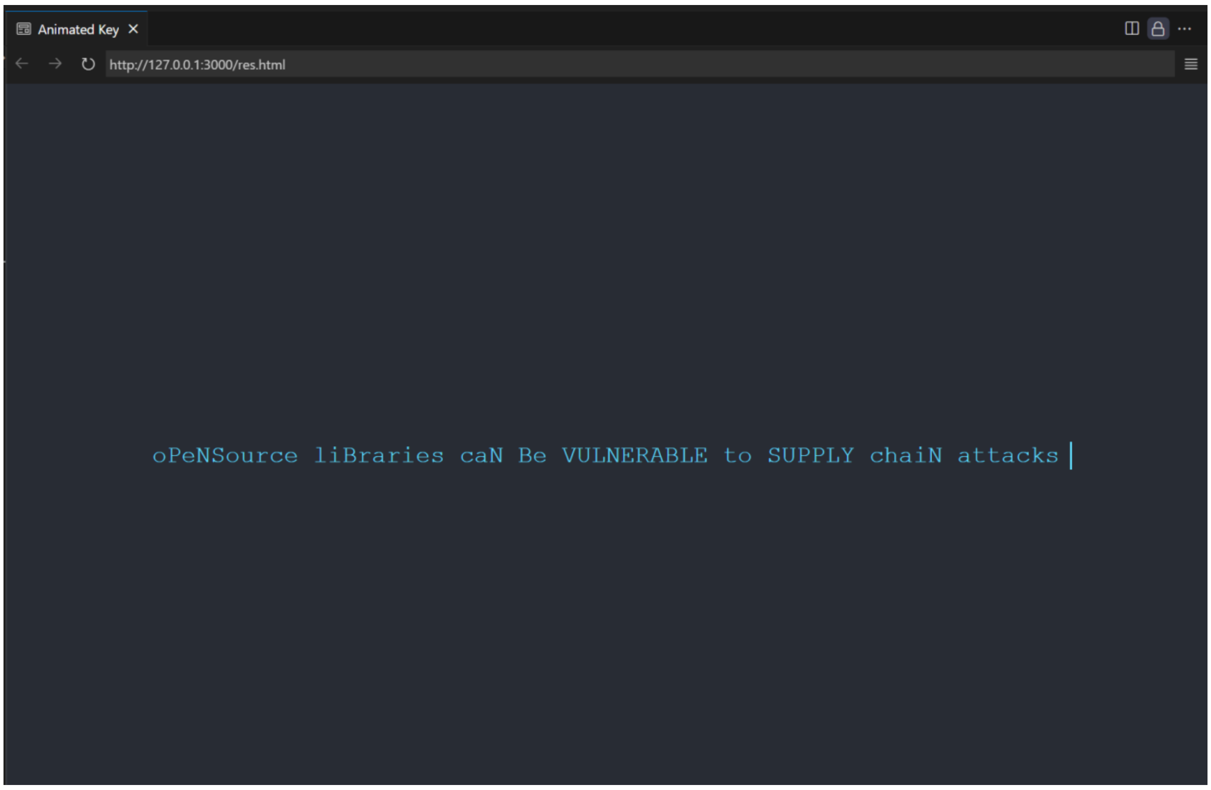
As mentioned in "Key Points in Algorithm Design," I will demonstrate the regular expressions for each character, paying attention to their order.



```
replacements = {
    r'/\\/\\': 'M',
    r'/\\/': 'N',
    r'/\\': 'A',
    r'\\/\\/': 'W',
    r'\\/': 'V',
    r'\(\_,\)': 'Q',
    r'\(\_\)': 'U',
    r'\|3': 'B',
    r'\b><\b': 'X',
    r'-\\-': 'Z'
}
```

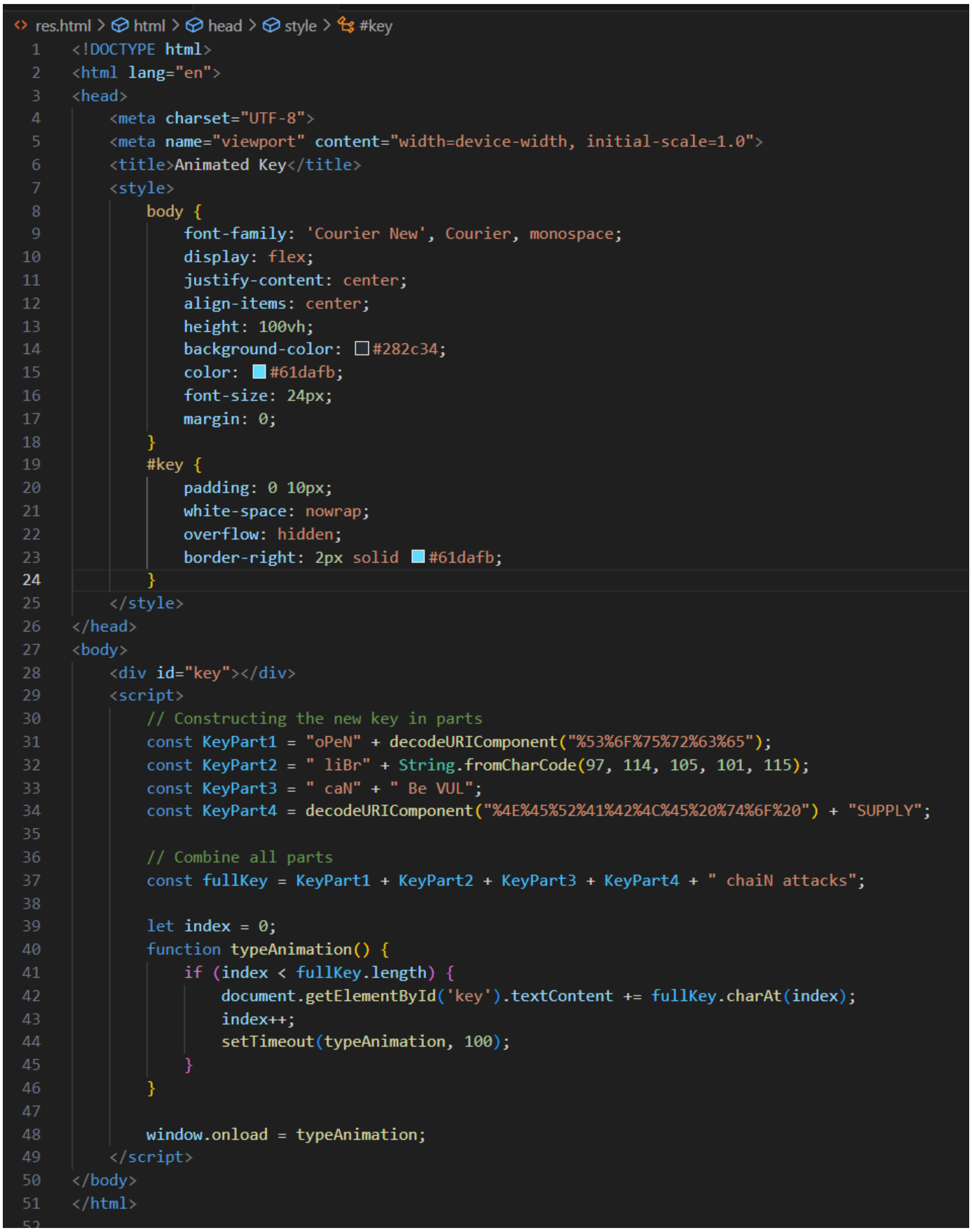
For problem 2, since the entire set of character correspondences is already provided, we just need to construct regular expressions to match each character accordingly.

Result



```
oPeNSource liBraries caN Be VULNERABLE to SUPPLY chaiN attacks
```

If we correctly convert the HTML file in problem 2, we can obtain the password key for the file in problem 3 as described above.



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Animated Key</title>
    <style>
        body {
            font-family: 'Courier New', Courier, monospace;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            background-color: #282c34;
            color: #61dafb;
            font-size: 24px;
            margin: 0;
        }
        #key {
            padding: 0 10px;
            white-space: nowrap;
            overflow: hidden;
            border-right: 2px solid #61dafb;
        }
    </style>
</head>
<body>
    <div id="key"></div>
    <script>
        // Constructing the new key in parts
        const KeyPart1 = "oPeN" + decodeURIComponent("%53%6F%75%72%63%65");
        const KeyPart2 = " liBr" + String.fromCharCode(97, 114, 105, 101, 115);
        const KeyPart3 = " caN" + " Be VUL";
        const KeyPart4 = decodeURIComponent("%4E%45%52%41%42%4C%45%20%74%6F%20") + "SUPPLY";

        // Combine all parts
        const fullKey = KeyPart1 + KeyPart2 + KeyPart3 + KeyPart4 + " chaiN attacks";

        let index = 0;
        function typeAnimation() {
            if (index < fullKey.length) {
                document.getElementById('key').textContent += fullKey.charAt(index);
                index++;
                setTimeout(typeAnimation, 100);
            }
        }

        window.onload = typeAnimation;
    </script>
</body>
</html>
```

Problem #3

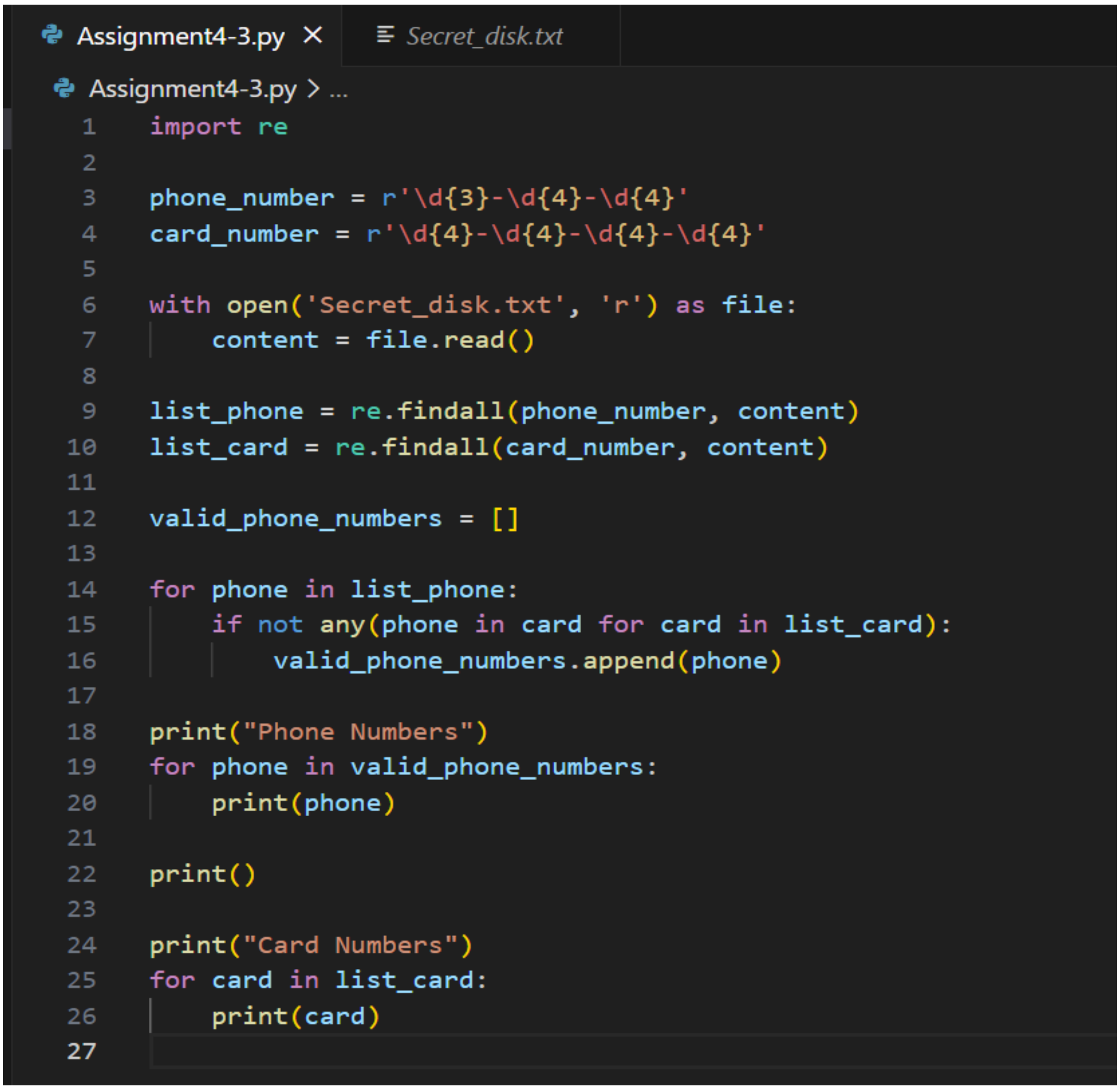
Problem analysis

Problem 3 assumes that the disk recovery process did not complete properly, resulting in invalid characters being inserted between valid pieces of information. The goal is to extract and print valid information such as phone numbers and card numbers from the given `Secret disk.txt` file.

Solving environment

To solve problem #3, I use Python3.

My code



```python
import re

phone_number = r'\d{3}-\d{4}-\d{4}'
card_number = r'\d{4}-\d{4}-\d{4}-\d{4}'

with open('Secret_disk.txt', 'r') as file:
    content = file.read()

list_phone = re.findall(phone_number, content)
list_card = re.findall(card_number, content)

valid_phone_numbers = []

for phone in list_phone:
    if not any(phone in card for card in list_card):
        valid_phone_numbers.append(phone)

print("Phone Numbers")
for phone in valid_phone_numbers:
    print(phone)

print()

print("Card Numbers")
for card in list_card:
    print(card)

```

Algorithm

1. To handle regular expressions, import the `re` module.
2. Define the regular expressions for phone numbers and card numbers. Note that phone numbers should be in the format of three digits, a hyphen, four digits, a hyphen, and four more digits; card numbers should be in the format of four groups of four digits separated by hyphens.
3. Open the given `Secret_disk.txt` file in read mode and store its contents in the `content` variable.
4. Use the findall method from the re module to retrieve only the strings that match the given regular expressions from the content variable and store them in lists.
5. Since card number substrings can be included in phone numbers, declare an empty list to store only valid phone numbers that are not substrings of card numbers.
6. To extract only valid phone numbers that are not included in any card numbers, iterate through the existing phone number list and add only phone numbers that are not substrings of any card numbers to the `valid_phone_numbers` list.
   While it's possible to simply check conditions and remove items directly from the original list in this process, removing items from the original list can lead to unintended results. Therefore, I declared an empty list and used a filtering method instead.
7. Print the valid phone number and card number information according to the provided output format.

Regular Expression

The criteria for valid information as presented in the problem are as follows:

Phone Numbers:

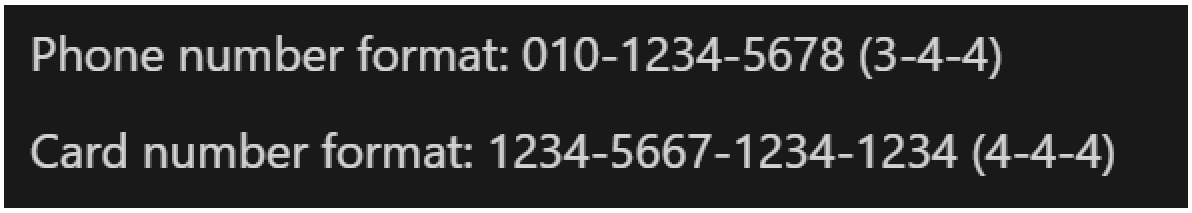
Must follow the pattern: xxx-xxxx-xxxx

Example: 123-4567-8901

Card Numbers:

Must follow the pattern: xxxx-xxxx-xxxx-xxxx

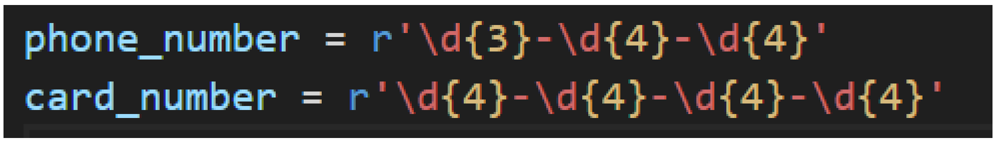
Example: 1234-5678-9012-3456



Phone number format: 010-1234-5678 (3-4-4)

Card number format: 1234-5667-1234-1234 (4-4-4)

The regular expressions that meet these criteria are as follows.



```
phone_number = r'\d{3}-\d{4}-\d{4}'
card_number = r'\d{4}-\d{4}-\d{4}-\d{4}'
```

Explanation: of phone number.

₩d{3}: Matches exactly three digits, representing the area code (e.g., 123).

-: Matches a hyphen.

₩d{4}: Matches exactly four digits, representing the central office code (e.g., 4567).

-: Matches another hyphen.

₩d{4}: Matches exactly four more digits, representing the line number (e.g., 8901).

Explanation: of card number.

₩d{4}: Matches exactly four digits, representing the first group in the credit card number.

-: Matches a hyphen.

₩d{4}: Matches another four digits, representing the second group in the credit card number.

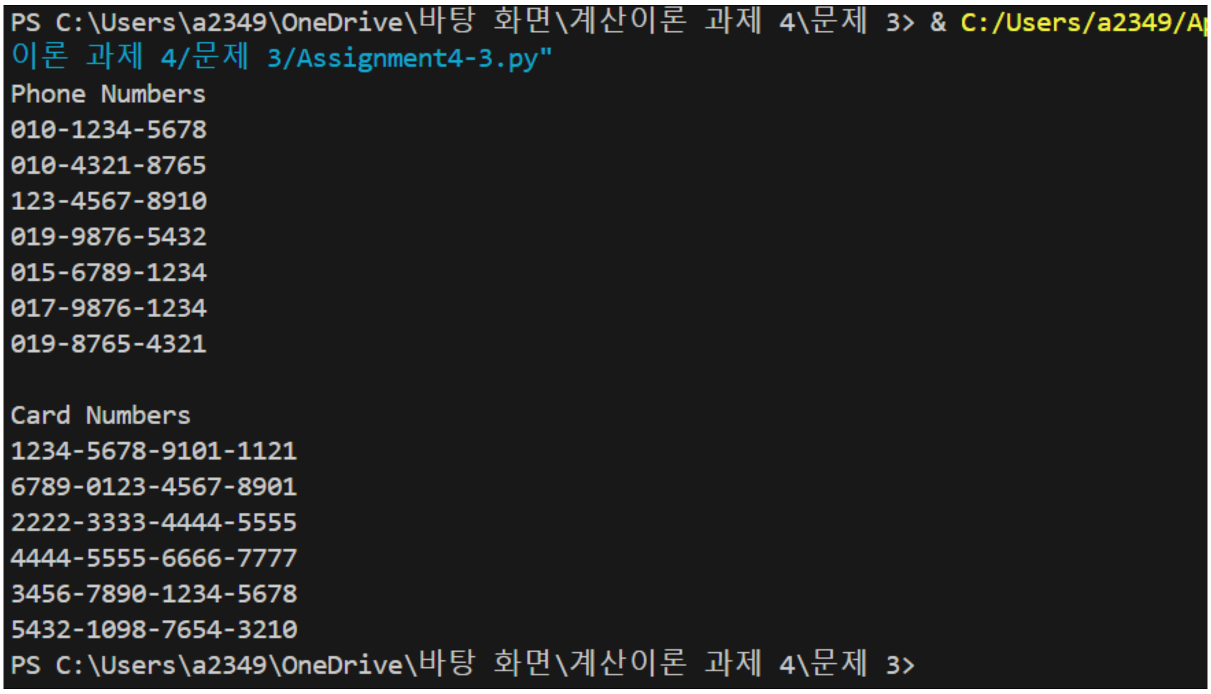
-: Matches another hyphen.

₩d{4}: Matches another four digits, representing the third group in the credit card number.

-: Matches a hyphen.

₩d{4}: Matches the final four digits, representing the fourth group in the credit card number.

Result



```
PS C:\Users\a2349\OneDrive\바탕 화면\계산이론 과제 4\문제 3> & C:/Users/a2349/A
이론 과제 4/문제 3/Assignment4-3.py"
Phone Numbers
010-1234-5678
010-4321-8765
123-4567-8910
019-9876-5432
015-6789-1234
017-9876-1234
019-8765-4321

Card Numbers
1234-5678-9101-1121
6789-0123-4567-8901
2222-3333-4444-5555
4444-5555-6666-7777
3456-7890-1234-5678
5432-1098-7654-3210
PS C:\Users\a2349\OneDrive\바탕 화면\계산이론 과제 4\문제 3>
```