FİLE ORGANIZATION TERM PROJECT

Name: Mustafa
Surname: Çirci
Number: 1306210018

1-) Introduction

This project aims to develop a system that enables fast searching on a large password dataset. The system stores and searches for passwords efficiently using techniques hashing and alphabetical indexing. Python os library was used for

2-) Tools and Technologies

The code of the project was written using Python 3.10.2. Os, hashlib, shutil, string and time libraries were used in the code file.

```
1   import os
2   import shutil
3   import hashlib
4   import string
5   import time
```

- From the os library os.makedir(), os.path.exists(), os.path.join(), os.walk() functions were used for file operations.
- To move “Unprocessed-Passwords” files to “Processed” file, shutil libraries shutil.move() function were used.

- The Shutil.move() function of the Shutil library was used to move the “Unprocessed-Passwords” files to the “Processed” file.
- To hash passwords to MD5, Sha128, Sha256 hash format, hashlib library’s hashlib.md5(), hashlib.sha1(), hashlib.sha256() functions were used.
- To take ascii letters and digits, string library were used.
- To calculate time of process, time library’s time() function were used.

3-) Project Design and Configuration

I added the files in the Common-Credentials folder that we downloaded from GitHub Repository to the Unprocessed-Password folder. The files in this file are moved to the folder named "Processed" after being passed through the python code in the "Code" file. The passwords in the files being processed are divided into files, each consisting of characters, under the folder named "Index". The first letter of the password determines which folder the password will be saved under. The password is also saved in the relevant .txt file under this folder. Passwords starting with unknown characters are saved in a txt file in the "Other" folder under the "Index" folder.

4-) Indexing Process

Some of the functions I will use in the indexing process are given below.

```
7  def create_folder_if_not_exists(folder_path):
8      if not os.path.exists(folder_path):
9          os.makedirs(folder_path)
10
11 def get_valid_filename(name):
12     valid_chars = "-_.() %s%s" % (string.ascii_letters, string.digits)
13     return ''.join(c for c in name if c in valid_chars)
14
15 def get_index_folder_name(char):
16     if char.islower():
17         return char.lower()
18     elif char.isupper():
19         return char.upper() + "_"
20     elif char.isdigit():
21         return char
22     else:
23         return "Other"
```

Picture 4.1

- With the “create_folder_if_not_exists()” function, if there is no file with the given file name, it is created.
- With the “get_valid_filename()” function, it retrieves valid chars from the string library and returns them.
- With the “get_index_folder_name()” function, it is decided whether the first letter of the given password is uppercase or lowercase, a number or a symbol. For file types with symbols, “Other” is returned and saved in the file named “Other”. The reason for adding "_" after the uppercase letter is that uppercase and lowercase filenames are considered the same when creating a file.

The function I use in the indexing process is given below. In this function, I calculated the total function running time with the time module.(Picture 4.2)

```python
def process_passwords(processed_passwords):
    total_time = 0.0
    start_time = time.time()
    source_folder = "Unprocessed-Passwords"
    target_folder = "Index"
    create_folder_if_not_exists(target_folder)

    index_line_counts = {}

    for root, dirs, files in os.walk(source_folder):
        for file in files:
            file_path = os.path.join(root, file)

            with open(file_path, 'r', encoding='utf-8') as input_file:
                for line in input_file:
                    password = line.strip()
                    if password and password not in processed_passwords:
                        processed_passwords.add(password)
                        first_char = password[0]
                        target_subfolder = os.path.join(target_folder, get_index_folder_name(first_char))
                        create_folder_if_not_exists(target_subfolder)

                        password_hash = hashlib.md5(password.encode()).hexdigest()
                        sha128_hash = hashlib.sha1(password.encode()).hexdigest()
                        sha256_hash = hashlib.sha256(password.encode()).hexdigest()
                        output_line = f"{password}|{password_hash}|{sha128_hash}|{sha256_hash}|{file}"

                        safe_filename = get_valid_filename(first_char)
                        global output_file_path

                        if first_char in index_line_counts:
                            index_line_counts[first_char] += 1
                        else:
                            index_line_counts[first_char] = 1

                        if index_line_counts[first_char] > 10000:
                            new_output_file_path = os.path.join(target_subfolder, f"output_{safe_filename}_{index_line_counts[first_char] // 10000}.txt")
                            output_file_path = new_output_file_path
```

```python
                        else:
                            output_file_path = os.path.join(target_subfolder, f"output_{safe_filename}.txt")

                        with open(output_file_path, 'a', encoding='utf-8') as output_file:
                            output_file.write(output_line + '\n')

            processed_file_path = os.path.join(target_folder, file)
            shutil.move(file_path, "Processed")

    end_time = time.time()
    search_time = end_time - start_time
    total_time += search_time
    print(f"Total index time: {total_time} seconds")
```

Picture 4.2

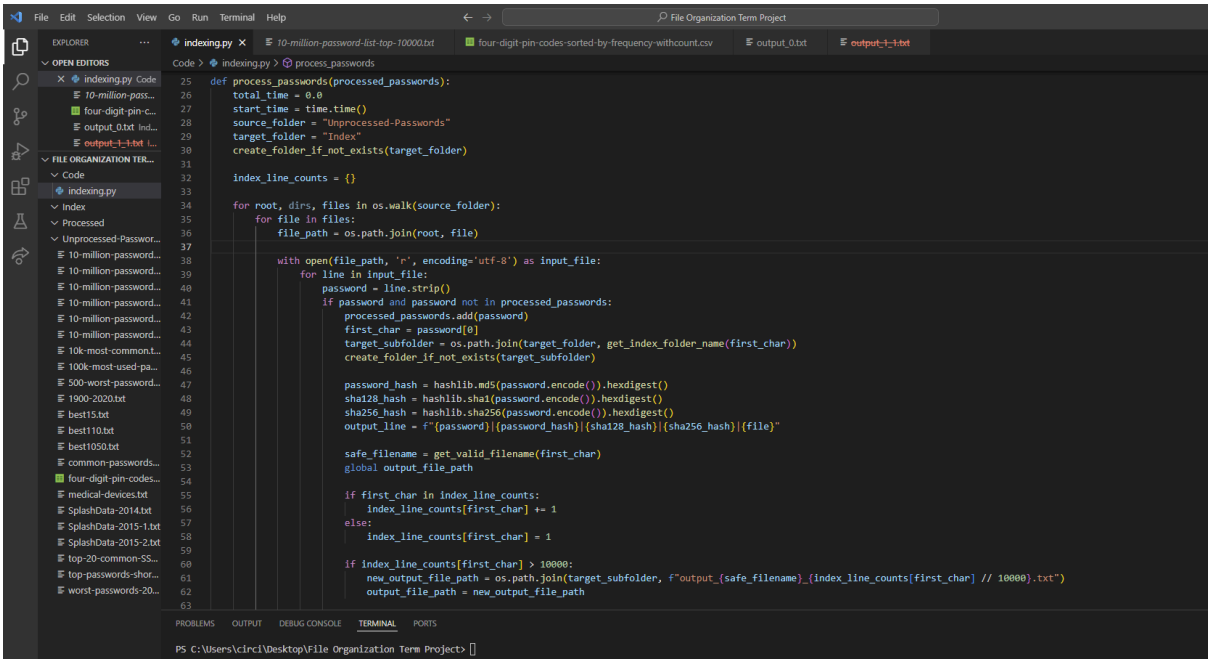
```python
            elif file.endswith(".csv"):
                with open(file_path, 'r', encoding='utf-8') as input_file:
                    reader = csv.reader(input_file)
                    for line in reader:
                        password = line[0].strip()
                        if password and password not in processed_passwords:
                            processed_passwords.add(password)  # Add password to the set
                            first_char = password[0]
                            target_subfolder = os.path.join(target_folder, get_index_folder_name(first_char))
                            create_folder_if_not_exists(target_subfolder)

                            password_hash = hashlib.md5(password.encode()).hexdigest()
                            sha128_hash = hashlib.sha1(password.encode()).hexdigest()
                            sha256_hash = hashlib.sha256(password.encode()).hexdigest()
                            output_line = f"{password},{password_hash},{sha128_hash},{sha256_hash},{file}"

                            safe_filename = get_valid_filename(first_char)
                            # Check if the index has reached the line limit
                            if first_char in index_line_counts:
                                index_line_counts[first_char] += 1
                            else:
                                index_line_counts[first_char] = 1

                            # Create a new output file if the line count exceeds 10,000
                            if index_line_counts[first_char] > 10000:
                                new_output_file_path = os.path.join(target_subfolder,
                                                                    f"output_{safe_filename}_{index_line_counts[first_char] // 10000}.txt")
                                output_file_path = new_output_file_path
                            else:
                                output_file_path = os.path.join(target_subfolder, f"output_{safe_filename}.txt")

                            with open(output_file_path, 'a', newline='', encoding='utf-8') as output_file:
                                writer = csv.writer(output_file)
                                writer.writerow(output_line.split(','))
```

Picture 4.3

First of all, since there was a file with the .csv extension among the files that needed to be processed, I wrote separate code for both .txt and .csv (Picture 4.3). However, since I realized that both files with .csv and .txt extensions can be read in a single piece of code, I preferred the other piece of code to increase performance and reduce the number of lines of code (Picture 4.2).

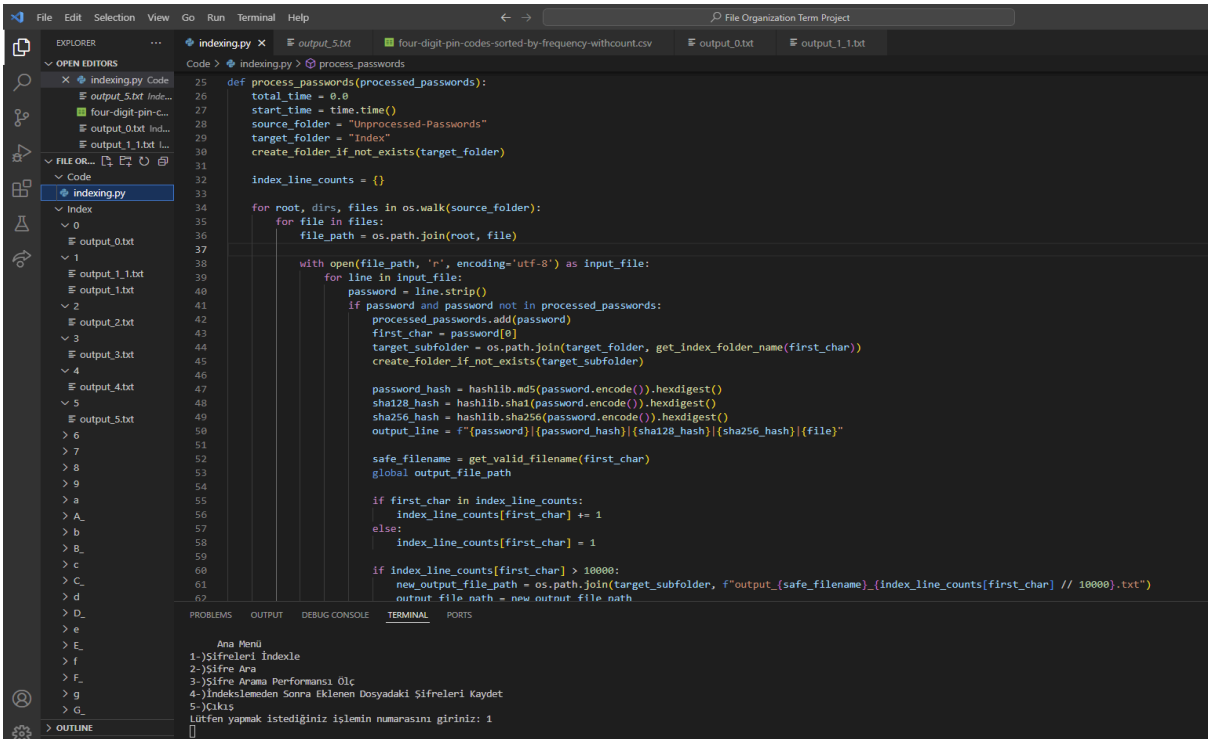
I created an empty dictionary named “index_line_counts” and kept track of whether the number of passwords in the txt file exceeded 10000 for each file name. Then, I used the loop to navigate through each file in the “Unprocessed-Password” folder. While taking the password in each line, I removed the spaces with the .strip() function. The passwords previously written to the files are saved with the processed_passwords set sent to the function as a parameter. Inside the function, it is checked whether the password in each line is in this set. If the password is not included in the set, the password is hashed in md5, sha128 and sha256 types with the help of hashlib. Password, hashed passwords and folder information containing the password are combined under a single string and made ready to be written to the file. While selecting the file where the password would be written, I used the "index_line_counts" dictionary to find the name of the txt file that was last written. After finding the file name, I opened the relevant file in writing mode and had it written to the file. When each line of each file in the "Unprocessed-Password" folder was read, I ensured that the relevant file was moved to the "Processed" folder using the shutil library. In the last part, I took the end time with the time library and subtracted it from the start time. Thus, I found the total indexing time.

Before Processing:

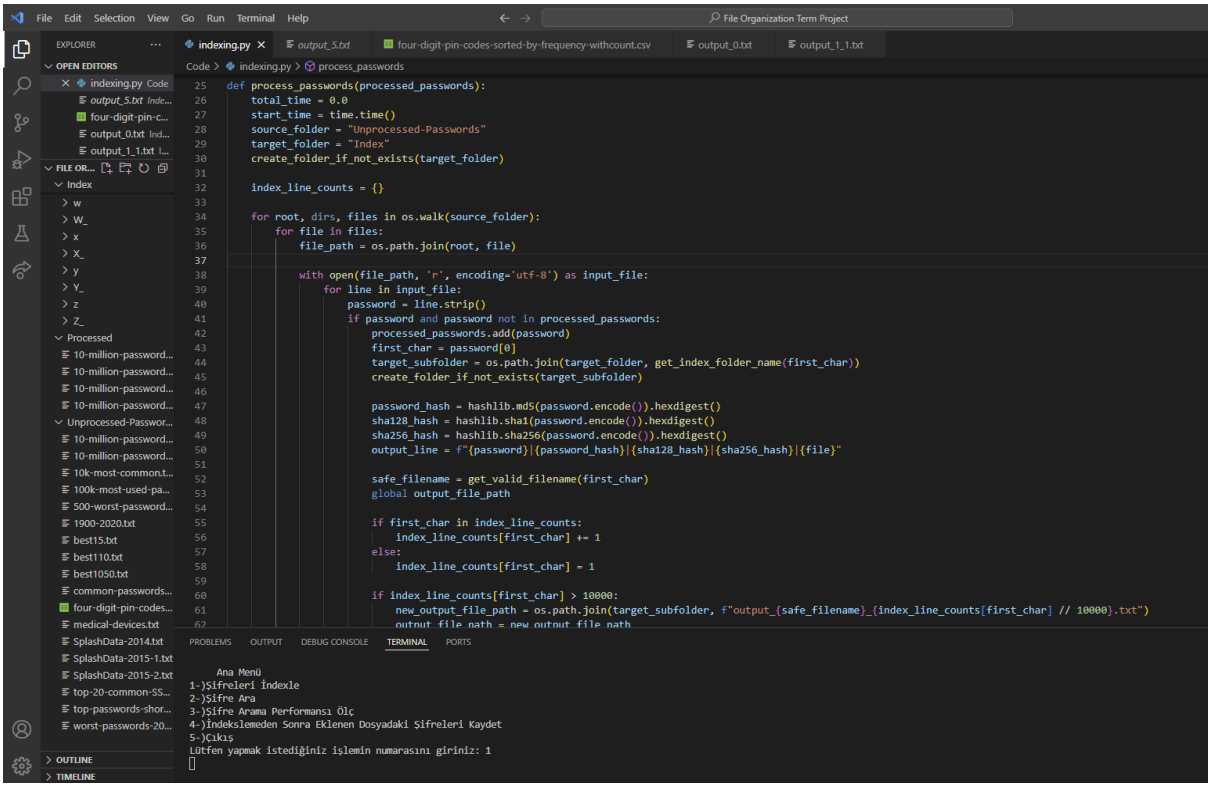


Picture 4.4

While processing:

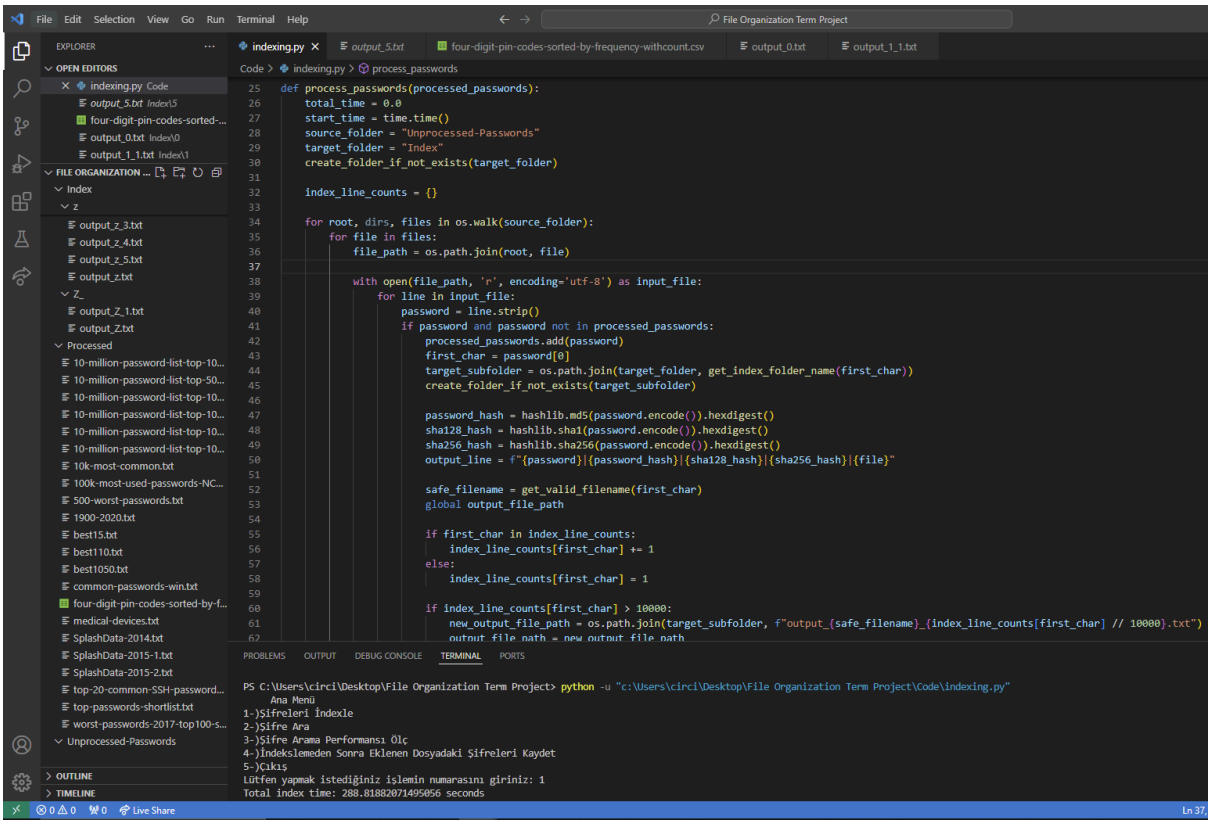


Picture 4.5

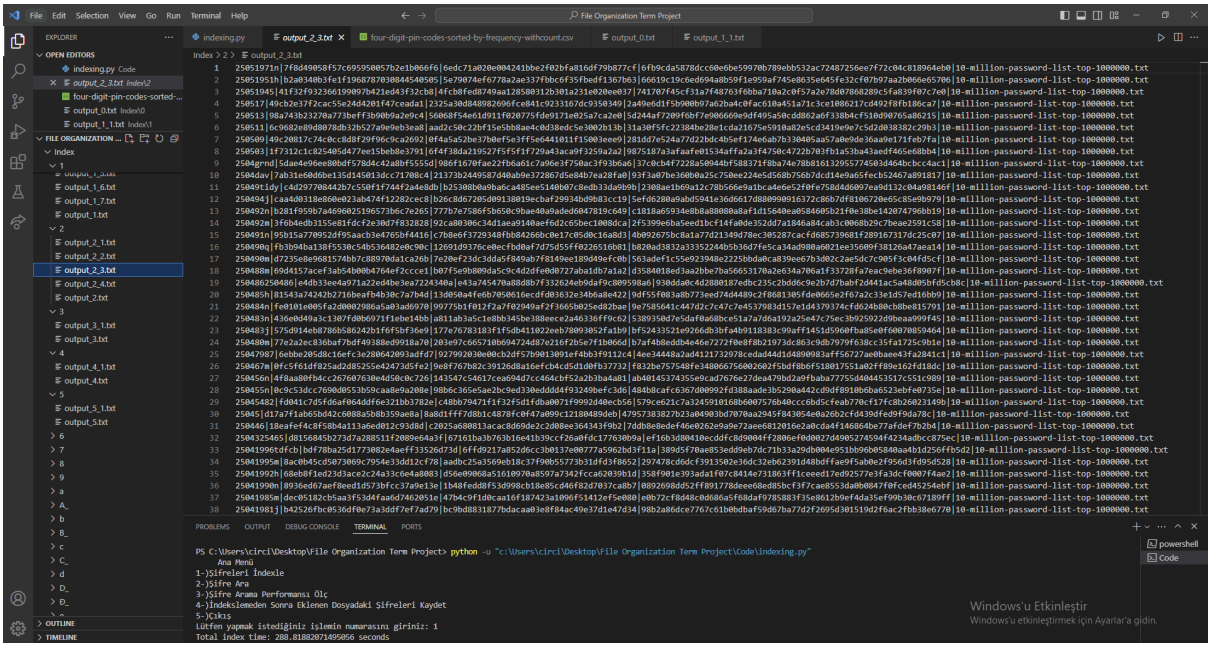


Picture 4.6

After Processing:



Picture 4.7



Picture 4.8

Overview:

While processing files, we can see the created files on the left tab of VS code(Picture 4.4). Also, we can see files moved to the “Processed” file from the “Unprocessed-Passwords” file(Picture 4.5).

After processing files, we can see the moved all files to the “Processed” file from the “Unprocessed-Passwords” file(Picture 4.6). The passwords on the output.txt files can be seen on the Picture 4.7.

5-) Search Function and Performance Test

```python
def search_password(query, isPrint):
    start_time = time.time()
    target_folder = "Index"
    search_results = []
    password = ""
    query = query.strip()
    first_char = query[0]
    target_subfolder = os.path.join(target_folder, get_index_folder_name(first_char))

    file_path = os.path.join(target_subfolder, f"output_{get_valid_filename(first_char)}.txt")

    if os.path.exists(file_path):
        with open(file_path, 'r+', encoding='utf-8') as input_file:
            for line in input_file:
                line = line.strip()
                password = line.split("|")[0]

                if password == query:
                    search_results.append(line)
            if(isPrint):
                if(search_results):
                    for asd in search_results:
                        print(asd)

    else:
        with open(file_path, 'a+', encoding='utf-8') as input_file:
            password_hash = hashlib.md5(password.encode()).hexdigest()
            sha128_hash = hashlib.sha1(password.encode()).hexdigest()
            sha256_hash = hashlib.sha256(password.encode()).hexdigest()
            output_line = f"{query}|{password_hash}|{sha128_hash}|{sha256_hash}| FromUserSearch"
            input_file.write(output_line + '\n')
        if(isPrint):
            print(f"Password not found: {query} and added to index list")
    end_time = time.time()
    search_time = end_time - start_time
    print(f"Total search time: {search_time} seconds")

```

Picture 5.1

To improve the performance of the code, I took the first letter of the password I wanted to search and entered it into the file corresponding to that letter. Then, it starts reading each line by entering the .txt files under the relevant file. When it finds a line that matches the relevant password, it adds this password to the list named "search_results". If the password is not found, it adds the searched password to the last line of the relevant .txt file. This ensures that it can be found in other searches.

Picture 5.2

The performance of the search function given on the Picture 5.2. To calculate the time of the search process, time library was used.

```python
def measure_search_performance():
    query_passwords = ["password1", "123456", "qwerty", "password123", "letmein", "admin", "batman", "monkey", "password", "abc123"]
    total_time = 0.0

    for password in query_passwords:
        start_time = time.time()
        search_password(password, False)
        end_time = time.time()
        search_time = end_time - start_time
        total_time += search_time

    average_time = total_time / len(query_passwords)
    print(f"Average search time: {average_time} seconds")

```

Picture 5.3

To find the average search speed of 10 passwords, I found the search time for each password. Then, I summed this number with the variable containing the total time. Finally, I found the average search time by dividing by the number of passwords.

Picture 5.4

The average search time in the code I wrote can be seen on the picture 5.4.

6-) Results and Evaluation

With this project, we saw and learned how to efficiently sort and search large datasets. We have seen that when we use many loops, our code will slow down and how we can eliminate the complexities that may occur in many file operations. By listing all the passwords under the required conditions, I successfully completed the project and reached my goal. The project can be improved by making it work for different file types.

7-) References

- https://python-istihza.yazbel.com/temel_dosya_islemleri.html
- https://www.w3schools.com/python/python_file_write.asp
- https://www.geeksforgeeks.org/python-how-to-search-for-a-string-in-text-files/
- https://www.geeksforgeeks.org/python-hash-method/
- https://www.programiz.com/python-programming/time
- https://docs.python.org/3.4/library/stdtypes.html
- https://www.python-engineer.com/posts/file-organization/
- https://www.geeksforgeeks.org/file-handling-python/
- https://realpython.com/working-with-files-in-python/
- Os, shutil, hashlib, string, time and csv libraries were used in the project.