

Data Privacy Assignment

Name: Mehul Jhunjhunwala

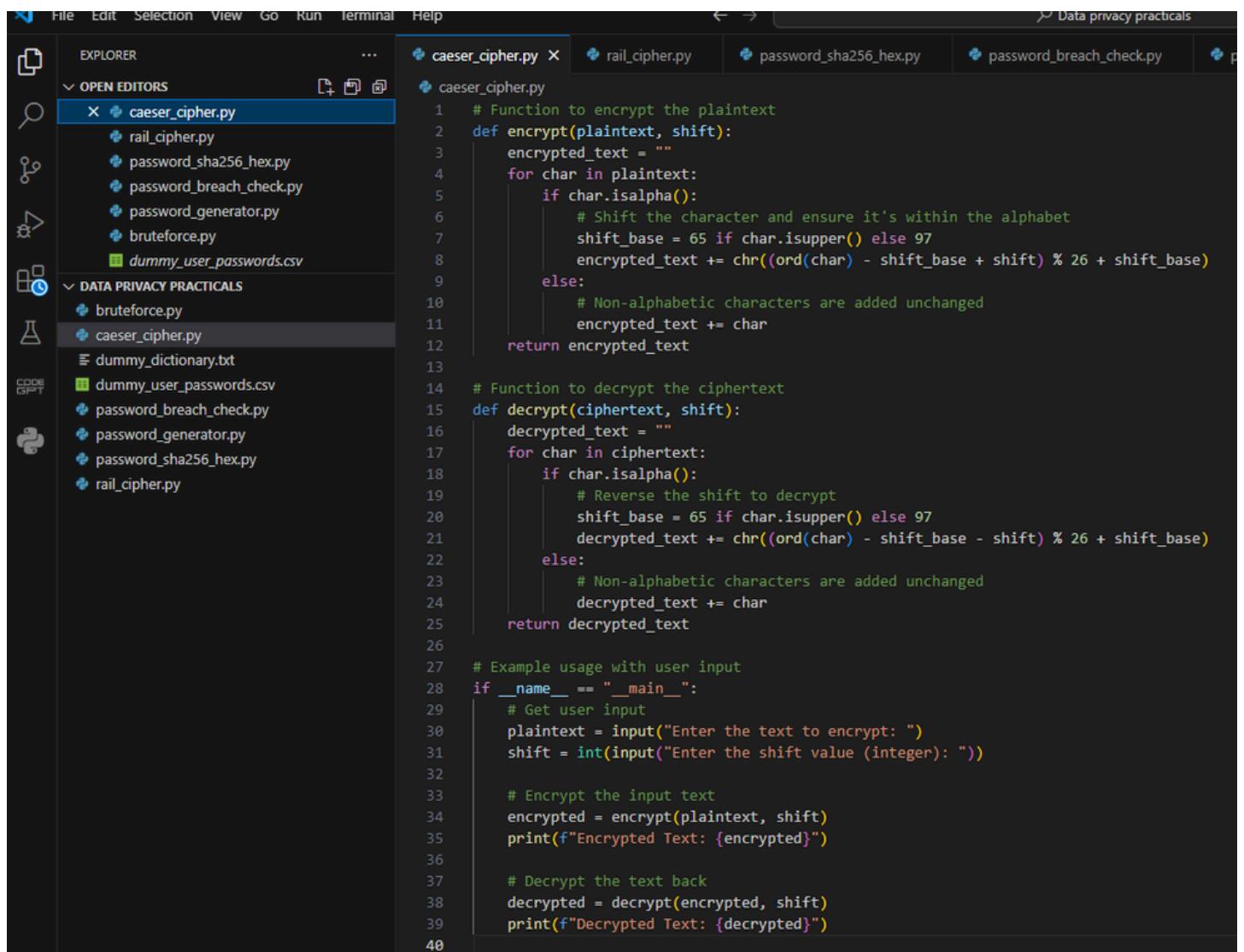
College roll no: 20221428

Exam roll no: 22020570025

Practical 1:

Write a program to perform encryption and decryption using Caesar cipher (substitutional cipher).

Source code:



The screenshot shows a code editor interface with a dark theme. The left sidebar has icons for file operations like Open, Save, and Find. The Explorer panel shows a file tree with several Python files and CSV files. The main editor area displays the code for 'caeser_cipher.py'. The code implements a Caesar cipher for both encryption and decryption.

```
File Edit Selection View Go Run Terminal Help Data privacy practicals caeser_cipher.py rail_cipher.py password_sha256_hex.py password_breach_check.py EXPLORER OPEN EDITORS caeser_cipher.py rail_cipher.py password_sha256_hex.py password_breach_check.py password_generator.py bruteforce.py dummy_user_passwords.csv DATA PRIVACY PRACTICALS bruteforce.py caeser_cipher.py dummy_dictionary.txt dummy_user_passwords.csv password_breach_check.py password_generator.py password_sha256_hex.py rail_cipher.py
# Function to encrypt the plaintext
def encrypt(plaintext, shift):
    encrypted_text = ""
    for char in plaintext:
        if char.isalpha():
            # Shift the character and ensure it's within the alphabet
            shift_base = 65 if char.isupper() else 97
            encrypted_text += chr((ord(char) - shift_base + shift) % 26 + shift_base)
        else:
            # Non-alphabetic characters are added unchanged
            encrypted_text += char
    return encrypted_text

# Function to decrypt the ciphertext
def decrypt(ciphertext, shift):
    decrypted_text = ""
    for char in ciphertext:
        if char.isalpha():
            # Reverse the shift to decrypt
            shift_base = 65 if char.isupper() else 97
            decrypted_text += chr((ord(char) - shift_base - shift) % 26 + shift_base)
        else:
            # Non-alphabetic characters are added unchanged
            decrypted_text += char
    return decrypted_text

# Example usage with user input
if __name__ == "__main__":
    # Get user input
    plaintext = input("Enter the text to encrypt: ")
    shift = int(input("Enter the shift value (integer): "))

    # Encrypt the input text
    encrypted = encrypt(plaintext, shift)
    print(f"Encrypted Text: {encrypted}")

    # Decrypt the text back
    decrypted = decrypt(encrypted, shift)
    print(f"Decrypted Text: {decrypted}")
```

Explanation:

This code is a simple implementation of a **Caesar Cipher**, a technique used to encrypt and decrypt messages by shifting the letters of the alphabet by a certain number of places.

1. **Encryption Function** (encrypt):
 - This function takes a **plaintext** (the original message) and a **shift value** (the number of positions to shift each letter).
 - It loops through each character in the text.
 - If the character is a letter, it shifts it forward in the alphabet by the specified number of positions.
 - For example, with a shift of 3, 'A' becomes 'D'.
 - If the character is not a letter (e.g., a number, space, or punctuation), it adds it unchanged to the result.
 - Finally, it returns the encrypted text.
2. **Decryption Function** (decrypt):
 - This function does the opposite of encryption.
 - It takes the **ciphertext** (the encrypted message) and reverses the shift to get back the original message.
 - Non-letter characters remain unchanged, just like in encryption.
3. **User Input and Example Usage:**
 - The program starts by asking the user for:
 - The text they want to encrypt.
 - The shift value (an integer).
 - It then uses the encrypt function to encrypt the input text and displays the result.
 - Next, it uses the decrypt function to confirm that the encrypted text can be decrypted back to the original text.

Outputs:

```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d
:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\caeser_cipher.py"

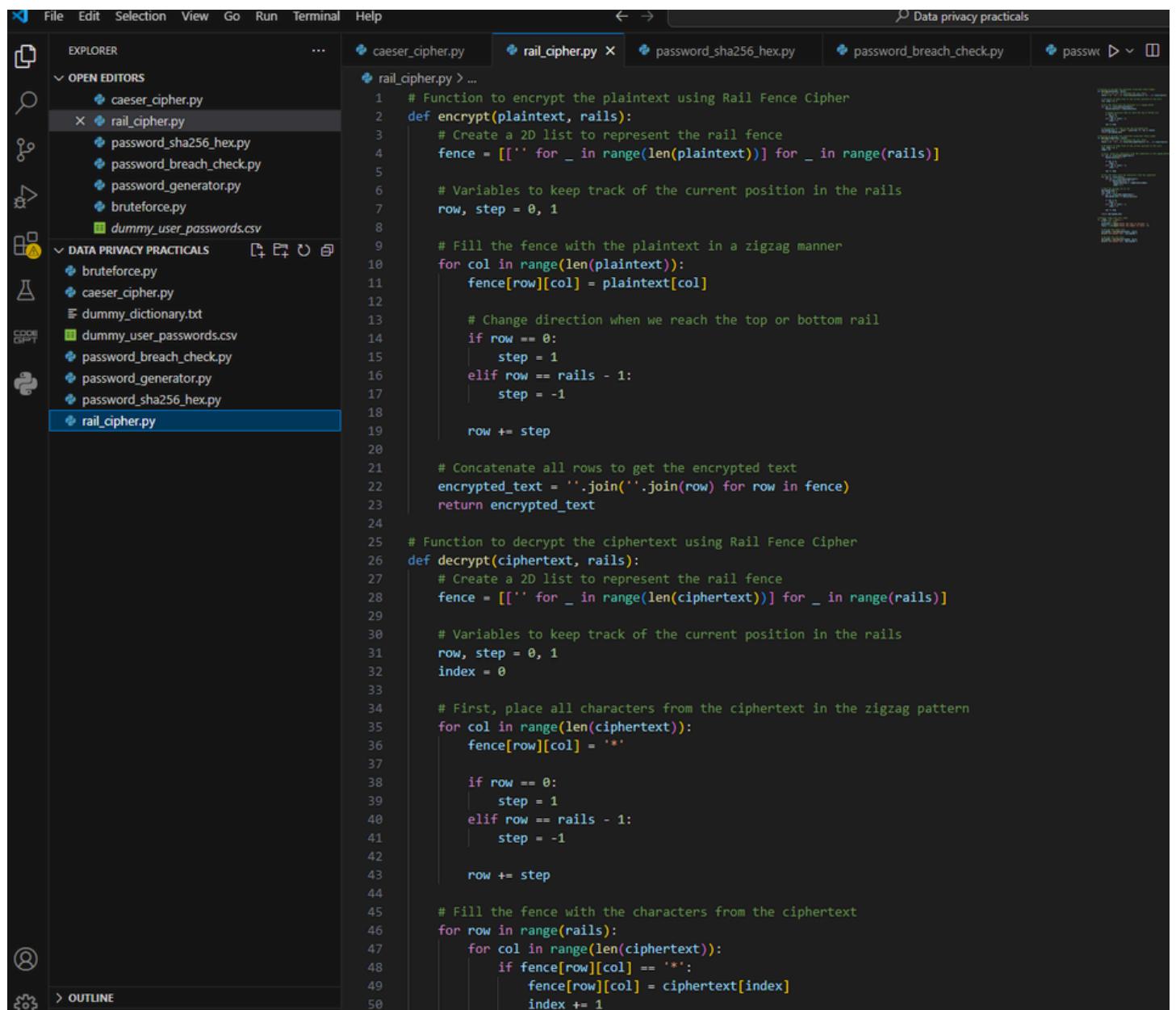
Enter the text to encrypt: Computer science
Enter the shift value (integer): 3
Encrypted Text: Frpsxwhu vflhqf
Decrypted Text: Computer science
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals>
```

```
Enter the text to encrypt: Caesar cipher
Enter the shift value (integer): 5
Encrypted Text: Hfjxfw hnumjw
Decrypted Text: Caesar cipher
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals>
```

Practical 2:

Write a program to perform encryption and decryption using Rail Fence Cipher(transpositional cipher)

Source code:



The screenshot shows a code editor interface with a dark theme. On the left is a sidebar with various icons and a tree view of files. The main area has tabs for multiple files, with 'rail_cipher.py' currently active. The code itself is a Python script for performing rail fence encryption and decryption.

```
# Function to encrypt the plaintext using Rail Fence Cipher
def encrypt(plaintext, rails):
    # Create a 2D list to represent the rail fence
    fence = [['' for _ in range(len(plaintext))] for _ in range(rails)]  
  
    # Variables to keep track of the current position in the rails
    row, step = 0, 1  
  
    # Fill the fence with the plaintext in a zigzag manner
    for col in range(len(plaintext)):
        fence[row][col] = plaintext[col]  
  
        # Change direction when we reach the top or bottom rail
        if row == 0:
            step = 1
        elif row == rails - 1:
            step = -1  
  
        row += step  
  
    # Concatenate all rows to get the encrypted text
    encrypted_text = ''.join(''.join(row) for row in fence)
    return encrypted_text  
  
# Function to decrypt the ciphertext using Rail Fence Cipher
def decrypt(ciphertext, rails):
    # Create a 2D list to represent the rail fence
    fence = [['' for _ in range(len(ciphertext))] for _ in range(rails)]  
  
    # Variables to keep track of the current position in the rails
    row, step = 0, 1
    index = 0  
  
    # First, place all characters from the ciphertext in the zigzag pattern
    for col in range(len(ciphertext)):
        fence[row][col] = '*'  
  
        if row == 0:
            step = 1
        elif row == rails - 1:
            step = -1  
  
        row += step  
  
    # Fill the fence with the characters from the ciphertext
    for row in range(rails):
        for col in range(len(ciphertext)):
            if fence[row][col] == '*':
                fence[row][col] = ciphertext[index]
                index += 1
```

The screenshot shows a code editor interface with the title "Data privacy practicals". The left sidebar has sections for "EXPLORER" and "OPEN EDITORS". Under "OPEN EDITORS", there are files: caeser_cipher.py, rail_cipher.py (which is selected), password_sha256_hex.py, password_breach_check.py, and passwr...py. Under "DATA PRIVACY PRACTICALS", there are files: bruteforce.py, caeser_cipher.py, dummy_dictionary.txt, dummy_user_passwords.csv, password_breach_check.py, password_generator.py, password_sha256_hex.py, and rail_cipher.py. The main editor area displays the following Python code:

```
caeser_cipher.py rail_cipher.py X password_sha256_hex.py password_breach_check.py passwr...py ...  
rail_cipher.py > encrypt  
26 def decrypt(ciphertext, rails):  
51     # Read the message row by row  
52     decrypted_text = ''  
53     row, step = 0, 1  
54     for col in range(len(ciphertext)):  
55         decrypted_text += fence[row][col]  
56  
57         if row == 0:  
58             step = 1  
59         elif row == rails - 1:  
60             step = -1  
61  
62         row += step  
63  
64     return decrypted_text  
65  
66     # Example usage with user input  
67     if __name__ == "__main__":  
68         # Get user input  
69         plaintext = input("Enter the text to encrypt: ")  
70         rails = int(input("Enter the number of rails: "))  
71  
72         # Encrypt the input text  
73         encrypted = encrypt(plaintext, rails)  
74         print(f"Encrypted Text: {encrypted}")  
75  
76         # Decrypt the text back  
77         decrypted = decrypt(encrypted, rails)  
78         print(f"Decrypted Text: {decrypted}")  
79  
80
```

Explanation:

This code implements the **Rail Fence Cipher**, a classical encryption method where the text is written in a zigzag pattern on multiple "rails" (or rows) and then read off row by row to form the ciphertext.

1. Encryption Function (encrypt):

- The function takes the **plaintext** (original message) and the number of **rails** (rows).
- It creates a 2D list (a kind of table) to represent the zigzag pattern of the rails.
- The plaintext is placed on the rails in a zigzag pattern:
 - Starts from the top rail, moves downward to the bottom rail, and then moves back up to the top rail, repeating this process.
- After filling the zigzag pattern, all characters are read row by row to form the encrypted text (ciphertext).

2. Decryption Function (decrypt):

- The function takes the **ciphertext** (encrypted message) and the number of **rails**.
- It creates a similar 2D list to represent the zigzag pattern.
- First, it marks the zigzag positions by filling the pattern with placeholders (*) to determine the arrangement of the rails.
- Then, it replaces the placeholders with the actual characters from the ciphertext.
- Finally, the function reconstructs the original message by reading the zigzag pattern in the same order as it was filled during encryption.

3. User Input:

- The program asks the user for:
 - The text to be encrypted.
 - The number of rails to use in the cipher.
- It uses the encrypt function to generate the encrypted text and displays it.

- Then, it uses the decrypt function to verify that the encrypted text can be decoded back to the original plaintext.

Outputs:

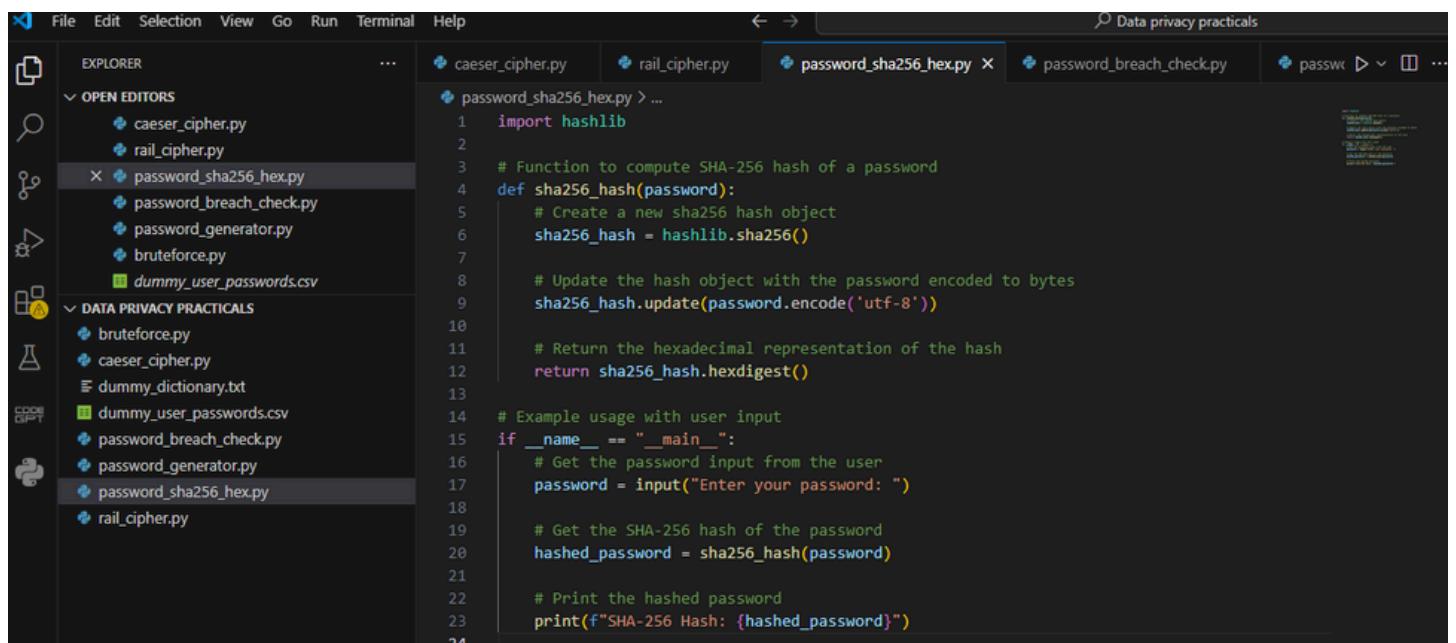
```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d
: \Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\rail_cipher.py"
Enter the text to encrypt: mumbai indians
Enter the number of rails: 4
Encrypted Text: m nuiiasmanibd
Decrypted Text: mumbai indians
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals>
```

```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d
: \Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\rail_cipher.py"
Enter the text to encrypt: Royal challengers Banglore
Enter the number of rails: 5
Encrypted Text: Rasrohlr oeycleBla egaglnn
Decrypted Text: Royal challengers Banglore
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals>
```

Practical 3:

Write a Python program that defines a function and takes a password string as input and returns its SHA-256 hashed representation as a hexadecimal string.

Source code:



The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files in the current directory: caeser_cipher.py, rail_cipher.py, password_sha256_hex.py (selected), password_breach_check.py, password_generator.py, bruteforce.py, and dummy_user_passwords.csv.
- Terminal:** Shows the command `python -u "d : \Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\rail_cipher.py"`. The output of the previous run is also visible.
- Code Editor:** The file `password_sha256_hex.py` is open and contains the following Python code:

```

EXPLORER File Edit Selection View Go Run Terminal Help ↻ → Data privacy practicals
OPEN EDITORS caeser_cipher.py rail_cipher.py password_sha256_hex.py X password_breach_check.py passwo ...
password_sha256_hex.py > ...
1 import hashlib
2
3 # Function to compute SHA-256 hash of a password
4 def sha256_hash(password):
5     # Create a new sha256 hash object
6     sha256_hash = hashlib.sha256()
7
8     # Update the hash object with the password encoded to bytes
9     sha256_hash.update(password.encode('utf-8'))
10
11    # Return the hexadecimal representation of the hash
12    return sha256_hash.hexdigest()
13
14 # Example usage with user input
15 if __name__ == "__main__":
16     # Get the password input from the user
17     password = input("Enter your password: ")
18
19     # Get the SHA-256 hash of the password
20     hashed_password = sha256_hash(password)
21
22     # Print the hashed password
23     print(f"SHA-256 Hash: {hashed_password}")
24

```

Explanation:

This code provides a simple implementation of hashing a password using the **SHA-256 algorithm**, a cryptographic hash function that generates a fixed-length, 256-bit hash value.

1. Hashing Function (sha256_hash):

- The function takes a **password** (string) as input.
- It creates a new SHA-256 hash object using the hashlib module.
- The password is converted to bytes using .encode('utf-8') because the hash object works with binary data, not plain strings.
- The hash object processes the password, and the resulting hash is converted to a **hexadecimal string** using .hexdigest(). This hexadecimal format makes the hash easy to read and store.

2. Example Usage:

- The program asks the user to input a password.
- The entered password is passed to the sha256_hash function, which computes its SHA-256 hash.
- The program then prints the hashed password.

Outputs:

```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d :\\Study\\B.Sc (Hons.) CS\\SEMESTER 5\\Data privacy practicals\\password_sha256_h ex.py"
Enter your password: 12345678
SHA-256 Hash: ef797c8118f02dfb649607dd5d3f8c7623048c9c063d532cc95c5ed7a898a6
4f
PS D:\\Study\\B.Sc (Hons.) CS\\SEMESTER 5\\Data privacy practicals>
```

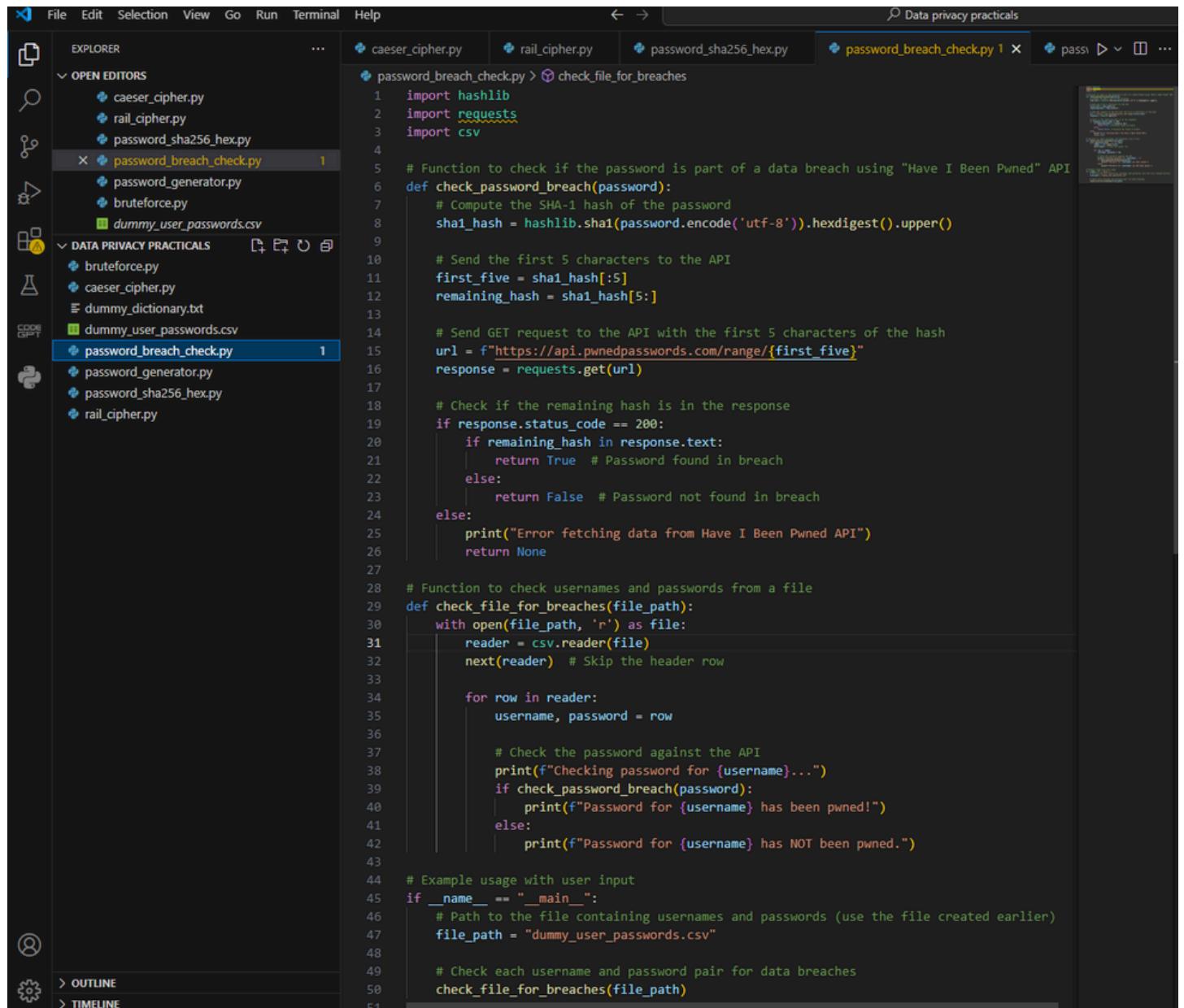
```
PS D:\\Study\\B.Sc (Hons.) CS\\SEMESTER 5\\Data privacy practicals> python -u "d :\\Study\\B.Sc (Hons.) CS\\SEMESTER 5\\Data privacy practicals\\password_sha256_h ex.py"
Enter your password: qwerty123
SHA-256 Hash: daaad6e5604e8e17bd9f108d91e26afe6281dac8fd0091040a7a6d7bd9b43
b5
PS D:\\Study\\B.Sc (Hons.) CS\\SEMESTER 5\\Data privacy practicals>
```

Practical 4:

Write a Python program that reads a file containing a list of usernames and passwords, one pair per line (separated by

a comma). It checks each password to see if it has been leaked in a data breach. You can use the “Have I Been Pwned” API (<https://haveibeenpwned.com/API/v3>) to check if a password has been leaked.

Source code:



The screenshot shows a code editor interface with the title "Data privacy practicals". The left sidebar lists files in the "EXPLORER" and "DATA PRIVACY PRACTICALS" sections. The "password_breach_check.py" file is selected and shown in the main editor area. The code uses the hashlib, requests, and csv modules to check if a password has been leaked in a data breach. It defines two functions: "check_password_breach" and "check_file_for_breaches". The "check_password_breach" function takes a password, computes its SHA-1 hash, sends a GET request to the Have I Been Pwned API, and checks if the remaining hash is present in the response. The "check_file_for_breaches" function reads a CSV file containing usernames and passwords, checks each one against the API, and prints the results. A terminal window in the background shows some command-line output.

```
password_breach_check.py > check_file_for_breaches
1 import hashlib
2 import requests
3 import csv
4
5 # Function to check if the password is part of a data breach using "Have I Been Pwned" API
6 def check_password_breach(password):
7     # Compute the SHA-1 hash of the password
8     sha1_hash = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()
9
10    # Send the first 5 characters to the API
11    first_five = sha1_hash[:5]
12    remaining_hash = sha1_hash[5:]
13
14    # Send GET request to the API with the first 5 characters of the hash
15    url = f"https://api.pwnedpasswords.com/range/{first_five}"
16    response = requests.get(url)
17
18    # Check if the remaining hash is in the response
19    if response.status_code == 200:
20        if remaining_hash in response.text:
21            return True # Password found in breach
22        else:
23            return False # Password not found in breach
24    else:
25        print("Error fetching data from Have I Been Pwned API")
26        return None
27
28    # Function to check usernames and passwords from a file
29    def check_file_for_breaches(file_path):
30        with open(file_path, 'r') as file:
31            reader = csv.reader(file)
32            next(reader) # Skip the header row
33
34            for row in reader:
35                username, password = row
36
37                # Check the password against the API
38                print(f"Checking password for {username}...")
39                if check_password_breach(password):
40                    print(f"Password for {username} has been pwned!")
41                else:
42                    print(f"Password for {username} has NOT been pwned.")
43
44    # Example usage with user input
45    if __name__ == "__main__":
46        # Path to the file containing usernames and passwords (use the file created earlier)
47        file_path = "dummy_user_passwords.csv"
48
49        # Check each username and password pair for data breaches
50        check_file_for_breaches(file_path)
```

Explanation:

This program checks whether passwords have been exposed in a **data breach** by using the “Have I Been Pwned” API.

1. Function to Check a Password Against Breaches (check_password_breach):

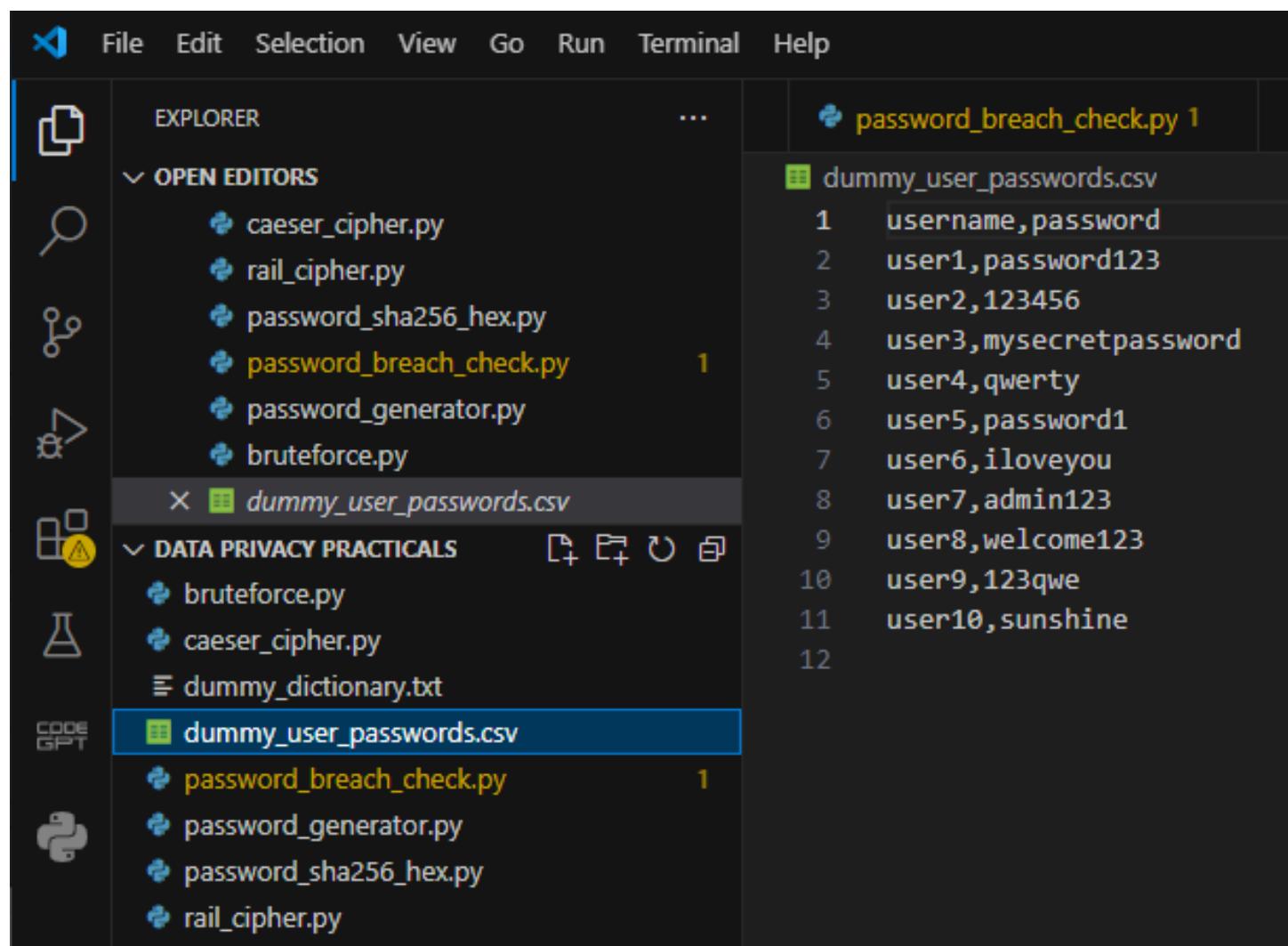
- **Password Hashing:**
 - The password is hashed using **SHA-1**. This converts the password into a 40-character hexadecimal string.

- The hash is split into two parts: the **first 5 characters** (first_five) and the **remaining characters** (remaining_hash).
- API Request:**
 - A request is sent to the **Have I Been Pwned API** using the first 5 characters of the hash (first_five).
 - The API responds with a list of hashed passwords starting with those 5 characters, along with the number of times each has been seen in breaches.
- Check for a Match:**
 - The program searches the API response for the remaining_hash.
 - If a match is found, it means the password has been exposed in a data breach, and the function returns True. Otherwise, it returns False.

2. Function to Check Usernames and Passwords from a File (check_file_for_breaches):

- Reads a CSV file containing **username-password pairs**.
 - Assumes the file has a header row, which is skipped.
 - Each row contains a username and a password.
- For each username-password pair:
 - The password is checked against the API using the check_password_breach function.
 - The result (breached or not) is printed to the console.

CSV file:



```
password_breach_check.py 1
dummy_user_passwords.csv
1 username,password
2 user1,password123
3 user2,123456
4 user3,mysecretpassword
5 user4,qwerty
6 user5,password1
7 user6,iloveyou
8 user7,admin123
9 user8,welcome123
10 user9,123qwe
11 user10,sunshine
12
```

The screenshot shows a code editor interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. On the left, there's an Explorer sidebar with icons for files, folders, and other project details. The main workspace shows several Python files in the 'OPEN EDITORS' list: caeser_cipher.py, rail_cipher.py, password_sha256_hex.py, password_breach_check.py (which is currently active), password_generator.py, and bruteforce.py. Below this, a 'DATA PRIVACY PRACTICALS' section contains files like bruteforce.py, caeser_cipher.py, and dummy_dictionary.txt. A CSV file named 'dummy_user_passwords.csv' is open in the editor, showing 12 rows of data with columns 'username' and 'password'. The first few rows are: user1,password123, user2,123456, user3,mysecretpassword, user4,qwerty, user5,password1, user6,iloveyou, user7,admin123, user8,welcome123, user9,123qwe, user10,sunshine.

Outputs:

```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d
:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\password_breach_c
heck.py"
Checking password for user1...
Password for user1 has been pwned!
Checking password for user2...
Password for user2 has been pwned!
Checking password for user3...
Password for user3 has been pwned!
Checking password for user4...
Password for user4 has been pwned!
Checking password for user5...
Password for user5 has been pwned!
Checking password for user6...
Password for user6 has been pwned!
Checking password for user7...
Password for user7 has been pwned!
Checking password for user8...
Password for user8 has been pwned!
Checking password for user9...
Password for user9 has been pwned!
Checking password for user10...
Password for user10 has been pwned!
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> []
```

Practical 5:

Write a Python program that generates a password using a random combination of words from a dictionary file.

Source code:

The screenshot shows a code editor interface with several tabs open. The tabs include 'password_breach_check.py' (active), 'password_generator.py', 'bruteforce.py', and 'dummy_user_passwords.csv'. The left sidebar shows a file tree under 'EXPLORER' and 'DATA PRIVACY PRACTICALS' sections, containing files like 'caeser_cipher.py', 'rail_cipher.py', 'password_sha256_hex.py', 'password_breach_check.py', 'bruteforce.py', and 'dummy_user_passwords.csv'. The main area displays the code for 'password_generator.py'.

```
password_generator.py > ...
1 import random
2
3 # Function to generate a strong password from the dictionary file
4 def generate_password(length, dictionary_file_path):
5     # Open the dictionary file and load words
6     with open(dictionary_file_path, mode="r") as file:
7         words = [line.strip() for line in file]
8
9     # Generate password by randomly selecting words from the dictionary
10    password = random.sample(words, length)
11
12    # Combine the words into a single password string
13    return ''.join(password)
14
15 # Example usage:
16 if __name__ == "__main__":
17     # Define the path to the dummy dictionary file
18     dictionary_file_path = "dummy_dictionary.txt" # Replace with the correct path on your
19
20     # Ask the user for password length
21     password_length = int(input("Enter the number of words for the password: "))
22
23     # Generate the password
24     generated_password = generate_password(password_length, dictionary_file_path)
25     print(f"Generated Password: {generated_password}")
26
```

Explanation:

This program generates a strong, randomized password using words from a dictionary file.

1. Function to Generate a Password (generate_password):

- **Inputs:**

- length: The number of words to include in the password.
- dictionary_file_path: Path to a text file containing a list of words, one per line.

- **How It Works:**

1. The program reads all words from the dictionary file and stores them in a list.
2. It uses the random.sample() function to randomly select the specified number of words (length) from the list.
3. The selected words are joined together into a single string without spaces or separators to form the password.

2. Example Usage:

- The script asks the user to specify the number of words to include in the password.
- It then generates a password using the specified word count and prints it.

Dictionary file:

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows several Python files and a CSV file under the "OPEN EDITORS" and "DATA PRIVACY PRACTICALS" sections.
- Editor:** The "dummy_dictionary.txt" file is open in the main editor area. It contains the following content:

```
1 apple
2 banana
3 cherry
4 date
5 elephant
6 frog
7 grape
8 honey
9 ice
10 jungle
11 kiwi
12 lemon
13 mango
14 nectar
15 orange
16 pear
17 quilt
18 rose
19 sunshine
20 tiger
21 umbrella
22 violet
23 watermelon
24 xylophone
25 yellow
26 zebra
27
```

Outputs:

```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d
:Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\password_generato
r.py"
Enter the number of words for the password: 3
Generated Password: umbrellabanananawatermelon
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals>
```

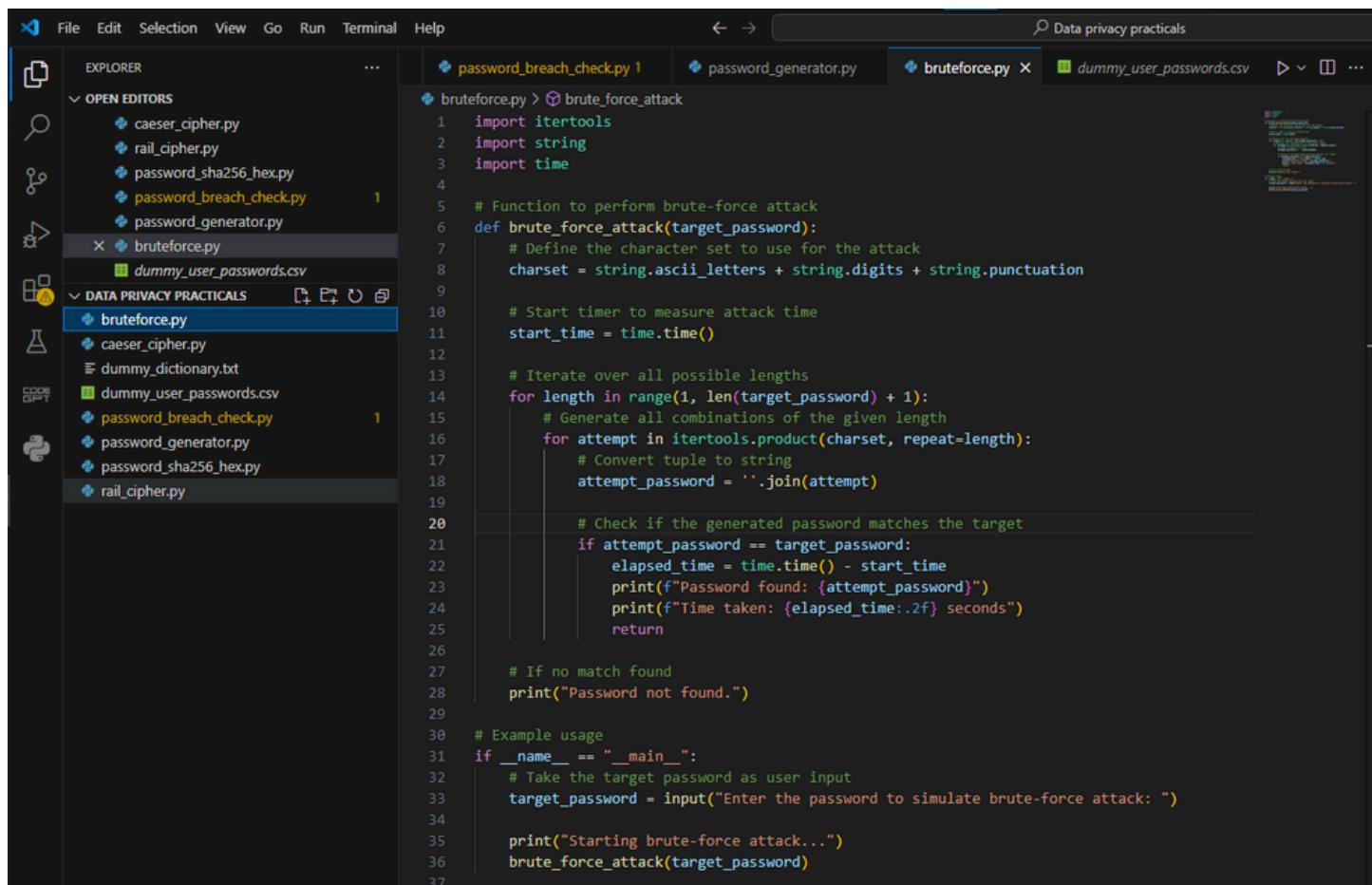
```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d
:Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\password_generato
r.py"
Enter the number of words for the password: 2
Generated Password: sunshinebanana
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals>
```

Practical 6:

Write a Python program that simulates a brute-force attack on a password by trying out all possible character

combinations.

Source code:



The screenshot shows a code editor interface with multiple files open in the sidebar and tabs at the top. The tabs include password_breach_check.py, password_generator.py, bruteforce.py (active), and dummy_user_passwords.csv. The code editor displays the contents of the bruteforce.py file, which contains a function for performing a brute-force attack. The code uses the string module's ascii_letters, digits, and punctuation constants to create a character set. It then uses itertools.product to generate all combinations of characters for each length from 1 to the length of the target password. The attempt is converted back into a string and compared against the target password. If a match is found, the elapsed time is printed. If no match is found after all attempts, it prints "Password not found." An example usage block shows how to run the function with user input for the target password.

```
password_breach_check.py 1 password_generator.py bruteforce.py > brute_force_attack
import itertools
import string
import time

# Function to perform brute-force attack
def brute_force_attack(target_password):
    # Define the character set to use for the attack
    charset = string.ascii_letters + string.digits + string.punctuation

    # Start timer to measure attack time
    start_time = time.time()

    # Iterate over all possible lengths
    for length in range(1, len(target_password) + 1):
        # Generate all combinations of the given length
        for attempt in itertools.product(charset, repeat=length):
            # Convert tuple to string
            attempt_password = ''.join(attempt)

            # Check if the generated password matches the target
            if attempt_password == target_password:
                elapsed_time = time.time() - start_time
                print(f"Password found: {attempt_password}")
                print(f"Time taken: {elapsed_time:.2f} seconds")
                return

    # If no match found
    print("Password not found.")

# Example usage
if __name__ == "__main__":
    # Take the target password as user input
    target_password = input("Enter the password to simulate brute-force attack: ")

    print("Starting brute-force attack...")
    brute_force_attack(target_password)
```

Explanation:

This program simulates a **brute-force attack**, attempting to guess a target password by systematically trying every possible combination of characters.

1. Function to Perform the Brute-Force Attack (brute_force_attack):

- **Input:**
 - target_password: The password the function tries to guess.
- **Character Set:**
 - Includes **uppercase letters**, **lowercase letters**, **digits**, and **punctuation** from the `stringmodule`:
 - `string.ascii_letters + string.digits + string.punctuation`
- **Attack Process:**
 1. The function starts a timer to measure how long the attack takes.
 2. For each possible password length (starting from 1 up to the length of the `target_password`):
 - It generates all possible combinations of characters of that length using `itertools.product()`.
 - Converts each combination (a tuple) into a string.
 - Compares the generated string with the `target_password`.

3. If a match is found:

- Prints the password and the time taken for the attack.
- Stops further attempts.

- **Stopping Condition:**

- The loop exits as soon as the target password is found.
- If the password is not found within the specified character length range, it prints "Password not found."

Outputs:

```
.) CS\SEMESTER 5\Data privacy practicals> python -u "d:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\bruteforce.py"
Enter the password to simulate brute-force attack: p123
Starting brute-force attack...
Password found: p123
Time taken: 1.99 seconds
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals>
```

```
.) CS\SEMESTER 5\Data privacy practicals> python -u "d:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\bruteforce.py"
Enter the password to simulate brute-force attack: 1234
Starting brute-force attack...
Password found: 1234
Time taken: 8.74 seconds
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals>
```

Practical 8:

Students needs to conduct a data privacy audit of an organization to identify potential vulnerabilities and risks in their data privacy practices.

Source code:

The screenshot shows a code editor interface with multiple tabs and a sidebar. The top bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search field labeled 'Data privacy practicals'. The left sidebar has sections for EXPLORER, OPEN EDITORS, and DATA PRIVACY PRACTICALS, with various files listed. The main area displays the content of the 'practical_8.py' file.

```
1 import os
2 import json
3 import datetime
4
5 # Define audit categories
6 audit_criteria = {
7     "data_collection": [
8         "Are users informed about data collection?",
9         "Is data collection limited to what's necessary?",
10        "Are consent mechanisms in place?"
11    ],
12    "data_storage": [
13        "Is data encrypted at rest?",
14        "Is sensitive data stored securely?",
15        "Are backup policies in place?"
16    ],
17    "data_access": [
18        "Is access to data restricted based on roles?",
19        "Are access logs maintained and monitored?",
20        "Are strong authentication mechanisms used?"
21    ],
22    "compliance": [
23        "Is the organization GDPR compliant?",
24        "Are data retention policies clearly defined?",
25        "Is there a process for handling data subject requests?"
26    ]
27 }
28
29 # Sample responses for vulnerabilities
30 responses = {}
31
32 def collect_responses():
33     print("Starting Data Privacy Audit...\n")
34     for category, questions in audit_criteria.items():
35         print(f"Category: {category.upper()}")
36         category_responses = []
37         for question in questions:
38             response = input(f" - {question} (Yes/No): ").strip().lower()
39             while response not in ["yes", "no"]:
40                 print("Please enter 'Yes' or 'No'.")
41                 response = input(f" - {question} (Yes/No): ").strip().lower()
42             category_responses.append({"question": question, "response": response})
43         responses[category] = category_responses
44     print("\n")
45
46 def analyze_responses():
47     print("\nAnalyzing Responses...\n")
48     vulnerabilities = {}
49     for category, answers in responses.items():
50         category_vulnerabilities = [item["question"] for item in answers if item["response"]
51                                     vulnerabilities[category] = category_vulnerabilities
```

The screenshot shows a code editor interface with multiple tabs open. The tabs include 'tor.py', 'dummy_dictionary.txt', 'bruteforce.py', 'practical_8.py' (which is the active tab), 'practical_9.py', 'prac_10.py', and 'Data privacy practicals'. The left sidebar shows a file tree under 'OPEN EDITORS' and 'DATA PRIVACY PRACTICALS' sections, with 'practical_8.py' also selected. The main area displays the content of the 'practical_8.py' script:

```
def analyze_responses():
    return vulnerabilities

def generate_report(vulnerabilities):
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
    report_filename = f"data_privacy_audit_report_{timestamp}.json"

    report_content = {
        "timestamp": timestamp,
        "audit_results": responses,
        "vulnerabilities": vulnerabilities
    }

    with open(report_filename, "w") as report_file:
        json.dump(report_content, report_file, indent=4)

    print(f"\nAudit report generated: {report_filename}")
    return report_filename

def display_summary(vulnerabilities):
    print("\nSummary of Findings:")
    for category, issues in vulnerabilities.items():
        print(f"- {category.upper()}: {len(issues)} issues identified.")
        for issue in issues:
            print(f"  * {issue}")

# Main script execution
if __name__ == "__main__":
    collect_responses()
    vulnerabilities = analyze_responses()
    display_summary(vulnerabilities)
    generate_report(vulnerabilities)
```

Explanation:

This script facilitates a **Data Privacy Audit** by asking questions, analyzing responses, and generating a detailed audit report.

1. Audit Categories and Questions

- The audit_criteriadictionary defines the audit categories and their respective questions, such as:
 - **Data Collection:** Ensures user consent and minimal data collection.
 - **Data Storage:** Checks encryption, storage security, and backup policies.
 - **Data Access:** Verifies restricted access, logging, and authentication.
 - **Compliance:** Ensures adherence to regulations like GDPR.

2. User Interaction

- **collect_responses():**
 - Prompts the user with questions under each category.
 - Accepts responses as "**Yes**" or "**No**".
 - Ensures valid input by re-prompting for incorrect responses.
 - Stores the answers in the responses dictionary.

3. Response Analysis

- **analyze_responses():**
 - Scans through the collected responses.
 - Identifies vulnerabilities (questions with a "No" response).

- Returns a vulnerabilities dictionary with issues grouped by category.

4. Report Generation

- **generate_report():**
 - Creates a timestamped JSON report (data_privacy_audit_report_<timestamp>.json) containing:
 - Audit timestamp.
 - Full audit results (responses).
 - Identified vulnerabilities.
 - Saves the report to the current working directory.
 - Example filename: data_privacy_audit_report_2024-11-26_14-30-45.json.

5. Summary Display

- **display_summary():**
 - Prints a concise summary of vulnerabilities by category.
 - For each category, it lists:
 - The number of identified issues.
 - Each specific issue.

Applications

- Useful for organizations conducting internal data privacy assessments.
- Helps identify areas requiring improvement to comply with regulations like GDPR or CCPA.
- Provides a documented record of the audit process.

Output:

```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\practical_8.py"
Starting Data Privacy Audit...
```

Category: DATA_COLLECTION

- Are users informed about data collection? (Yes/No): Yes
- Is data collection limited to what's necessary? (Yes/No): Yes
- Are consent mechanisms in place? (Yes/No): Yes

Category: DATA_STORAGE

- Is data encrypted at rest? (Yes/No): No
- Is sensitive data stored securely? (Yes/No): Yes
- Are backup policies in place? (Yes/No): Yes

Category: DATA_ACCESS

- Is access to data restricted based on roles? (Yes/No): Yes
- Are access logs maintained and monitored? (Yes/No): Yes
- Are strong authentication mechanisms used? (Yes/No): Yes

Category: COMPLIANCE

- Is the organization GDPR compliant? (Yes/No): Yes
- Are data retention policies clearly defined? (Yes/No): No
- Is there a process for handling data subject requests? (Yes/No): Yes

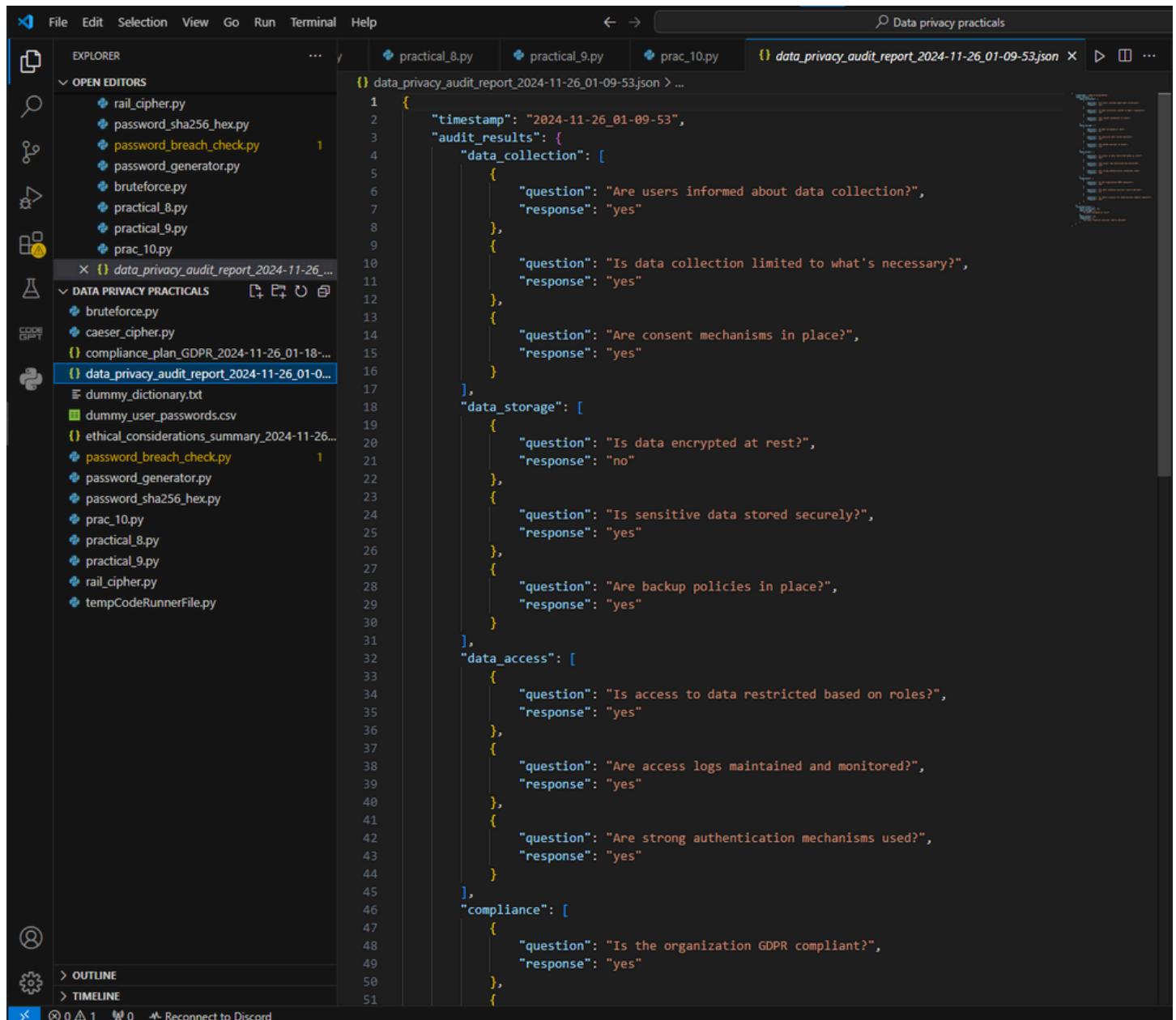
Analyzing Responses...

Summary of Findings:

- DATA_COLLECTION: 0 issues identified.
- DATA_STORAGE: 1 issues identified.
 - * Is data encrypted at rest?
- DATA_ACCESS: 0 issues identified.
- COMPLIANCE: 1 issues identified.
 - * Are data retention policies clearly defined?

Audit report generated: data_privacy_audit_report_2024-11-26_01-09-53.json

```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> █
```



A screenshot of a code editor interface, likely Visual Studio Code, displaying a JSON file titled "data_privacy_audit_report_2024-11-26_01-09-53.json". The file contains a structured audit report with sections for data collection, storage, access, and compliance. The code editor shows line numbers from 1 to 51. The "data_collection" section includes questions about user informed consent, data minimization, and consent mechanisms. The "data_storage" section includes questions about data encryption at rest and secure storage. The "data_access" section includes questions about role-based access restrictions, log maintenance, and strong authentication. The "compliance" section includes a question about GDPR compliance.

```
1  {
2      "timestamp": "2024-11-26_01-09-53",
3      "audit_results": [
4          {
5              "data_collection": [
6                  {
7                      "question": "Are users informed about data collection?",
8                      "response": "yes"
9                  },
10                 {
11                     "question": "Is data collection limited to what's necessary?",
12                     "response": "yes"
13                 },
14                 {
15                     "question": "Are consent mechanisms in place?",
16                     "response": "yes"
17                 }
18             ],
19             "data_storage": [
20                 {
21                     "question": "Is data encrypted at rest?",
22                     "response": "no"
23                 },
24                 {
25                     "question": "Is sensitive data stored securely?",
26                     "response": "yes"
27                 },
28                 {
29                     "question": "Are backup policies in place?",
30                     "response": "yes"
31                 }
32             ],
33             "data_access": [
34                 {
35                     "question": "Is access to data restricted based on roles?",
36                     "response": "yes"
37                 },
38                 {
39                     "question": "Are access logs maintained and monitored?",
40                     "response": "yes"
41                 },
42                 {
43                     "question": "Are strong authentication mechanisms used?",
44                     "response": "yes"
45                 }
46             ],
47             "compliance": [
48                 {
49                     "question": "Is the organization GDPR compliant?",
50                     "response": "yes"
51                 }
52             ]
53         }
54     ]
55 }
```

Practical 9:

Students needs to explore the requirements of the Data Protection Regulations and develop a plan for ensuring compliance with the regulation.

Source code:

The screenshot shows a code editor interface with multiple tabs open. The tabs include: tor.py, dummy_dictionary.txt, bruteforce.py, practical_8.py, practical_9.py (which is the active tab), and prac_10.py. The left sidebar displays a file tree under 'EXPLORER' and 'DATA PRIVACY PRACTICALS'. The 'DATA PRIVACY PRACTICALS' section contains files: bruteforce.py, caeser_cipher.py, data_privacy_audit_report_2024-11-26_01-0..., dummy_dictionary.txt, dummy_user_passwords.csv, password_breach_check.py, password_generator.py, password_sha256_hex.py, prac_10.py, practical_8.py, practical_9.py, and rail_cipher.py. The main editor area shows the content of the practical_9.py script:

```
1 import json
2 from datetime import datetime
3
4 # Define data protection regulation requirements
5 regulation_requirements = {
6     "GDPR": [
7         "data_processing": [
8             "Ensure lawful, fair, and transparent processing of personal data.",
9             "Obtain explicit consent from data subjects.",
10            "Provide data subjects with access to their data and the right to correct or de
11        ],
12        "data_security": [
13            "Implement appropriate technical and organizational measures.",
14            "Ensure encryption and pseudonymization of data.",
15            "Maintain data integrity and confidentiality."
16        ],
17        "compliance_monitoring": [
18            "Conduct regular data protection impact assessments (DPIA).",
19            "Maintain records of processing activities.",
20            "Appoint a Data Protection Officer (DPO) if required."
21        ],
22    ],
23    "HIPAA": [
24        "privacy_rule": [
25            "Ensure protected health information (PHI) is safeguarded.",
26            "Provide patients with rights over their PHI.",
27            "Limit disclosures of PHI to the minimum necessary."
28        ],
29        "security_rule": [
30            "Implement administrative safeguards (e.g., training, risk analysis).",
31            "Establish physical safeguards (e.g., facility access controls).",
32            "Use technical safeguards (e.g., encryption, access control)."
33        ],
34        "breach_notification_rule": [
35            "Notify affected individuals within 60 days of discovering a breach.",
36            "Report breaches affecting more than 500 individuals to the Department of Health
37        ],
38    ],
39 }
40
41 # Function to develop a compliance plan
42 def develop_compliance_plan(regulation, selected_requirements):
43     if regulation not in regulation_requirements:
44         print(f"Regulation '{regulation}' not recognized.")
45         return
46
47     print(f"\nDeveloping Compliance Plan for {regulation}...\n")
48     selected_plan = {}
49     for category, requirements in regulation_requirements[regulation].items():
50         if category in selected_requirements:
```

```
EXPLORER          tor.py    dummy_dictionary.txt  bruteforce.py  practical_8.py  practical_9.py  prac_10.py
OPEN EDITORS       practical_9.py > ...
rail_cipher.py
password_sha256_hex.py
password_breach_check.py
password_generator.py
dummy_dictionary.txt
bruteforce.py
practical_8.py
X practical_9.py
prac_10.py

DATA PRIVACY PRACTICALS
bruteforce.py
caeser_cipher.py
data_privacy_audit_report_2024-11-26_01-0...
dummy_dictionary.txt
dummy_user_passwords.csv
password_breach_check.py
password_generator.py
password_sha256_hex.py
prac_10.py
practical_8.py
practical_9.py
rail_cipher.py

practical_9.py
42 def develop_compliance_plan(regulation, selected_requirements):
50     if regulation in selected_requirements:
51         selected_plan[regulation] = requirements
52
53     return selected_plan
54
55 # Function to generate a compliance plan report
56 def generate_report(regulation, compliance_plan):
57     timestamp = datetime.now().strftime("%Y-%m-%d %H-%M-%S")
58     report_filename = f"compliance_plan_{regulation}_{timestamp}.json"
59
60     report_content = {
61         "timestamp": timestamp,
62         "regulation": regulation,
63         "compliance_plan": compliance_plan
64     }
65
66     with open(report_filename, "w") as report_file:
67         json.dump(report_content, report_file, indent=4)
68
69     print(f"\nCompliance plan report generated: {report_filename}")
70     return report_filename
71
72 # Main execution
73 if __name__ == "__main__":
74     print("Available Regulations:")
75     for regulation in regulation_requirements.keys():
76         print(f"- {regulation}")
77
78     regulation = input("\nEnter the regulation to comply with (e.g., GDPR, HIPAA): ").strip()
79
80     if regulation in regulation_requirements:
81         print(f"\nCategories for {regulation}:")
82         for category in regulation_requirements[regulation].keys():
83             print(f"- {category}")
84
85         selected_categories = input(
86             "\nEnter the categories to include in the compliance plan (comma-separated): "
87         ).strip().split(",")
88
89         selected_categories = [cat.strip().lower() for cat in selected_categories]
90         compliance_plan = develop_compliance_plan(regulation, selected_categories)
91
92         if compliance_plan:
93             print("\nCompliance Plan:")
94             for category, actions in compliance_plan.items():
95                 print(f"- {category.capitalize()}:")
96                 for action in actions:
97                     print(f"  * {action}")
98
99         generate_report(regulation, compliance_plan)
100    else:
101        print(f"Regulation '{regulation}' is not supported.")
```

```
EXPLORER          tor.py    dummy_dictionary.txt  bruteforce.py  practical_8.py  practical_9.py  prac_10.py
OPEN EDITORS       practical_9.py > ...
rail_cipher.py
password_sha256_hex.py
password_breach_check.py
password_generator.py
dummy_dictionary.txt
bruteforce.py
practical_8.py
X practical_9.py
prac_10.py

practical_9.py
98
99         generate_report(regulation, compliance_plan)
100    else:
101        print(f"Regulation '{regulation}' is not supported.")
```

Explanation:

This script helps organizations **develop and document compliance plans** for specific data protection regulations such as **GDPR** and **HIPAA**. Here's an explanation of its features and workflow:

1. Regulation Requirements

- The script defines compliance requirements for:
 - GDPR:** Covers data processing, security, and compliance monitoring.
 - HIPAA:** Focuses on privacy, security, and breach notification rules.

- Each regulation has categories like **data processing** or **security rule**, with detailed actions required for compliance.

2. Workflow

Step 1: User Input

- **Available Regulations:**
 - Lists supported regulations: **GDPR**, **HIPAA**, etc.
- **Regulation Selection:**
 - The user chooses a regulation to comply with (e.g., **GDPR**).
- **Category Selection:**
 - Displays available categories for the chosen regulation.
 - The user specifies the categories (e.g., **data_security**).

Step 2: Plan Development

- **develop_compliance_plan():**
 - Matches the selected categories with the regulation's requirements.
 - Returns a dictionary containing the compliance plan.

Step 3: Plan Report Generation

- **generate_report():**
 - Creates a timestamped JSON file documenting:
 - Selected regulation and categories.
 - Compliance actions for each category.
 - Example filename: compliance_plan_GDPR_2024-11-26_14-30-45.json.

Output:

```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\practical_9.py"
Available Regulations:
- GDPR
- HIPAA

Enter the regulation to comply with (e.g., GDPR, HIPAA): GDPR

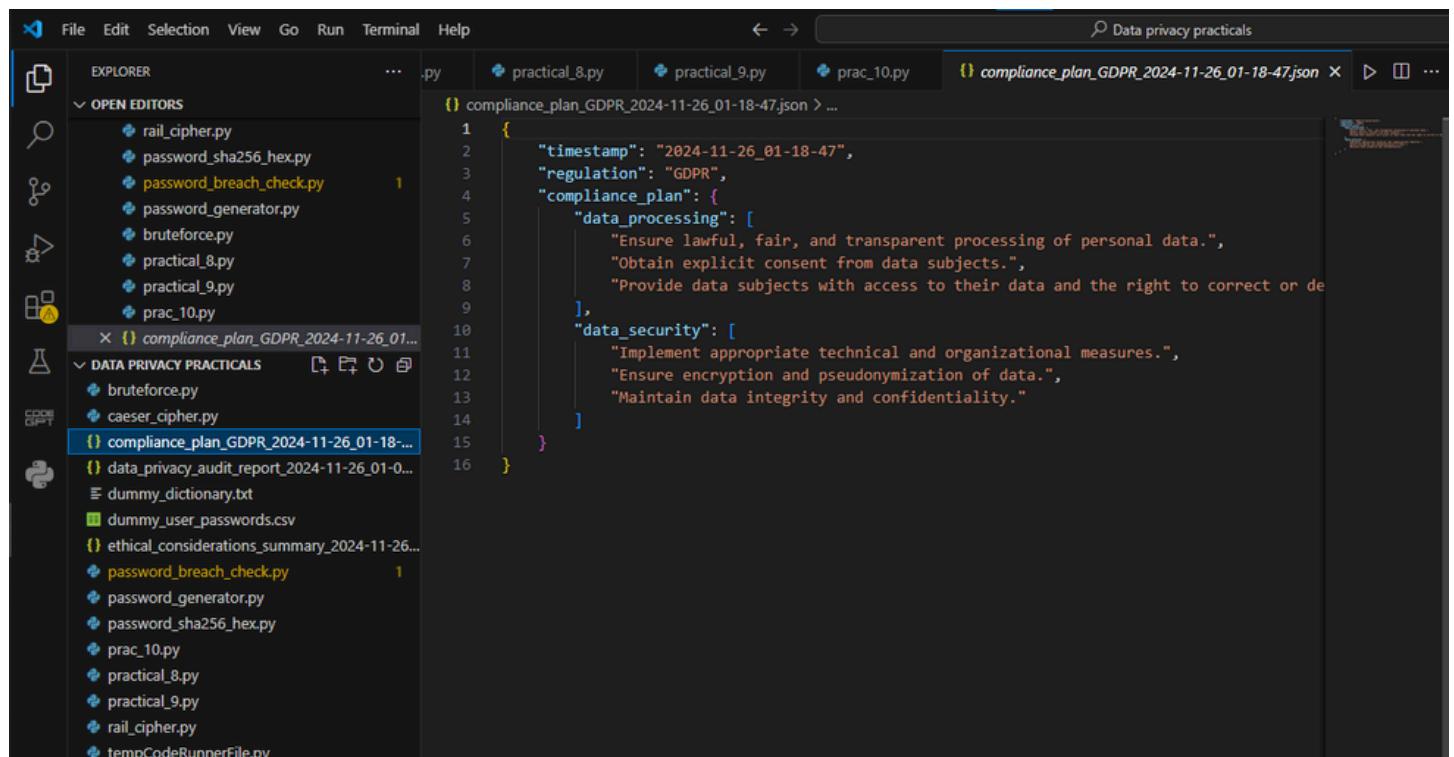
Categories for GDPR:
- data_processing
- data_security
- compliance_monitoring

Enter the categories to include in the compliance plan (comma-separated): data_processing,data_security

Developing Compliance Plan for GDPR...

Compliance Plan:
- Data_processing:
  * Ensure lawful, fair, and transparent processing of personal data.
  * Obtain explicit consent from data subjects.
  * Provide data subjects with access to their data and the right to correct or delete it.
- Data_security:
  * Implement appropriate technical and organizational measures.
  * Ensure encryption and pseudonymization of data.
  * Maintain data integrity and confidentiality.

Compliance plan report generated: compliance_plan_GDPR_2024-11-26_01-18-47.json
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals>
```



The screenshot shows a code editor interface with the following details:

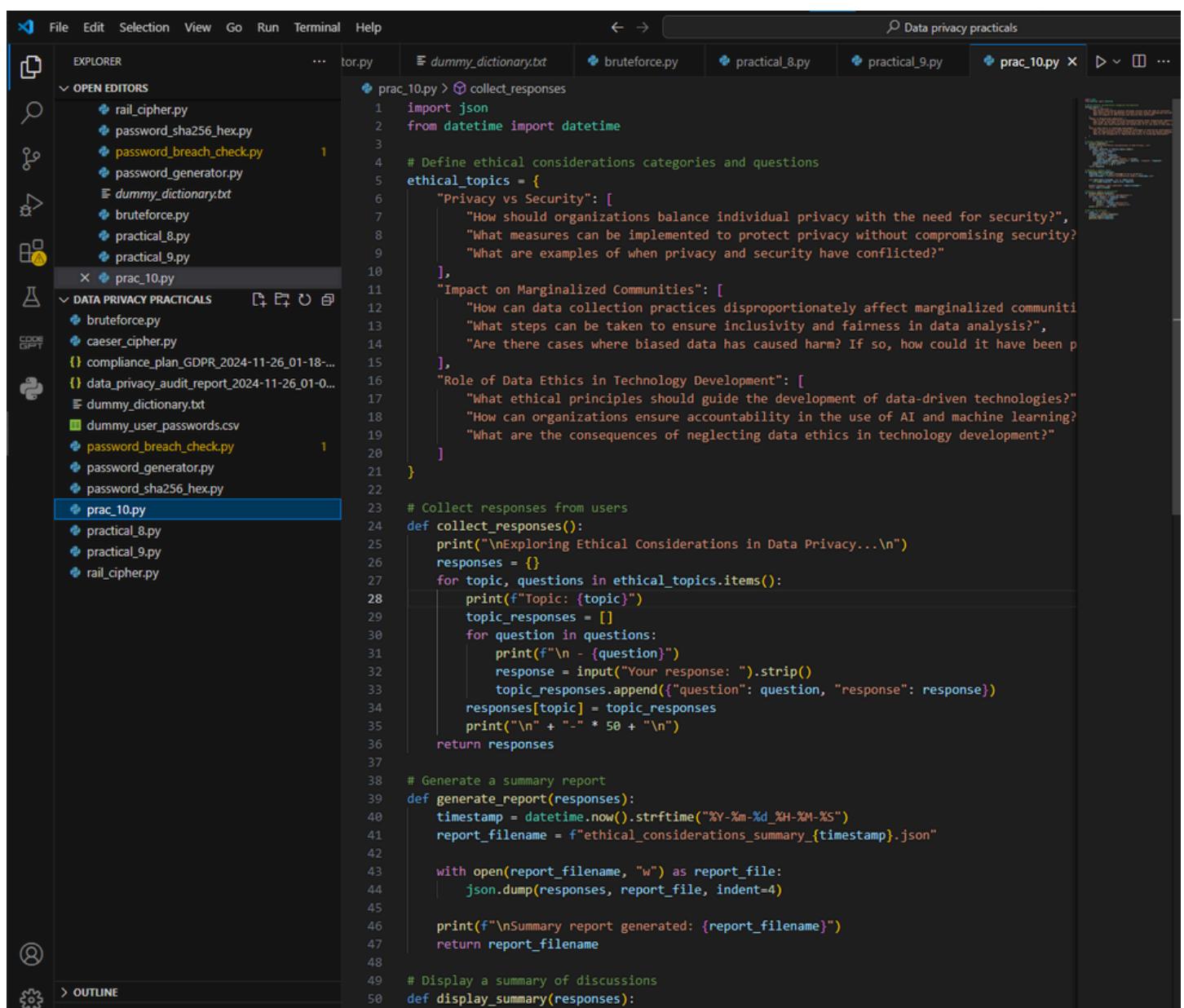
- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Data privacy practicals
- Explorer:** Shows files and folders:
 - OPEN EDITORS: compliance_plan_GDPR_2024-11-26_01-18-47.json
 - DATA PRIVACY PRACTICALS:
 - bruteforce.py
 - caeser_cipher.py
 - compliance_plan_GDPR_2024-11-26_01-18-47.json
 - data_privacy_audit_report_2024-11-26_01-0...
 - dummy_dictionary.txt
 - dummy_user_passwords.csv
 - ethical_considerations_summary_2024-11-26...
 - password_breach_check.py
 - password_generator.py
 - password_sha256_hex.py
 - prac_10.py
 - practical_8.py
 - practical_9.py
 - rail_cipher.py
 - tempCodeRunnerFile.py
- Code Editor:** Displays the JSON content of the compliance plan file:

```
1  {
2      "timestamp": "2024-11-26_01-18-47",
3      "regulation": "GDPR",
4      "compliance_plan": {
5          "data_processing": [
6              "Ensure lawful, fair, and transparent processing of personal data.",
7              "Obtain explicit consent from data subjects.",
8              "Provide data subjects with access to their data and the right to correct or delete it."
9          ],
10         "data_security": [
11             "Implement appropriate technical and organizational measures.",
12             "Ensure encryption and pseudonymization of data.",
13             "Maintain data integrity and confidentiality."
14         ]
15     }
16 }
```

Practical 10:

Students needs to explore ethical considerations in data privacy, such as the balance between privacy and security, the impact of data collection and analysis on marginalized communities, and the role of data ethics in technology development.

Source code:



The screenshot shows a code editor interface with the title "Data privacy practicals". The left sidebar displays a file tree under "EXPLORER" with several files listed, including "prac_10.py" which is currently selected. The main editor area contains the following Python code:

```
File Edit Selection View Go Run Terminal Help
EXPLORER OPEN EDITORS DATA PRIVACY PRACTICALS CODE DRAFTS
prac_10.py > collect_responses
1 import json
2 from datetime import datetime
3
4 # Define ethical considerations categories and questions
5 ethical_topics = {
6     "Privacy vs Security": [
7         "How should organizations balance individual privacy with the need for security?",
8         "What measures can be implemented to protect privacy without compromising security?",
9         "What are examples of when privacy and security have conflicted?"
10    ],
11    "Impact on Marginalized Communities": [
12        "How can data collection practices disproportionately affect marginalized communities?",
13        "What steps can be taken to ensure inclusivity and fairness in data analysis?",
14        "Are there cases where biased data has caused harm? If so, how could it have been prevented?"
15    ],
16    "Role of Data Ethics in Technology Development": [
17        "What ethical principles should guide the development of data-driven technologies?",
18        "How can organizations ensure accountability in the use of AI and machine learning?",
19        "What are the consequences of neglecting data ethics in technology development?"
20    ]
21 }
22
23 # Collect responses from users
24 def collect_responses():
25     print("\nExploring Ethical Considerations in Data Privacy...\n")
26     responses = {}
27     for topic, questions in ethical_topics.items():
28         print(f"\nTopic: {topic}")
29         topic_responses = []
30         for question in questions:
31             print(f"\n - {question}")
32             response = input("Your response: ").strip()
33             topic_responses.append({"question": question, "response": response})
34         responses[topic] = topic_responses
35         print("\n" + "-" * 50 + "\n")
36     return responses
37
38 # Generate a summary report
39 def generate_report(responses):
40     timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
41     report_filename = f"ethical_considerations_summary_{timestamp}.json"
42
43     with open(report_filename, "w") as report_file:
44         json.dump(responses, report_file, indent=4)
45
46     print(f"\nSummary report generated: {report_filename}")
47     return report_filename
48
49 # Display a summary of discussions
50 def display_summary(responses):
```

The screenshot shows a code editor interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** Data privacy practicals.
- Explorer:** Shows a tree view of files and folders. Under "OPEN EDITORS", "prac_10.py" is selected. Under "DATA PRIVACY PRACTICALS", there are several files: "bruteforce.py", "caeser_cipher.py", "compliance_plan_GDPR_2024-11-26_01-18...", "data_privacy_audit_report_2024-11-26_01-0...", "dummy_dictionary.txt", "dummy_user_passwords.csv", "password_breach_check.py", "password_generator.py", "password_sha256_hex.py", and "prac_10.py".
- Editor Area:** The code for "prac_10.py" is displayed:

```
prac_10.py > collect_responses
49 # Display a summary of discussions
50 def display_summary(responses):
51     print("\nSummary of Ethical Considerations:")
52     for topic, answers in responses.items():
53         print(f"\nTopic: {topic}")
54         for answer in answers:
55             print(f" - {answer['question']}")
56             print(f" * {answer['response']}")
57     print("\n" + "-" * 50 + "\n")
58
59 # Main script execution
60 if __name__ == "__main__":
61     responses = collect_responses()
62     display_summary(responses)
63     generate_report(responses)
```

Explanation:

This script facilitates a structured **discussion on ethical considerations** in data privacy, guiding users through key topics and generating a report of their responses.

1. Ethical Topics

The script focuses on three critical areas:

1. Privacy vs Security:

- Balancing privacy and security in organizational practices.
- Examples of conflicts and strategies for resolution.

2. Impact on Marginalized Communities:

- Addressing biases in data collection and analysis.
- Ensuring inclusivity and preventing harm.

3. Role of Data Ethics in Technology Development:

- Ethical principles guiding AI/ML.
- Accountability and consequences of neglecting data ethics.

Each topic contains thought-provoking questions to gather detailed insights.

2. Workflow

Step 1: Collect User Responses

- **collect_responses():**

- Iterates through topics and their questions.
- Prompts the user to provide answers, which are stored in a structured format.

Step 2: Display Summary

- **display_summary():**
 - Summarizes the discussion by displaying:
 - Topics.
 - Questions and the user's responses for each topic.

Step 3: Generate Report

- **generate_report():**
 - Creates a timestamped JSON report with the collected responses.
 - Example filename: ethical_considerations_summary_2024-11-26_15-10-22.json.

Output:

```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\tempCodeRunnerFile.py"

Exploring Ethical Considerations in Data Privacy...

Topic: Privacy vs Security

- How should organizations balance individual privacy with the need for security?
Your response: Organizations should use techniques like anonymization to ensure privacy while addressing security concerns.

- What measures can be implemented to protect privacy without compromising security?
Your response: Use encryption, minimize data collection, and conduct regular privacy audits.

- What are examples of when privacy and security have conflicted?
Your response: The Apple-FBI case in 2016 is a key example where unlocking a device for security conflicted with user privacy.
```

Topic: Impact on Marginalized Communities

- How can data collection practices disproportionately affect marginalized communities?

Your response: Data collection practices can perpetuate systemic biases when datasets are not representative of diverse populations. For example, facial recognition systems often misidentify individuals from marginalized racial or ethnic groups due to lack of training data diversity. Additionally, excessive surveillance in low-income neighborhoods can lead to stigmatization and unequal treatment.

- What steps can be taken to ensure inclusivity and fairness in data analysis?

Your response: Inclusivity can be achieved by ensuring datasets are diverse and representative of all demographic groups. Implementing bias detection tools, involving marginalized communities in decision-making, and conducting fairness audits regularly can further promote equitable outcomes in data analysis.

- Are there cases where biased data has caused harm? If so, how could it have been prevented?

Your response: A notable example is the use of biased algorithms in criminal justice systems, such as COMPAS, which disproportionately labeled Black defendants as high-risk for recidivism. This could have been prevented by using transparent methodologies, auditing algorithms for racial biases, and incorporating domain experts in the review process.

Topic: Role of Data Ethics in Technology Development

- What ethical principles should guide the development of data-driven technologies?

* Principles such as fairness, transparency, accountability, and respect for user privacy should guide technology development. Developers should ensure technologies do not discriminate, protect user rights, and adhere to relevant regulations like GDPR or HIPAA.

- How can organizations ensure accountability in the use of AI and machine learning?

* Organizations can ensure accountability by maintaining clear documentation of model development, auditing AI systems for unintended biases, and involving external reviewers for ethical assessments. Appointing ethics officers and establishing review boards can also enforce accountability.

- What are the consequences of neglecting data ethics in technology development?

* Neglecting data ethics can lead to widespread harm, such as discrimination, privacy breaches, and loss of public trust. For instance, unethical data usage by Cambridge Analytica significantly impacted trust in social media platforms. Legal consequences, such as lawsuits and fines, can also arise from non-compliance with regulations.

The screenshot shows a code editor interface with several files open in the Explorer and Editor panes.

OPEN EDITORS:

- practical_8.py
- practical_9.py
- prac_10.py
- ethical_considerations_summary_2024-11-26_01-32-37.json

DATA PRIVACY PRACTICALS:

- bruteforce.py
- caeser_cipher.py
- compliance_plan_GDPR_2024-11-26_01-18-37.json
- data_privacy_audit_report_2024-11-26_01-0-37.json
- dummy_dictionary.txt
- dummy_user_passwords.csv
- ethical_considerations_summary_2024-11-26_01-32-37.json
- password_breach_check.py
- password_generator.py
- password_sha256_hex.py
- prac_10.py
- practical_8.py
- practical_9.py
- rail_cipher.py
- tempCodeRunnerFile.py

ethical_considerations_summary_2024-11-26_01-32-37.json Content:

```
1  {
2      "Privacy vs Security": [
3          {
4              "question": "How should organizations balance individual privacy with the need for security?",
5              "response": "Organizations should use techniques like anonymization to ensure privacy while maintaining security measures like encryption and access controls."
6          },
7          {
8              "question": "What measures can be implemented to protect privacy without compromising security?",
9              "response": "Use encryption, minimize data collection, and conduct regular privacy audits to protect privacy without significantly impacting security."
10         },
11         {
12             "question": "What are examples of when privacy and security have conflicted?",  
13             "response": "The Apple-FBI case in 2016 is a key example where unlocking a device for law enforcement purposes conflicted with user privacy rights."  
14         }
15     ],
16     "Impact on Marginalized Communities": [
17         {
18             "question": "How can data collection practices disproportionately affect marginalized communities?",  
19             "response": "Data collection practices can perpetuate systemic biases when data is collected from diverse populations, leading to unequal treatment and outcomes."  
20         },
21         {
22             "question": "What steps can be taken to ensure inclusivity and fairness in data collection?",  
23             "response": "Inclusivity can be achieved by ensuring datasets are diverse and representative, and by implementing fair machine learning models that do not discriminate based on protected characteristics."  
24         },
25         {
26             "question": "Are there cases where biased data has caused harm? If so, how could this be addressed?",  
27             "response": "A notable example is the use of biased algorithms in criminal justice systems, which can lead to discriminatory outcomes for certain demographic groups."  
28         }
29     ],
30     "Role of Data Ethics in Technology Development": [
31         {
32             "question": "What ethical principles should guide the development of data-driven technologies?",  
33             "response": "Principles such as fairness, transparency, accountability, and respect for individual privacy should guide the development of ethical AI systems."  
34         },
35         {
36             "question": "How can organizations ensure accountability in the use of AI and machine learning models?",  
37             "response": "Organizations can ensure accountability by maintaining clear documentation of their AI processes, conducting regular audits, and being transparent about how AI decisions are made."  
38         },
39         {
40             "question": "What are the consequences of neglecting data ethics in technology development?",  
41             "response": "Neglecting data ethics can lead to widespread harm, such as discrimination, privacy violations, and loss of trust in AI systems."  
42         }
43     ]
44 }
```

Complexities of each problem:

Code Description	Time Complexity	Space Complexity	Explanation
Caesar Cipher Encryption/Decryption	$O(n)$	$O(n)$	Iterates over the plaintext/ciphertext of length n. Requires space proportional to the output text.
Rail Fence Cipher Encryption/Decryption	$O(n)$	$O(n)$	Iterates through the plaintext for zigzag filling and reading. Space required for the zigzag pattern matrix and final output.
SHA-256 Hashing	$O(n)$	$O(1)$	Iterates through the password to compute the hash. Space usage is constant as the hash output size is fixed.
Checking Password Breach using API	$O(n)+O(m)$	$O(1)$	Computes SHA-1 hash in $O(n)O(n)$, and searches the API response in $O(m)$, where m is the length of the API response.
Password Generation from Dictionary	$O(k)$	$O(k)$	Randomly selects k words from the dictionary file. Space depends on storing k words for the password.

Brute-Force Attack on Password	$O(cl)$	$O(l)$	Tries all combinations up to length l using a character set of size c . Space used for the generated password attempt.
Data Privacy Audit with Vulnerabilities Analysis	$O(q)$	$O(q)$	Iterates through q questions to collect responses. Space depends on storing the responses and vulnerabilities.
Compliance Plan Development and Report Generation	$O(r+a)$	$O(r+a)$	Parses r regulation requirements and generates a compliance plan with a actions. Space for storing the plan and output.
Ethical Considerations Discussion and Summary Report	$O(q)$	$O(q)$	Processes q ethical questions. Space depends on storing responses and generating the summary report.