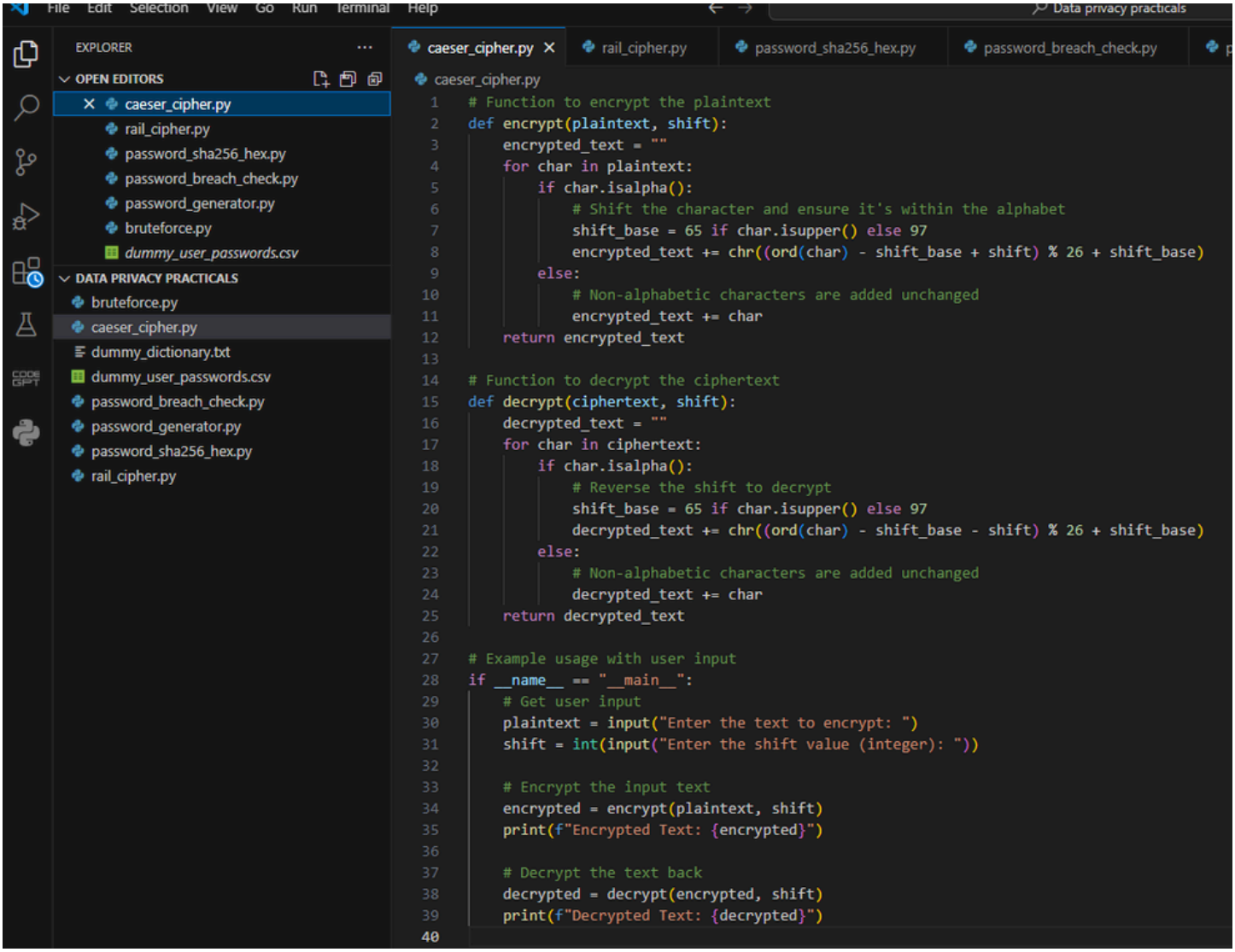
Data Privacy Assignment

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:



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:

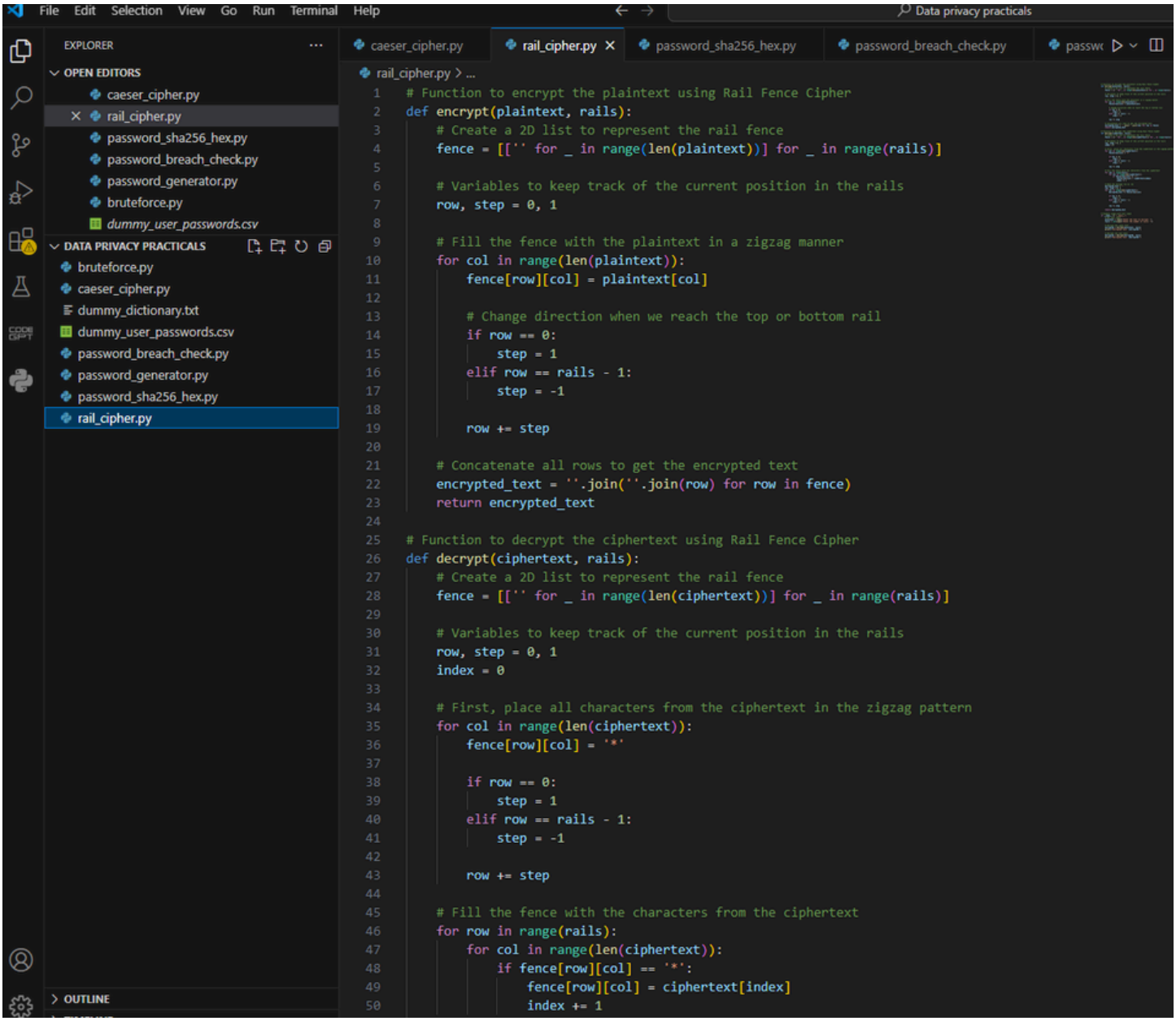
```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d
:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\caeser_cipher.py"

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

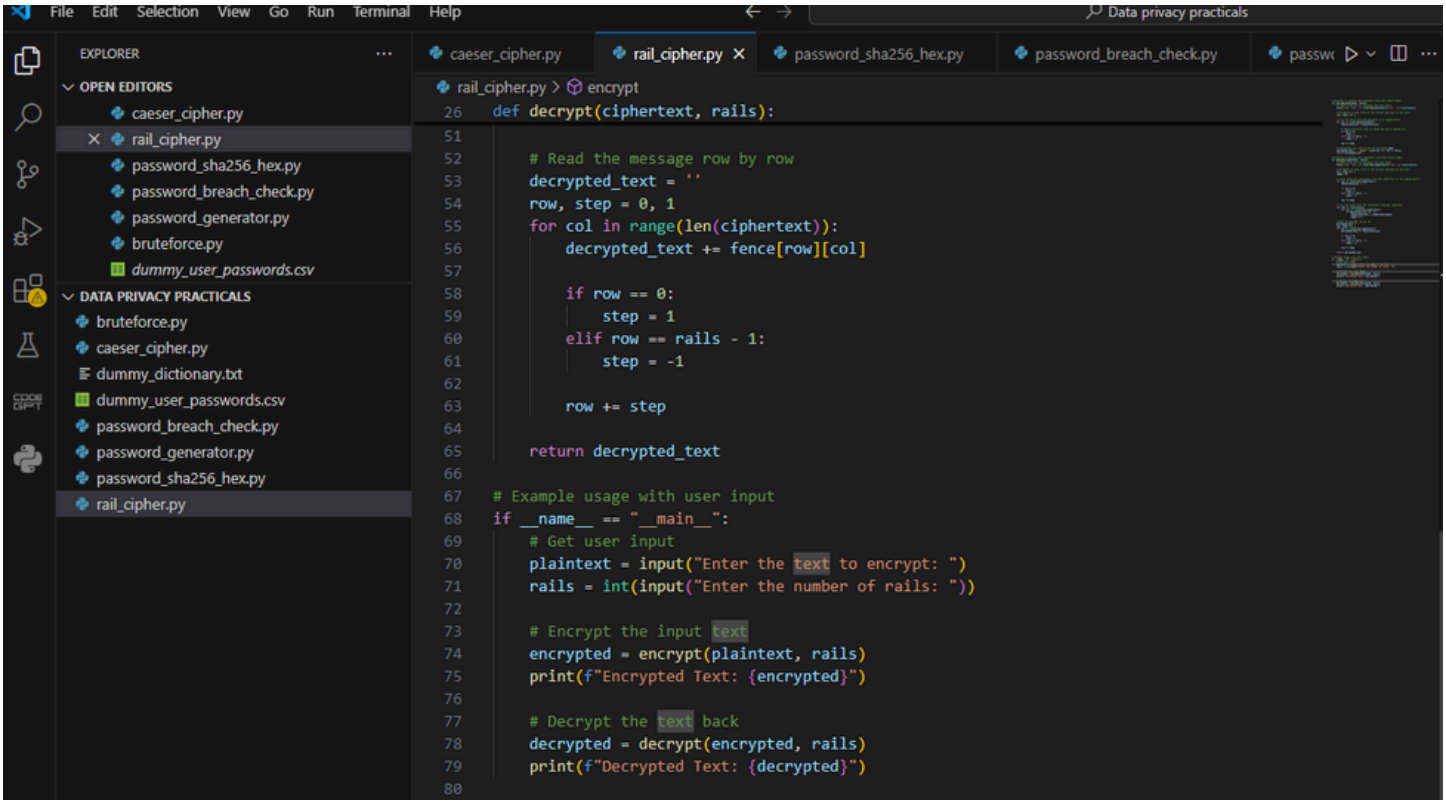
```
Enter the text to encrypt: Caesar cipher
Enter the shift value (integer): 5
Encrypted Text: Hfjxfw hnumjw
Decrypted Text: Caesar cipher
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals>
```

Practical 2:

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

Source code:





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:

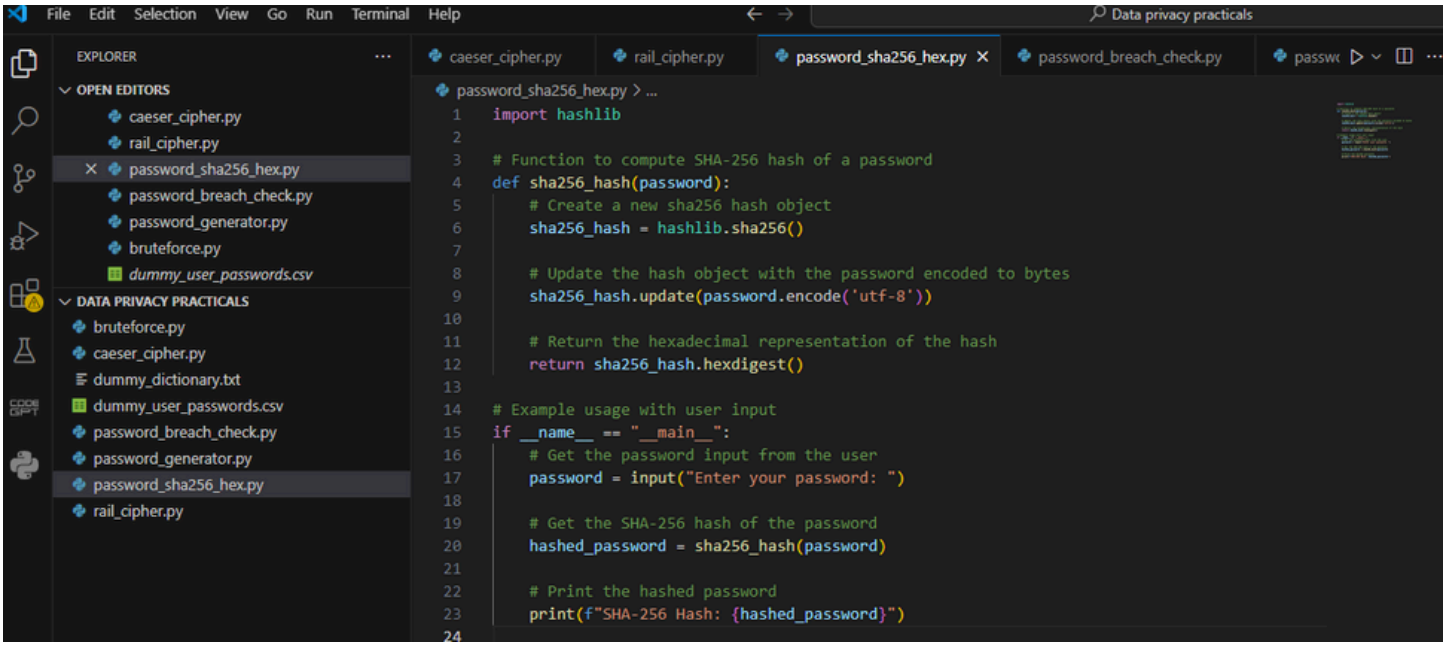
```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d
:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\rail_cipher.py"
Enter the text to encrypt: mumbai indians
Enter the number of rails: 4
Encrypted Text: m nuiiasmanibd
Decrypted Text: mumbai indians
PS D:\Study\B.Sc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



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>
```

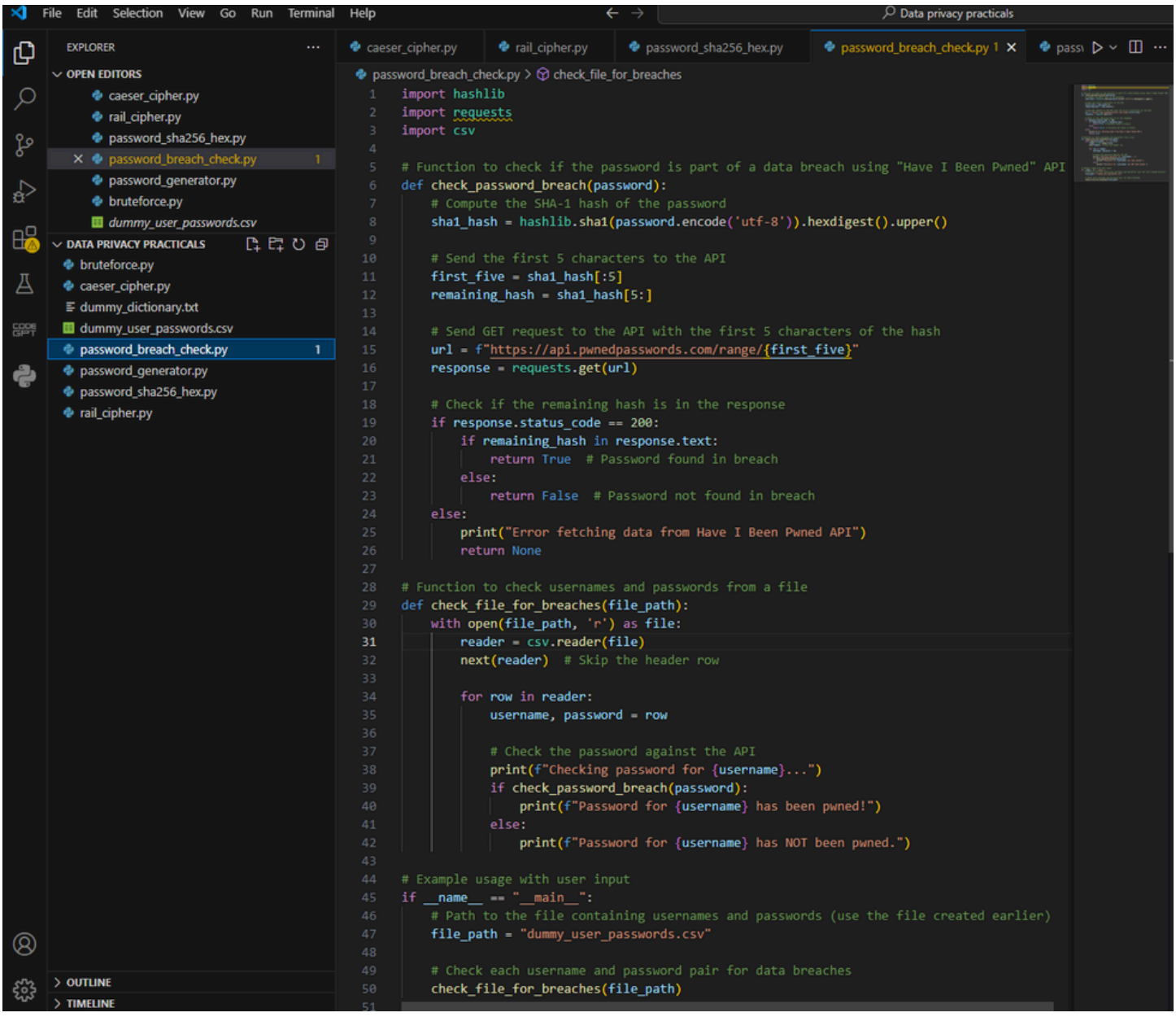
```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d
:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\password_sha256_h
ex.py"
Enter your password: qwerty123
SHA-256 Hash: daaad6e5604e8e17bd9f108d91e26afe6281dac8fda0091040a7a6d7bd9b43
b5
PS D:\Study\B.Sc (Hons.) 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:



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):

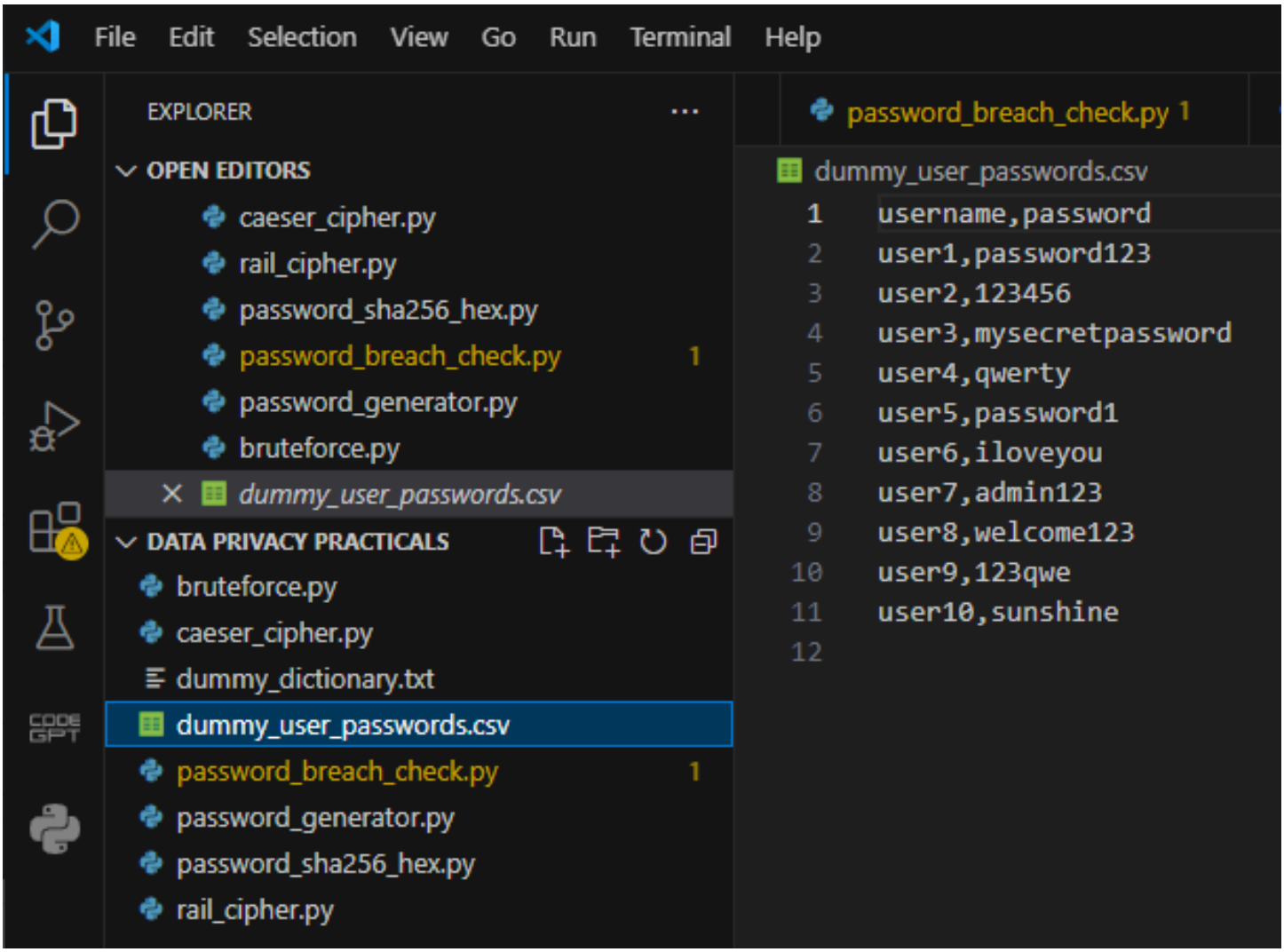
- **Password Hashing**:
  - The password is hashed using **SHA-1**. This converts the password into a 40-character hexadecimal string.

    - 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:



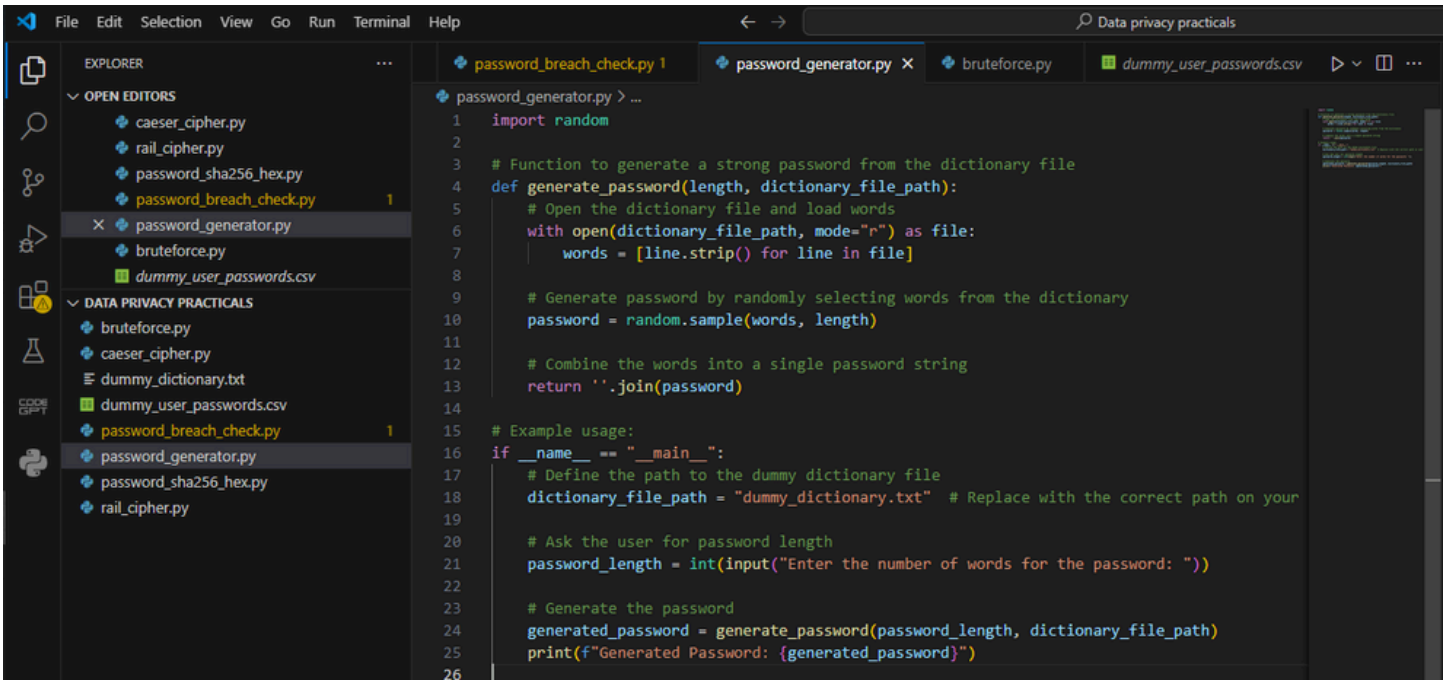
Outputs:

```
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:



Explanation:

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

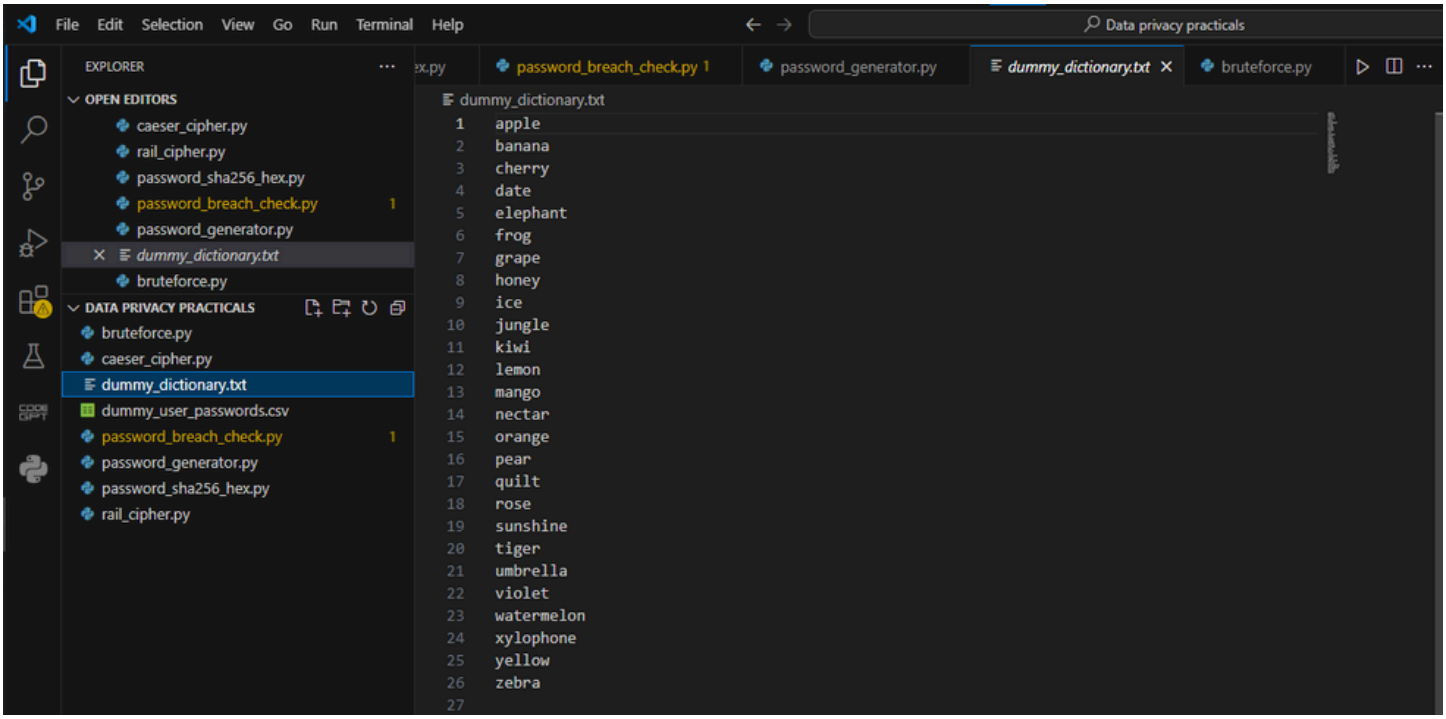
1. Function to Generate a Password (generate_password):

- **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.

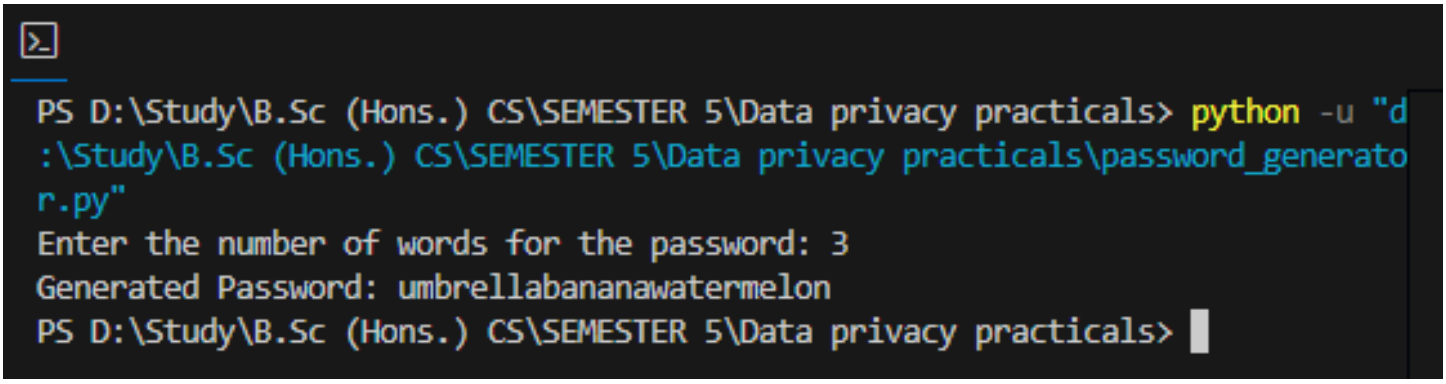
2. Example Usage:

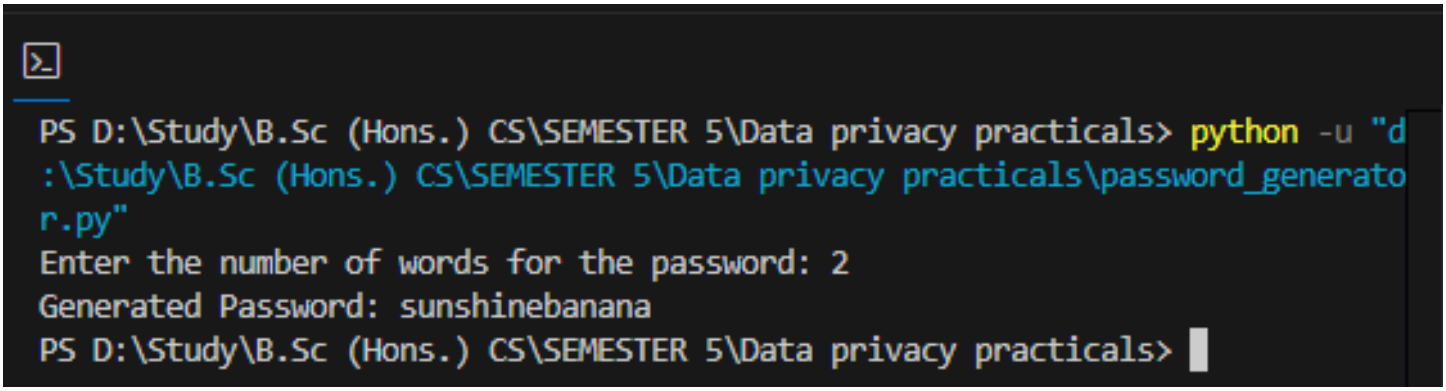
- The script asks the user to specify the number of words to include in the password.
- It then generates a password using the specified word count and prints it.

Dictionary file:



Outputs:



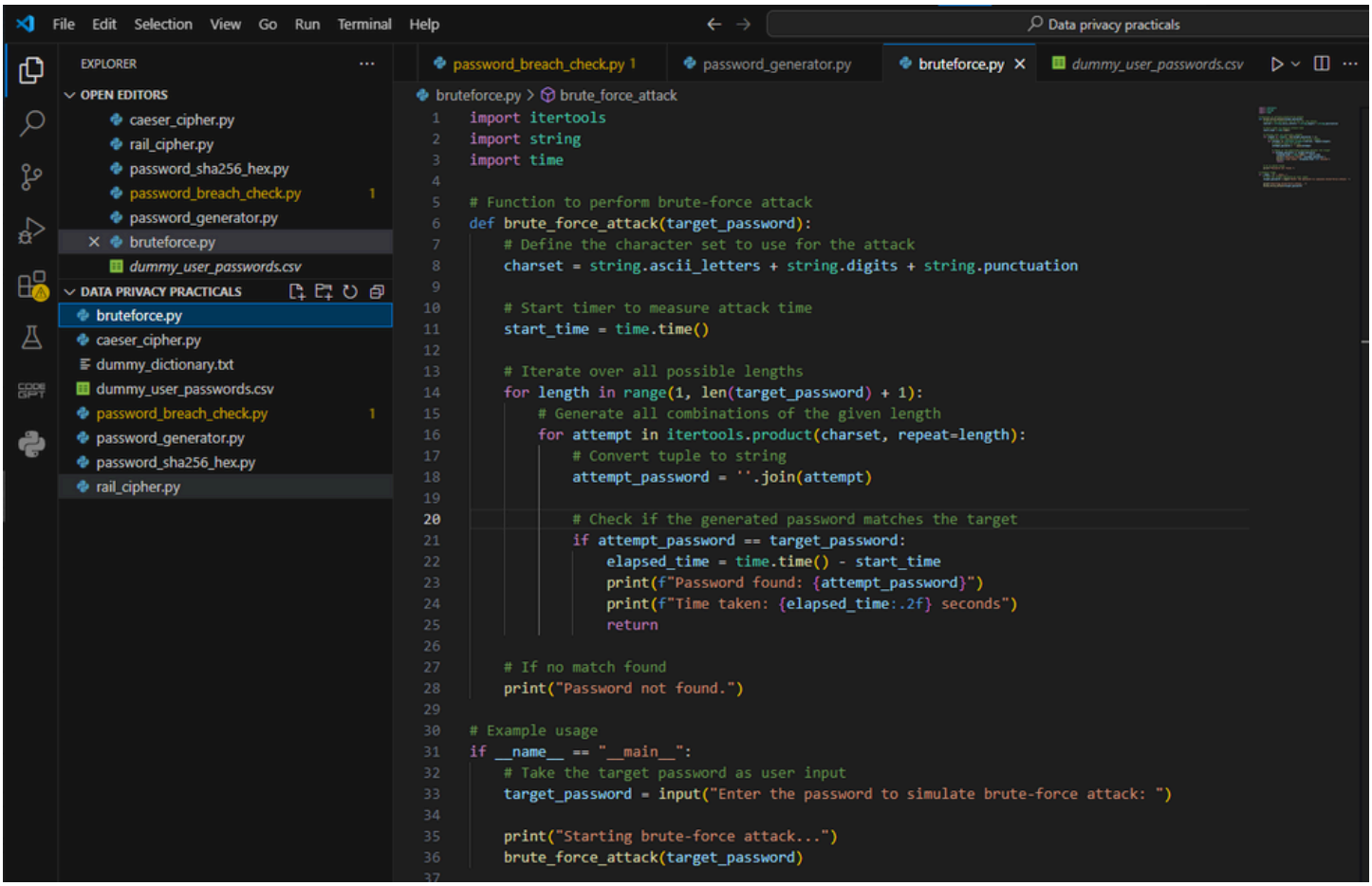


Practical 6:

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

combinations.

Source code:



Explanation:

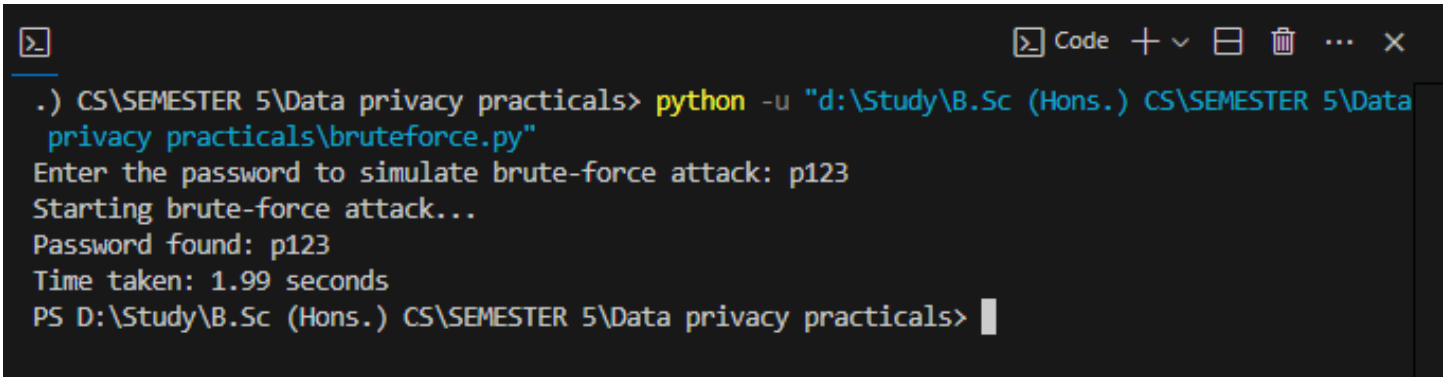
This program simulates a **brute-force attack**, attempting to guess a target password by systematically trying every possible combination of characters.

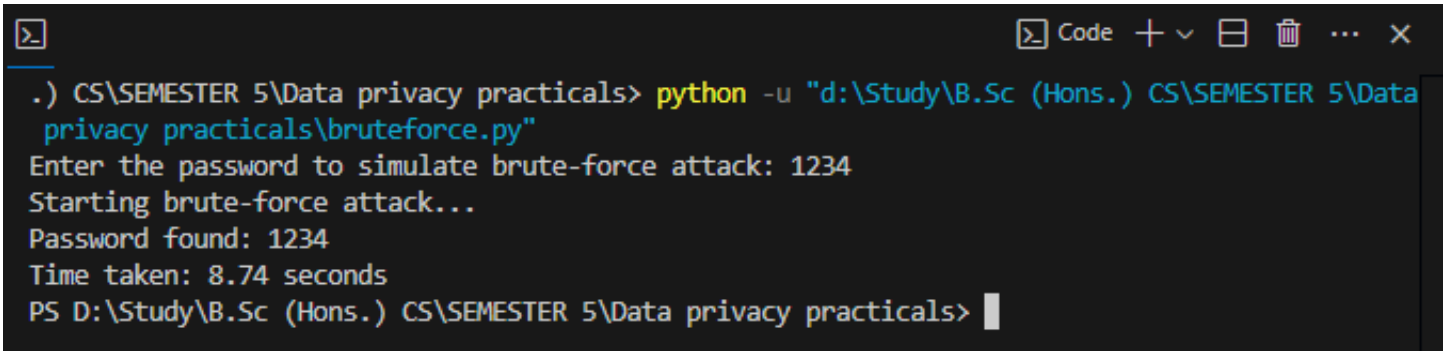
1. Function to Perform the Brute-Force Attack (brute_force_attack):

- **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:

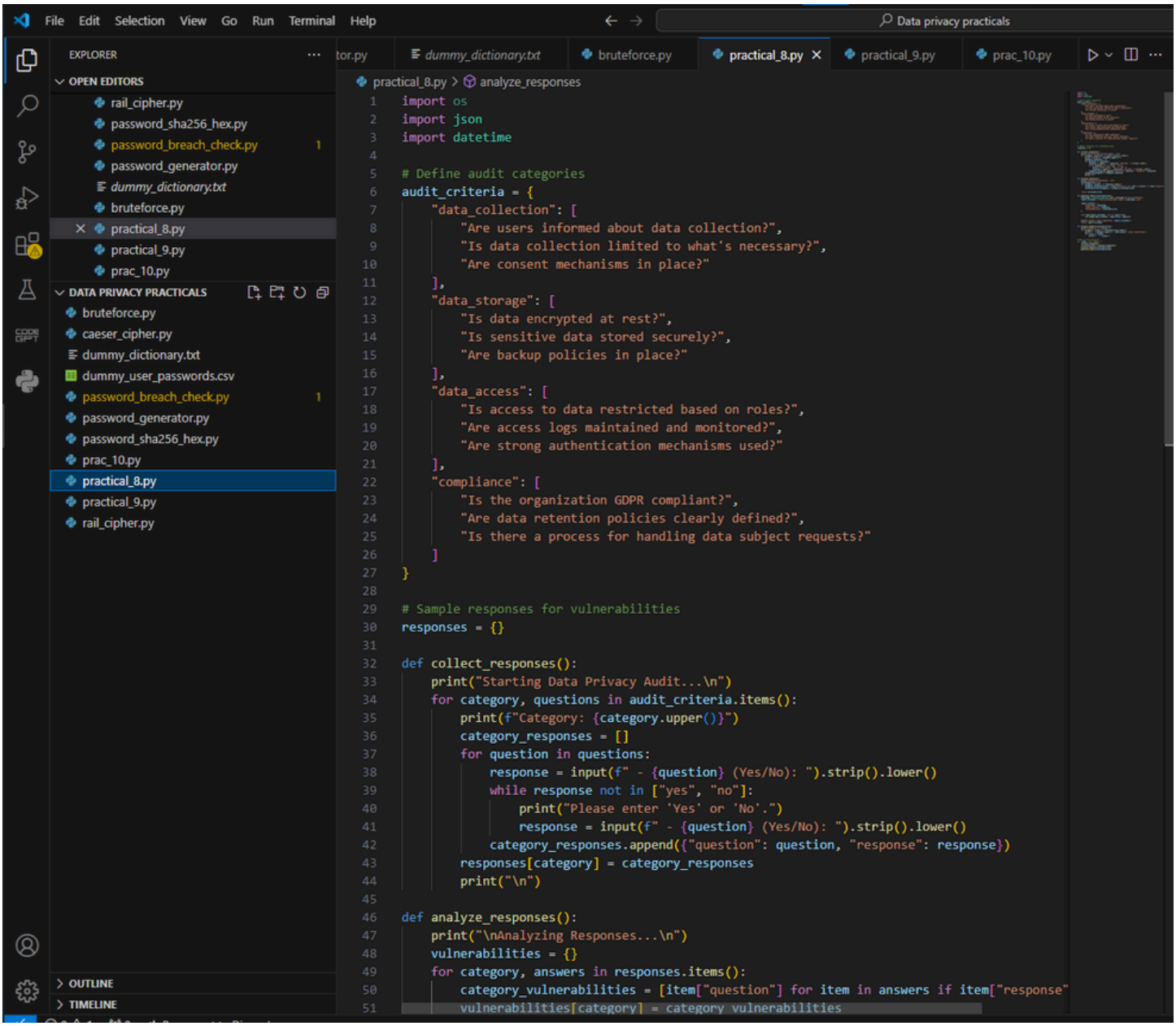


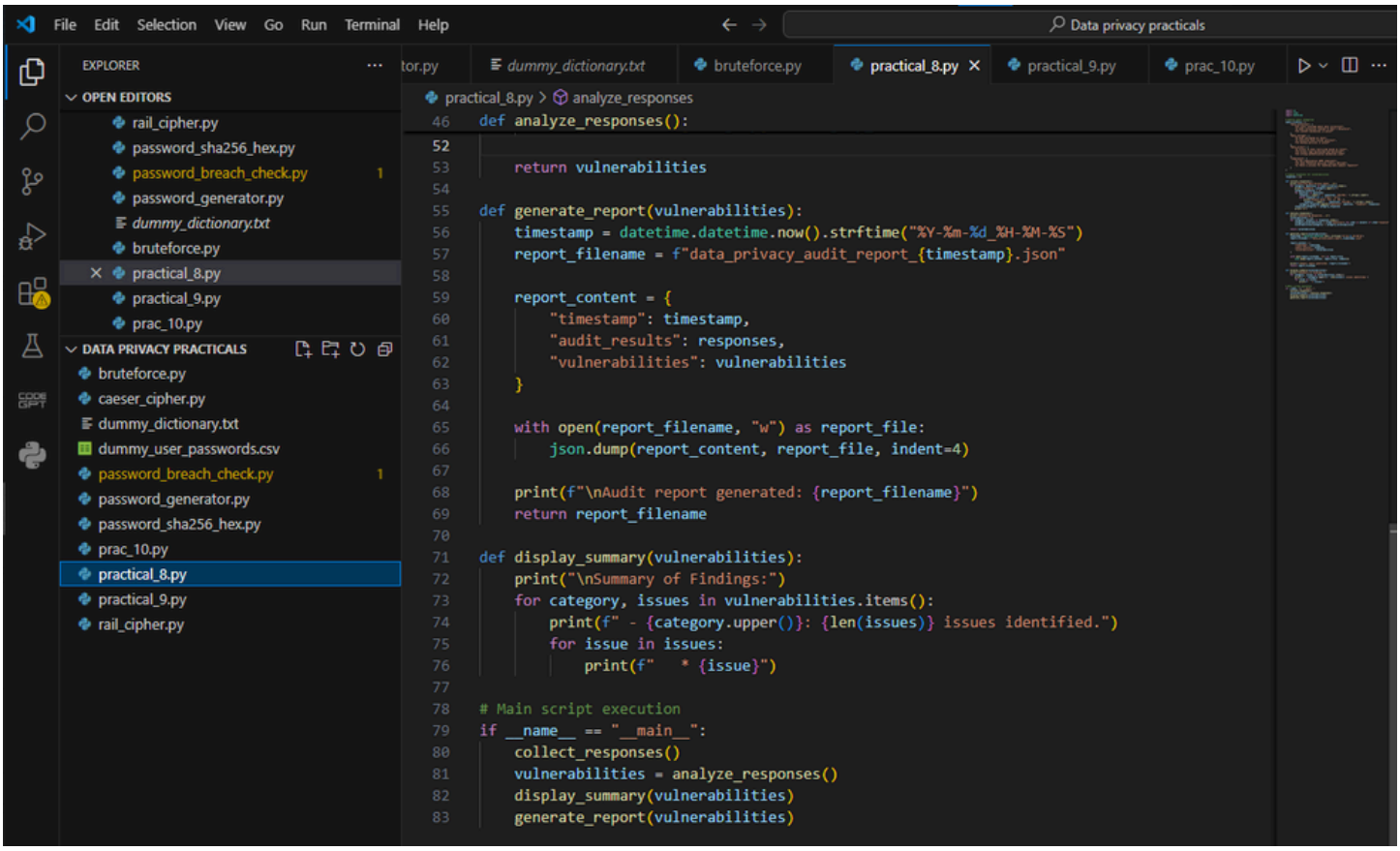


Practical 8:

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

Source code:


File Edit Selection View Go Run Terminal Help
Data privacy practicals
EXPLORER
OPEN EDITORS
rail_cipher.py
password_sha256_hex.py
password_breach_check.py 1
password_generator.py
dummy_dictionary.txt
bruteforce.py
practical_8.py
practical_9.py
prac_10.py
DATA PRIVACY PRACTICALS
bruteforce.py
caeser_cipher.py
dummy_dictionary.txt
dummy_user_passwords.csv
password_breach_check.py 1
password_generator.py
password_sha256_hex.py
prac_10.py
practical_8.py
practical_9.py
rail_cipher.py
OUTLINE
TIMELINE
tor.py
dummy_dictionary.txt
bruteforce.py
practical_8.py
practical_9.py
prac_10.py
practical_8.py > analyze_responses
import os
import json
import datetime
# Define audit categories
audit_criteria = {
"data_collection": [
"Are users informed about data collection?",
"Is data collection limited to what's necessary?",
"Are consent mechanisms in place?"
],
"data_storage": [
"Is data encrypted at rest?",
"Is sensitive data stored securely?",
"Are backup policies in place?"
],
"data_access": [
"Is access to data restricted based on roles?",
"Are access logs maintained and monitored?",
"Are strong authentication mechanisms used?"
],
"compliance": [
"Is the organization GDPR compliant?",
"Are data retention policies clearly defined?",
"Is there a process for handling data subject requests?"
]
}
# Sample responses for vulnerabilities
responses = {}
def collect_responses():
print("Starting Data Privacy Audit...\n")
for category, questions in audit_criteria.items():
print(f"Category: {category.upper()}")
category_responses = []
for question in questions:
response = input(f" - {question} (Yes/No): ").strip().lower()
while response not in ["yes", "no"]:
print("Please enter 'Yes' or 'No'.")
response = input(f" - {question} (Yes/No): ").strip().lower()
category_responses.append({"question": question, "response": response})
responses[category] = category_responses
print("\n")
def analyze_responses():
print("\nAnalyzing Responses...\n")
vulnerabilities = {}
for category, answers in responses.items():
category_vulnerabilities = [item["question"] for item in answers if item["response"]
vulnerabilities[category] = category_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

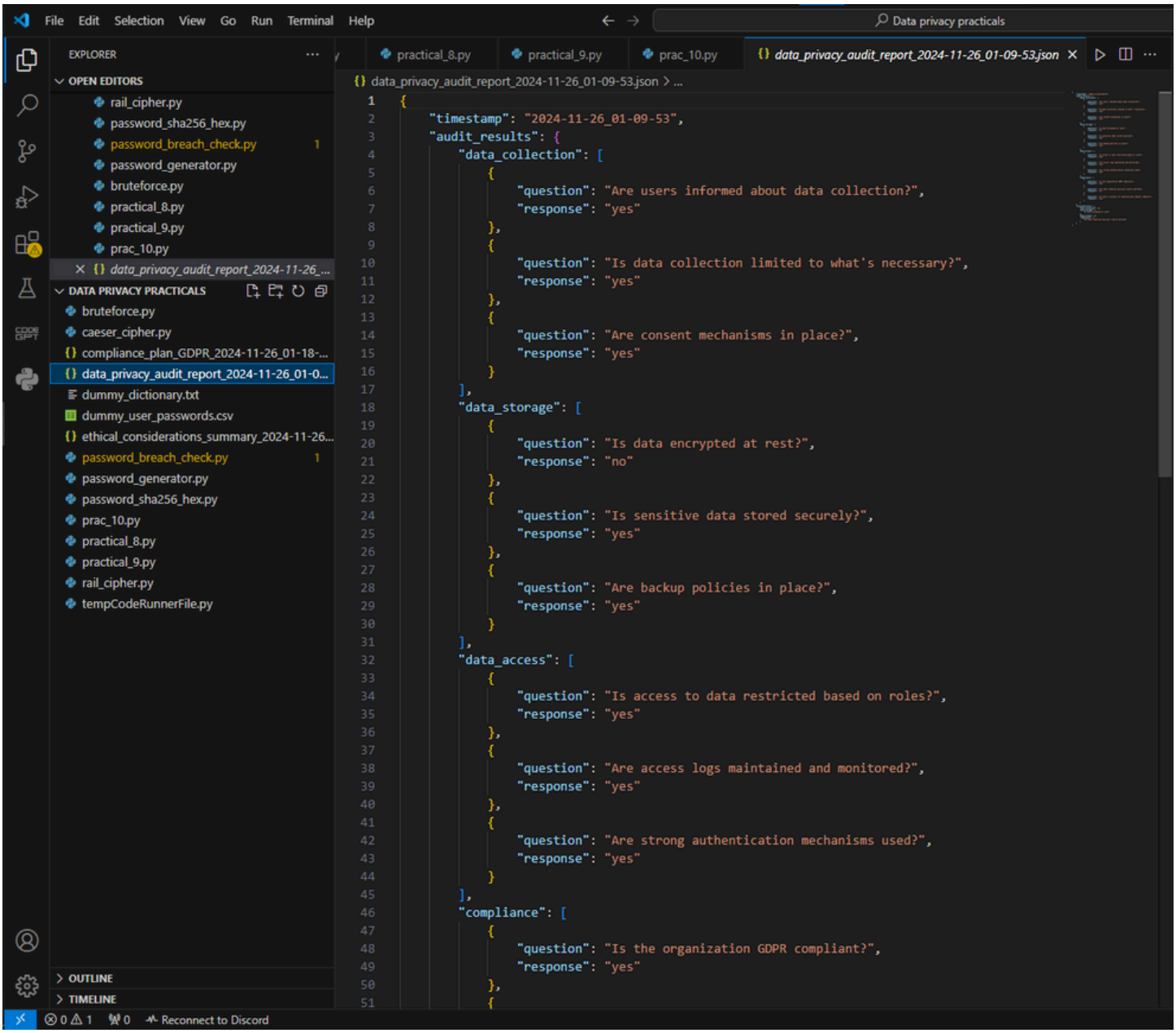
- **display_summary()**:
  - Prints a concise summary of vulnerabilities by category.
  - For each category, it lists:
    - The number of identified issues.
    - Each specific issue.

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:

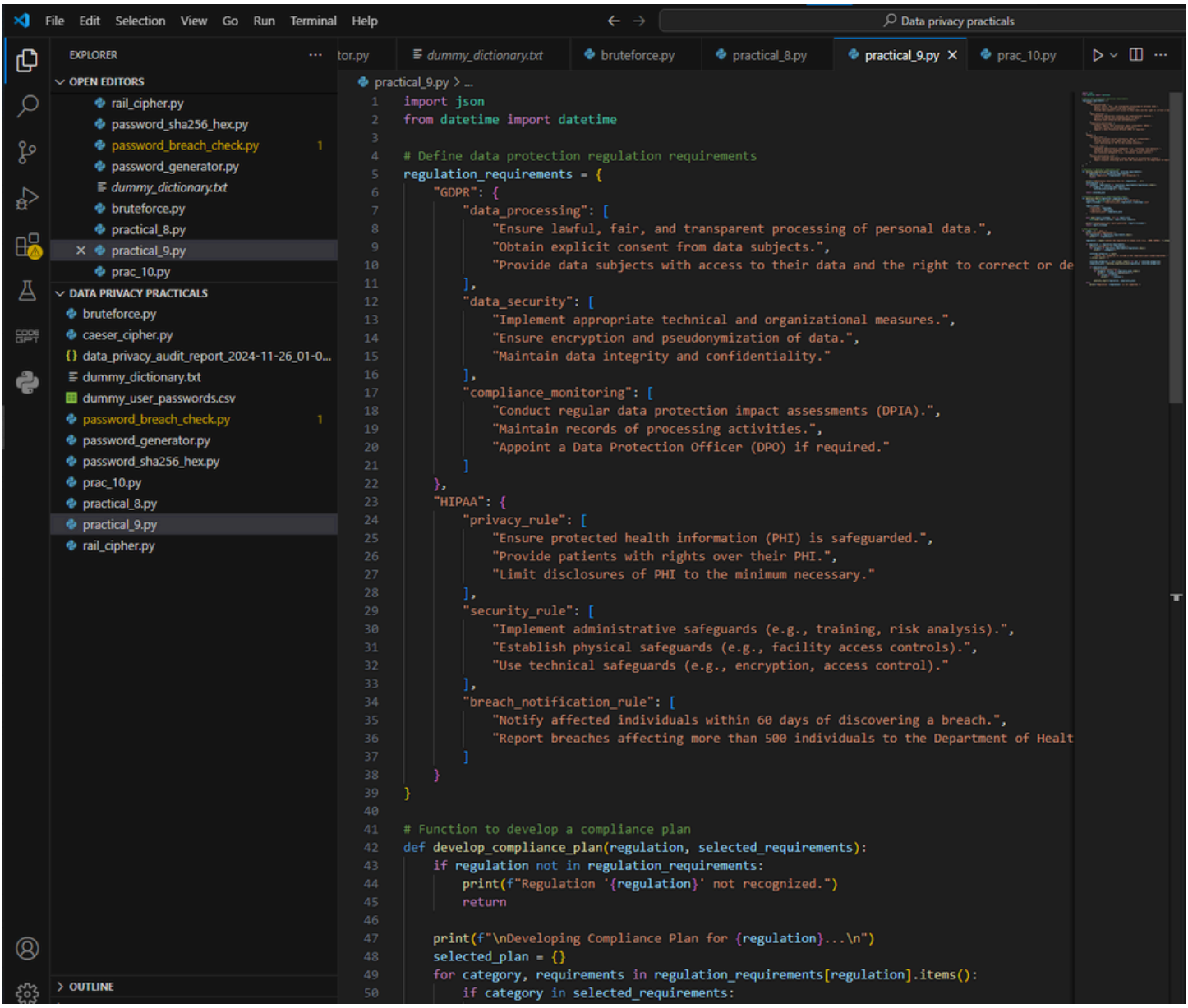
```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d:\Study\B.Sc (Hon
s.) CS\SEMESTER 5\Data privacy practicals\practical_8.py"
Starting Data Privacy Audit...

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>
```

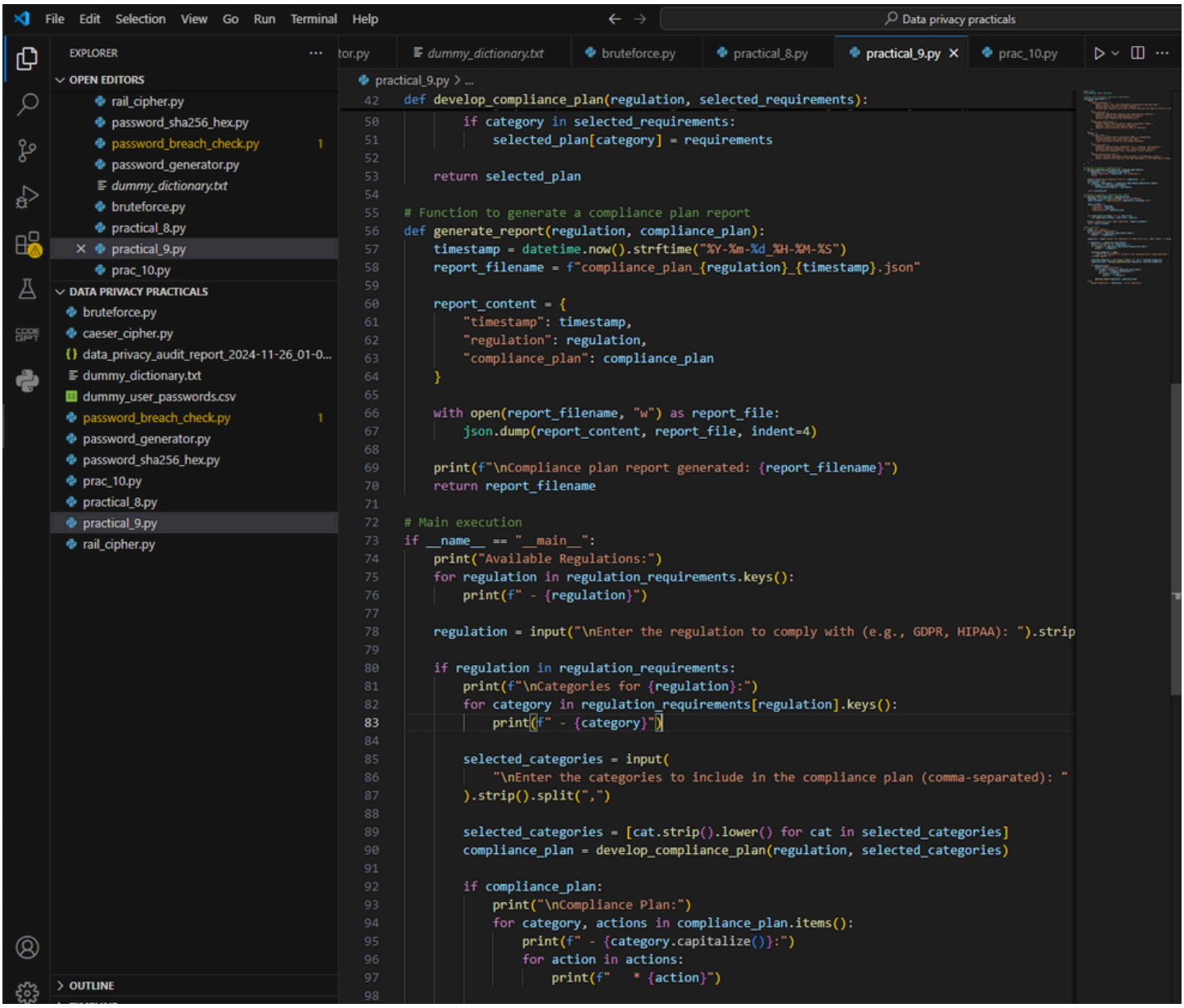


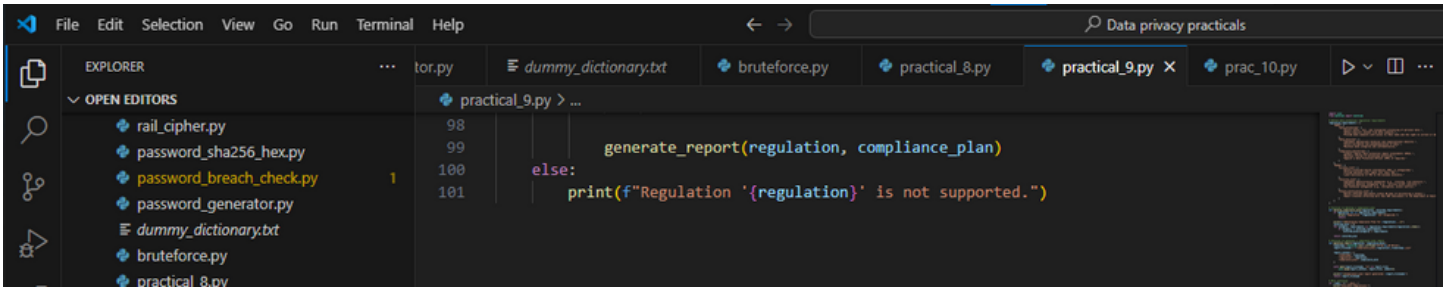
Practical 9:

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

Source code:


File Edit Selection View Go Run Terminal Help
Data privacy practicals
EXPLORER
OPEN EDITORS
rail_cipher.py
password_sha256_hex.py
password_breach_check.py 1
password_generator.py
dummy_dictionary.txt
bruteforce.py
practical_8.py
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 1
password_generator.py
password_sha256_hex.py
prac_10.py
practical_8.py
practical_9.py
rail_cipher.py
OUTLINE
tor.py
dummy_dictionary.txt
bruteforce.py
practical_8.py
practical_9.py
prac_10.py
practical_9.py > ...
import json
from datetime import datetime
# Define data protection regulation requirements
regulation_requirements = {
"GDPR": {
"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 de
],
"data_security": [
"Implement appropriate technical and organizational measures.",
"Ensure encryption and pseudonymization of data.",
"Maintain data integrity and confidentiality."
],
"compliance_monitoring": [
"Conduct regular data protection impact assessments (DPIA).",
"Maintain records of processing activities.",
"Appoint a Data Protection Officer (DPO) if required."
]
},
"HIPAA": {
"privacy_rule": [
"Ensure protected health information (PHI) is safeguarded.",
"Provide patients with rights over their PHI.",
"Limit disclosures of PHI to the minimum necessary."
],
"security_rule": [
"Implement administrative safeguards (e.g., training, risk analysis).",
"Establish physical safeguards (e.g., facility access controls).",
"Use technical safeguards (e.g., encryption, access control)."
],
"breach_notification_rule": [
"Notify affected individuals within 60 days of discovering a breach.",
"Report breaches affecting more than 500 individuals to the Department of Healt
]
}
}
# Function to develop a compliance plan
def develop_compliance_plan(regulation, selected_requirements):
if regulation not in regulation_requirements:
print(f"Regulation '{regulation}' not recognized.")
return
print(f"\nDeveloping Compliance Plan for {regulation}...\n")
selected_plan = {}
for category, requirements in regulation_requirements[regulation].items():
if category in selected_requirements:





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

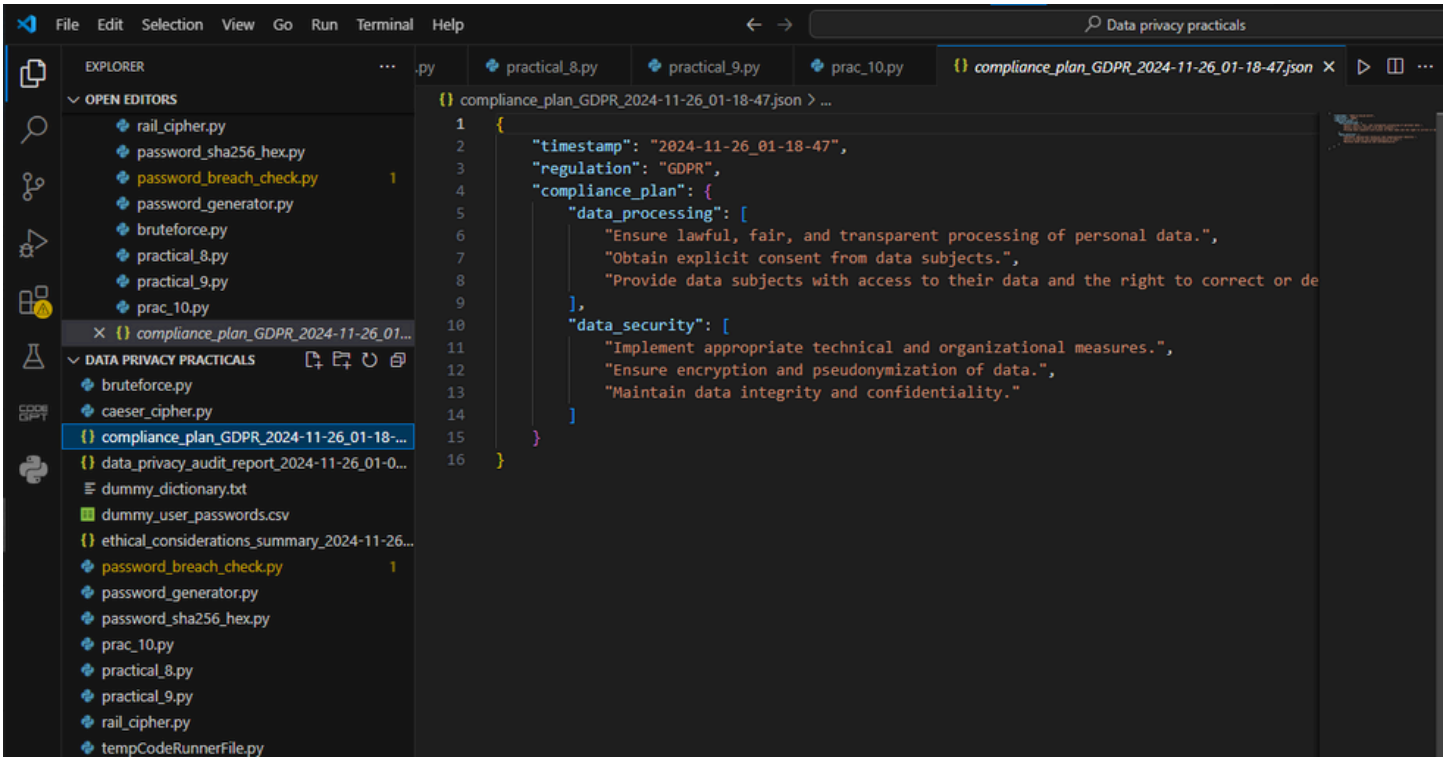
- **develop_compliance_plan()**:
  - Matches the selected categories with the regulation's requirements.
  - Returns a dictionary containing the compliance plan.

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:

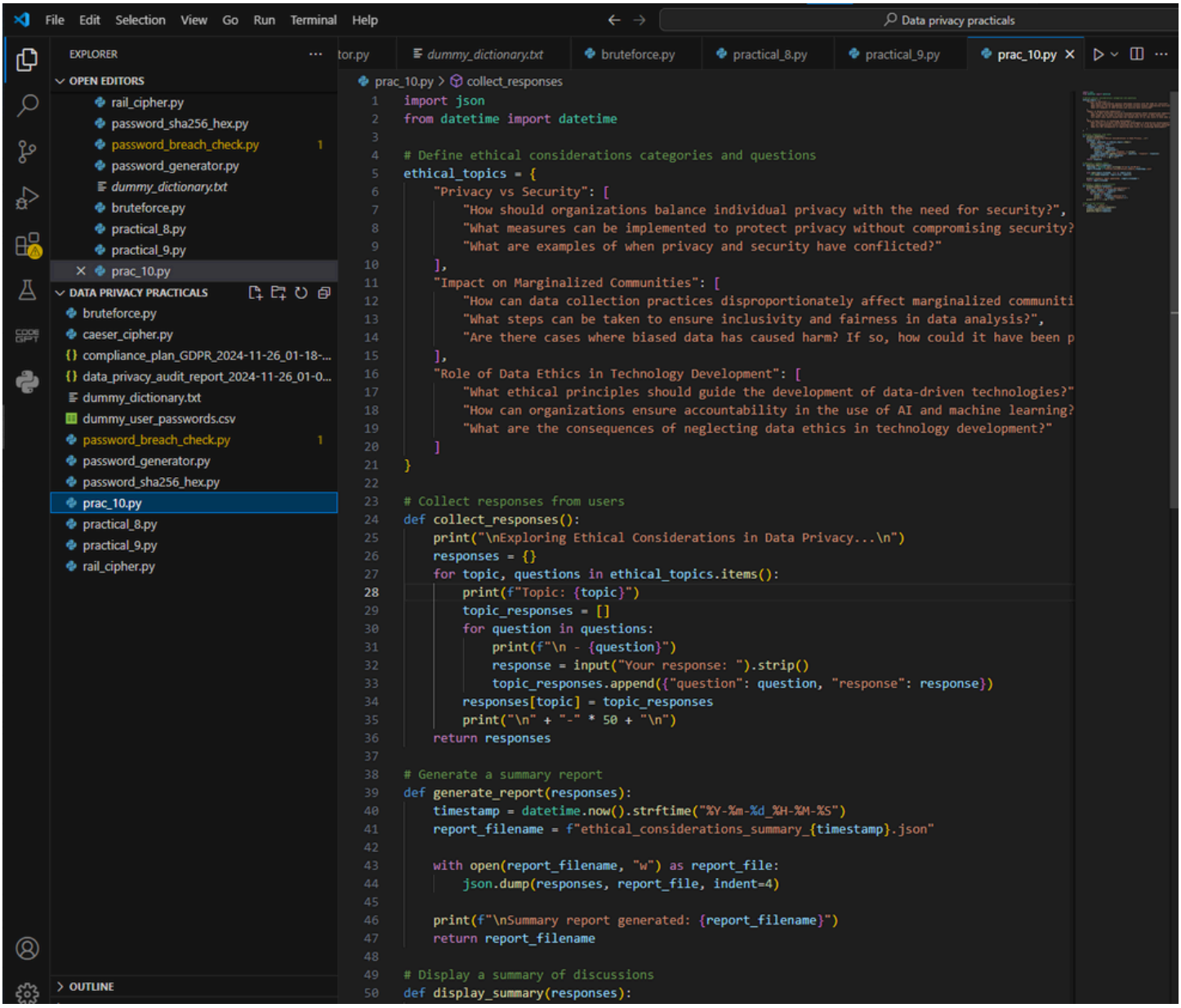
```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals\practical_9.py"
Available Regulations:
 - GDPR
 - HIPAA

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>
```

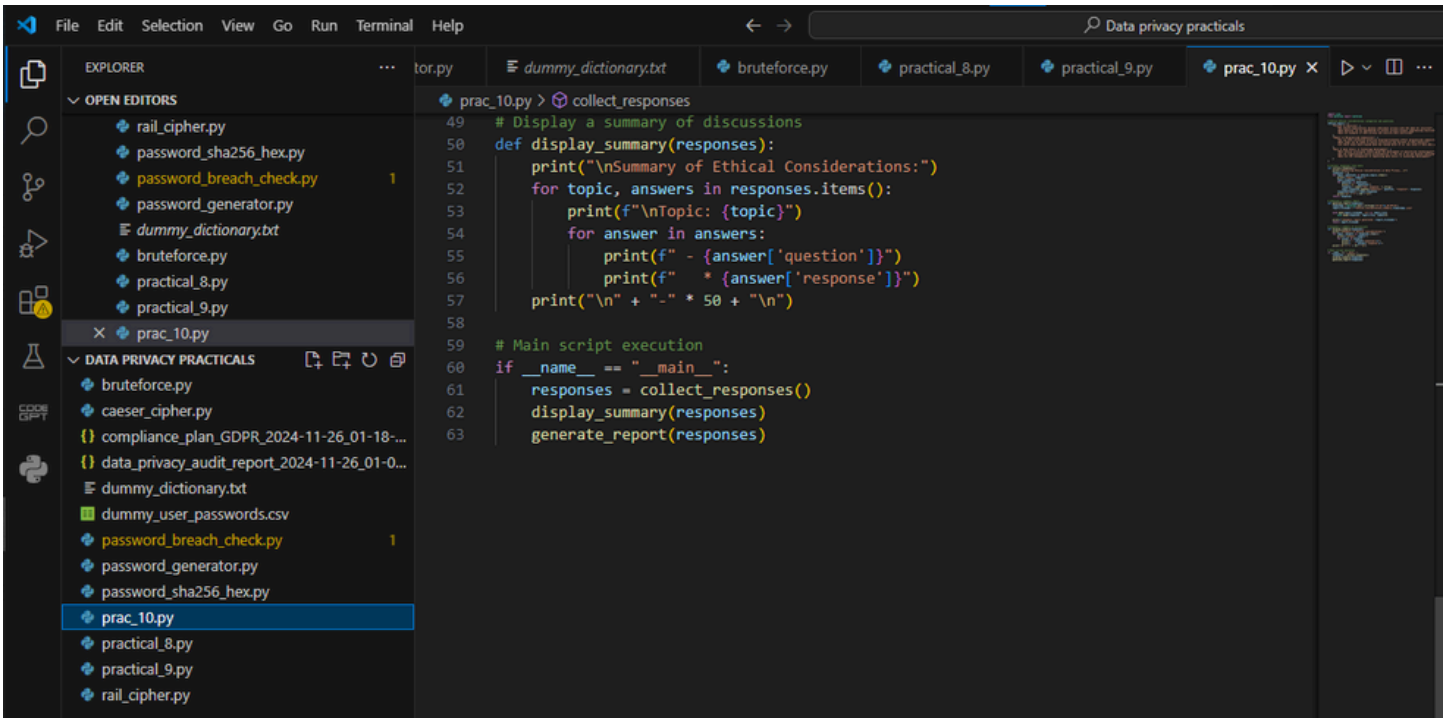


Practical 10:

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:





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

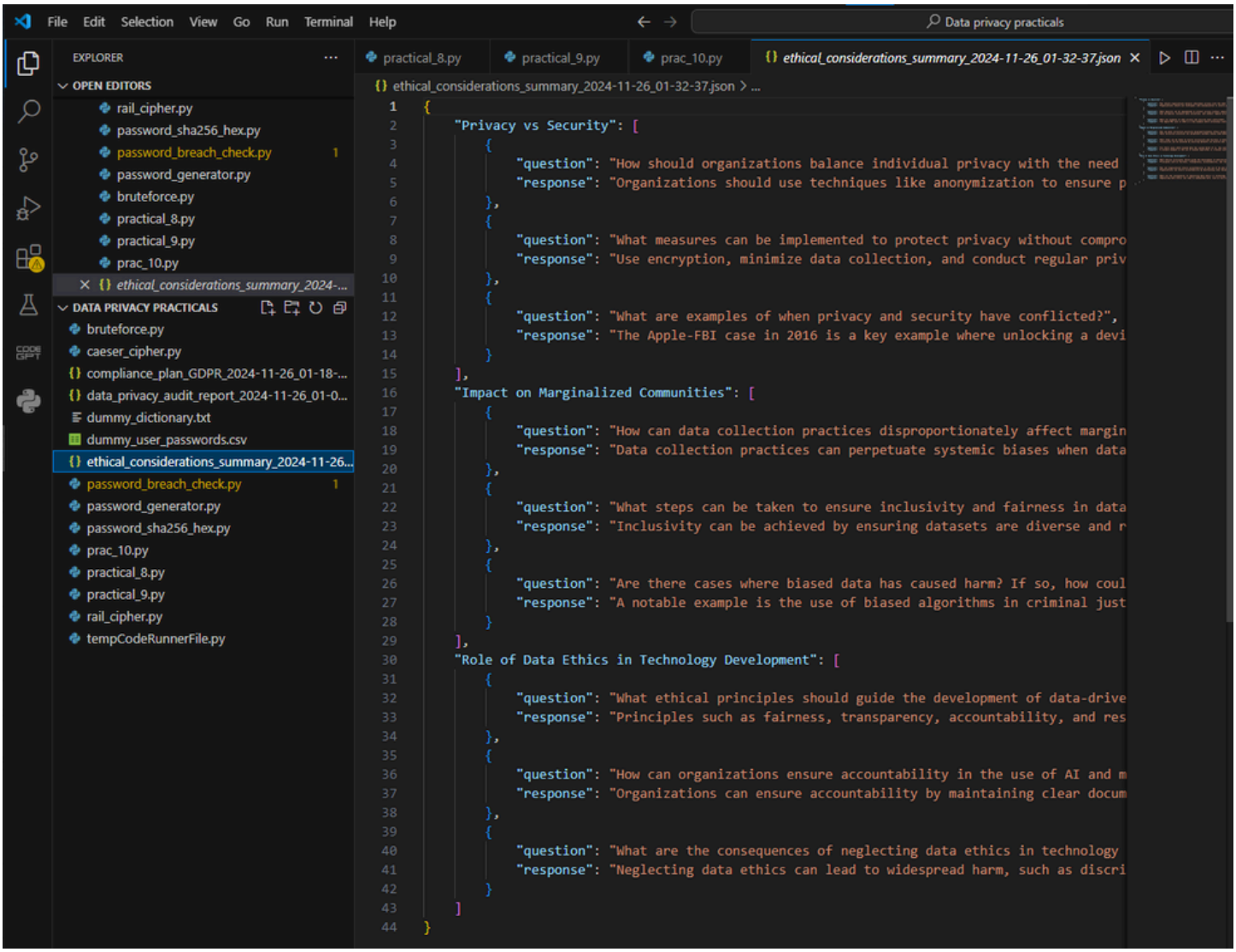
- **generate_report()**:
  - Creates a timestamped JSON report with the collected responses.
  - Example filename: ethical_considerations_summary_2024-11-26_15-10-22.json.

Output:

```
PS D:\Study\B.Sc (Hons.) CS\SEMESTER 5\Data privacy practicals> python -u "d:\Study\B.Sc (Hon
s.) 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 secur
ity 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 misiden
tify individuals from marginalized racial or ethnic groups due to lack of training data diver
sity. Additionally, excessive surveillance in low-income neighborhoods can lead to stigmatiza
tion 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 representativ
e of all demographic groups. Implementing bias detection tools, involving marginalized commun
ities in decision-making, and conducting fairness audits regularly can further promote equita
ble 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 recidivis
m. 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 discriminat
e, 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 deve
lopment, auditing AI systems for unintended biases, and involving external reviewers for ethi
cal assessments. Appointing ethics officers and establishing review boards can also enforce a
ccountability.
 - What are the consequences of neglecting data ethics in technology development?
   * Neglecting data ethics can lead to widespread harm, such as discrimination, privacy brea
ches, and loss of public trust. For instance, unethical data usage by Cambridge Analytica sig
nificantly impacted trust in social media platforms. Legal consequences, such as lawsuits and
 fines, can also arise from non-compliance with regulations.
```



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*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.