

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | Assignment | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | Prashant Mishra (Submitted By) |  | | | |
|  | | | | Data Privacy(DSE)—Ms. Ashmita (Submitted to) |  | | | |
|  | | |  | | |  | | |

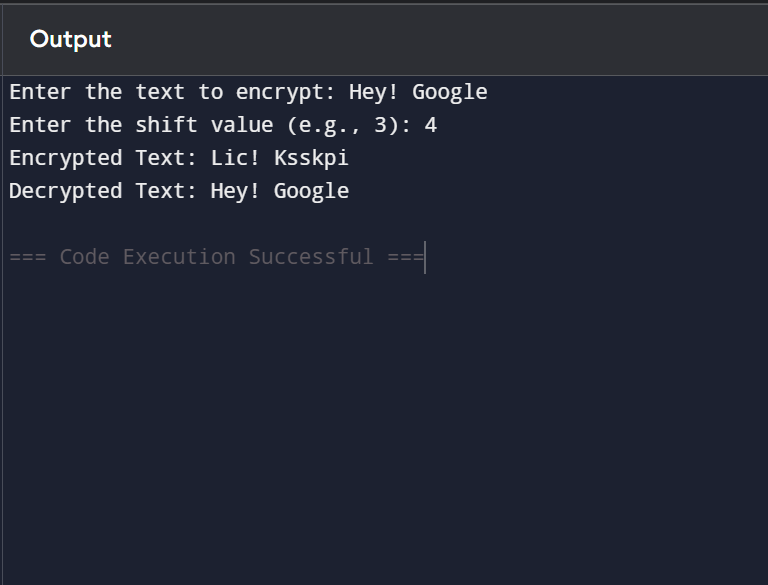
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | |  | | |
|  |  | | |  | | |
|  |  |  |  | |  | | |
|  |  |  | | |  |  | | |

Q1. Write a program to perform encryption and decryption using Caesar cipher (substitutional

cipher).

1. # Caesar Cipher Implementation
2. def caesar\_cipher\_encrypt(text, shift):
3. """Encrypt the text using Caesar Cipher."""
4. encrypted\_text = ""
5. for char in text:
6. if char.isalpha(): # Check if character is a letter
7. # Determine whether it's uppercase or lowercase
8. start = ord('A') if char.isupper() else ord('a')
9. # Perform the shift and wrap around using modulo
10. encrypted\_char = chr((ord(char) - start + shift) % 26 + start)
11. encrypted\_text += encrypted\_char
12. else:
13. # Non-alphabetic characters remain unchanged
14. encrypted\_text += char
15. return encrypted\_text
16. def caesar\_cipher\_decrypt(text, shift):
17. """Decrypt the text using Caesar Cipher."""
18. return caesar\_cipher\_encrypt(text, -shift) # Negative shift for decryption
19. # Input from the user
20. plain\_text = input("Enter the text to encrypt: ")
21. shift\_value = int(input("Enter the shift value (e.g., 3): "))
22. # Encryption
23. encrypted\_text = caesar\_cipher\_encrypt(plain\_text, shift\_value)
24. print("Encrypted Text:", encrypted\_text)
25. # Decryption
26. decrypted\_text = caesar\_cipher\_decrypt(encrypted\_text, shift\_value)
27. print("Decrypted Text:", decrypted\_text)

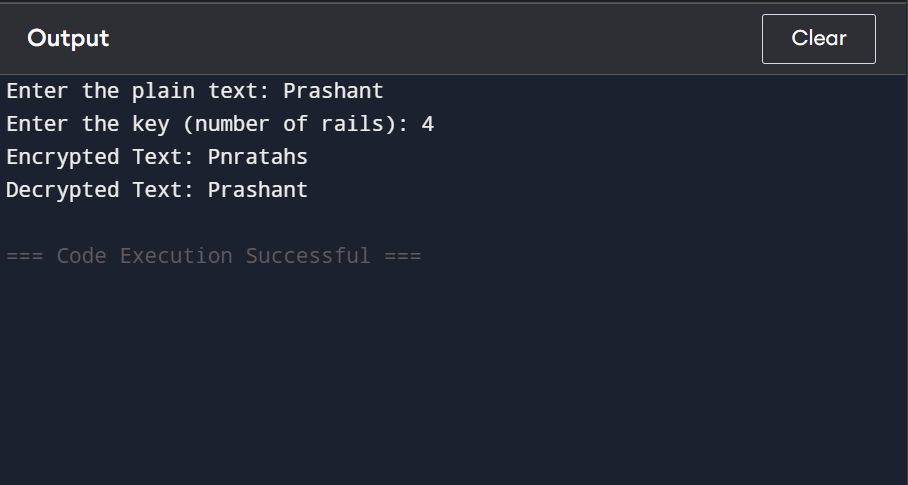
Output:



Q2. Write a program to perform encryption and decryption using Rail Fence Cipher (transpositional cipher).

1. def encrypt\_rail\_fence(text, key):
2. # Create the rail matrix
3. rail = [['\n' for \_ in range(len(text))] for \_ in range(key)]
4. # Mark the positions for the characters
5. direction\_down = False
6. row, col = 0, 0
7. for char in text:
8. if row == 0 or row == key - 1:
9. direction\_down = not direction\_down
10. rail[row][col] = char
11. col += 1
12. row += 1 if direction\_down else -1
13. # Collect the characters in the rail order
14. result = []
15. for i in range(key):
16. for j in range(len(text)):
17. if rail[i][j] != '\n':
18. result.append(rail[i][j])
19. return ''.join(result)
20. def decrypt\_rail\_fence(cipher, key):
21. # Create the rail matrix
22. rail = [['\n' for \_ in range(len(cipher))] for \_ in range(key)]
23. # Mark the positions for the characters
24. direction\_down = None
25. row, col = 0, 0
26. for i in range(len(cipher)):
27. if row == 0:
28. direction\_down = True
29. if row == key - 1:
30. direction\_down = False
31. rail[row][col] = '\*'
32. col += 1
33. row += 1 if direction\_down else -1
34. # Fill the marked positions with the ciphertext
35. index = 0
36. for i in range(key):
37. for j in range(len(cipher)):
38. if rail[i][j] == '\*' and index < len(cipher):
39. rail[i][j] = cipher[index]
40. index += 1
41. # Read the text in a zigzag pattern
42. result = []
43. row, col = 0, 0
44. for i in range(len(cipher)):
45. if row == 0:
46. direction\_down = True
47. if row == key - 1:
48. direction\_down = False
49. if rail[row][col] != '\n':
50. result.append(rail[row][col])
51. col += 1
52. row += 1 if direction\_down else -1
53. return ''.join(result)
54. # User Input
55. plain\_text = input("Enter the plain text: ")
56. key = int(input("Enter the key (number of rails): "))
57. # Encrypt the plain text
58. cipher\_text = encrypt\_rail\_fence(plain\_text, key)
59. print(f"Encrypted Text: {cipher\_text}")
60. # Decrypt the cipher text
61. decrypted\_text = decrypt\_rail\_fence(cipher\_text, key)
62. print(f"Decrypted Text: {decrypted\_text}")

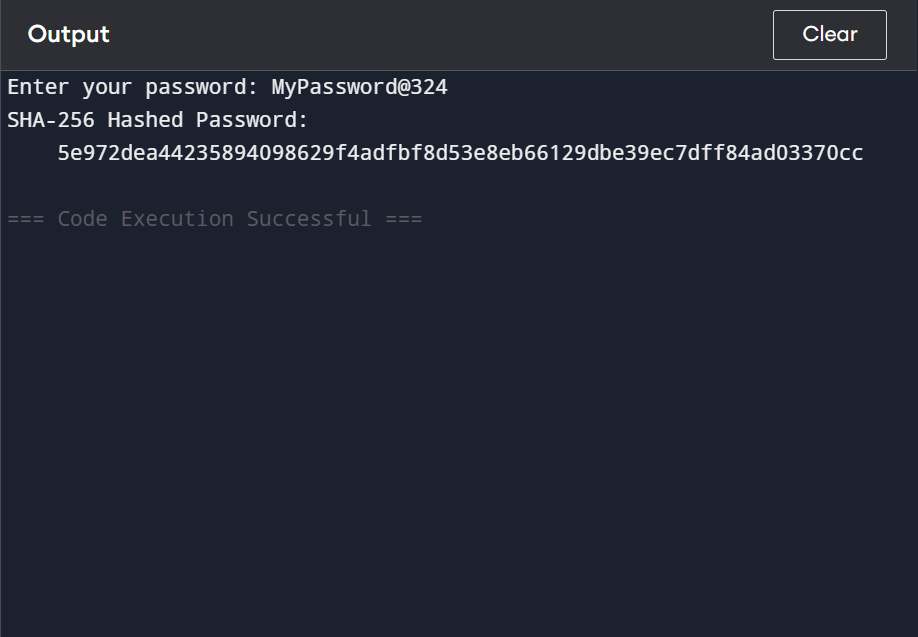
Output:



Q3. 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.

1. import hashlib
2. def hash\_password(password):
3. """
4. Hashes the input password using SHA-256 and returns its hexadecimal representation.
5. Args:
6. password (str): The password string to hash.
7. Returns:
8. str: SHA-256 hashed representation of the password.
9. """
10. # Encode the password to bytes
11. password\_bytes = password.encode('utf-8')
12. # Create a SHA-256 hash object
13. sha256\_hash = hashlib.sha256(password\_bytes)
14. # Return the hexadecimal digest
15. return sha256\_hash.hexdigest()
16. # Input from the user
17. user\_password = input("Enter your password: ")
18. # Hash the password
19. hashed\_password = hash\_password(user\_password)
20. # Display the hashed password
21. print("SHA-256 Hashed Password:", hashed\_password)

Output:



Q4. 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 &quot;Have I Been Pwned&quot; API

(https://haveibeenpwned.com/API/v3) to check if a password has been leaked.

import hashlib

import requests

def check\_pwned\_password\_free(password):

sha1\_hash = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()

prefix = sha1\_hash[:5]

suffix = sha1\_hash[5:]

url = f"https://api.pwnedpasswords.com/range/{prefix}"

response = requests.get(url)

hashes = response.text.splitlines()

for line in hashes:

hash\_suffix, count = line.split(':')

if hash\_suffix == suffix:

return True, count

return False, 0

password = input("Enter password to check: ")

is\_pwned, count = check\_pwned\_password\_free(password)

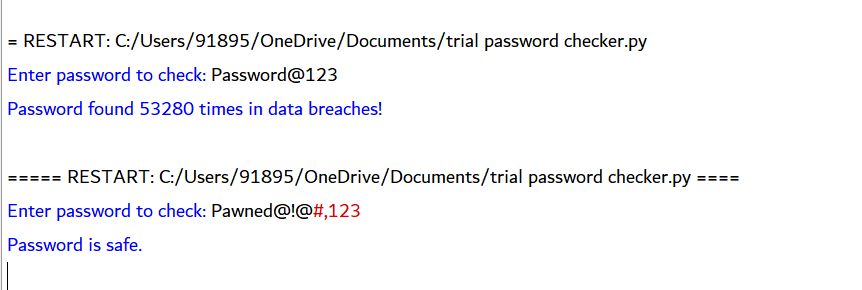
if is\_pwned:

print(f"Password found {count} times in data breaches!")

else:

print("Password is safe.")

Output:



Q5. Write a Python program that generates a password using a random combination of words

from a dictionary file.

import random

def load\_words\_from\_file(file\_path):

"""

Loads words from a dictionary file.

Args:

file\_path (str): Path to the dictionary file.

Returns:

list: List of words in the file.

"""

try:

with open(file\_path, 'r') as file:

# Read all lines, strip whitespace, and return as a list

return [line.strip() for line in file if line.strip()]

except FileNotFoundError:

print(f"Error: File '{file\_path}' not found.")

return []

except Exception as e:

print(f"An error occurred: {e}")

return []

def generate\_password(word\_list, num\_words=4):

"""

Generates a password using a random combination of words.

Args:

word\_list (list): List of words to choose from.

num\_words (int): Number of words to include in the password.

Returns:

str: Generated password.

"""

if len(word\_list) < num\_words:

print("Error: Not enough words in the dictionary to generate the password.")

return ""

# Randomly select words and join them with a separator (e.g., "-")

password = "-".join(random.choice(word\_list) for \_ in range(num\_words))

return password

# Main script

if \_\_name\_\_ == "\_\_main\_\_":

# Input: Path to the dictionary file

dictionary\_file = input("Enter the path to the dictionary file: ")

# Load words from the file

words = load\_words\_from\_file(dictionary\_file)

if words:

# Generate the password

num\_words = int(input("Enter the number of words for the password: "))

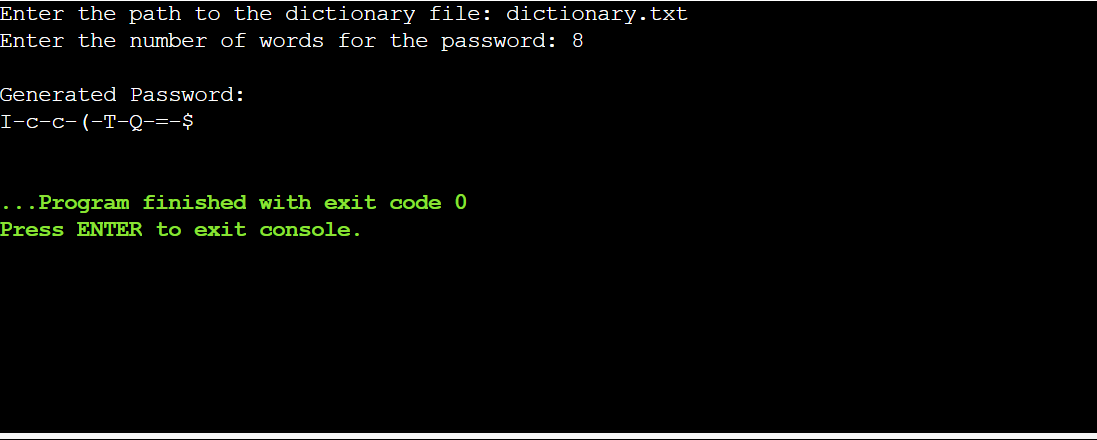
password = generate\_password(words, num\_words)

if password:

print("\nGenerated Password:")

print(password)

Output



Dictionary.txt: a b c d e f g h i j k l m n o p q r s t u v w x y z

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

! " # $ % & ' ( ) \* + , - . / : ; < = > ? @ [ \ ] ^ \_ ` { | } ~

Q6. Write a Python program that simulates a brute-force attack on a password by trying out all possible character combinations.

import itertools

def brute\_force\_attack(target\_password, max\_length):

"""

Simulates a brute-force attack to guess the target password.

Args:

target\_password (str): The password to guess.

max\_length (int): Maximum length of passwords to try.

Returns:

str: The guessed password if found.

"""

# Define the character set

characters = (

"abcdefghijklmnopqrstuvwxyz"

"ABCDEFGHIJKLMNOPQRSTUVWXYZ"

"0123456789"

"!\"#$%&'()\*+,-./:;<=>?@[\\]^\_`{|}~"

)

attempts = 0

# Try passwords of increasing lengths up to max\_length

for length in range(1, max\_length + 1):

for combination in itertools.product(characters, repeat=length):

# Join the characters to form a password

guessed\_password = ''.join(combination)

attempts += 1

# Print progress for beginners to visualize

print(f"Trying: {guessed\_password}")

# Check if the guessed password matches the target

if guessed\_password == target\_password:

print(f"\nPassword found after {attempts} attempts!")

return guessed\_password

# If the password isn't found

print("\nPassword not found within the given length.")

return None

# Main script

if \_\_name\_\_ == "\_\_main\_\_":

# Ask the user for the target password and maximum length

target\_password = input("Enter the password to guess: ")

max\_length = int(input("Enter the maximum password length to try: "))

# Start the brute-force attack

print("\nStarting brute-force attack...")

result = brute\_force\_attack(target\_password, max\_length)

if result:

print(f"The password is: {result}")

else:

print("Could not guess the password.")

Output



Q7. Demonstrate the usage/sending of a digitally signed document.

import os

from cryptography.hazmat.primitives import hashes

from cryptography.hazmat.primitives.asymmetric import rsa, padding

from cryptography.hazmat.primitives.serialization import Encoding, PublicFormat, PrivateFormat, NoEncryption

from cryptography.hazmat.primitives.serialization import load\_pem\_private\_key, load\_pem\_public\_key

# Step 1: Generate RSA Key Pair

def generate\_keys():

private\_key = rsa.generate\_private\_key(

public\_exponent=65537,

key\_size=2048,

)

public\_key = private\_key.public\_key()

return private\_key, public\_key

# Step 2: Digitally Sign the File

def sign\_file(file\_path, private\_key):

with open(file\_path, "rb") as file:

file\_data = file.read()

signature = private\_key.sign(

file\_data,

padding.PSS(

mgf=padding.MGF1(hashes.SHA256()),

salt\_length=padding.PSS.MAX\_LENGTH,

),

hashes.SHA256(),

)

return signature

# Step 3: Verify the File's Signature

def verify\_file(file\_path, signature, public\_key):

with open(file\_path, "rb") as file:

file\_data = file.read()

try:

public\_key.verify(

signature,

file\_data,

padding.PSS(

mgf=padding.MGF1(hashes.SHA256()),

salt\_length=padding.PSS.MAX\_LENGTH,

),

hashes.SHA256(),

)

return True

except Exception:

return False

# Step 4: Save and Load Keys

def save\_key(key, file\_name, is\_private=True):

with open(file\_name, 'wb') as key\_file:

if is\_private:

key\_file.write(key.private\_bytes(

Encoding.PEM,

PrivateFormat.PKCS8,

NoEncryption()

))

else:

key\_file.write(key.public\_bytes(

Encoding.PEM,

PublicFormat.SubjectPublicKeyInfo

))

def load\_key(file\_name, is\_private=True):

with open(file\_name, 'rb') as key\_file:

if is\_private:

return load\_pem\_private\_key(key\_file.read(), password=None)

else:

return load\_pem\_public\_key(key\_file.read())

# Main demonstration

if \_\_name\_\_ == "\_\_main\_\_":

# 1. Generate keys

private\_key, public\_key = generate\_keys()

# Save keys to files

save\_key(private\_key, "private\_key.pem", is\_private=True)

save\_key(public\_key, "public\_key.pem", is\_private=False)

# Load keys back from files

private\_key = load\_key("private\_key.pem", is\_private=True)

public\_key = load\_key("public\_key.pem", is\_private=False)

# 2. Ask the user to provide a file to sign

file\_path = input("Enter the path of the file to sign: ")

if not os.path.exists(file\_path):

print("File does not exist. Please check the path and try again.")

else:

# 3. Sign the file

signature = sign\_file(file\_path, private\_key)

print("\nFile signed successfully.")

# Save the signature to a file

signature\_file = file\_path + ".sig"

with open(signature\_file, "wb") as sig\_file:

sig\_file.write(signature)

print(f"Signature saved to: {signature\_file}")

# 4. Verify the file's signature

is\_valid = verify\_file(file\_path, signature, public\_key)

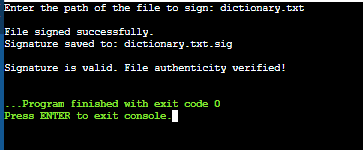
if is\_valid:

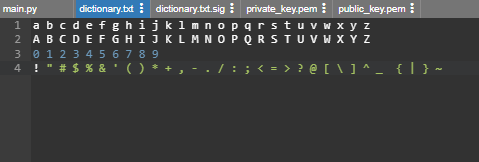
print("\nSignature is valid. File authenticity verified!")

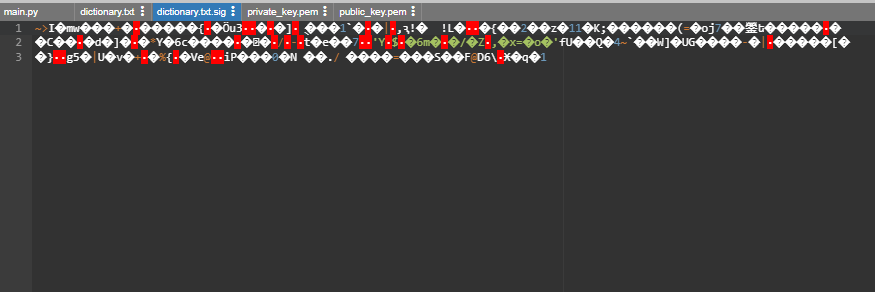
else:

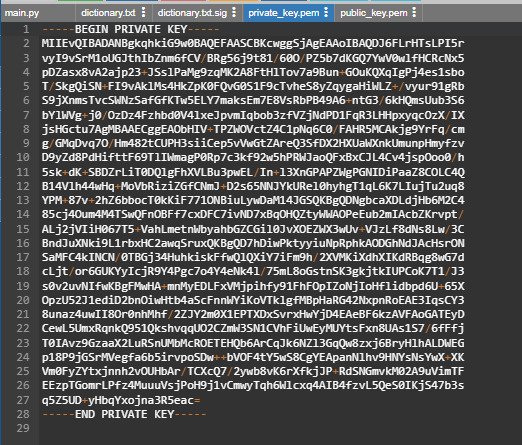
print("\nSignature is invalid. File authenticity cannot be verified.")

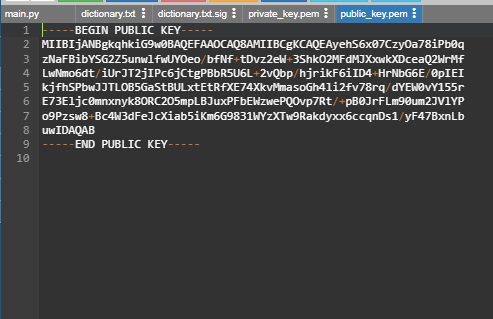
Outputs:











Q8. Students needs to conduct a data privacy audit of an organization to identify potential vulnerabilities and risks in their data privacy practices.

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"] == "no"]

vulnerabilities[category] = category\_vulnerabilities

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\_\_":

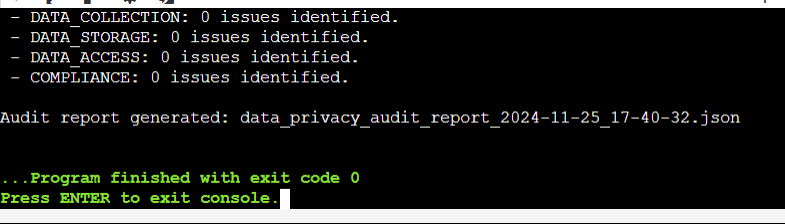
collect\_responses()

vulnerabilities = analyze\_responses()

display\_summary(vulnerabilities)

generate\_report(vulnerabilities)

Output



Q9. Students needs to explore the requirements of the Data Protection Regulations and develop a plan for ensuring compliance with the regulation.

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 delete it."

],

"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 Health and Human Services."

]

}

}

# 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:

selected\_plan[category] = requirements

return selected\_plan

# Function to generate a compliance plan report

def generate\_report(regulation, compliance\_plan):

timestamp = datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")

report\_filename = f"compliance\_plan\_{regulation}\_{timestamp}.json"

report\_content = {

"timestamp": timestamp,

"regulation": regulation,

"compliance\_plan": compliance\_plan

}

with open(report\_filename, "w") as report\_file:

json.dump(report\_content, report\_file, indent=4)

print(f"\nCompliance plan report generated: {report\_filename}")

return report\_filename

# Main execution

if \_\_name\_\_ == "\_\_main\_\_":

print("Available Regulations:")

for regulation in regulation\_requirements.keys():

print(f" - {regulation}")

regulation = input("\nEnter the regulation to comply with (e.g., GDPR, HIPAA): ").strip().upper()

if regulation in regulation\_requirements:

print(f"\nCategories for {regulation}:")

for category in regulation\_requirements[regulation].keys():

print(f" - {category}")

selected\_categories = input(

"\nEnter the categories to include in the compliance plan (comma-separated): "

).strip().split(",")

selected\_categories = [cat.strip().lower() for cat in selected\_categories]

compliance\_plan = develop\_compliance\_plan(regulation, selected\_categories)

if compliance\_plan:

print("\nCompliance Plan:")

for category, actions in compliance\_plan.items():

print(f" - {category.capitalize()}:")

for action in actions:

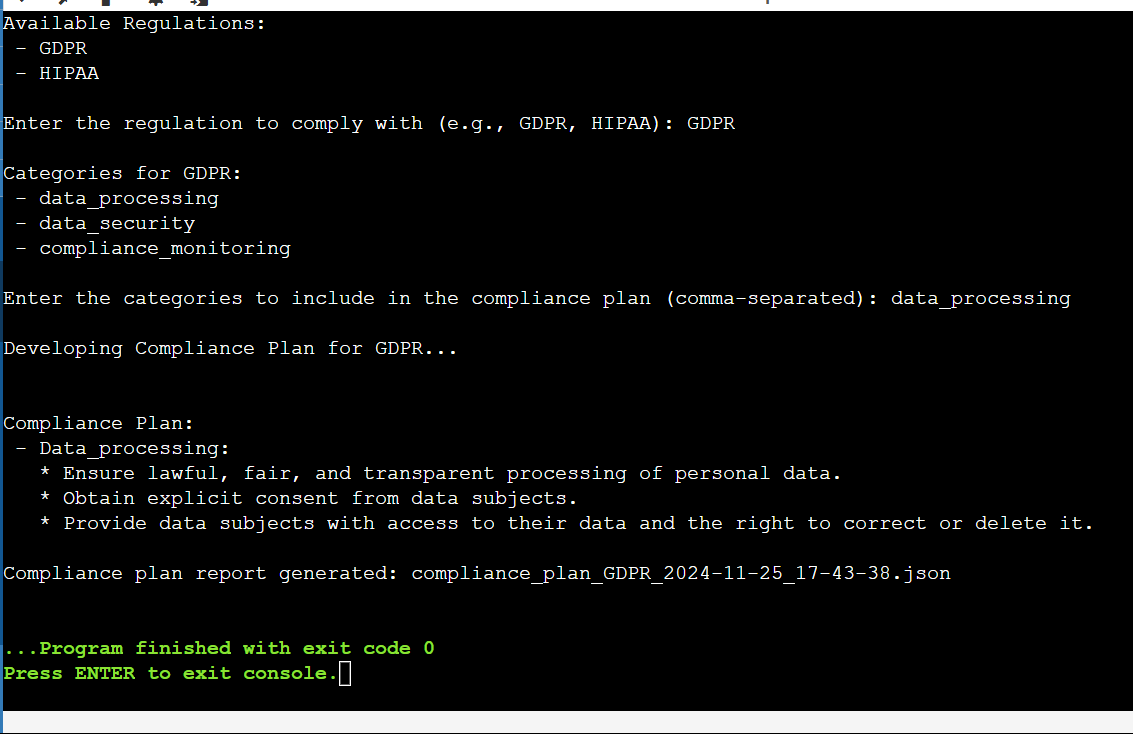
print(f" \* {action}")

generate\_report(regulation, compliance\_plan)

else:

print(f"Regulation '{regulation}' is not supported.")

Output:



Q10. 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.

import json

from datetime import datetime

# Define ethical considerations categories and questions

ethical\_topics = {

"Privacy vs Security": [

"How should organizations balance individual privacy with the need for security?",

"What measures can be implemented to protect privacy without compromising security?",

"What are examples of when privacy and security have conflicted?"

],

"Impact on Marginalized Communities": [

"How can data collection practices disproportionately affect marginalized communities?",

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

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

],

"Role of Data Ethics in Technology Development": [

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

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

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

]

}

# Collect responses from users

def collect\_responses():

print("\nExploring Ethical Considerations in Data Privacy...\n")

responses = {}

for topic, questions in ethical\_topics.items():

print(f"Topic: {topic}")

topic\_responses = []

for question in questions:

print(f"\n - {question}")

response = input("Your response: ").strip()

topic\_responses.append({"question": question, "response": response})

responses[topic] = topic\_responses

print("\n" + "-" \* 50 + "\n")

return responses

# Generate a summary report

def generate\_report(responses):

timestamp = datetime.now().strftime("%Y-%m-%d\_%H-%M-%S")

report\_filename = f"ethical\_considerations\_summary\_{timestamp}.json"

with open(report\_filename, "w") as report\_file:

json.dump(responses, report\_file, indent=4)

print(f"\nSummary report generated: {report\_filename}")

return report\_filename

# Display a summary of discussions

def display\_summary(responses):

print("\nSummary of Ethical Considerations:")

for topic, answers in responses.items():

print(f"\nTopic: {topic}")

for answer in answers:

print(f" - {answer['question']}")

print(f" \* {answer['response']}")

print("\n" + "-" \* 50 + "\n")

# Main script execution

if \_\_name\_\_ == "\_\_main\_\_":

responses = collect\_responses()

display\_summary(responses)

generate\_report(responses)

Output:

