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

def caesar\_cipher(text, shift, mode='encrypt'):

result = ""

shift = shift if mode == 'encrypt' else -shift

for char in text:

if char.isalpha():

shift\_base = ord('A') if char.isupper() else ord('a')

shifted\_char = chr((ord(char) - shift\_base + shift) % 26 + shift\_base)

result += shifted\_char

else:

result += char

return result

text = input("Enter the text: ")

shift = int(input("Enter the shift value: "))

mode = input("Type 'encrypt' for encryption or 'decrypt' for decryption: ").strip().lower()

print("Result:", caesar\_cipher(text, shift, mode))



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

def rail\_fence\_encrypt(text, key):

rail = [['\n' for i in range(len(text))] for j in range(key)]

direction\_down = False

row, col = 0, 0

for char in text:

if row == 0 or row == key - 1:

direction\_down = not direction\_down

rail[row][col] = char

col += 1

row += 1 if direction\_down else -1

result = ''.join(rail[row][col] for row in range(key) for col in range(len(text)) if rail[row][col] != '\n')

return result

def rail\_fence\_decrypt(cipher\_text, key):

rail = [['\n' for i in range(len(cipher\_text))] for j in range(key)]

direction\_down = None

row, col = 0, 0

for i in range(len(cipher\_text)):

if row == 0:

direction\_down = True

if row == key - 1:

direction\_down = False

rail[row][col] = '\*'

col += 1

row += 1 if direction\_down else -1

index = 0

for i in range(key):

for j in range(len(cipher\_text)):

if rail[i][j] == '\*' and index < len(cipher\_text):

rail[i][j] = cipher\_text[index]

index += 1

result = []

row, col = 0, 0

for i in range(len(cipher\_text)):

if row == 0:

direction\_down = True

if row == key - 1:

direction\_down = False

result.append(rail[row][col])

col += 1

row += 1 if direction\_down else -1

return ''.join(result)

text = input("Enter the text: ")

key = int(input("Enter the key (number of rails): "))

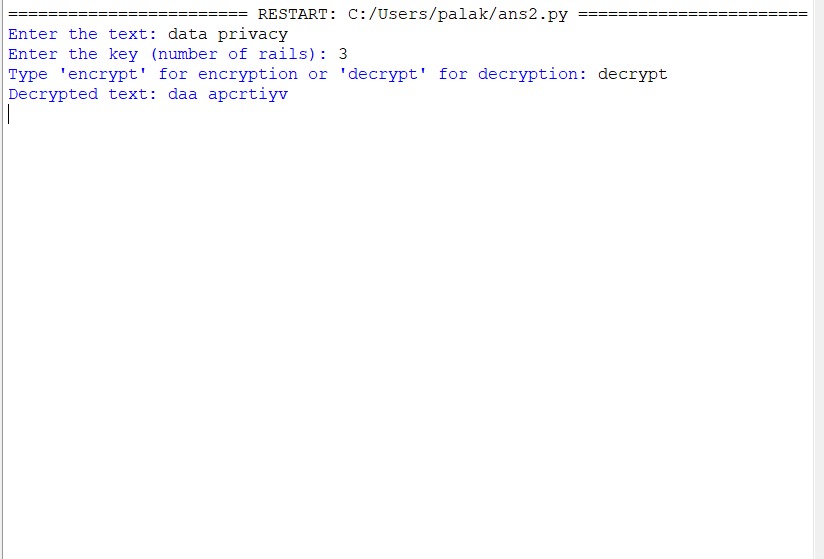
mode = input("Type 'encrypt' for encryption or 'decrypt' for decryption: ").strip().lower()

if mode == 'encrypt':

print("Encrypted text:", rail\_fence\_encrypt(text, key))

else:

print("Decrypted text:", rail\_fence\_decrypt(text, key))



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.

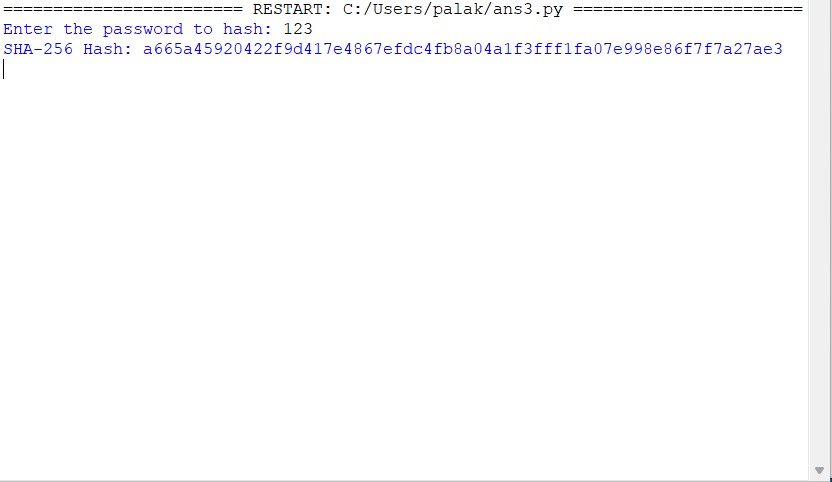
import hashlib

def hash\_password(password):

return hashlib.sha256(password.encode()).hexdigest()

password = input("Enter the password to hash: ")

print("SHA-256 Hash:", hash\_password(password))



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.

import hashlib

import requests

import os

def check\_password\_pwned(password):

# Hash the password using SHA-1

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

# Send the first 5 characters of the hash to the API

prefix = sha1\_password[:5]

suffix = sha1\_password[5:]

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

response = requests.get(url)

# Check if the response contains the suffix

if response.status\_code == 200:

hashes = (line.split(':') for line in response.text.splitlines())

for hash\_suffix, count in hashes:

if hash\_suffix == suffix:

return int(count) # Return the number of times the password has been pwned

return 0 # Return 0 if not found in the breached database

def check\_passwords\_from\_file(file\_path):

# Check if the file exists

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

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

return

# Open and read the file

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

for line in file:

# Skip blank lines

if not line.strip():

continue

# Split username and password

try:

username, password = line.strip().split(',', 1) # Split on the first comma only

except ValueError:

print(f"Error: Line '{line.strip()}' is not formatted correctly. Skipping.")

continue

# Check if the password is pwned

pwned\_count = check\_password\_pwned(password)

if pwned\_count > 0:

print(f"WARNING: Password for {username} has been pwned {pwned\_count} times!")

else:

print(f"Password for {username} is safe (not found in breach database).")

# Example usage

file\_path = "usernames\_passwords.txt" # Replace with your file path if needed

check\_passwords\_from\_file(file\_path)



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

import random

import random

def generate\_password(word\_count=3):

with open("dictionary.txt", 'r') as file:

words = [line.strip() for line in file]

password = ''.join(random.choices(words, k=word\_count))

return password

word\_count = int(input("Enter the number of words for the password: "))

print("Generated password:", generate\_password(word\_count))



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

import itertools

import string

def brute\_force\_attack(target\_password):

# Define the set of characters to consider (lowercase letters in this case)

characters = string.ascii\_lowercase

# Try all possible combinations of characters

for length in range(1, len(target\_password) + 1): # Loop through different password lengths

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

# Join the tuple into a string to form the password guess

guess = ''.join(combination)

print(f"Trying password: {guess}")

# Check if the guess matches the target password

if guess == target\_password:

print(f"Password found: {guess}")

return guess

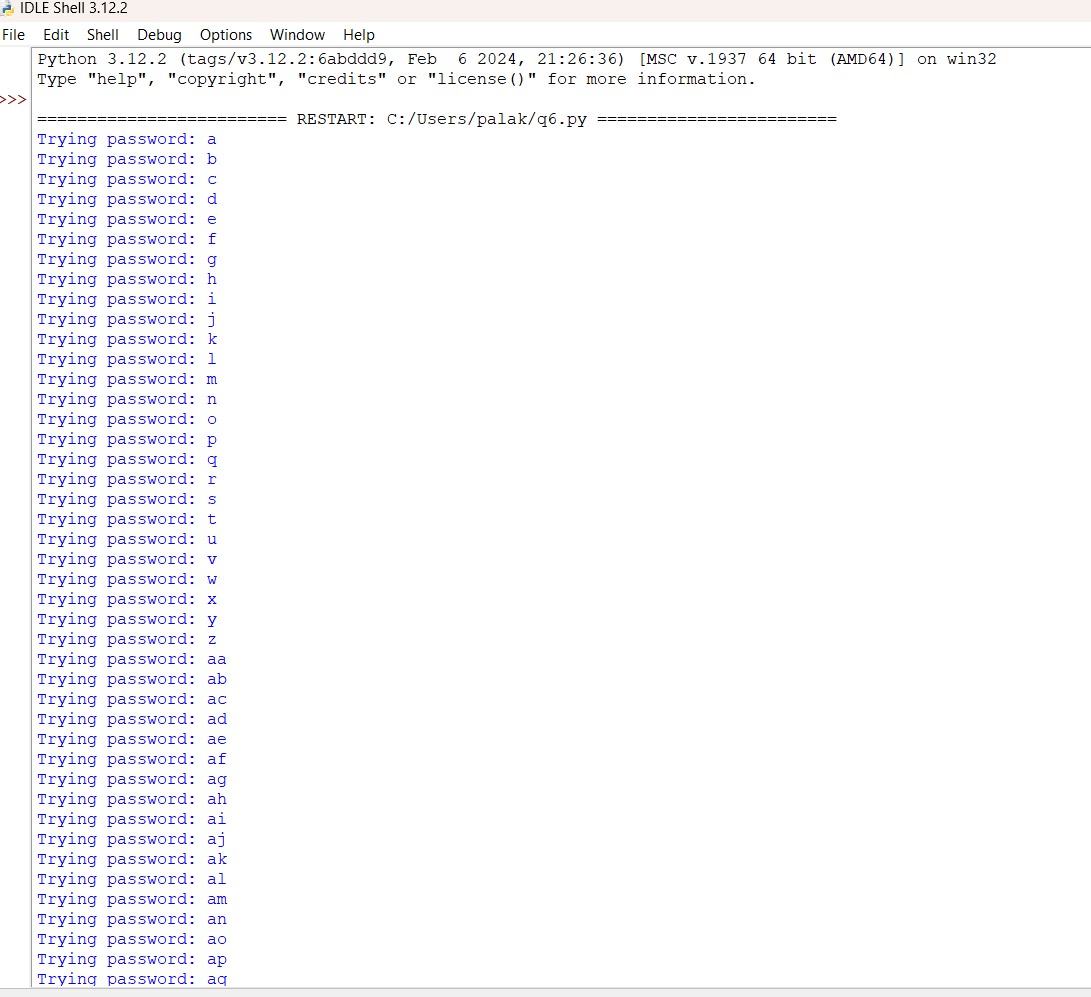
print("Password not found")

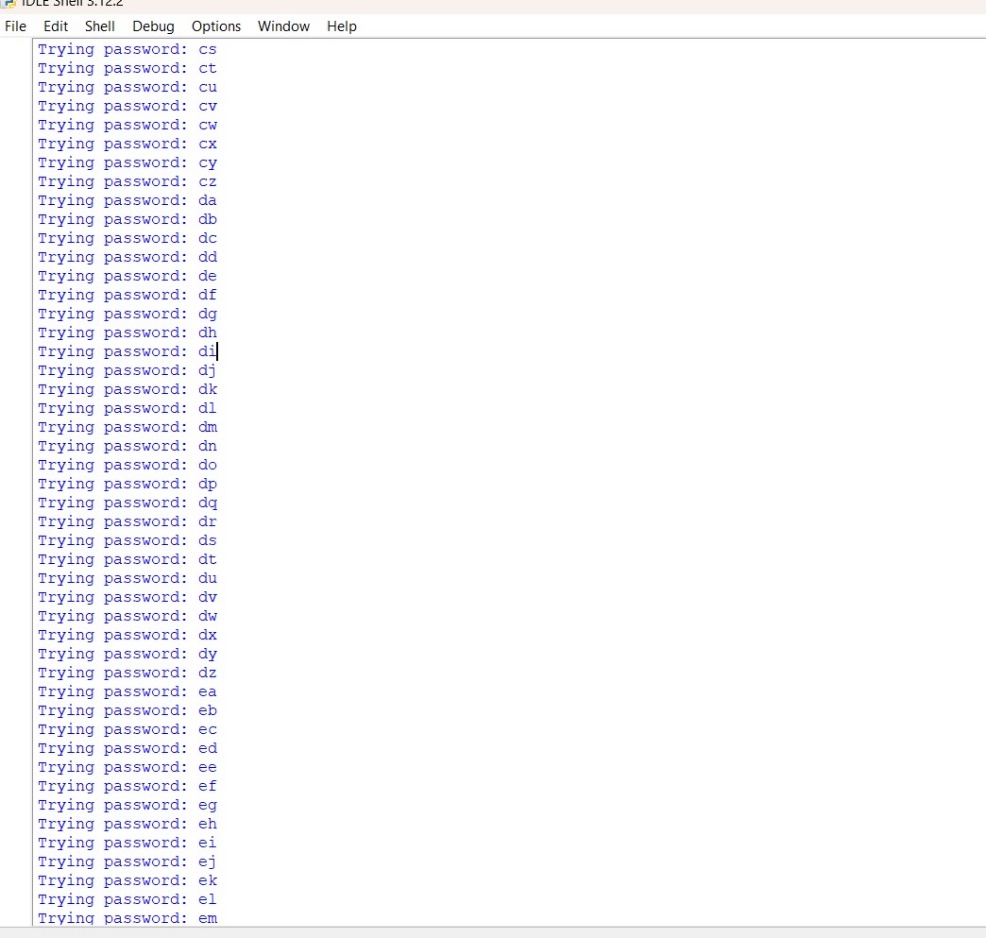
return None

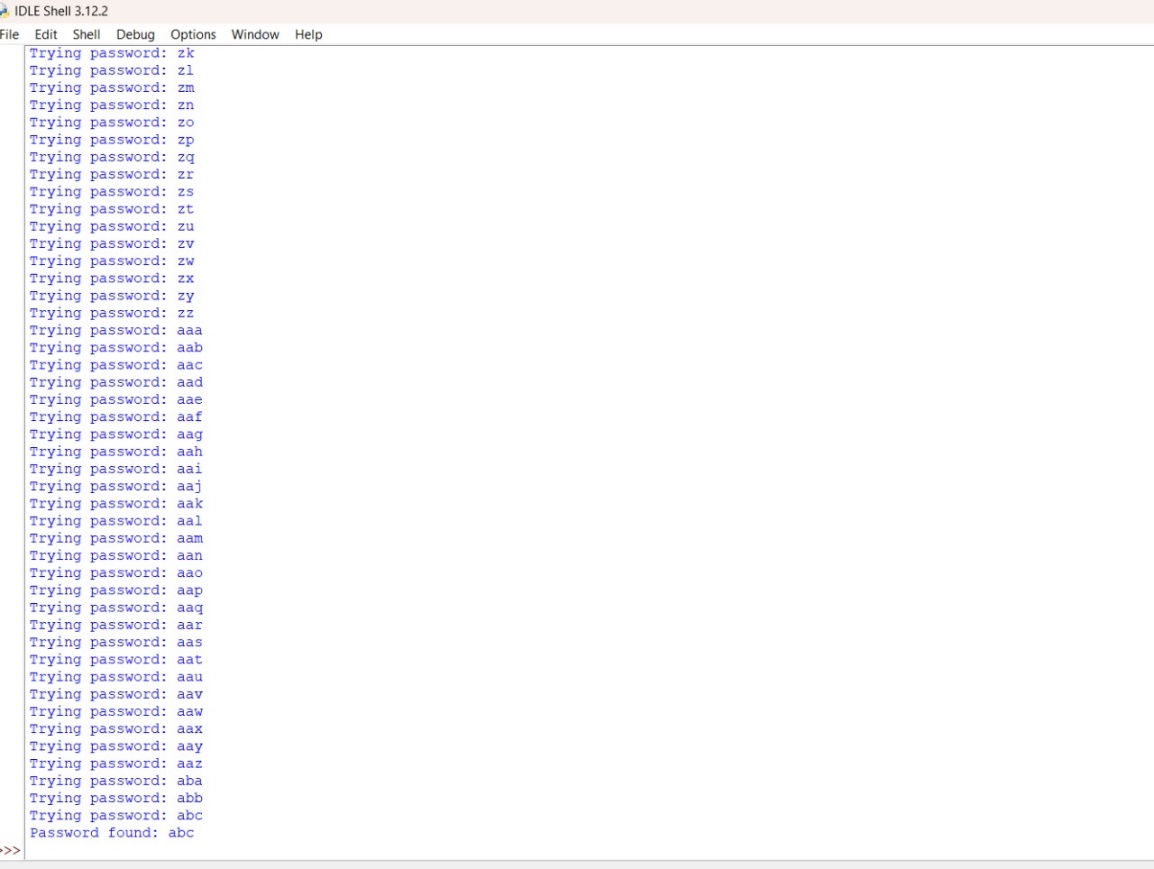
# Example usage

target\_password = "abc" # Replace with your desired target password

brute\_force\_attack(target\_password)







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

from cryptography.hazmat.primitives import hashes

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

from cryptography.hazmat.primitives import serialization

from cryptography.hazmat.primitives.asymmetric.utils import Prehashed

# Step 1: Generate RSA keys

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

public\_exponent=65537,

key\_size=2048

)

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

# Step 2: Sign the document

def sign\_document(document, private\_key):

document\_bytes = document.encode() # Convert document to bytes

signature = private\_key.sign(

document\_bytes,

padding.PSS(

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

salt\_length=padding.PSS.MAX\_LENGTH

),

hashes.SHA256()

)

return signature

# Step 3: Verify the document signature

def verify\_signature(document, signature, public\_key):

document\_bytes = document.encode() # Convert document to bytes

try:

public\_key.verify(

signature,

document\_bytes,

padding.PSS(

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

salt\_length=padding.PSS.MAX\_LENGTH

),

hashes.SHA256()

)

print("Signature is valid. Document is authentic.")

except Exception as e:

print("Signature is invalid. Document may have been tampered with.")

# Example usage

document = "This is a confidential document."

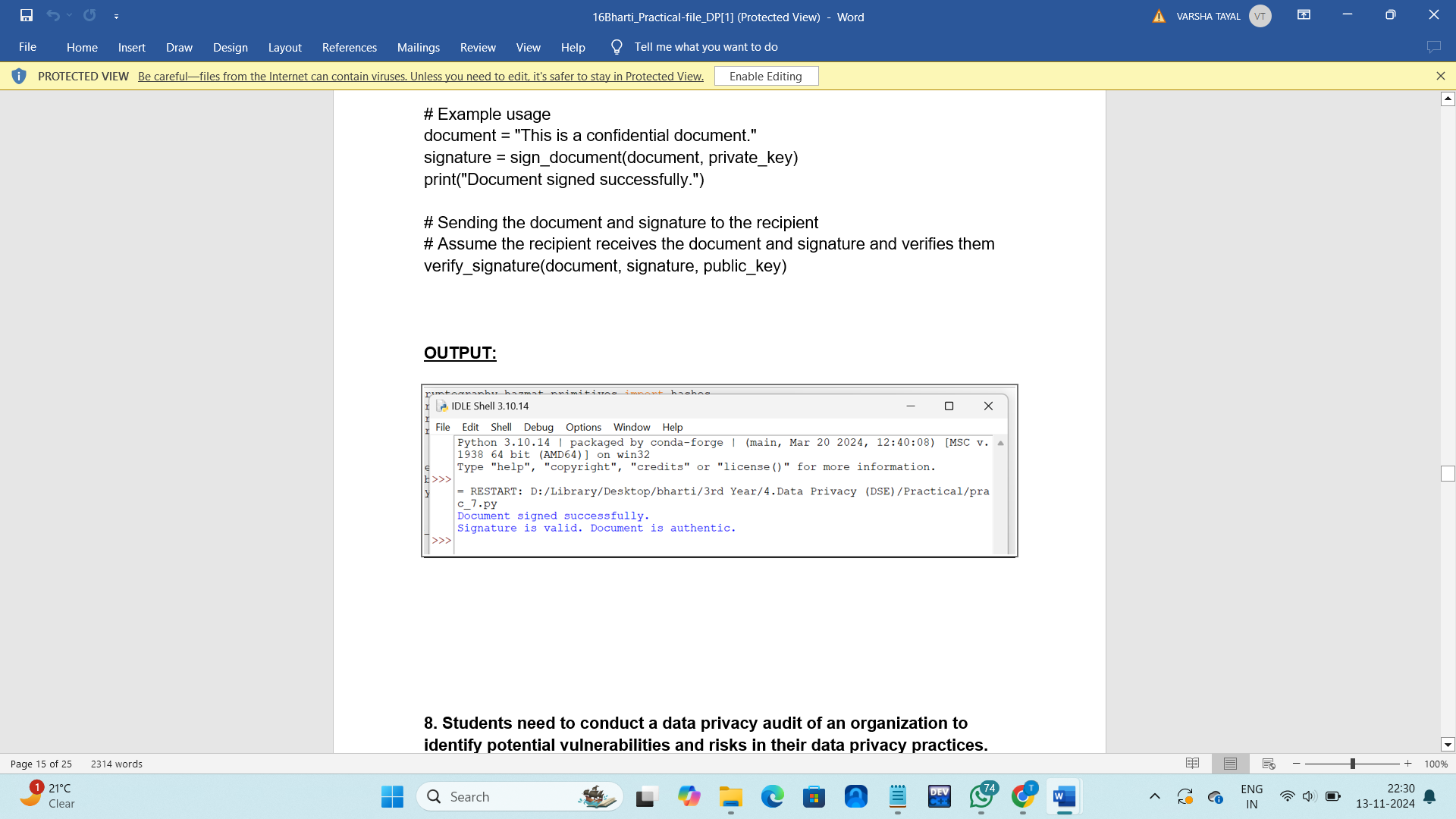
signature = sign\_document(document, private\_key)

print("Document signed successfully.")

# Sending the document and signature to the recipient

# Assume the recipient receives the document and signature and verifies them

verify\_signature(document, signature, public\_key)



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

import pandas as pd

def load\_data(filename):

"""Load data from CSV file."""

try:

return pd.read\_csv(filename)

except FileNotFoundError:

print(f"File '{filename}' not found.")

return None

def analyze\_privacy\_risks(data):

"""Analyze and flag privacy risks based on common privacy standards."""

risk\_flags = []

for \_, row in data.iterrows():

risks = []

# Check if data is encrypted

if row['Encryption'] == 'No':

risks.append("Data not encrypted")

# Check access control level

if row['Access\_Control'] not in ['Restricted', 'Limited']:

risks.append("Weak access control")

# Check retention period: flag if retention is 'Indefinite' or exceeds recommended limits

if row['Retention\_Period'] == 'Indefinite' or int(row['Retention\_Period'].split()[0]) > 3:

risks.append("Long retention period")

# Check if data is shared with third parties

if row['Shared\_With\_Third\_Parties'] == 'Yes':

risks.append("Data shared with third parties")

# Compile all identified risks for the data type

if risks:

risk\_flags.append({"Data\_Type": row['Data\_Type'], "Risks": ", ".join(risks)})

return risk\_flags

def report\_risks(risk\_flags):

"""Print out a summary report of identified risks."""

if not risk\_flags:

print("No significant privacy risks found.")

return

print("Privacy Risk Audit Report")

print("=========================")

for risk in risk\_flags:

print(f"Data Type: {risk['Data\_Type']}")

print(f"Risks: {risk['Risks']}")

print("-------------------------")

# Main program

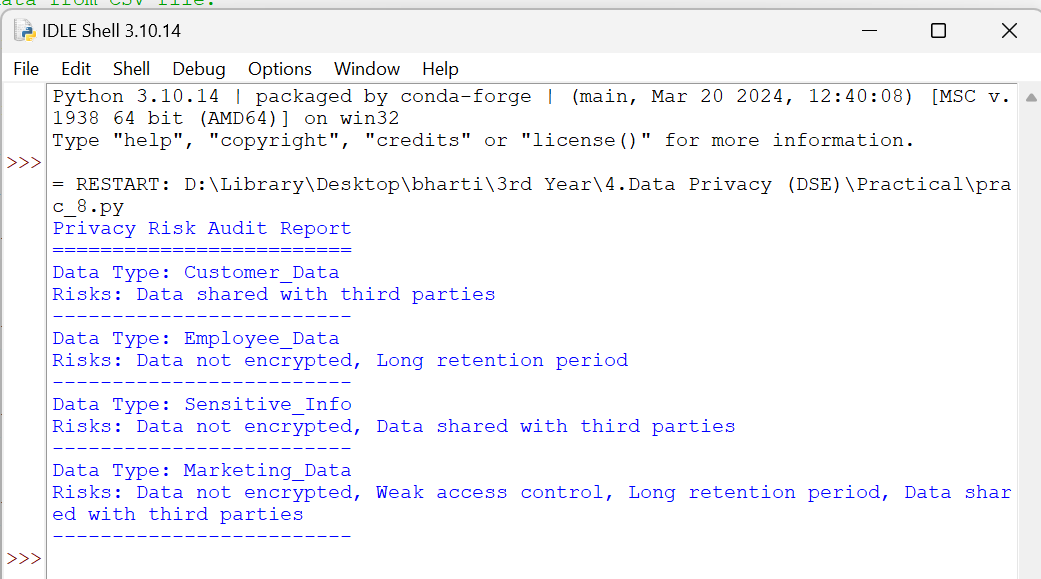
filename = 'data\_policies.csv'

data = load\_data(filename)

if data is not None:

risk\_flags = analyze\_privacy\_risks(data)

report\_risks(risk\_flags)



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

import datetime

import json

# Mock database for storing user data, consents, and audit logs

database = {

"users": [],

"consents": [],

"audit\_logs": []

}

# 1. Data Collection Audit - Track PII data stored in the system

def data\_collection\_audit(data):

pii\_fields = ["name", "email", "phone", "address", "ssn"]

pii\_data = {k: v for k, v in data.items() if k in pii\_fields}

database["users"].append(pii\_data)

log\_audit("Data Collection", f"Collected PII data: {pii\_data}")

return pii\_data

# 2. Consent Management - Track and manage user consents

def add\_user\_consent(user\_id, consent\_type):

consent\_record = {

"user\_id": user\_id,

"consent\_type": consent\_type,

"timestamp": datetime.datetime.now().isoformat()

}

database["consents"].append(consent\_record)

log\_audit("Consent Management", f"User {user\_id} provided consent for {consent\_type}")

# 3. Access Control - Check user permissions for accessing PII data

def check\_access(user\_role, data\_type):

allowed\_roles = {"admin": ["all"], "user": ["self"], "guest": []}

access\_allowed = data\_type in allowed\_roles.get(user\_role, [])

log\_audit("Access Control", f"{user\_role} access to {data\_type}: {'Allowed' if access\_allowed else 'Denied'}")

return access\_allowed

# 4. Data Minimization - Ensure only necessary data is collected

def data\_minimization\_check(data):

required\_fields = ["name", "email"] # Example of necessary fields

unnecessary\_data = {k: v for k, v in data.items() if k not in required\_fields}

if unnecessary\_data:

log\_audit("Data Minimization", f"Unnecessary data collected: {unnecessary\_data}")

else:

log\_audit("Data Minimization", "Only necessary data collected.")

return unnecessary\_data

# 5. Logging & Record-Keeping - Maintain audit log of data access and processing activities

def log\_audit(action, message):

audit\_entry = {

"timestamp": datetime.datetime.now().isoformat(),

"action": action,

"message": message

}

database["audit\_logs"].append(audit\_entry)

# Example Usage

# Collect data for a new user

user\_data = {

"user\_id": 1,

"name": "John Doe",

"email": "johndoe@example.com",

"phone": "123-456-7890",

"ssn": "987-65-4321"

}

# Run data collection audit

collected\_data = data\_collection\_audit(user\_data)

# Add consent for data usage

add\_user\_consent(user\_id=1, consent\_type="email\_marketing")

# Check access control for a guest user trying to access PII

check\_access(user\_role="guest", data\_type="all")

# Check if any unnecessary data is collected

unnecessary\_data = data\_minimization\_check(user\_data)

# Print audit logs as a JSON record for review

print("Audit Logs:", json.dumps(database["audit\_logs"], indent=2))



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.

import datetime

# Ethical assessment logs

ethics\_report = []

# 1. Privacy vs. Security Trade-off Assessment

def assess\_privacy\_vs\_security(data\_type, security\_level, justification):

# Evaluate the necessity of security measures against privacy intrusion

if security\_level > 3 and justification:

ethics\_report.append({

"timestamp": datetime.datetime.now().isoformat(),

"assessment": "Privacy vs Security",

"data\_type": data\_type,

"security\_level": security\_level,

"justification": justification,

"outcome": "Security justification adequate"

})

return "Security measures justified by context."

else:

ethics\_report.append({

"timestamp": datetime.datetime.now().isoformat(),

"assessment": "Privacy vs Security",

"data\_type": data\_type,

"security\_level": security\_level,

"justification": justification,

"outcome": "Security justification lacking"

})

return "Security measures may be excessive for this data type."

# 2. Impact Assessment on Marginalized Communities

def assess\_impact\_on\_marginalized\_communities(data\_collection\_purpose, affected\_groups, risk\_level):

# Check if data collection may negatively affect certain groups

if "marginalized" in affected\_groups and risk\_level >= 3:

ethics\_report.append({

"timestamp": datetime.datetime.now().isoformat(),

"assessment": "Impact on Marginalized Groups",

"purpose": data\_collection\_purpose,

"affected\_groups": affected\_groups,

"risk\_level": risk\_level,

"outcome": "Potential negative impact detected"

})

return "High risk of negative impact on marginalized communities."

else:

ethics\_report.append({

"timestamp": datetime.datetime.now().isoformat(),

"assessment": "Impact on Marginalized Groups",

"purpose": data\_collection\_purpose,

"affected\_groups": affected\_groups,

"risk\_level": risk\_level,

"outcome": "Impact appears minimal"

})

return "Minimal impact on marginalized communities."

# 3. Data Ethics Checklist - Ensure compliance with ethical practices

def data\_ethics\_checklist(data\_type, purpose, consent\_obtained, bias\_evaluation):

if not consent\_obtained:

ethics\_report.append({

"timestamp": datetime.datetime.now().isoformat(),

"assessment": "Ethics Checklist",

"data\_type": data\_type,

"purpose": purpose,

"consent\_obtained": consent\_obtained,

"bias\_evaluation": bias\_evaluation,

"outcome": "Consent missing"

})

return "Consent is required for ethical data collection."

elif bias\_evaluation:

ethics\_report.append({

"timestamp": datetime.datetime.now().isoformat(),

"assessment": "Ethics Checklist",

"data\_type": data\_type,

"purpose": purpose,

"consent\_obtained": consent\_obtained,

"bias\_evaluation": bias\_evaluation,

"outcome": "Potential bias detected"

})

return "Bias detected; review necessary to avoid discrimination."

else:

ethics\_report.append({

"timestamp": datetime.datetime.now().isoformat(),

"assessment": "Ethics Checklist",

"data\_type": data\_type,

"purpose": purpose,

"consent\_obtained": consent\_obtained,

"bias\_evaluation": bias\_evaluation,

"outcome": "Compliant with ethical practices"

})

return "Data collection aligns with ethical practices."

# Generate ethics report for audit and transparency

def generate\_ethics\_report(report\_path="ethics\_report.txt"):

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

report\_file.write("Ethical Data Privacy Assessment Report\n")

report\_file.write("=" \* 50 + "\n")

for entry in ethics\_report:

report\_file.write(f"Timestamp: {entry['timestamp']}\n")

report\_file.write(f"Assessment: {entry['assessment']}\n")

for key, value in entry.items():

if key not in ["timestamp", "assessment"]:

report\_file.write(f"{key.capitalize()}: {value}\n")

report\_file.write("-" \* 50 + "\n")

print(f"Ethics report generated: {report\_path}")

# Example Usage

# Privacy vs. Security Assessment

print(assess\_privacy\_vs\_security(data\_type="location", security\_level=4, justification="Location data needed for fraud detection"))

# Impact Assessment on Marginalized Communities

print(assess\_impact\_on\_marginalized\_communities(data\_collection\_purpose="ad targeting", affected\_groups=["marginalized"], risk\_level=4))

# Data Ethics Checklist

print(data\_ethics\_checklist(data\_type="purchase history", purpose="recommendations", consent\_obtained=True, bias\_evaluation=True))

# Generate the ethics report

generate\_ethics\_report()

