Data Privacy Codes

Ques1. Write a program to perform encryption and decryption using Caesar cipher (substitution cipher).

```
def encrypt(text, s):
    result = ""
    for i in range(len(text)):
        char = text[i]
        if char.isupper():
            result += chr((ord(char) + s - 65) % 26 + 65)
        else :
            result += chr((ord(char) + s - 97) % 26 + 97)
    return result

def decrypt(text, s):
    result = ""
    for i in range(len(text)):
        char = text[i]
        if char.isupper():
            result += chr((ord(char) - s - 65) % 26 + 65)
        else :
            result += chr((ord(char) - s - 97) % 26 + 97)
    return result

def menu():
    while True:
        print("\nMenu:")
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Exit")
        choice = input("Enter choice (1/2/3): ")

        if choice == '1':
            text = input("Enter text to encrypt: ")
            s = int(input("Enter shift value: "))
            print("Encrypted text:", encrypt(text, s))
        elif choice == '2':
            text = input("Enter text to decrypt: ")
            s = int(input("Enter shift value: "))
            print("Decrypted text:", decrypt(text, s))
        elif choice == '3':
            print("Exiting...")
            break
        else:
            print("Invalid choice, please try again.")

if __name__ == "__main__":
    menu()
```

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

```
import numpy as np
def encryption(plain_text, depth):
    plain_text = plain_text.replace(" ", "")
    col = len(plain_text)
    arr = np.empty((depth, col), dtype='str')
    arr.fill("")

    k = 0
    j = 0
    flag = 0
    cipher = []

    for i in range(col):
        arr[j, i] = plain_text[k]

        if j == depth - 1:
            flag = 1
        elif j == 0:
            flag = 0

        j += 1 if flag == 0 else -1
        k += 1

    for i in range(depth):
        for j in range(col):
            if arr[i, j] != "":
                cipher.append(arr[i, j])

    return ''.join(cipher)

def decryption(cipher, depth):
    col = len(cipher)
    arr = np.empty((depth, col), dtype='str')
    arr.fill("*")

    k = 0
    j = 0
    flag = 0

    for i in range(col):
        if j < depth:
            arr[j, i] = '*'

        if j == depth - 1:
            flag = 1
        elif j == 0:
```

```
            flag = 0

        j += 1 if flag == 0 else -1


    for i in range(depth):
        for j in range(col):
            if arr[i, j] == '*' and k < len(cipher):
                arr[i, j] = cipher[k]
                k += 1

    decrypted_text = []
    j = 0
    flag = 0

    for i in range(col):
        decrypted_text.append(arr[j, i])

        if j == depth - 1:
            flag = 1
        elif j == 0:
            flag = 0

        j += 1 if flag == 0 else -1

    return ''.join(decrypted_text)

def main():
    plain_text = "thank you very much"
    depth = 2

    print("Encryption")
    cipher = encryption(plain_text, depth)
    print("Encrypted text:", cipher)

    print("Decryption")
    decrypted_text = decryption(cipher, depth)
    print("Decrypted text:", decrypted_text)

if __name__ == "__main__":
    main()
```

Ques 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):
    sha256_hash = hashlib.sha256(password.encode('utf-8')).hexdigest()
    return sha256_hash

password = input("Enter the password: ")
hashed_password = hash_password(password)
print("SHA-256 Hashed Password:", hashed_password)
```

Ques 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.

Credentials :
Ben:admin123
Helen:password123
Kim:er$#%DFF

```
import hashlib
import requests

def get_sha1_hash(password):
    return hashlib.sha1(password.encode('utf-8')).hexdigest().upper()

def check_pwned_password(username, password):
    sha1_password = get_sha1_hash(password)
    prefix, suffix = sha1_password[:5], sha1_password[5:]
    url = f"https://api.pwnedpasswords.com/range/{prefix}"
    response = requests.get(url)

    if response.status_code != 200:
        raise RuntimeError(f"Error fetching data: {response.status_code}")

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

    for hash_suffix, count in hashes:
        if hash_suffix == suffix:
            return f"Password for '{username}' found {count} times in breaches!"

    return f"Password for '{username}' not found in any breaches."

def check_credentials_from_file(filename):
```

```
    try:
        with open(filename, 'r') as file:
            credentials = file.readlines()
    except FileNotFoundError:
        print(f"File {filename} not found.")
        return

    for line in credentials:
        line = line.strip()
        if line:
            username, password = line.split(":")
            result = check_pwned_password(username, password)
            print(result)

filename = "credentials.txt"
check_credentials_from_file(filename)
```

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

```
import random
import string

def generate_password(num_letters):
    try:
        with open('dictionary.txt', 'r') as file:
            words = [line.strip() for line in file.readlines() if line.strip()]
    except FileNotFoundError:
        print("Dictionary file not found!")
        return None

    password = ''

    while len(password) < num_letters:
        word = random.choice(words)
        if len(password) + len(word) <= num_letters:
            password += word

    password = password[:num_letters]

    return password

def main():
    while True:
        try:
            num_letters = int(input("Enter the number of letters for your password (max 15): "))
            if num_letters <= 15:
```

```
            break
        else:
            print("Password length must not exceed 15 characters. Try again.")
    except ValueError:
        print("Please enter a valid number.")

  password = generate_password(num_letters)
  if password:
    print(f"Generated password: {password}")

if __name__ == '__main__':
  main()
```

Ques 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
import time

def brute_force_attack(target_password):
  characters = string.ascii_lowercase
  password_length = 1

  while True:
    for guess in itertools.product(characters, repeat=password_length):
      guess = ''.join(guess)
      print(f"Trying: {guess}")

      if guess == target_password:
        return guess

    password_length += 1

target_password = "abc"

start_time = time.time()

found_password = brute_force_attack(target_password)

end_time = time.time()

print(f"Password '{found_password}' found in {end_time - start_time:.2f} seconds.")
```

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

Verify File:

```
import rsa
with open('samplepdf.pdf','rb') as f:
    pdf=f.read()
with open('public_key_file.pem','rb') as puk:
    public_key=rsa.PublicKey.load_pkcs1(puk.read())
with open('signature_file','rb') as sf:
    signature_file=sf.read()
verify_file=rsa.verify(pdf,signature_file,public_key)
verify_file
```

Sign File:

```
import rsa
with open('samplepdf.pdf','rb') as f:
    pdf=f.read()
with open('private_key_file.pem','rb') as pr:
    private_key=rsa.PrivateKey.load_pkcs1(pr.read())
signature_file=rsa.sign(pdf,private_key,'SHA-256')
print(len(signature_file))
with open('signature_file','wb') as sf:
    sf.write(signature_file)
```

Create Keys:

```
!pip install rsa
import rsa
public_key,private_key=rsa.newkeys(2048)
with open('public_key_file.pem','wb') as puk:
    puk.write(public_key.save_pkcs1('PEM'))
with open('private_key_file.pem','wb') as prk:
    prk.write(private_key.save_pkcs1('PEM'))
```

Sample pdf : This is not the right file