

**RAMANUJAN COLLEGE**

**University Of Delhi**

**NAAC Grade A++ with CGPA 3.71**

**DATA PRIVACY**

**ASSIGNMENT**

**NAME – Aditya Swaroop**

**ROLL NO – 20221405**

**EXAM ROLL NO - 22020570028**

**SEMESTER – 5th**

**SUBMITTED TO – ASMITA MA’AM**

# Write a program to perform encryption and decryption using Caesar cipher (substitutional cipher). Ans:-

def encrypt(text, shift):

result = ""

for char in text:

if char.isalpha():

# Determine the case and base ASCII value

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

# Convert to 0-25 range, apply shift and wrap around using modulo

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

# Convert back to ASCII and append

result += chr(shifted + ascii\_base)

else:

# Keep non-alphabetic characters unchanged

result += char

return result

def decrypt(text, shift):

# Decryption is just encryption with the opposite shift

return encrypt(text, -shift)

def main():

# Get input from user

message = input("Enter the message: ")

shift = int(input("Enter the shift value (1-25): "))

# Ensure shift is in valid range

shift = shift % 26

# Perform encryption

encrypted\_text = encrypt(message, shift)

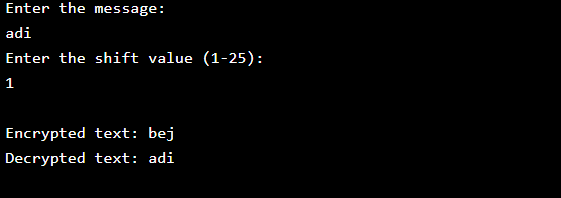
print(f"\nEncrypted text: {encrypted\_text}")

# Perform decryption

decrypted\_text = decrypt(encrypted\_text, shift)

print(f"Decrypted text: {decrypted\_text}")

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

main()  
  


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

Ans:--

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

if rails <= 1:

return text # No encryption needed if only one rail or less

# Create a rail matrix

fence = [['' for \_ in range(len(text))] for \_ in range(rails)]

direction\_down = False

row, col = 0, 0

# Fill the rail matrix

for char in text:

fence[row][col] = char

col += 1

# Change direction at top or bottom rail

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

direction\_down = not direction\_down

# Move up or down

row += 1 if direction\_down else -1

# Extract the encrypted text from the rail matrix

encrypted\_text = ''

for rail in fence:

encrypted\_text += ''.join(rail)

return encrypted\_text

def rail\_fence\_decrypt(ciphertext, rails):

if rails <= 1:

return ciphertext # No decryption needed if only one rail or less

# Create a rail matrix

fence = [['' for \_ in range(len(ciphertext))] for \_ in range(rails)]

direction\_down = None

row, col = 0, 0

# Mark the positions to fill

for \_ in range(len(ciphertext)):

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

col += 1

# Change direction at top or bottom rail

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

direction\_down = not direction\_down

# Move up or down

row += 1 if direction\_down else -1

# Fill the marked positions with the ciphertext

index = 0

for r in range(rails):

for c in range(len(ciphertext)):

if fence[r][c] == '\*' and index < len(ciphertext):

fence[r][c] = ciphertext[index]

index += 1

# Read the plaintext

plaintext = ''

row, col = 0, 0

direction\_down = None

for \_ in range(len(ciphertext)):

plaintext += fence[row][col]

col += 1

# Change direction at top or bottom rail

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

direction\_down = not direction\_down

# Move up or down

row += 1 if direction\_down else -1

return plaintext

def main():

print("Rail Fence Cipher Encryption and Decryption")

print("1. Encrypt a message")

print("2. Decrypt a message")

choice = input("Choose an option (1/2): ")

if choice not in ['1', '2']:

print("Invalid choice!")

return

text = input("Enter the text: ").replace(" ", "") # Remove spaces for simplicity

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

if choice == '1':

result = rail\_fence\_encrypt(text, rails)

print("Encrypted text:", result)

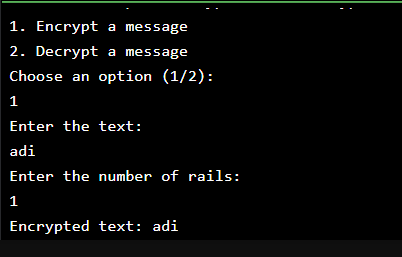
else:

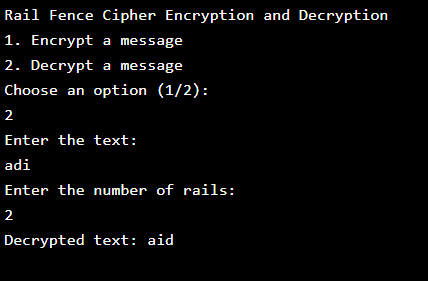
result = rail\_fence\_decrypt(text, rails)

print("Decrypted text:", result)

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

main()





# 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.

Ans:--

import hashlib

def hash\_password(password):

# Encode the password to bytes, then hash it

sha256\_hash = hashlib.sha256(password.encode())

# Return the hexadecimal representation of the hash

return sha256\_hash.hexdigest()

def main():

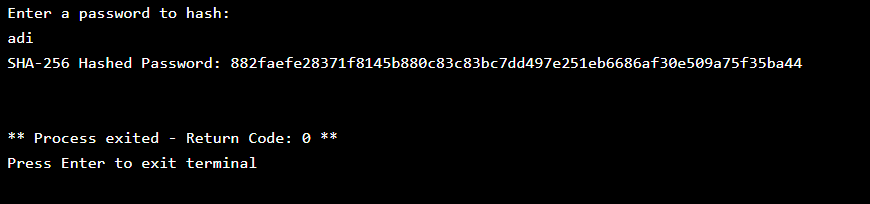
password = input("Enter a password to hash: ")

hashed\_password = hash\_password(password)

print(f"SHA-256 Hashed Password: {hashed\_password}")

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

main()



# 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.

Ans:---

import hashlib

import requests

import os

def check\_password\_pwned(password):

# Hash the password using SHA-1

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

prefix, suffix = sha1\_hash[:5], sha1\_hash[5:]

# Query the API with the hash prefix

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

try:

response = requests.get(url)

if response.status\_code != 200:

raise Exception(f"Error querying API: {response.status\_code}")

# Check if the hash suffix is in the API's response

hashes = response.text.splitlines()

for line in hashes:

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

if hash\_suffix == suffix:

return int(count)

return 0

except requests.RequestException as e:

raise Exception(f"Network error: {e}")

def process\_file(filename):

try:

with open(filename, 'r') as file:

lines = file.readlines()

for line in lines:

username, password = line.strip().split(',')

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

if pwned\_count > 0:

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

else:

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

except FileNotFoundError:

print("Error: File not found.")

except Exception as e:

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

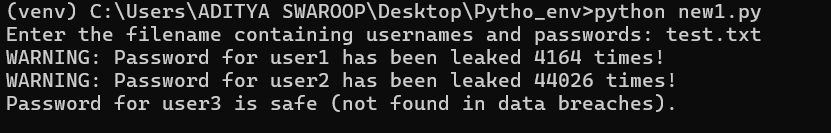
def main():

filename = input("Enter the filename containing usernames and passwords: ")

process\_file(filename)

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

main()



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

import random

def generate\_password(filename, desired\_length=12):

try:

with open(filename, 'r') as f:

words = [word.strip() for word in f.readlines()]

password = ""

while len(password) < desired\_length:

# Randomly pick a word from the list

word = random.choice(words)

# Shuffle the letters of the selected word

shuffled\_word = ''.join(random.sample(word, len(word)))

password += shuffled\_word

# Trim the password to the desired length

password = password[:desired\_length]

return password

except FileNotFoundError:

return "Error: Dictionary file not found"

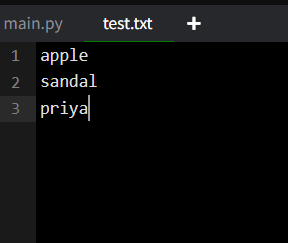
def main():

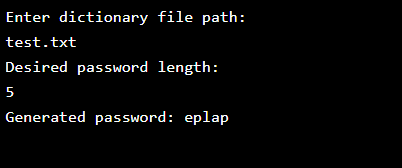
filename = input("Enter dictionary file path: ")

desired\_length = int(input("Desired password length: "))

print(f"Generated password: {generate\_password(filename, desired\_length)}")

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

main()  




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

import itertools

import string

import time

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

# Define the character set (lowercase letters, uppercase letters, digits)

charset = string.ascii\_letters + string.digits

# Start brute-forcing

start\_time = time.time()

# Generate all possible combinations of the charset with lengths starting from 1

for length in range(1, len(target\_password) + 1):

for attempt in itertools.product(charset, repeat=length):

guess = ''.join(attempt)

print(f"Trying: {guess}") # You can remove or comment this to stop printing guesses

if guess == target\_password:

elapsed\_time = time.time() - start\_time

print(f"\nPassword found: {guess}")

print(f"Time taken: {elapsed\_time:.2f} seconds")

return guess

print("Password not found")

return None

def main():

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

brute\_force\_attack(target\_password)

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

main()

