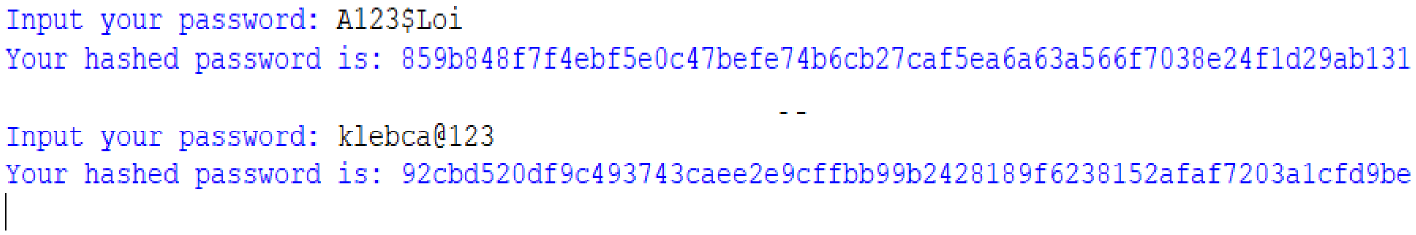
1. **Write a Python program that defines a function and takes a password string as input and returns its SHA-256 hashed representation as a hexadecimal string.**

```
import hashlib
def hash_password(password):
   # Encode the password as bytes
   password_bytes = password.encode('utf-8')

   # Use SHA-256 hash function to create a hash object
   hash_object = hashlib.sha256(password_bytes)

   # Get the hexadecimal representation of the hash
   password_hash = hash_object.hexdigest()

   return password_hash

password = input("Input your password: ")
hashed_password = hash_password(password)
print(f"Your hashed password is: {hashed_password}")
```

output:

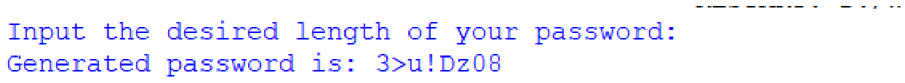


```
Input your password: A123$Loi
Your hashed password is: 859b848f7f4ebf5e0c47befe74b6cb27caf5ea6a63a566f7038e24f1d29ab131
                                                                       --
Input your password: klebca@123
Your hashed password is: 92cbd520df9c493743caee2e9cffbb99b2428189f6238152afaf7203a1cfd9be
```

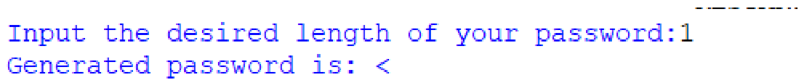
2. **Write a Python program that defines a function to generate random passwords of a specified length. The function takes an optional parameter length, which is set to 8 by default. If no length is specified by the user, the password will have 8 characters.**

```
import random
import string
def generate_password(length=8):
    # Define the characters to use in the password
    all_characters = string.ascii_letters + string.digits + string.punctuation

    # Use the random module to generate the password
    password = ''.join(random.choice(all_characters) for i in range(length))

    return password

password_length_str = input("Input the desired length of your password:")
if password_length_str:
    password_length = int(password_length_str)
else:
    password_length = 8

password = generate_password(password_length)
print(f"Generated password is: {password}")
```

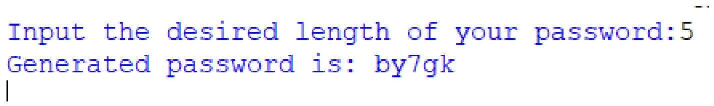
Output :



```
Input the desired length of your password:
Generated password is: 3>u!Dz08
```



```
Input the desired length of your password:1
Generated password is: <
```

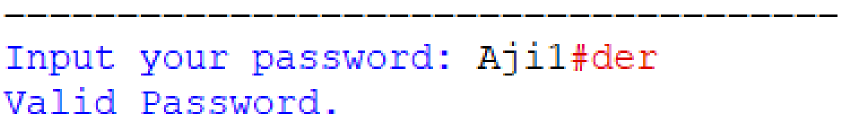


```
Input the desired length of your password:5
Generated password is: by7gk
```

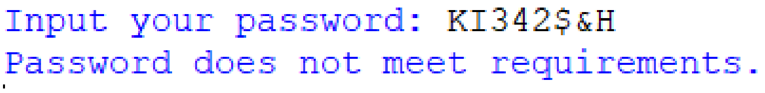
3. **Write a Python program to check if a password meets the following criteria: a. At least 8 characters long, b. Contains at least one uppercase letter, one lowercase letter, one digit, and one special character (!, @, #, $, %, or &), c. If the password meets the criteria, print a message that says "Valid Password." If it doesn'tmeet the criteria, print a message that says "Password does not meet requirements."**

```
import re


def validate_password(password):
    # Check if the password has at least 8 characters
    if len(password) < 8:
        return False

    # Check if the password contains at least one uppercase letter
    if not re.search(r'[A-Z]', password):
        return False

    # Check if the password contains at least one lowercase letter
    if not re.search(r'[a-z]', password):
        return False

    # Check if the password contains at least one digit
    if not re.search(r'\d', password):
        return False

    # Check if the password contains at least one special character
    if not re.search(r'[!@#$%^&*(),.?":{}|<>]', password):
        return False

    # If all the conditions are met, the password is valid
    return True

password = input("Input your password: ")
is_valid = validate_password(password)

if is_valid:
    print("Valid Password.")
else:
    print("Password does not meet requirements.")
```

Output :



```
Input your password: Aji1#der
Valid Password.
```



```
Input your password: KI342$&H
Password does not meet requirements.
```

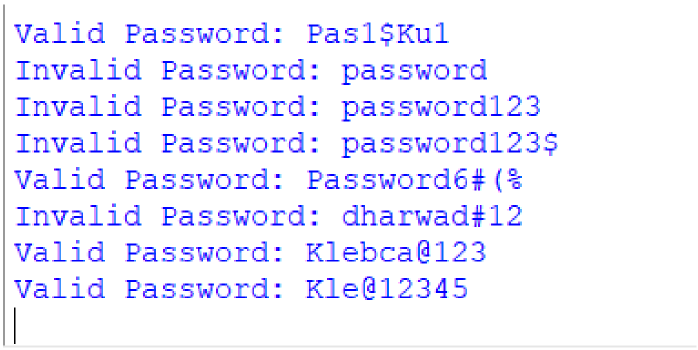
4. **Write a Python program that reads a file containing a list of passwords, one per line. It checks each password to see if it meets certain requirements (e.g. at least 8 characters, contains both uppercase and lowercase letters, and at least one number and one special character). Passwords that satisfy the requirements should be printed by the program.**

Passwords.txt

```
Pas1$Ku1
password
password123
password123$
Password6#(%
dharwad#12
Klebca@123
Kle@12345
```

```
import re
def check_password(password):
    # Define regular expressions for each requirement
    length_regex = re.compile(r'^.{8,}$')
    uppercase_regex = re.compile(r'[A-Z]')
    lowercase_regex = re.compile(r'[a-z]')
    digit_regex = re.compile(r'\d')
    special_regex = re.compile(r'[\W_]')

    # Check if password meets each requirement
    length_check = length_regex.search(password)
    uppercase_check = uppercase_regex.search(password)
    lowercase_check = lowercase_regex.search(password)
    digit_check = digit_regex.search(password)
    special_check = special_regex.search(password)

    # Return True if all requirements are met, False otherwise
    if length_check and uppercase_check and lowercase_check and digit_check and special_check:
        return True
    else:
        return False

# Open file containing passwords
with open('passwords.txt') as f:
    # Read each password from file and check if it meets requirements
    for password in f:
        password = password.strip()  # Remove newline character
        if check_password(password):
            print("Valid Password: "+password)
        else:
            print("Invalid Password: "+password)
```

Output :



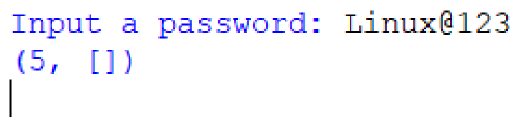
```
Valid Password: Pas1$Ku1
Invalid Password: password
Invalid Password: password123
Invalid Password: password123$
Valid Password: Password6#(%
Invalid Password: dharwad#12
Valid Password: Klebca@123
Valid Password: Kle@12345
```

5. **Write a Python program that creates a password strength meter. The program should prompt the user to enter a password and check its strength based on criteria such as length, complexity, and randomness. Afterwards, the program should provide suggestions for improving the password's strength.**

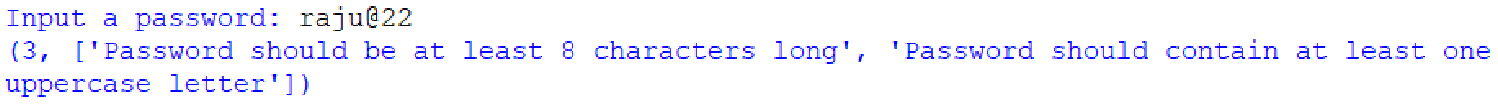
```
import re


def check_password_strength(password):
    score = 0
    suggestions = []

    # check length
    if len(password) >= 8:
        score += 1
    else:
        suggestions.append("Password should be at least 8 characters long")

    # check for uppercase letter
    if re.search(r"[A-Z]", password):
        score += 1
    else:
        suggestions.append("Password should contain at least one uppercase letter")

    # check for lowercase letter
    if re.search(r"[a-z]", password):
        score += 1
    else:
        suggestions.append("Password should contain at least one lowercase letter")

  # check for numeric digit
    if re.search(r"\d", password):
        score += 1
    else:
        suggestions.append("Password should contain at least one numeric digit")

    # check for special character
    if re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
        score += 1
    else:
        suggestions.append("Password should contain at least one special character (!@#$%^&*(),.?\":{}|<>)")

    return score, suggestions
password = input("Input a password: ")
```

print(check_password_strength(password))

Output :



```
Input a password: Linux@123
(5, [])
```



```
Input a password: raju@22
(3, ['Password should be at least 8 characters long', 'Password should contain at least one
uppercase letter'])
```

6. **Write a Python program that reads a file containing a list of usernames and passwords, one pair per line (separatized 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 Owned" API (https://haveibeenpwned.com/API/v3) to check if a password has been leaked.**

Passwords.txt

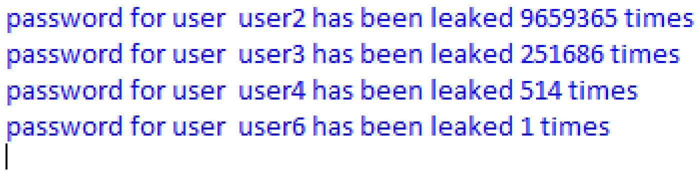
```
user1,pas1$Ku1

user2,password

user3,password123

user4,password123$

user5,password6#(%

user6,Germany#12
```

```
import requests

import hashlib


#Read the file containing username and passwords
```

```
with open("passwords.txt",'r') as f:
    for line in f:
        #split the line into username and password
        username , password = line.strip().split(',')

        #Hash the password using SHA-1 algorithm
        passwords_hash = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()

        #Make a request to "Have i pwned" API to check if the password
        response = requests.get(f"https://api.pwnedpasswords.com/range/{passwords_hash[:5]}")

       #If the response status code is 200,it means the password has been leaked
        if response.status_code == 200:

            #Get the list of hashes of leaked passwords that start with the same 5 characters as the input password
            hashes = [line.split(':')for line in response.text.splitlines()]

            #Check if the hash of the input password matches any of the leaked password hashes
            for h,count in hashes:
                if passwords_hash[5:] == h:
```

```
             print(f"password for user  {username} has been leaked {count} times")

             break


       else:

           print(f"could not check password for user  {username}.")
```

output:

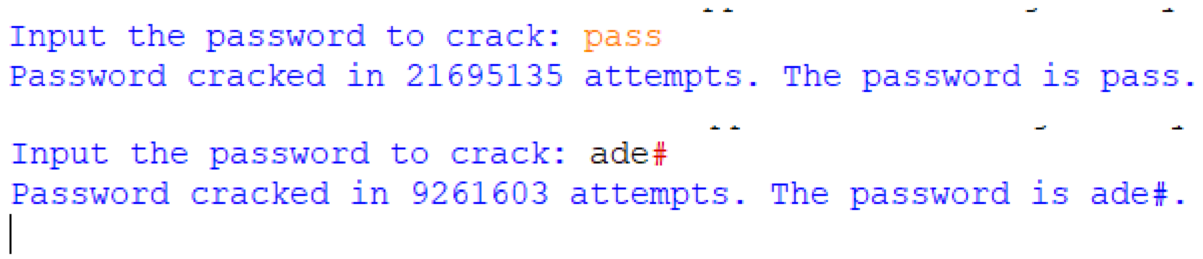


```
password for user  user2 has been leaked 9659365 times
password for user  user3 has been leaked 251686 times
password for user  user4 has been leaked 514 times
password for user  user6 has been leaked 1 times
```

7. **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 bruteforce_attack(password):

  chars = string.printable.strip()

  attempts = 0

  for length in range(1, len(password) + 1):

    for guess in itertools.product(chars, repeat=length):

      attempts += 1

      guess = ''.join(guess)

      if guess == password:
            return (attempts, guess)
    return (attempts, None)

password = input("Input the password to crack: ")
attempts, guess = bruteforce_attack(password)
if guess:
    print(f"Password cracked in {attempts} attempts. The password is {guess}.")
else:
    print(f"Password not cracked after {attempts} attempts.")
```

Output :

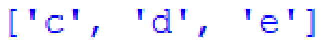


```
Input the password to crack: pass
Password cracked in 21695135 attempts. The password is pass.

Input the password to crack: ade#
Password cracked in 9261603 attempts. The password is ade#.
```

8. Python program for implementation symmetric encryption using Caesar cipher algorithm

```
# Define a function named caesar_encrypt that takes two arguments, 'realText' and 'step'.
def caesar_encrypt(realText, step):
    # Initialize two empty lists to store the output and the corresponding numeric values.
    outText = []
    cryptText = []
    # Define lists for uppercase and lowercase letters of the English alphabet.
    uppercase = ['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']
    lowercase = ['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']

    # Iterate through each letter in the 'realText' string.
    for eachLetter in realText:
        # Check if the letter is an uppercase letter.
        if eachLetter in uppercase:
            # Find the index of the letter in the 'uppercase' list.
            index = uppercase.index(eachLetter)

            # Perform Caesar cipher encryption by adding 'step' and taking the modulus 26.
            crypting = (index + step) % 26
            cryptText.append(crypting)

            # Find the new letter corresponding to the encrypted value and append it to the 'outText' list.
            newLetter = uppercase[crypting]
            outText.append(newLetter)
        # Check if the letter is a lowercase letter.
        elif eachLetter in lowercase:
            # Find the index of the letter in the 'lowercase' list.
            index = lowercase.index(eachLetter)
            # Perform Caesar cipher encryption by adding 'step' and taking the modulus 26.
            crypting = (index + step) % 26
            cryptText.append(crypting)

            # Find the new letter corresponding to the encrypted value and append it to the 'outText' list.
            newLetter = lowercase[crypting]
            outText.append(newLetter)

    # Return the 'outText' list containing the encrypted letters.
    return outText

# Call the caesar_encrypt function with the input 'abc' and a step of 2, and store the result in 'code'.
code = caesar_encrypt('abc', 2)

# Print an empty line for spacing.
print()

# Print the 'code', which contains the result of the Caesar cipher encryption.
print(code)

# Print an empty line for spacing.
print()
```

Output :



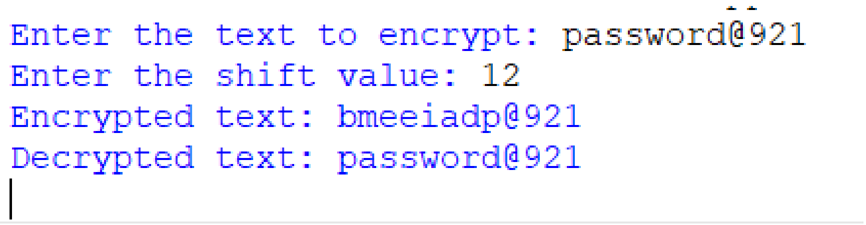
```
['c', 'd', 'e']
```

9. Python program implementation for hacking Caesar cipher algorithm

```
def encrypt(text, shift):
    result = ""

    for char in text:
        if char.isalpha():
            # Determine whether the character is uppercase or lowercase
            is_upper = char.isupper()

            # Shift the character and handle wrapping around the alphabet
            shifted_char = chr((ord(char) + shift - ord('A' if is_upper else 'a')) % 26 + ord('A' if is_upper else 'a'))

            result += shifted_char
        else:
            # If the character is not a letter, keep it unchanged
            result += char

    return result
```

```
def decrypt(text, shift):
    # Decryption is just encryption with a negative shift
    return encrypt(text, -shift)


def main():
    # Get user input
    plaintext = input("Enter the text to encrypt: ")
    shift = int(input("Enter the shift value: "))

    # Encrypt the input text
    ciphertext = encrypt(plaintext, shift)
    print("Encrypted text:", ciphertext)

    # Decrypt the encrypted text
    decrypted_text = decrypt(ciphertext, shift)
    print("Decrypted text:", decrypted_text)


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

Output :

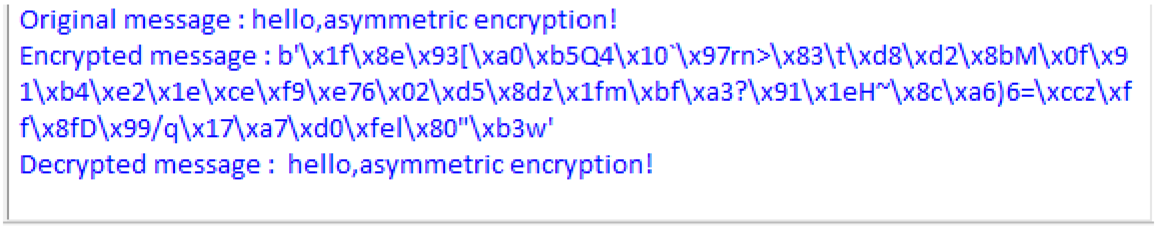


```
Enter the text to encrypt: password@921
Enter the shift value: 12
Encrypted text: bmeeiadp@921
Decrypted text: password@921
```

10. Python program to implement asymmetric encryption using rsa python library

```
import rsa

def generate_key_pair():
    # Generate a pair of public and private keys
    public_key, private_key = rsa.newkeys(512)  # You can adjust the key size as needed

    return public_key, private_key

def encrypt_message(message, public_key):
    # Encrypt the message using the public key
    encrypted_message = rsa.encrypt(message.encode('utf-8'), public_key)

    return encrypted_message

def decrypt_message(encrypted_message, private_key):
    # Decrypt the message using the private key
    decrypted_message = rsa.decrypt(encrypted_message, private_key).decode('utf-8')

    return decrypted_message

if __name__ == "__main__":
    # Generate key pair
    public_key, private_key = generate_key_pair()

    # Message to be encrypted
    original_message = "Hello, asymmetric encryption!"

    # Encrypt the message using the public key
    encrypted_message = encrypt_message(original_message, public_key)
    print("Original Message:", original_message)
    print("Encrypted Message:", encrypted_message)

    # Decrypt the message using the private key
    decrypted_message = decrypt_message(encrypted_message,
  private_key)

    print("Decrypted Message:", decrypted_message)
```

Output :



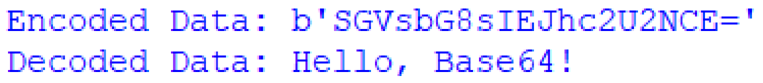
```
Original message : hello,asymmetric encryption!
Encrypted message : b'\x1f\x8e\x93[\xa0\xb5Q4\x10`\x97rn>\x83\t\xd8\xd2\x8bM\x0f\x9
1\xb4\xe2\x1e\xce\xf9\xe76\x02\xd5\x8dz\x1fm\xbf\xa3?\x91\x1eH~\x8c\xa6)6=\xccz\xf
f\x8fD\x99/q\x17\xa7\xd0\xfel\x80"\xb3w'
Decrypted message :  hello,asymmetric encryption!
```

11. Python program for encoding and decoding using Base64

```
import base64


def encode_base64(data):

  # Encode data to base64

  encoded_data = base64.b64encode(data)

  return encoded_data


def decode_base64(encoded_data):

  # Decode base64 data

  decoded_data = base64.b64decode(encoded_data)

  return decoded_data
```

```
# Example usage
original_data = b"Hello, Base64!"


# Encoding
encoded_data = encode_base64(original_data)
print("Encoded Data:", encoded_data)

# Decoding
decoded_data = decode_base64(encoded_data)
print("Decoded Data:", decoded_data.decode('utf-8'))
```

Output :



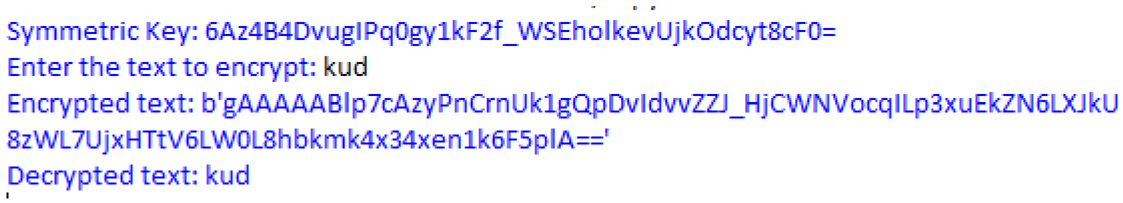
```
Encoded Data: b'SGVsbG8sIEJhc2U2NCE='
Decoded Data: Hello, Base64!
```

12. Python program to implement symmetric encryption using python library

```
from cryptography.fernet import Fernet


# Generate a key for symmetric encryption
def generate_key():
    return Fernet.generate_key()
```

```
# Encrypt the text using symmetric encryption
def encrypt_symmetric(text, key):
    cipher_suite = Fernet(key)
    encrypted_text = cipher_suite.encrypt(text.encode())
    return encrypted_text

# Decrypt the text using symmetric decryption
def decrypt_symmetric(encrypted_text, key):
    cipher_suite = Fernet(key)
    decrypted_text = cipher_suite.decrypt(encrypted_text).decode()
    return decrypted_text

def main():
    # Generate a key for symmetric encryption
    key = generate_key()
    print("Symmetric Key:", key.decode())

    # Get user input
    plaintext = input("Enter the text to encrypt: ")

    # Encrypt the plaintext using symmetric encryption
    encrypted_text = encrypt_symmetric(plaintext, key)
    print("Encrypted text:", encrypted_text)

    # Decrypt the text using symmetric decryption
    decrypted_text = decrypt_symmetric(encrypted_text, key)
    print("Decrypted text:", decrypted_text)

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

output:



```
Symmetric Key: 6Az4B4DvugIPq0gy1kF2f_WSEholkevUjkOdcyt8cF0=
Enter the text to encrypt: kud
Encrypted text: b'gAAAAABlp7cAzyPnCrnUk1gQpDvIdvvZZJ_HjCWNVocqILp3xuEkZN6LXJkU
8zWL7UjxHTtV6LW0L8hbkmk4x34xen1k6F5plA=='
Decrypted text: kud
```