#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_sha256(password):
    sha256=hashlib.sha256()
    sha256.update(password.encode('utf-8'))
    hashed_password=sha256.hexdigest()
    return hashed_password

password=input("Enter your password:")
hashed_password=hash_password_sha256(password)

print("SHA-256 Hashed Password:",hashed_password)
```

Output:

```
Enter your password:sai1234
SHA-256 Hashed Password:
fle0f63f10187d7bda3e66f00c812edbaac6498797c3216422f197234af8224
```

#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 string
import random

def generate_random_password(length=8):
    characters=string.ascii_letters+string.digits+string.punctuation
    password="".join(random.choice(characters) for _ in range(length))
    return password

password_length=int(input("Enter the length of the password(default is 8):"))
if password_length<=0:
    print("Invalid password length. Using default length of 8.")
    password=generate_random_password()
else:
    password=generate_random_password(password_length)

print("Generated Password:",password)
```

Output:

```
Enter the length of the password(default is 8):9
Generated Password: RV>?#B,9x
```

#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't meet the criteria, print a message that says "Password does not meet requirements."

```
import re

def is_valid_password(password):
    if len(password)<8:
        return False
    regex=re.compile(r'^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*["|!@#$%^&*()])')

    if not regex.search(password):
        return False
    return True

password=input("Enter your password:")
if is_valid_password(password):
    print("Valid Password")
else:
    print("Password does not meet requirement")
```

Output:

```
Enter your password:Sainath@8696
Valid Password
```

#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.

```
import re

def is_valid_password(password):
    if len(password)<8:
        return False
    if not any(char.isupper() for char in password) or not any(char.islower() for char in
password):
        return False
    if not any(char.isdigit() for char in password):
        return False
    special_characters=r"[!@#$%^&*()<>?\":{}|<>]"
    if not re.search(special_characters,password):
        return False
    return True

def main():
    file_path=input("Enter the path to the file containing passwords:")

    try:
        with open(file_path,'r') as file:
            passwords=file.read().splitlines()
            valid_passwords=[password for password in passwords if
is_valid_password(password)]
            print("Valid passwords:")
            for password in valid_passwords:
                print(password)

    except FileNotFoundError:
        print("File not found. Please check the file path and try again")
    except Exception as e:
        print(f"An error occured:{e}")

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

Output:

```
Enter the path to the file containing passwords:C:\Users\pc\Desktop\test1.txt
Valid passwords:
Sai@45678
```

#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):
    length = len(password)
    complexity = len(set(password))
    has_upper = any(char.isupper() for char in password)
    has_lower = any(char.islower() for char in password)
    has_digit = any(char.isdigit() for char in password)
    has_special = re.search(r"[!@#$%^&*(),.?\":{}|<>]", password)


    strength = 0
    if length >= 8:
        strength += 1

    if complexity >= length * 0.75:
        strength += 1

    if has_upper and has_lower:
        strength += 1

    if has_digit:
        strength += 1

    if has_special:
        strength += 1

    return strength

def main():
    password = input("Enter your password: ")

    strength = check_password_strength(password)

    print("\nPassword Strength:", strength, "out of 5")

    if strength < 5:
        print("\nSuggestions for improving password strength:")
        if len(password) < 8:
            print("- Password should be at least 8 characters long.")
        if len(set(password)) < len(password) * 0.75:
            print("- Use a mix of different characters to increase complexity.")
```

```
        if not any(char.isupper() for char in password) or not any(char.islower() for char in
password):
            print("- Include both uppercase and lowercase characters.")
        if not any(char.isdigit() for char in password):
            print("- Include at least one digit.")
        if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
            print("- Include at least one special character.")

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

Output:

Enter your password: Sainath#123

Password Strength: 5 out of 5

#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 databreach. You can use the "Have I Been Pwned" API (https://haveibeenpwned.com/API/v3) to check if a password has been leaked.

```
import requests
import hashlib
import os


def check_password_leak(password):
    sha1_hash = hashlib.sha1(password.encode()).hexdigest().upper()
    prefix, suffix = sha1_hash[:5], sha1_hash[5:]

    api_url = f'https://api.pwnedpasswords.com/range/{prefix}'
    response = requests.get(api_url)
    if suffix in response.text:
        return True
    else:
        return False


def main():
    file_name = 'test1.txt'
    file_path = os.path.abspath(file_name)

    if not os.path.isfile(file_path):
        print(f"File not found:{file_path}")
        return
    with open(file_path,'r') as file:
        for line in file:
            line = line.strip()
            if not line:
                continue
            parts = line.split(',')
            if len(parts) != 2:
                print(f"Invalid line format(expected 'username, password'):{line}")
                continue

            username, password = parts
            is_leaked = check_password_leak(password)

            if is_leaked is None:
                print(f"Could not check password for user '{username}'.")
            elif is_leaked:
                print(f"Password for user '{username}' has been leaked in a data breach.")
            else:
                print(f"Password for user '{username}' is secure.")
```

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

Output:

Password for user 'abcd123@gmail.com' is secure.

#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:abc
Password cracked in 98337 attempts.The password is abc.
```

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

```
def encrypt(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            if char.isupper():
                result += chr((ord(char) + shift - 65) % 26 + 65)
            else:
                result += chr((ord(char) + shift - 97) % 26 + 97)
        else:
            result += char
    return result

def decrypt(ciphertext, shift):
    return encrypt(ciphertext, -shift)

def main():
    plaintext = input("Enter the text to encrypt: ")
    shift = int(input("Enter the shift value: "))
    ciphertext = encrypt(plaintext, shift)
    print("Encrypted text:", ciphertext)
    decrypted_text = decrypt(ciphertext, shift)
    print("Decrypted text:", decrypted_text)

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

Output:

```
Enter the text to encrypt: Hello
Enter the shift value: 3
Encrypted text: Khoor
Decrypted text: Hello
```

#9. Python program implementation for hacking Caesar cipher algorithm

```
message = 'RNQTG HTRJGXIN APQ'
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
for key in range(len(LETTERS)):
    translated = ''
    for symbol in message:
        if symbol in LETTERS:
            num = LETTERS.find(symbol)
            num = num - key
            if num < 0:
                num = num + len(LETTERS)
            translated = translated + LETTERS[num]
        else:
            translated = translated + symbol

    print('Key #%s: %s' % (key, translated))
```

Output:

```
Key #0: RNQTG HTRJGXIN APQ
Key #1: QMPSF GSQIFWHM ZOP
Key #2: PLORE FRPHEVGL YNO
Key #3: OKNQD EQOGDUFK XMN
Key #4: NJMPC DPNFCTEJ WLM
Key #5: MILOB COMEBSDI VKL
Key #6: LHKNA BNLDARCH UJK
Key #7: KGJMZ AMKCZQBG TIJ
Key #8: JFILY ZLJBYPAF SHI
Key #9: IEHKX YKIAXOZE RGH
Key #10: HDGJW XJHZWNYD QFG
Key #11: GCFIV WIGYVMXC PEF
Key #12: FBEHU VHFXULWB ODE
Key #13: EADGT UGEWTKVA NCD
Key #14: DZCFS TFDVSJUZ MBC
Key #15: CYBER SECURITY LAB
Key #16: BXADQ RDBTQHSX KZA
Key #17: AWZCP QCASPGRW JYZ
Key #18: ZVYBO PBZROFQV IXY
Key #19: YUXAN OAYQNEPU HWX
Key #20: XTWZM NZXPMDOT GVW
Key #21: WSVYL MYWOLCNS FUV
Key #22: VRUXK LXVNKBMR ETU
Key #23: UQTWJ KWUMJALQ DST
Key #24: TPSVI JVTLIZKP CRS
Key #25: SORUH IUSKHYJO BQR
```

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

```
import rsa

def generate_key_pair():
    (public_key, private_key) = rsa.newkeys(512)
    return public_key, private_key

def encrypt_message(message, public_key):
    encrypted_message = rsa.encrypt(message.encode(), public_key)
    return encrypted_message

def decrypt_message(encrypted_message, private_key):
    decrypted_message = rsa.decrypt(encrypted_message, private_key)
    return decrypted_message.decode()

if __name__ == "__main__":
    public_key, private_key = generate_key_pair()
    original_message = "Hello, this is a secret message!"
    encrypted_message = encrypt_message(original_message, public_key)
    print(f"Original Message: {original_message}")
    print(f"Encrypted Message: {encrypted_message}")
    decrypted_message = decrypt_message(encrypted_message, private_key)
    print(f"Decrypted Message: {decrypted_message}")
```

Output:

Original Message: Hello, this is a secret message!
Encrypted Message:
b'\x17\xc4|\x97u\x92dRt/F\xe0\xb9hX\x1f\x82\xcbUe\xcc\xc9(\x0e\xc0\x05h\xd5\x17l\xdd\xa4\xb7\x9d\xc3\x15Jgj\x80@Z?0\xb1\x05\x07^w\xa1\x0e\xabBF\xbf\xe0\x97\xbc\xef\xf7\xf9\\\x99\xc3'
Decrypted Message: Hello, this is a secret message!

#11. Python program for encoding and decoding using Base64

```
import base64

def encode_base64(data):
    encoded_data = base64.b64encode(data.encode('utf-8'))
    return encoded_data.decode('utf-8')

def decode_base64(encoded_data):
    decoded_data = base64.b64decode(encoded_data.encode('utf-8'))
    return decoded_data.decode('utf-8')

original_data = "Hello, Base64 encoding and decoding in Python!"
encoded_data = encode_base64(original_data)
decoded_data = decode_base64(encoded_data)

print("Original Data:", original_data)
print("Encoded Data:", encoded_data)
print("Decoded Data:", decoded_data)
```

Output:

Original Data: Hello, Base64 encoding and decoding in Python!
Encoded Data:
SGVsbG8sIEJhc2U2NCBlbmNvZGluZyBhbmQgZGVjb2RpbmcgaW4gUHl0aG9uIQ==
Decoded Data: Hello, Base64 encoding and decoding in Python!

#12. Python program to implement symmetric encryption using python library

```
from cryptography.fernet import Fernet

key = Fernet.generate_key()
f = Fernet(key)
text = input("Enter the text:")
msg = text.encode()
encrypted_msg = f.encrypt(msg)
decrypted_msg = f.decrypt(encrypted_msg)

print("Original message:", msg.decode())
print("Encrypted message:", encrypted_msg)
print("Decrypted message:", decrypted_msg)
```

Output:

```
Enter the text:HeIIo world!
Original message: Hello world!
Encrypted message:
b’gAAAAABnWGsQsMgpxWpxpjHuHS7EiAmtwSqW656SDbb_zDIOEEaKqSaeHLIw103
8zRN7JIFssDW1sX01Us_IIzSOKg3sQiC5rQ==’
Decrypted message: b' Hello world! ‘
```