Program 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}")
```

Program 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
```

Program 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 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.")
```

Program 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 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
```

Program 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))
```

Program 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 Pwned" API (https://haveibeenpwned.com/API/v3) to check if a password has been leaked

```
import requests
import hashlib

# Read the file containing usernames and passwords
with open('password1.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
        password_hash = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()

        # Make a request to "Have I Been Pwned" API to check if the password has been leaked
        response = requests.get(f"https://api.pwnedpasswords.com/range/{password_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 password_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}.")
```

Program 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.")
```

Program 8: Python program for implementation symmetric encryption using Caesar cipher algorithm

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

    # traverse text
    for i in range(len(text)):
        char = text[i]

        # Encrypt uppercase characters
        if (char.isupper()):
            result += chr((ord(char) + s-65) % 26 + 65)

        # Encrypt lowercase characters
        else:
            result += chr((ord(char) + s - 97) % 26 + 97)

    return result

#check the above function
text = "ATTACKATONCE"
s = 7
print ("Text : " + text)
print ("Shift : " + str(s))
print ("Cipher: " + encrypt(text,s))
```

Program 9: Python program implementation for hacking Caesar cipher algorithm

```
# Get cipher text
print("Enter the Caesar Cipher text")
message = input("> ")

for shift in range(26):
    result = ''
    for char in message:
        if char.isalpha():
            ascii_offset = ord('A') if char.isupper() else ord('a')
            char = chr((ord(char) - ascii_offset - shift) % 26 + ascii_offset)
        result += char

    print(f"Key: {shift} | Decrypted message: {result}")
```

Program 10:

Python program to implement asymmetric encryption using RSA python library

```
from math import gcd

# defining a function to perform RSA approch
def RSA(p: int, q: int, message: int):
    # calculating n
    n = p * q

    # calculating totient, t
    t = (p - 1) * (q - 1)

    # selecting public key, e
    for i in range(2, t):
        if gcd(i, t) == 1:
            e = i
            break

    # selecting private key, d
    j = 0
    while True:
        if (j * e) % t == 1:
            d = j
            break
        j += 1

    # performing encryption
    ct = (message ** e) % n
    print(f"Encrypted message is {ct}")

    # performing decryption
    mes = (ct ** d) % n
    print(f"Decrypted message is {mes}")

# Testcase - 1
RSA(p=53, q=59, message=89)

# Testcase - 2
RSA(p=3, q=7, message=12)
```

Program 11: Python program for encoding and decoding using Base64

```
import base64
string1=input("Enter the text:")
byte1=string1.encode('ascii')
byte2=base64.b64encode(byte1)
print("Encoded string is:",byte2)
decoded_bytes=base64.b64decode(byte2)
decoded_string=decoded_bytes.decode('ascii')
print("Decoded string is:",decoded_string)
```

Program 12: Python program to implement symmetric encryption using python library

```
# Fernet module is imported from the
# cryptography package
from cryptography.fernet import Fernet

# key is generated
key = Fernet.generate_key()

# value of key is assigned to a variable
f = Fernet(key)

# the plaintext is converted to ciphertext
token = f.encrypt(b"welcome to cyber security lab")

# display the ciphertext
print(token)

# decrypting the ciphertext
d = f.decrypt(token)

# display the plaintext
print(d)
```