Lab Assignment: 01

Course title: Information Security

Name: Iqra Mubeen

Roll no. FA24_BSE_060

Semester: 04

Date: Feb. 28th 2026

Introduction:

What is Ceaser Cipher?

The **Caesar Cipher** is one of the earliest and simplest encryption techniques. It is a type of substitution cipher in which each letter in the original message (plaintext) is shifted by a fixed number of positions in the alphabet.

The method is named after **Julius Caesar**, who used it to protect his military communications.

Example:

- If the shift value is 3:

  A → D
  B → E
  C → F

- Original message:
  IQRA
- Encrypted message:
  LTUD

Each letter is shifted three positions forward in the alphabet.

What is Encryption and Decryption?

Encryption:

The process of converting readable text (plaintext) into unreadable text (ciphertext).

Decryption:

The process of converting ciphertext back into the original plaintext.

Purpose Of Assignment:

The purpose of this assignment is not only to write code but to understand fundamental concepts of information security and cryptography.

❑ To Understand Basic Encryption Concepts

This assignment helps students understand:

- What plaintext and ciphertext are
- How encryption and decryption work
- How simple cryptographic algorithms operate

❑ To Apply Programming Concepts

Students apply:

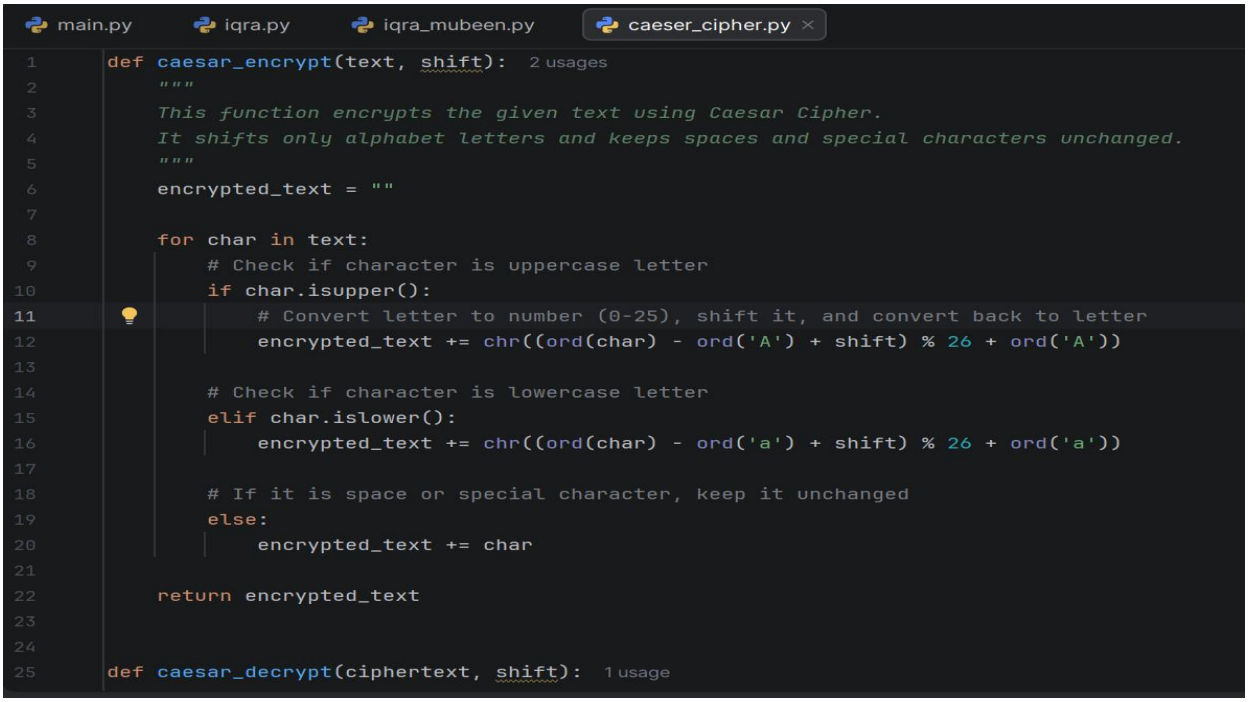
- Functions
- Loops
- Conditional statements (if-else)
- String manipulation
- ASCII conversion using `ord()` and `chr()`

❑ To Learn About Security Strength and Weakness

The Caesar Cipher is very simple and not secure for modern use.
By implementing it, students learn:

- Why simple encryption methods are weak
- How brute force attacks can break them
- Why modern cryptography is more complex

Python Code Screenshot:




```
def caesar_decrypt(ciphertext, shift):  1 usage
    """
    This function decrypts the encrypted text using Caesar Cipher.
    """
    return caesar_encrypt(ciphertext, -shift)


# Main Program
if __name__ == "__main__":
    print("=== Caesar Cipher Program ===")

    message = input("Enter your message: ")
    shift = int(input("Enter shift value: "))

    encrypted = caesar_encrypt(message, shift)
    print("Encrypted Message:", encrypted)

    decrypted = caesar_decrypt(encrypted, shift)
    print("Decrypted Message:", decrypted)
```

Output:

Line by Line Explanation of the Caesar Cipher Program

Program Title and Purpose:

❑ Encrypt Function

- **Line**: `def caesar_encrypt(text, shift):`
  Explanation: Defines a function named `caesar_encrypt` that takes two inputs: `text` (the message to encrypt) and `shift` (the number of positions each letter will move).
- **Line**: `encrypted_text = ""`
  Explanation: Initializes an empty string to store the final encrypted message.
- **Line**: `for char in text:`
  Explanation: Loops through every character in the input text to process it one by one.
- **Line**: `if char.isupper():`
  Explanation: Checks if the character is an uppercase letter (A-Z) because uppercase letters need to maintain their case.
- **Line**: `encrypted_text += chr((ord(char) - ord('A') + shift) % 26 + ord('A'))`
  Explanation:
  - `ord(char)` converts the letter to its ASCII number.
  - `- ord('A')` normalizes it to 0–25.
  - `+ shift` moves it forward by the shift value.
  - `% 26` ensures it wraps around after Z.
  - `chr(... + ord('A'))` converts the number back to a character.

    - The result is added to `encrypted_text`.
- **Line**: `elif char.islower():`
  Explanation: Checks if the character is a lowercase letter (a-z) to maintain lowercase.
- **Line**: `encrypted_text += chr((ord(char) - ord('a') + shift) % 26 + ord('a'))`
  Explanation: Same logic as uppercase but for lowercase letters.
- **Line**: `else:`
  Explanation: If the character is not a letter (space, number, punctuation), leave it unchanged.
- **Line**: `return encrypted_text`
  Explanation: Returns the final encrypted message after processing all characters.

❑ Decrypt Function

- **Line**: `def caesar_decrypt(ciphertext, shift):`
  Explanation: Defines a function to decrypt messages encrypted with the Caesar Cipher.
- **Line**: `return caesar_encrypt(ciphertext, -shift)`
  Explanation: Calls the `caesar_encrypt` function with a negative shift to reverse the encryption and recover the original message.

❑ Main Program Execution

- **Line**: `if __name__ == "__main__":`
  Explanation: Ensures the following code runs only when the file is executed directly, not when imported as a module.
- **Line**: `print("=== Caesar Cipher Program ===")`
  Explanation: Prints the program title in the console for clarity.
- **Line**: `message = input("Enter your message: ")`
  Explanation: Prompts the user to enter the message they want to encrypt.
- **Line**: `shift = int(input("Enter shift value: "))`
  Explanation: Prompts the user to enter the shift value (number of positions to shift letters).
- **Line**: `encrypted = caesar_encrypt(message, shift)`
  Explanation: Calls the encryption function to encrypt the user's message and stores the result in `encrypted`.

- **Line**: `print("Encrypted Message:", encrypted)`
  Explanation: Displays the encrypted message to the user.
- **Line**: `decrypted = caesar_decrypt(encrypted, shift)`
  Explanation: Calls the decryption function to convert the encrypted message back to the original text.
- **Line**: `print("Decrypted Message:", decrypted)`
  Explanation: Displays the decrypted (original) message to the user, confirming the program works correctly.

❑ Functions Used:

❑ **`def caesar_encrypt(text, shift):`**
- Function to encrypt the message.
- Inputs: `text` (message), `shift` (number of positions).
- Returns encrypted text.

❑ **`def caesar_decrypt(ciphertext, shift):`**
- Function to decrypt the message.
- Calls `caesar_encrypt` with `-shift` to reverse encryption.

❑ Loop Used

❑ **`for char in text:`**
- Loops through each character in the input text.
- Purpose: Check and process every character one by one.
- Ensures all letters are encrypted, while spaces and symbols are preserved.

❑ Conditions Used

❑ **`if char.isupper():`**

a. Checks if the character is an uppercase letter.

b. Only uppercase letters are shifted using ASCII conversion.

❑ **`elif char.islower():`**

c. Checks if the character is a lowercase letter.

d. Only lowercase letters are shifted using ASCII conversion.

❑ **`else:`**

a. Handles all other characters (spaces, numbers, punctuation).

b. Leaves them unchanged.

❑ Main Program Execution

- ❑ `if __name__ == "__main__":`
  - Ensures the code runs only when the file is executed directly.
- ❑ **User Input Lines:**
  - `message = input("Enter your message: ")` → Get message from user
  - `shift = int(input("Enter shift value: "))` → Get shift value
- ❑ **Function Calls:**
  - `encrypted = caesar_encrypt(message, shift)` → Encrypt message
  - `decrypted = caesar_decrypt(encrypted, shift)` → Decrypt message
- ❑ **Output Lines:**
  - `print("Encrypted Message:", encrypted)` → Show encrypted text
  - `print("Decrypted Message:", decrypted)` → Show decrypted (original) text.

Security Analysis:

❑ Strengths:

1. **Simple to implement**:
   a. Requires only basic programming knowledge.
   b. Good for educational purposes and understanding the concept of encryption and decryption.
2. **Maintains letter case and non-letter characters**:
   a. Uppercase and lowercase letters remain intact.
   b. Spaces, numbers, and special symbols are not altered, which helps maintain message readability.

❑ Weaknesses :

1. **Limited key space**:
   a. Only 25 possible shift values exist (1–25).
   b. This makes the cipher extremely vulnerable to **brute-force attacks**.
   c. An attacker can try all shifts in seconds to decrypt the message.
2. **Predictable pattern**:
   a. Each letter is shifted by the same fixed amount.

    b. Letter frequency is preserved. For example, ‘E’ remains the most common letter.
    c. This makes it vulnerable to **frequency analysis attacks**.
3. **Not suitable for modern security**:
    a. Cannot protect sensitive data in real-world applications.
    b. Easily breakable with minimal computational effort.

❑ Practical Implications:

- The Caesar Cipher is **mainly educational**:
    - Helps students understand the **basic concept of encryption, decryption, and modular arithmetic**.
- It demonstrates **why modern encryption methods** like AES, RSA, or DES are required for real-world data security.

Conclusion:

In this assignment, we successfully implemented the Caesar Cipher using Python to encrypt and decrypt messages. By creating separate functions for encryption and decryption, we applied core programming concepts such as **functions, loops, and conditional statements**. The program maintains uppercase and lowercase letters, preserves spaces and special characters, and handles all input efficiently, demonstrating careful string manipulation.

Through this exercise, we gained a clear understanding of the **basic principles of cryptography**, including how simple substitution ciphers work, their limitations, and the importance of modular arithmetic in encryption. While the Caesar Cipher is easy to implement and useful for educational purposes, it is **not secure for modern applications** due to its limited key space and vulnerability to brute-force and frequency analysis attacks.

Overall, this assignment highlights both the **practical implementation of encryption algorithms** and the **critical thinking required in information security**, preparing us for more advanced cryptographic techniques in the future.