COMSATS University Islamabad

Attock Campus

Department Of Computer Science

Course	***IS- LAB***
Instructor	***Ms. AMBREEN GUL***
Assignment No.	***01***

Student Details

Registration No.	***Name***
FA24-BSE-024	***ISMAIL AHMED***

1. Introduction :

The purpose of this assignment is to implement the Caesar Cipher encryption and decryption algorithm using Python. The Caesar Cipher is one of the oldest and simplest encryption techniques. It works by shifting letters in the alphabet by a fixed number of positions.

For example:

Plaintext:

HELLO

Shift: 3

Ciphertext:

KHOOR

2. Complete Python Code :

```
"""
Caesar Cipher Program
This program encrypts and decrypts text using the Caesar Cipher technique.
Only alphabetic characters are shifted.
Uppercase and lowercase letters are preserved.
"""
def caesar_encrypt(text, shift):
    encrypted_text = ""
    for char in text:
        if char.isupper():
            encrypted_text += chr((ord(char) - ord('A') + shift) % 26 + ord('A'))
        elif char.islower():
            encrypted_text += chr((ord(char) - ord('a') + shift) % 26 + ord('a'))
        else:
            encrypted_text += char
    return encrypted_text
def caesar_decrypt(ciphertext, shift):
    return caesar_encrypt(ciphertext, -shift)
if __name__ == "__main__":
    message = input("Enter your message: ")
    shift_value = int(input("Enter shift value: "))

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

    decrypted = caesar_decrypt(encrypted, shift_value)
```

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

3. Line-by-Line Explanation:

- **Program Description**

  """

  Caesar Cipher Program

  """

This is a multi-line comment (docstring) that explains the purpose of the program.

- **Function Definition: Encryption**

  def caesar_encrypt(text, shift):
  This defines a function named caesar_encrypt.

- text → the original message to encrypt
- shift → number of positions to shift letters

  encrypted_text = ""
  An empty string is created to store the encrypted result.

  for char in text:
  This loop goes through each character in the input text one by one.

- **Handling Uppercase Letters**

  if char.isupper():
  Checks if the character is an uppercase letter.

  encrypted_text += chr((ord(char) - ord('A') + shift) %

26 + ord('A'))

Step-by-step explanation:

1. ord(char) → Converts character to ASCII number
2. ord('A') → ASCII value of 'A' (65)
3. ord(char) - ord('A') → Normalizes position (0–25)
4. + shift → Shifts letter
5. % 26 → Wraps around alphabet
6. + ord('A') → Returns to ASCII uppercase range
7. chr() → Converts number back to character

- **Handling Lowercase Letters**

elif char.islower():
Checks if character is lowercase.

encrypted_text += chr((ord(char) - ord('a') + shift) % 26 + ord('a'))
Same logic as uppercase but uses ASCII of 'a' (97).

- **Handling Spaces & Special Characters**

else:
encrypted_text += char

If the character is:

- Space
- Number
- Symbol

It remains unchanged.
return encrypted_text

Returns the final encrypted message.

- **Decryption Function**

def caesar_decrypt(ciphertext, shift):
return caesar_encrypt(ciphertext, -shift)
Decryption works by shifting backward.
If encryption adds +3
Decryption subtracts −3
This avoids rewriting the entire logic.

- **Main Program Execution**

if __name__ == "__main__":
Ensures the program runs only when executed directly.

message = input("Enter your message: ")
shift_value = int(input("Enter shift value: "))
Takes input from user.

encrypted = caesar_encrypt(message, shift_value)
Calls encryption function.

decrypted = caesar_decrypt(encrypted, shift_value)
Calls decryption function.

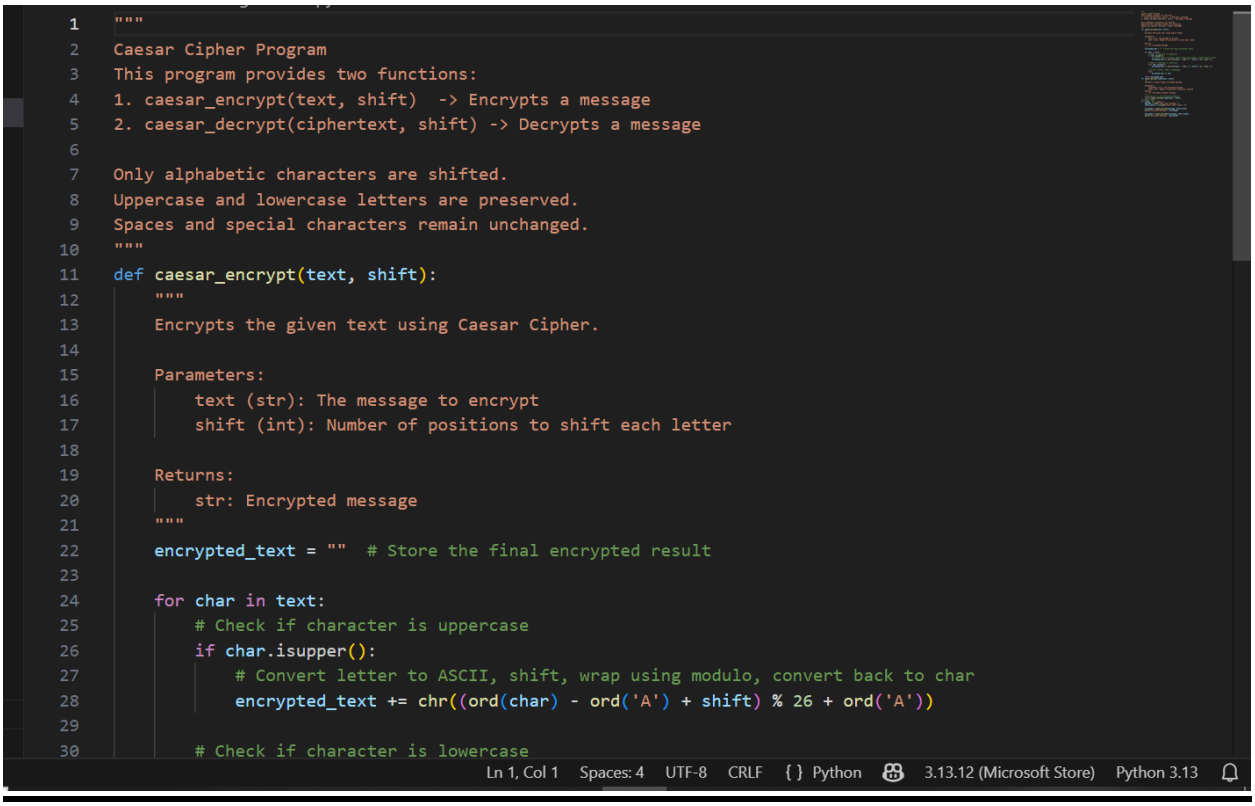
- **Sample Program Execution**

Example Input:
Enter your message: Hello World!
Enter shift value: 3

Output:
Encrypted Message: Khoor Zruog!
Decrypted Message: Hello World!

4. Screenshots Section:

CODE:



```
"""
Caesar Cipher Program
This program provides two functions:
1. caesar_encrypt(text, shift)  -> Encrypts a message
2. caesar_decrypt(ciphertext, shift) -> Decrypts a message

Only alphabetic characters are shifted.
Uppercase and lowercase letters are preserved.
Spaces and special characters remain unchanged.
"""
def caesar_encrypt(text, shift):
    """
    Encrypts the given text using Caesar Cipher.

    Parameters:
        text (str): The message to encrypt
        shift (int): Number of positions to shift each letter

    Returns:
        str: Encrypted message
    """
    encrypted_text = ""  # Store the final encrypted result

    for char in text:
        # Check if character is uppercase
        if char.isupper():
            # Convert letter to ASCII, shift, wrap using modulo, convert back to char
            encrypted_text += chr((ord(char) - ord('A') + shift) % 26 + ord('A'))

        # Check if character is lowercase
```

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.13.12 (Microsoft Store) Python 3.13

```python
        # Check if character is lowercase
        elif char.islower():
            encrypted_text += chr((ord(char) - ord('a') + shift) % 26 + ord('a'))

        # If not a letter, keep it unchanged
        else:
            encrypted_text += char

    return encrypted_text
def caesar_decrypt(ciphertext, shift):
    """
    Decrypts a Caesar Cipher encrypted message.

    Parameters:
        ciphertext (str): The encrypted message
        shift (int): Number of positions originally shifted
    Returns:
        str: Decrypted original message
    """
    # Decryption is just shifting backwards
    return caesar_encrypt(ciphertext, -shift)
# Example usage
if __name__ == "__main__":
    message = input("Enter your message: ")
    shift_value = int(input("Enter shift value: "))

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

```python
    shift_value = int(input("Enter shift value: "))

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

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

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.13.12 (Microsoft Store) Python 3.13

OUTPUT:

```
PS D:\IS-LAB> & "C:/Users/ISMAIL AHMED/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "d:/IS-LAB/LAB 1/Assignment 1.p
Enter shift value: 1
Decrypted Message: k
PS D:\IS-LAB> & "C:/Users/ISMAIL AHMED/AppData/Local/Microsoft/WindowsApps/python3.13.exe" "d:/IS-LAB/LAB 1/Assignment 1.p
y"
Enter your message: ISMAIL AHMED
Enter shift value: 4
Encrypted Message: MWQEMP ELQIH
Decrypted Message: ISMAIL AHMED
PS D:\IS-LAB>
```

5. Security Analysis

5.1 Key Space Limitation

The Caesar Cipher has only 25 possible shifts (1–25).
This means an attacker can try all possibilities easily.
This is called a **Brute Force Attack**.
Because the key space is very small, the cipher is weak.

5.2 Vulnerability to Frequency Analysis

In English language:

- E is the most common letter
- T, A, O appear frequently

An attacker can:

- Count letter frequency
- Compare with known language patterns
- Determine the shift value

This makes the cipher predictable.

5.3 No Protection Against Modern Attacks

The Caesar Cipher:

- Does not use complex keys
- Does not hide letter patterns
- Does not provide strong encryption

Modern encryption algorithms like:

- AES
- RSA

are mathematically complex and secure.

5.4 Conclusion of Security Analysis

The Caesar Cipher is:

✓ Good for educational purposes

✓ Useful for understanding basic encryption concepts

❌ Not secure for real-world applications

It can be broken easily using brute force or frequency analysis.

6. Overall Conclusion

This project demonstrates:

- Understanding of string manipulation
- Use of ASCII values
- Modular arithmetic
- Function implementation in Python
- Basic cryptographic concepts

Although the Caesar Cipher is simple and insecure, it provides a strong foundation for learning modern cryptography.

THE END