COMSATS UNIVERSITY ISLAMABAD

ATTOCK CAMPUS

Lab Assignment # 1

Course name: Information Security
Course Code: CSC232

Student Name: Samrina Manahil
Registration Number: FA24-BSE-052
Submitted to: Ms Ambreen Gul
Due Date: 28th Feb,2026

Task 1:

Implement the Caesar Cipher (10 Marks)
Write a Python program that encrypts and decrypts a message using the **Caesar Cipher**.

Requirements:

1. Write a function **`caesar_encrypt(text, shift)`** that takes a string and a shift value as input and returns the encrypted text.

2. Write a function **`caesar_decrypt(ciphertext, shift)`** that reverses the encryption and returns the original message.

3. The program **must consider spaces and special characters** only letters should be shifted.

4. The program should **maintain uppercase and lowercase letters** while encrypting and decrypting.

Python Implementation:

```python
def caesar_encrypt(text, shift):
    encrypted_text = ""

    for char in text:
        # Check if character is uppercase letter
        if char.isupper():
            encrypted_text += chr((ord(char) - 65 + shift) % 26 + 65)

        # Check if character is lowercase letter
        elif char.islower():
            encrypted_text += chr((ord(char) - 97 + shift) % 26 + 97)

        # Keep spaces and special characters unchanged
        else:
            encrypted_text += char

    return encrypted_text
```

```python
def caesar_decrypt(ciphertext, shift):
    decrypted_text = ""

    for char in ciphertext:
        if char.isupper():
            decrypted_text += chr((ord(char) - 65 - shift) % 26 + 65)

        elif char.islower():
            decrypted_text += chr((ord(char) - 97 - shift) % 26 + 97)

        else:
            decrypted_text += char

    return decrypted_text
```

```python
# Main Program
if __name__ == "__main__":
    message = input("Enter message: ")
    shift = int(input("Enter shift value: "))

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

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

Function: caesar_encrypt(text, shift)

Line 1: def caesar_encrypt(text, shift)
Defines a function named caesar_encrypt that takes two parameters:
text: The original message to encrypt (string)
shift: The number of positions to shift each letter (integer)

Line 2: encrypted_text = ""
Initializes an empty string variable to store the encrypted result

Line 3: for char in text
Starts a loop that iterates through each character in the input text

Line 4: # Check if character is uppercase letter
Comment indicating we're about to check for uppercase letters

Line 5: if char.isupper()
Uses the isupper() method to check if the current character is an uppercase letter
If True, executes the encryption for uppercase letters

Line 6: encrypted_text += chr((ord(char) - 65 + shift) % 26 + 65)
This complex line does the actual encryption:
ord(char): Converts the character to its ASCII value (e.g., 'A' = 65)
ord(char) - 65: Converts ASCII to 0-25 range (A=0, B=1, ..., Z=25)
+ shift: Adds the shift value to the position
% 26: Wraps around if result exceeds 25 (modulo operation)
+ 65: Converts back to ASCII range (65-90 for uppercase)
chr(): Converts ASCII number back to character
+=: Appends the encrypted character to the result string

Line 7: # Check if character is lowercase letter
Comment indicating we're about to check for lowercase letters

Line 8: elif char.islower()
Checks if the character is a lowercase letter
Uses islower() method

Line 9: encrypted_text += chr((ord(char) - 97 + shift) % 26 + 97)
Same logic as uppercase, but using 97 as base (ASCII for 'a')
Lowercase letters range from 97-122 in ASCII

Line 10: # Keep spaces and special characters unchanged
Comment explaining handling of non-alphabetic characters

Line 11: else
Catches all other characters (spaces, numbers, punctuation, etc.)

Line 12: encrypted_text += char
Appends the original character unchanged to the result

Line 13: return encrypted_text
Returns the complete encrypted string

Function: caesar_decrypt(ciphertext, shift)

Line 15: def caesar_decrypt(ciphertext, shift)
Defines decryption function with same parameters as encryption

Line 16: decrypted_text = ""
Initializes empty string for decrypted result

Line 17-24: Similar to encryption but with subtraction

Line 19: (ord(char) - 65 - shift) % 26 + 65 - Subtracts shift instead of adding

Line 21: (ord(char) - 97 - shift) % 26 + 97 - Subtracts shift for lowercase
This reverses the encryption process

Main Program Section

Line 27: if *__name__* == "*__main__*":
Checks if this script is being run directly (not imported as a module)
Common Python idiom to prevent code execution when importing

Line 28: message = input("Enter message: ")
Prompts user and stores input message as string

Line 29: shift = int(input("Enter shift value: "))
Prompts for shift value and converts it to integer

Line 30: encrypted = caesar_encrypt(message, shift)
Calls encryption function with user inputs
Stores result in encrypted variable

Line 31: print("Encrypted Text:", encrypted)
Displays the encrypted message

Line 32: decrypted = caesar_decrypt(encrypted, shift)
Calls decryption function with the encrypted text
Uses same shift value to decrypt

Line 33: print("Decrypted Text:", decrypted)
Displays the decrypted message to verify it matches original

Security Analysis:

The Caesar Cipher is a symmetric key substitution cipher. It has only 25 possible shift values, making it highly vulnerable to brute-force attacks. Frequency analysis can also easily break the cipher. Therefore, it is not suitable for secure modern communication but is useful for understanding basic cryptographic concepts.

Output:

Encryption:

- Each letter is converted to its alphabet position.
- The shift value is added.
- `% 26` ensures wrap-around within A–Z or a–z.
- The result is converted back to a letter.
- Non-letter characters remain unchanged.
- All processed characters form the encrypted text.

Decryption:

- Each encrypted letter is converted to its alphabet position.
- The shift value is subtracted.
- `% 26` ensures proper wrapping.
- The result is converted back to the original letter.
- The final output matches the original message.