Caesar Cipher Encryption & Decryption Program in Python

Pallabi Poria

04/10/2025

Aim:

Create a Python Program that can encrypt and decrypt text using the Caesar Cipher algorithm. Allow users to input a message and a shift value to perform encryption and decryption.

Objective:

The objective of this project is to implement a Python program that can securely encrypt and decrypt text messages using the Caesar Cipher Algorithm, while allowing user interaction via a menu-driven interface. The program also validates inputs to ensure correct execution.

Description:

The Caesar Cipher is a classical encryption technique where each letter in the plaintext is shifted by a fixed number if positions down or up the alphabet. Non-alphabetic characters (numbers, spaces, punctuation) remain unchanged.

The program provides a menu to either encrypt or decrypt a message. Users enter:

- The message to process.
- The shift value (number of positions to shift)

The program then performs the required operation and displays the output.

Features:

- Menu-driven interface: Users can choose between encryption and decryption.
- Input-validation: Ensures the message is not empty and shift value is numeric.
- Case-sensitive encryption: Preserves uppercase and lowercase letters.
- Non-alphabetic character handling: Punctuation, spaces, and numbers remain unchanged.
- Reversible encryption: Decryption restores the original message.

Algorithm / Logic:

Encryption:

1. Take input message from the user.
2. Take shift value from the user.
3. For each character in the message:
    - If alphabetic: shift it by the specified value.
    - If non-alphabetic: leave it unchanged.
4. Output the encrypted message.

Decryption:

Decryption is performed by shifting letters in the opposite direction using the same logic as encryption.

Python Code:

```
# Caesar Cipher Program

def encrypt(text, shift):
    if text is None:
        return ""
    result = ""
    for char in text:
        if char.isalpha():
            shift_base = 65 if char.isupper() else 97
            result += chr((ord(char) - shift_base + shift) % 26 + shift_base)
        else:
            result += char
    return result

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

def get_shift():
    while True:
        shift_input = input("Enter shift value (e.g. 3):  ")
        if shift_input.isdigit():
            return int(shift_input)
        else:
            print(" Shift must be a number. try again. ")

def get_message():
    while True:
        message = input("Enter your message: ")
        if message.strip() != " ":
            return message
        else:
            print(" Message cannot be empty. Try again. ")

# Main Program
print("=== Caesar Cipher Menu ===")
print("1. Encrypt")
print("2. Decrypt")
choice = input("Choose an option (1 or 2): ")
```

```
if choice not in ['1', '2']:
    print(" Invalid choice. Exiting. ")
    exit()

message = get_message()
shift = get_shift()

if choice =='1':
    encrypted_message = encrypt(message, shift)
    print(f"\n Encrypted message: {encrypted_message}")
elif choice == '2':
    decrypted_message = decrypt(message, shift)
    print(f"\n Decrypted message: {decrypted_message}")
```

Code explanation

```
def encrypt(text, shift):
```

- Defines a function encrypt that takes a string text and a numeric shift.
- Used to encrypt messages by shifting letters.

```
if text is None:
        return ""
```

- Checks *if text is None* (empty input)
- Returns an empty string to prevent errors when logging.

```
result = ""
```

- Initializes an empty string result to store the encrypted message.

```
for char in text:
```

- Loops through each character in the message.

```
if char.isalpha():
            shift_base = 65 if char.isupper() else 97
            result += chr((ord(char) - shift_base + shift) % 26 + shift_base)
```

- Checks if the character is a **letter.**
- Determines whether it's uppercase (65 for 'A') or lowercase (97 for 'a').
- Uses *ord()* and *chr()* to shift letters correctly and wraps around using % 26.

```
else:
            result += char
```

- Leaves non-alphabetic characters (space, punctuation, numbers) unchanged.

```
def decrypt(text, shift):
```

```
    return encrypt(text, -shift)
```

- Defines the *decrypt* function.
- Reverses the encryption by shifting letters **in the opposite direction**.

```
def get_shift():
    while True:
        shift_input = input("Enter shift value (e.g. 3):  ")
        if shift_input.isdigit():
            return int(shift_input)
        else:
            print(" Shift must be a number. try again. ")
```

- Function to safely get the shift value from the user.
- Loops until the user enters a valid numeric shift.

```
def get_message():
    while True:
        message = input("Enter your message: ")
        if message.strip() != " ":
            return message
        else:
            print(" Message cannot be empty. Try again. ")
```

- Function to safely get the message from the user.
- Ensures input is not empty or only spaces.

```
print("=== Caesar Cipher Menu ===")
print("1. Encrypt")
print("2. Decrypt")
choice = input("Choose an option (1 or 2): ")
```

- Displays a menu for the user to choose Encrypt or Decrypt.

```
if choice not in ['1', '2']:
    print(" Invalid choice. Exiting. ")
    exit()
```

- Validates the menu choice.
- Exits the program if the user enters an invalid option.

```
message = get_message()
shift = get_shift()
```

Calls the helper functions to get message and shift safely.

```
if choice =='1':
    encrypted_message = encrypt(message, shift)
    print(f"\n Encrypted message: {encrypted_message}")
elif choice == '2':
```

```
    decrypted_message = decrypt(message, shift)
    print(f"\n Decrypted message: {decrypted_message}")
```

- Performs encryption or decryption based on the user's choice.
- Displays the result clearly for encryption and decryption.

Sample output:

Encryption Example:

```
=== Caesar Cipher Menu ===
1. Encrypt
2. Decrypt
Choose an option (1 or 2): 1
Enter your message: My name
Enter shift value (e.g. 3):  2

 Encrypted message: Oa pcog
```

Decryption Example:

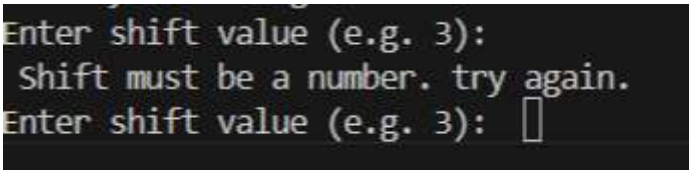
```
=== Caesar Cipher Menu ===
1. Encrypt
2. Decrypt
Choose an option (1 or 2): 2
Enter your message: Oa pcog
Enter shift value (e.g. 3):  2

 Decrypted message: My name
```

If you leave the option empty:

```
=== Caesar Cipher Menu ===
1. Encrypt
2. Decrypt
Choose an option (1 or 2):
 Invalid choice. Exiting.
```

If you leave the shift value empty:



Project Justification

- The developed Python program successfully fulfills the stated aim of creating a Caesar Cipher tool for encrypting and decrypting text.
- The program allows users to input any message and a numeric shift value, performing encryption and decryption accurately.
- Its menu-driven interface ensures that users can choose between encryption and decryption easily, while input validation prevents errors from empty messages or invalid shift values.
- Additionally, the program preserves the case of letters and leaves non-alphabetic characters unchanged, making it reliable and user friendly.
- Overall, the implementation demonstrates a correct and practical application of the Caesar Cipher algorithm as intended.

Conclusion and Scope

Conclusion:

- The menu-driven Caesar Cipher program effectively demonstrates the principles of classical encryption and decryption using Python.
- It allows users to input any message and a shift value, providing accurate encryption and decryption results while preserving the original formatting of letters and non-alphabetic characters.
- The program's user-friendly interface, along with input validation, ensures smooth execution and prevents common input errors.
- Overall, it achieves the intended aim by offering a functional and reliable tool for learning and practicing the Caesar Cipher algorithm.

Scope:

- This program can serve as a foundation for more advanced encryption projects. It can be extended to handle file encryption, larger datasets, or integrated into secure messaging systems.
- Additionally, the logic can be adapted to implement other classical ciphers, such as Vigenère or substitution ciphers, providing a practical learning experience in the field of cryptography and cybersecurity.
- The program is also suitable for educational purposes, helping beginners understand the concepts of encryption, decryption, and text manipulation using Python.