1. **Write a Python program to reverse the content of the string.**

Do not use built in

🡪def reverse\_string(s):

reversed\_str = ""

for char in s:

reversed\_str = char + reversed\_str

return reversed\_str

# Test the function

input\_string = "hello"

print("Reversed string:", reverse\_string(input\_string))

**2. Create a program that performs basic string compression using the counts of repeated characters. For example, the string “aabcccccaaa” would become “a2b1c5a3”.**

**🡪**def compress\_string(s):

compressed = ""

count = 1

for i in range(1, len(s)):

if s[i] == s[i-1]:

count += 1

else:

compressed += s[i-1] + str(count)

count = 1

compressed += s[-1] + str(count)

return compressed

# Test the function

input\_string = "aabcccccaaa"

print("Compressed string:", compress\_string(input\_string))

**3. Get the Caesar cipher from the user Decrypt the cipher**

**🡪**def decrypt\_caesar(cipher, shift):

decrypted = ""

for char in cipher:

if char.isalpha():

offset = 65 if char.isupper() else 97

decrypted += chr((ord(char) - offset - shift) % 26 + offset)

else:

decrypted += char

return decrypted

# Test the function

cipher\_text = "Uifsf jt b tfdsfu nfttbhf!"

shift = 1

print("Decrypted message:", decrypt\_caesar(cipher\_text, shift))

**4. Get the cipher encrypted using shift cipher. Identify the key used to encrypt using brute force**

**ie all the values in the key space**

**🡪**def brute\_force\_caesar(cipher):

for shift in range(1, 26):

print(f"Trying shift {shift}: {decrypt\_caesar(cipher, shift)}")

# Test the function

cipher\_text = "Uifsf jt b tfdsfu nfttbhf!"

brute\_force\_caesar(cipher\_text)

**5. Find the k value , Provided cipher text and plain text**

🡪def atbash\_cipher(s):

atbash\_encrypted = ""

for char in s:

if char.isalpha():

offset = 65 if char.isupper() else 97

atbash\_encrypted += chr(offset + (25 - (ord(char) - offset)))

else:

atbash\_encrypted += char

return atbash\_encrypted

# Test the function

input\_string = "hello"

encrypted\_string = atbash\_cipher(input\_string)

print("Atbash Encrypted:", encrypted\_string)

print("Atbash Decrypted:", atbash\_cipher(encrypted\_string))

**6. Encrypt and decrypt the string using Atbash cipher**

**🡪**def atbash\_cipher(s):

atbash\_encrypted = ""

for char in s:

if char.isalpha():

offset = 65 if char.isupper() else 97

atbash\_encrypted += chr(offset + (25 - (ord(char) - offset)))

else:

atbash\_encrypted += char

return atbash\_encrypted

# Test the function

input\_string = "hello"

encrypted\_string = atbash\_cipher(input\_string)

print("Atbash Encrypted:", encrypted\_string)

print("Atbash Decrypted:", atbash\_cipher(encrypted\_string))

**7. Encrypt and decrypt using Affine cipher**

**add validation**

🡪import math

def affine\_encrypt(text, a, b):

encrypted = ""

for char in text:

if char.isalpha():

offset = 65 if char.isupper() else 97

encrypted += chr(((a \* (ord(char) - offset) + b) % 26) + offset)

else:

encrypted += char

return encrypted

def affine\_decrypt(cipher, a, b):

decrypted = ""

a\_inv = pow(a, -1, 26) # modular inverse of a

for char in cipher:

if char.isalpha():

offset = 65 if char.isupper() else 97

decrypted += chr(((a\_inv \* ((ord(char) - offset) - b)) % 26) + offset)

else:

decrypted += char

return decrypted

# Test the function

a = 5 # 'a' and 26 must be coprime

b = 8

text = "AFFINECIPHER"

cipher\_text = affine\_encrypt(text, a, b)

print("Affine Encrypted:", cipher\_text)

print("Affine Decrypted:", affine\_decrypt(cipher\_text, a, b))