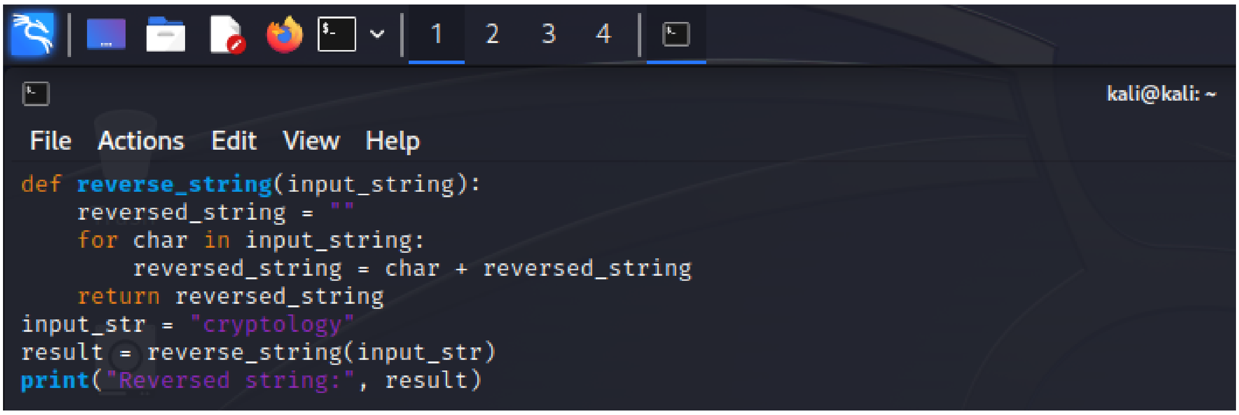
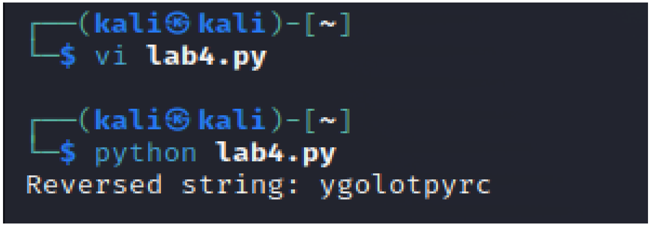
1. Write a Python program to reverse the content of the string. Do not use built in

Sol:



```
def reverse_string(input_string):
    reversed_string = ""
    for char in input_string:
        reversed_string = char + reversed_string
    return reversed_string
input_str = "cryptology"
result = reverse_string(input_str)
print("Reversed string:", result)
```

Output:

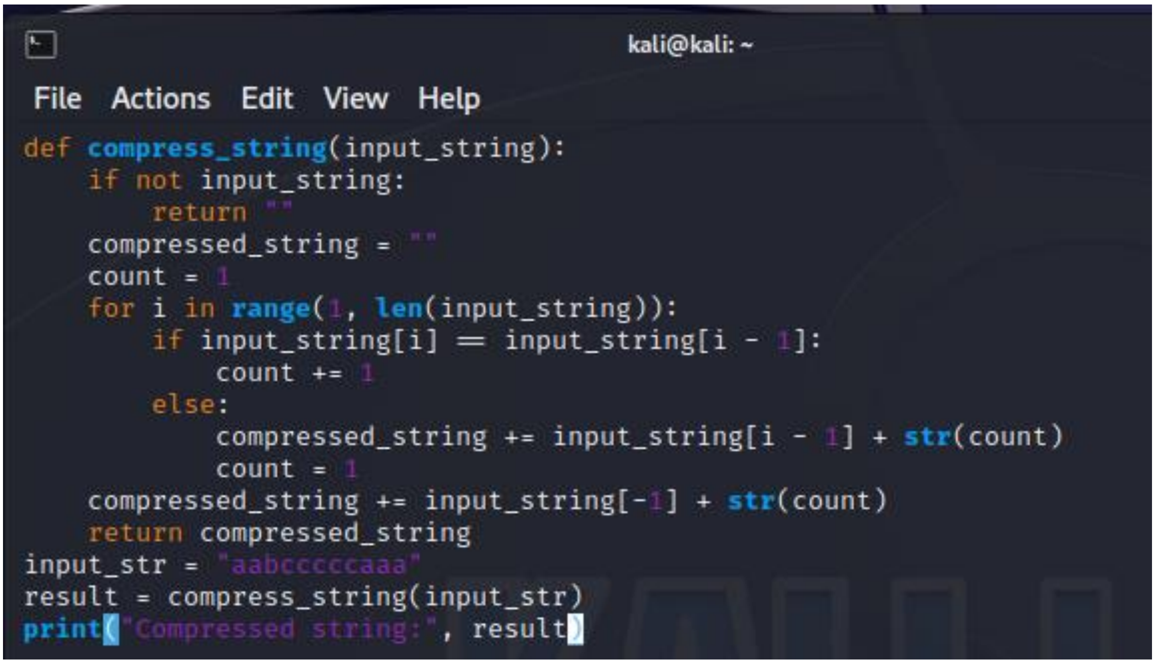


```
┌──(kali㉿kali)-[~]
└─$ vi lab4.py

┌──(kali㉿kali)-[~]
└─$ python lab4.py
Reversed string: ygolotpyrc
```

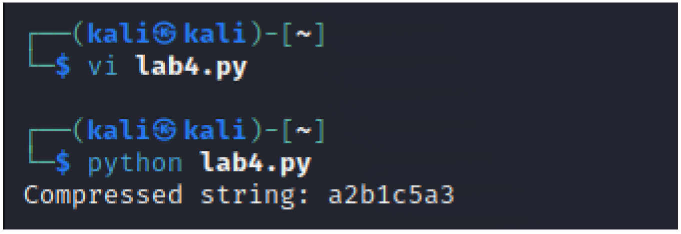
2. Create a program that performs basic string compression using the counts of repeated characters. For example, the string "aabcccccaaa" would become "a2b1c5a3".

Sol:



```
def compress_string(input_string):
    if not input_string:
        return ""
    compressed_string = ""
    count = 1
    for i in range(1, len(input_string)):
        if input_string[i] == input_string[i - 1]:
            count += 1
        else:
            compressed_string += input_string[i - 1] + str(count)
            count = 1
    compressed_string += input_string[-1] + str(count)
    return compressed_string
input_str = "aabcccccaaa"
result = compress_string(input_str)
print("Compressed string:", result)
```

Output:

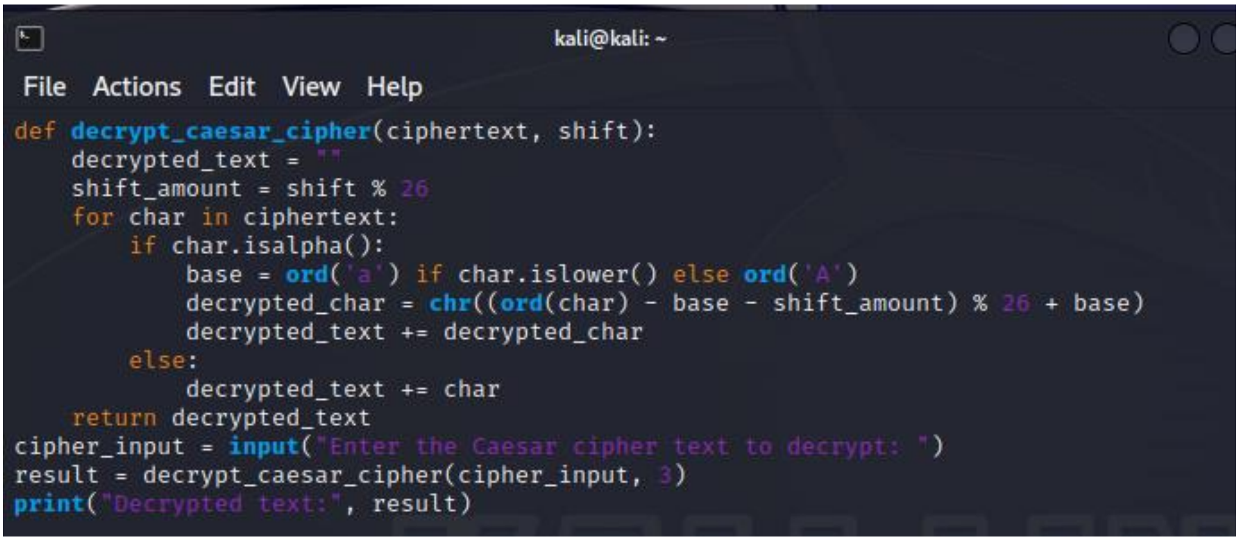


```
┌──(kali㉿kali)-[~]
└─$ vi lab4.py

┌──(kali㉿kali)-[~]
└─$ python lab4.py
Compressed string: a2b1c5a3
```

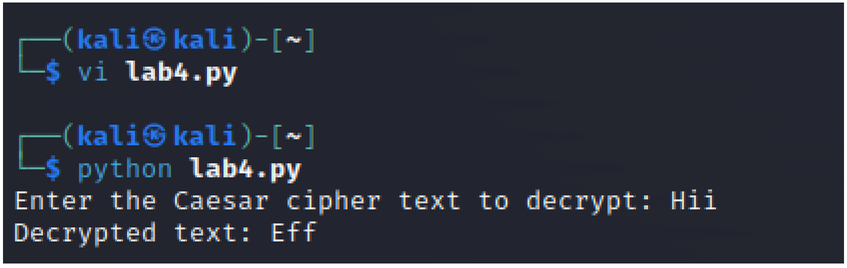
3. Get the Caesar cipher from the user Decrypt the cipher

Sol:



```
def decrypt_caesar_cipher(ciphertext, shift):
    decrypted_text = ""
    shift_amount = shift % 26
    for char in ciphertext:
        if char.isalpha():
            base = ord('a') if char.islower() else ord('A')
            decrypted_char = chr((ord(char) - base - shift_amount) % 26 + base)
            decrypted_text += decrypted_char
        else:
            decrypted_text += char
    return decrypted_text
cipher_input = input("Enter the Caesar cipher text to decrypt: ")
result = decrypt_caesar_cipher(cipher_input, 3)
print("Decrypted text:", result)
```

Output:

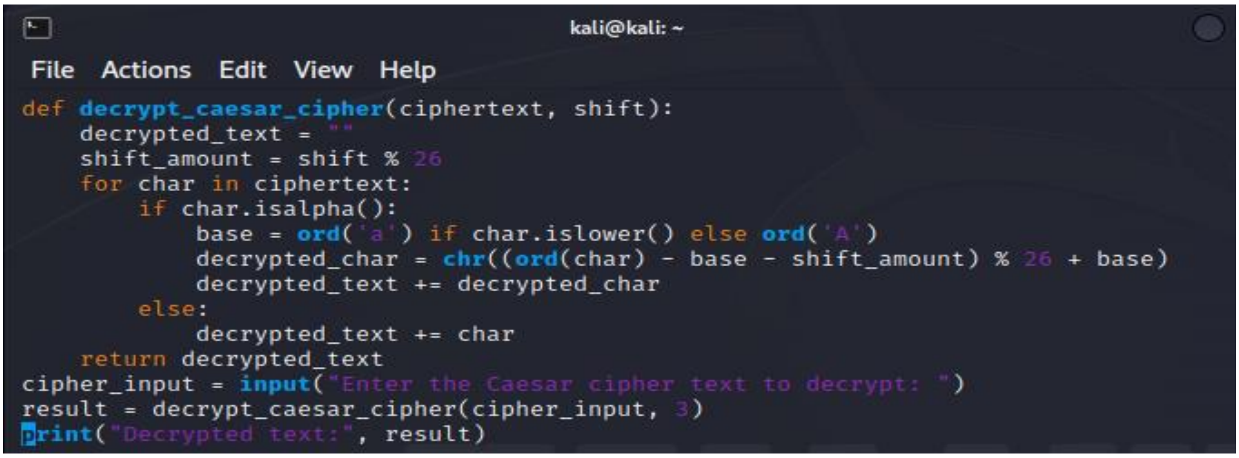


```
┌──(kali㉿kali)-[~]
└─$ vi lab4.py

┌──(kali㉿kali)-[~]
└─$ python lab4.py
Enter the Caesar cipher text to decrypt: Hii
Decrypted text: Eff
```

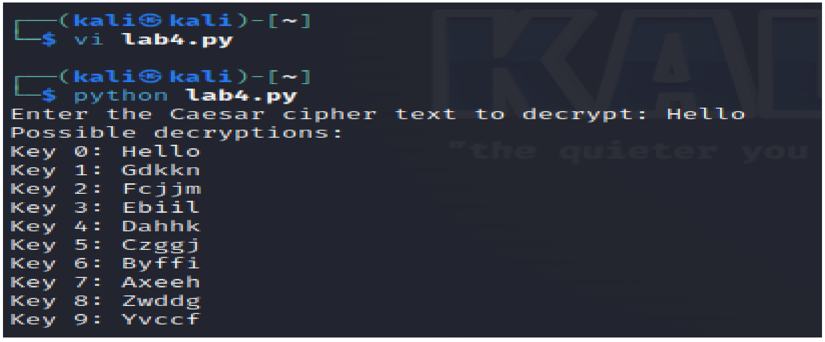
4. Get the cipher encrypted using shift cipher. Identify the key used to encrypt using brute force i.e all the values in the key space

Sol:



```
def decrypt_caesar_cipher(ciphertext, shift):
    decrypted_text = ""
    shift_amount = shift % 26
    for char in ciphertext:
        if char.isalpha():
            base = ord('a') if char.islower() else ord('A')
            decrypted_char = chr((ord(char) - base - shift_amount) % 26 + base)
            decrypted_text += decrypted_char
        else:
            decrypted_text += char
    return decrypted_text
cipher_input = input("Enter the Caesar cipher text to decrypt: ")
result = decrypt_caesar_cipher(cipher_input, 3)
print("Decrypted text:", result)
```

Output:

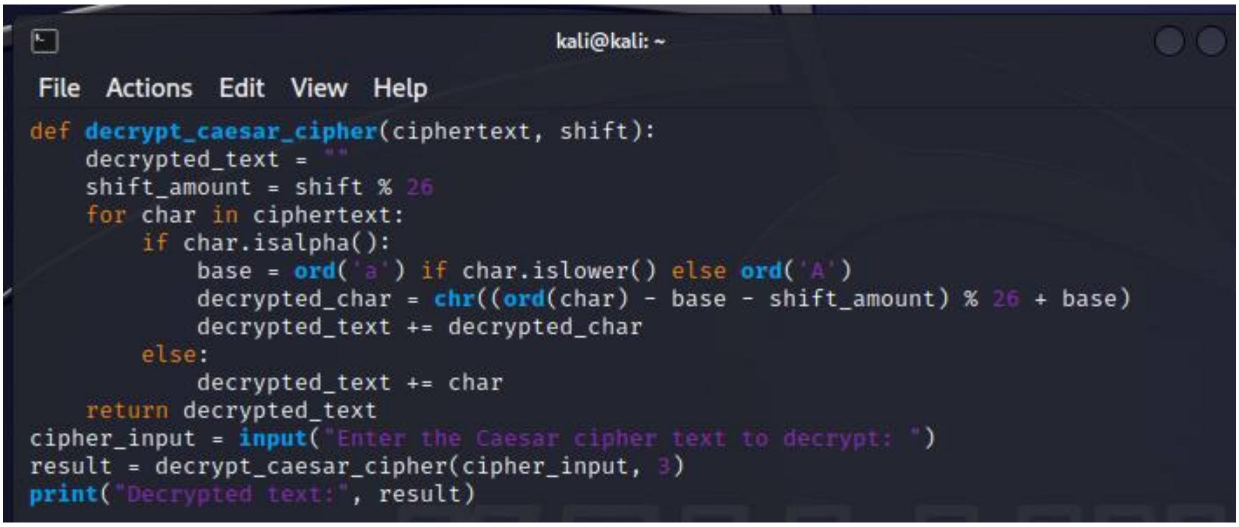


```
┌──(kali㉿kali)-[~]
└─$ vi lab4.py

┌──(kali㉿kali)-[~]
└─$ python lab4.py
Enter the Caesar cipher text to decrypt: Hello
Possible decryptions:
Key 0: Hello
Key 1: Gdkkn
Key 2: Fcjjm
Key 3: Ebiil
Key 4: Dahhk
Key 5: Czggj
Key 6: Byffi
Key 7: Axeeh
Key 8: Zwddg
Key 9: Yvccf
```

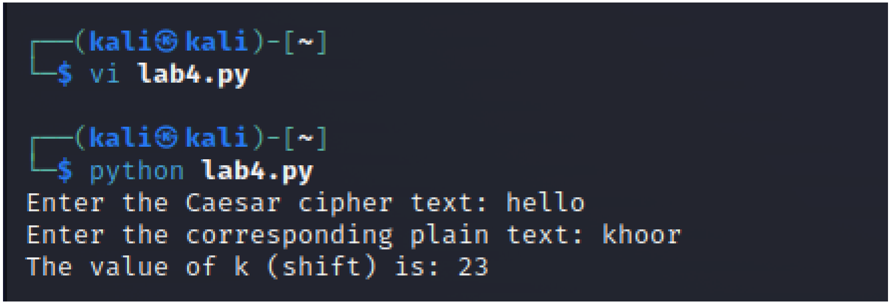
5. Find the k value, Provided cipher text and plain text

Sol:



```
def decrypt_caesar_cipher(ciphertext, shift):
    decrypted_text = ""
    shift_amount = shift % 26
    for char in ciphertext:
        if char.isalpha():
            base = ord('a') if char.islower() else ord('A')
            decrypted_char = chr((ord(char) - base - shift_amount) % 26 + base)
            decrypted_text += decrypted_char
        else:
            decrypted_text += char
    return decrypted_text
cipher_input = input("Enter the Caesar cipher text to decrypt: ")
result = decrypt_caesar_cipher(cipher_input, 3)
print("Decrypted text:", result)
```

Output:

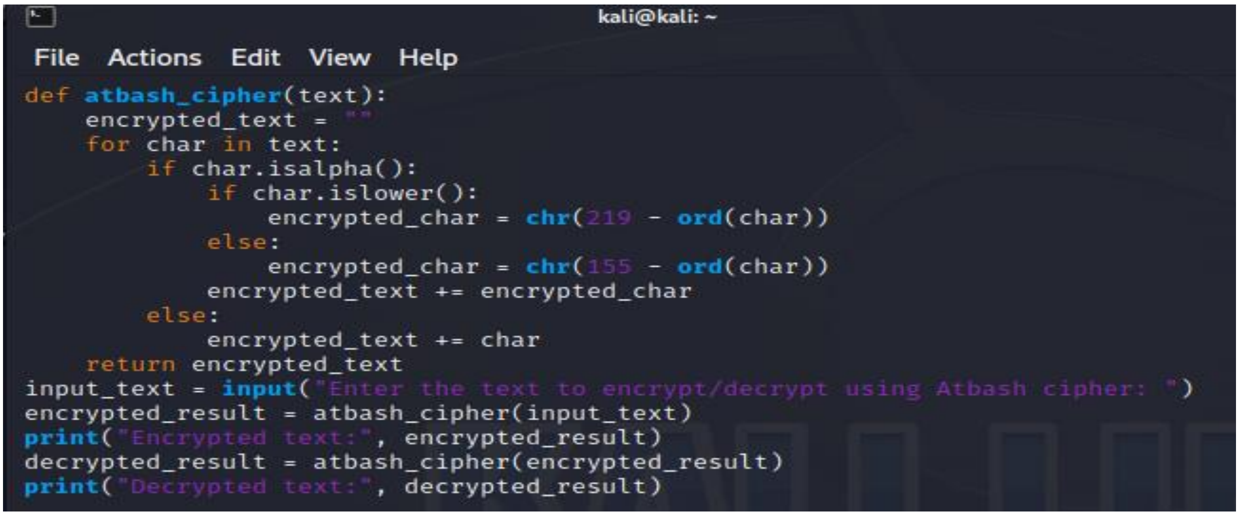


```
┌──(kali㉿kali)-[~]
└─$ vi lab4.py

┌──(kali㉿kali)-[~]
└─$ python lab4.py
Enter the Caesar cipher text: hello
Enter the corresponding plain text: khoor
The value of k (shift) is: 23
```

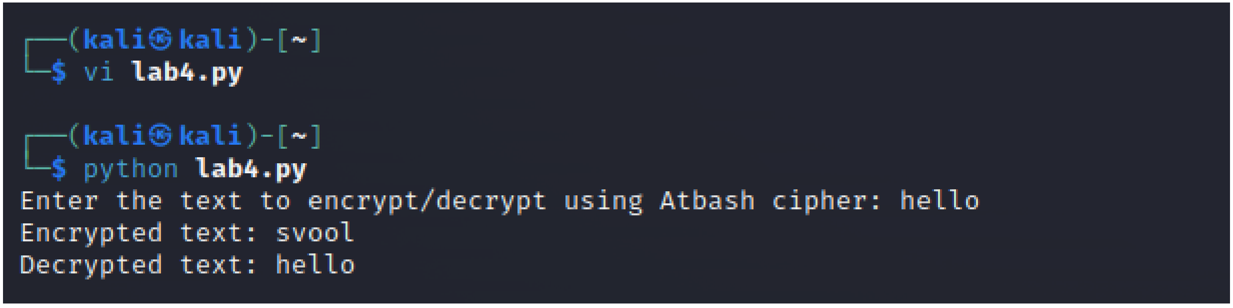
6. Encrypt and decrypt the string using Atbash cipher

Sol:



```
def atbash_cipher(text):
    encrypted_text = ""
    for char in text:
        if char.isalpha():
            if char.islower():
                encrypted_char = chr(219 - ord(char))
            else:
                encrypted_char = chr(155 - ord(char))
            encrypted_text += encrypted_char
        else:
            encrypted_text += char
    return encrypted_text
input_text = input("Enter the text to encrypt/decrypt using Atbash cipher: ")
encrypted_result = atbash_cipher(input_text)
print("Encrypted text:", encrypted_result)
decrypted_result = atbash_cipher(encrypted_result)
print("Decrypted text:", decrypted_result)
```

Output:

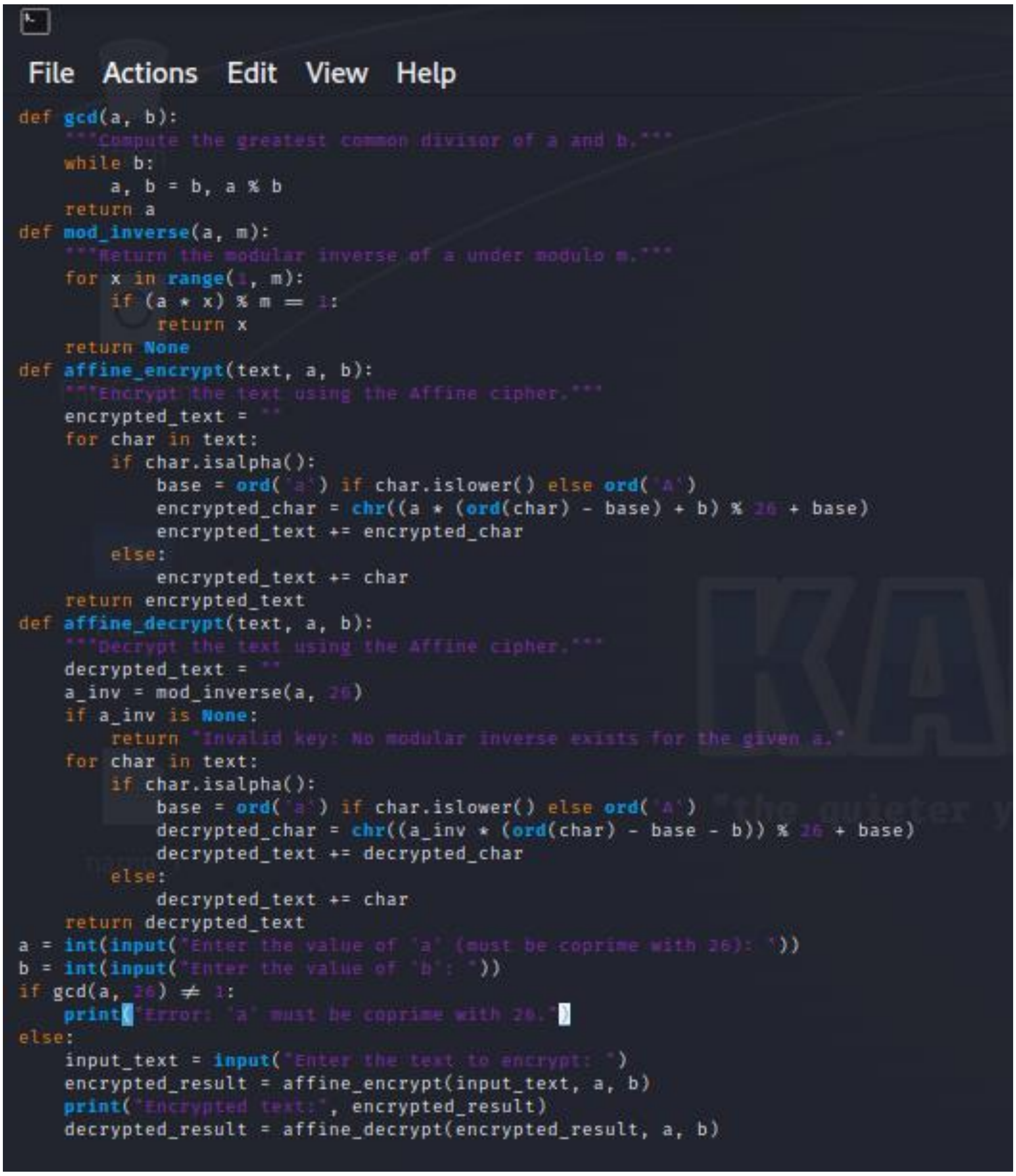


```
┌──(kali㉿kali)-[~]
└─$ vi lab4.py

┌──(kali㉿kali)-[~]
└─$ python lab4.py
Enter the text to encrypt/decrypt using Atbash cipher: hello
Encrypted text: svool
Decrypted text: hello
```

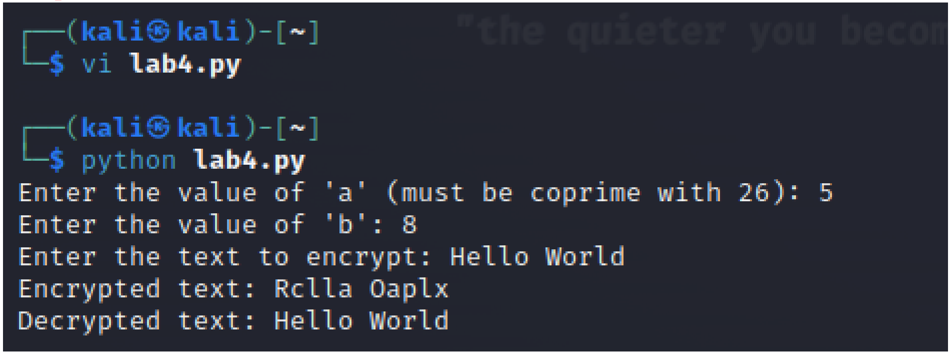
7. Encrypt and decrypt using Affine cipher add validation

Sol:



```
def gcd(a, b):
    """Compute the greatest common divisor of a and b."""
    while b:
        a, b = b, a % b
    return a
def mod_inverse(a, m):
    """Return the modular inverse of a under modulo m."""
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None
def affine_encrypt(text, a, b):
    """Encrypt the text using the Affine cipher."""
    encrypted_text = ""
    for char in text:
        if char.isalpha():
            base = ord('a') if char.islower() else ord('A')
            encrypted_char = chr((a * (ord(char) - base) + b) % 26 + base)
            encrypted_text += encrypted_char
        else:
            encrypted_text += char
    return encrypted_text
def affine_decrypt(text, a, b):
    """Decrypt the text using the Affine cipher."""
    decrypted_text = ""
    a_inv = mod_inverse(a, 26)
    if a_inv is None:
        return "Invalid key: No modular inverse exists for the given a."
    for char in text:
        if char.isalpha():
            base = ord('a') if char.islower() else ord('A')
            decrypted_char = chr((a_inv * (ord(char) - base - b)) % 26 + base)
            decrypted_text += decrypted_char
        else:
            decrypted_text += char
    return decrypted_text
a = int(input("Enter the value of 'a' (must be coprime with 26): "))
b = int(input("Enter the value of 'b': "))
if gcd(a, 26) != 1:
    print("Error: 'a' must be coprime with 26.")
else:
    input_text = input("Enter the text to encrypt: ")
    encrypted_result = affine_encrypt(input_text, a, b)
    print("Encrypted text:", encrypted_result)
    decrypted_result = affine_decrypt(encrypted_result, a, b)
```

Output:



```
┌──(kali㉿kali)-[~]
└─$ vi lab4.py

┌──(kali㉿kali)-[~]
└─$ python lab4.py
Enter the value of 'a' (must be coprime with 26): 5
Enter the value of 'b': 8
Enter the text to encrypt: Hello World
Encrypted text: Rclla Oaplx
Decrypted text: Hello World
```