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

Do not use built in

def reverse\_string(input\_string):

# Initialize an empty string to store the reversed content

reversed\_string = ''

# Loop through the input string from the end to the start

for i in range(len(input\_string) - 1, -1, -1):

reversed\_string += input\_string[i]

return reversed\_string

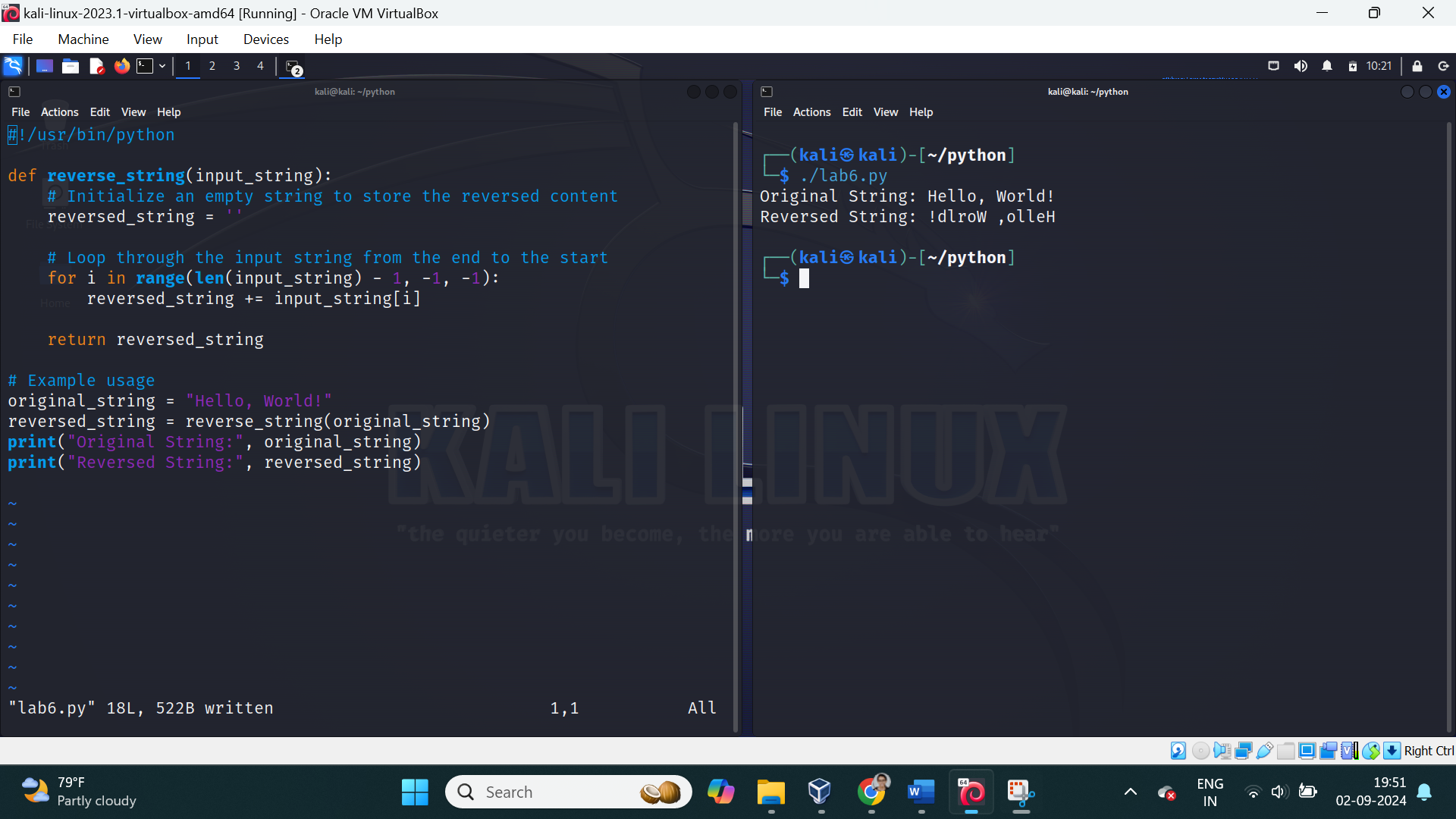
# Example usage

original\_string = "Hello, World!"

reversed\_string = reverse\_string(original\_string)

print("Original String:", original\_string)

print("Reversed String:", reversed\_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(input\_string):

# Initialize an empty string for the compressed result

compressed = ""

# Initialize a counter for consecutive characters

count = 1

# Loop through the string

for i in range(1, len(input\_string)):

# If the current character is the same as the previous one

if input\_string[i] == input\_string[i - 1]:

count += 1

else:

# Append the previous character and its count to the result

compressed += input\_string[i - 1] + str(count)

count = 1 # Reset the count

# Don't forget to add the last set of characters

compressed += input\_string[-1] + str(count)

return compressed

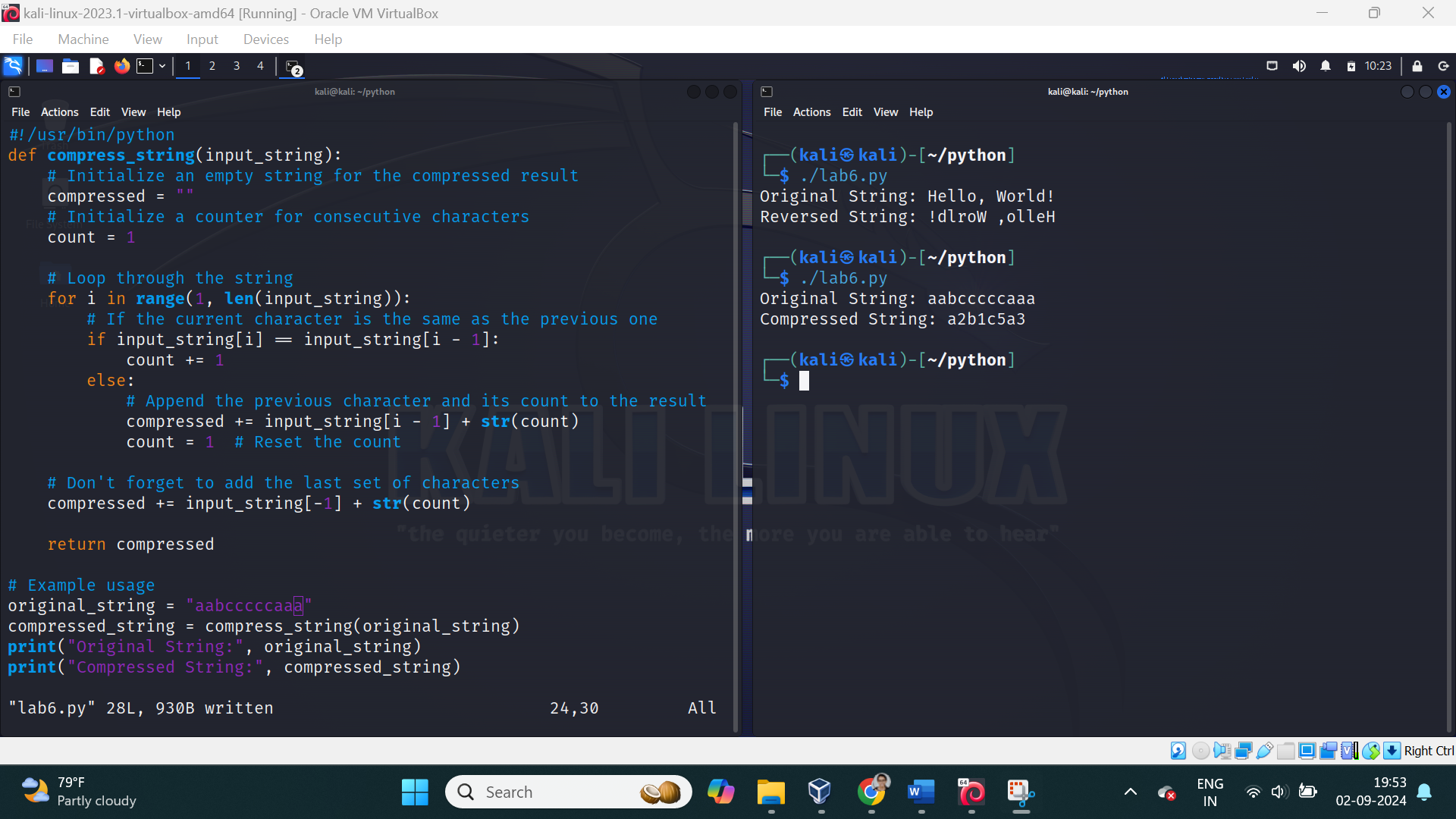
# Example usage

original\_string = "aabcccccaaa"

compressed\_string = compress\_string(original\_string)

print("Original String:", original\_string)

print("Compressed String:", compressed\_string)



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

def decrypt\_caesar\_cipher(cipher\_text, shift):

decrypted\_text = ""

for char in cipher\_text:

# Check if the character is an uppercase letter

if 'A' <= char <= 'Z':

decrypted\_char = chr((ord(char) - shift - 65) % 26 + 65)

# Check if the character is a lowercase letter

elif 'a' <= char <= 'z':

decrypted\_char = chr((ord(char) - shift - 97) % 26 + 97)

else:

# If it's not a letter, don't change it

decrypted\_char = char

decrypted\_text += decrypted\_char

return decrypted\_text

# Get the cipher text and shift from the user

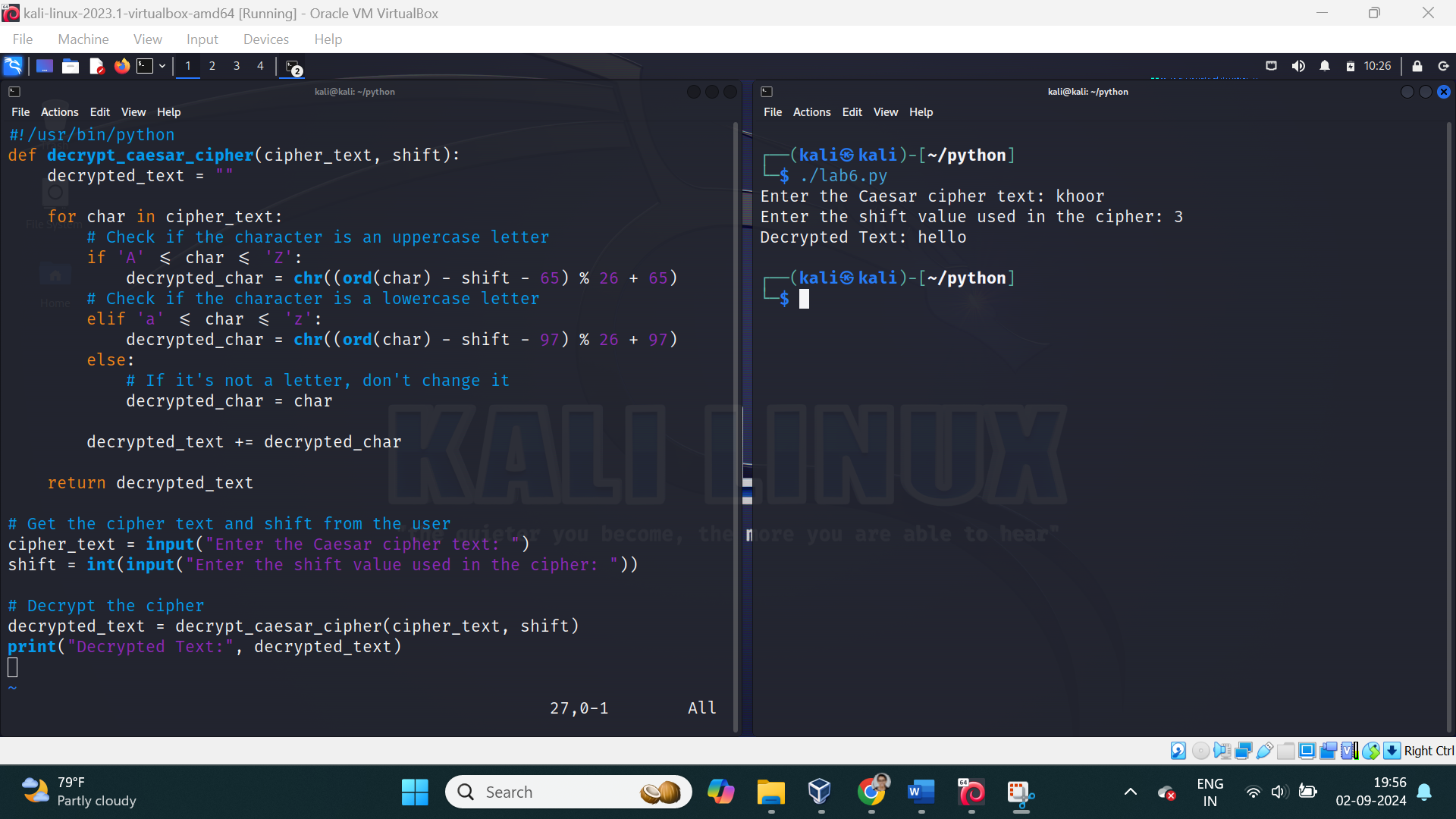
cipher\_text = input("Enter the Caesar cipher text: ")

shift = int(input("Enter the shift value used in the cipher: "))

# Decrypt the cipher

decrypted\_text = decrypt\_caesar\_cipher(cipher\_text, shift)

print("Decrypted Text:", decrypted\_text)



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 decrypt\_caesar\_cipher(cipher\_text, shift):

decrypted\_text = ""

for char in cipher\_text:

# Check if the character is an uppercase letter

if 'A' <= char <= 'Z':

decrypted\_char = chr((ord(char) - shift - 65) % 26 + 65)

# Check if the character is a lowercase letter

elif 'a' <= char <= 'z':

decrypted\_char = chr((ord(char) - shift - 97) % 26 + 97)

else:

# If it's not a letter, don't change it

decrypted\_char = char

decrypted\_text += decrypted\_char

return decrypted\_text

def brute\_force\_caesar(cipher\_text):

for shift in range(1, 26):

decrypted\_text = decrypt\_caesar\_cipher(cipher\_text, shift)

print(f"Key {shift}: {decrypted\_text}")

# Get the cipher text from the user

cipher\_text = input("Enter the Caesar cipher text: ")

# Perform brute force attack to identify the key

brute\_force\_caesar(cipher\_text)



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

def find\_shift\_value(plain\_text, cipher\_text):

# Assuming plain\_text and cipher\_text are of the same length

shift\_values = []

for pt\_char, ct\_char in zip(plain\_text, cipher\_text):

if pt\_char.isalpha(): # Only consider alphabetic characters

# Calculate the shift value for uppercase letters

if pt\_char.isupper():

shift = (ord(ct\_char) - ord(pt\_char)) % 26

# Calculate the shift value for lowercase letters

elif pt\_char.islower():

shift = (ord(ct\_char) - ord(pt\_char)) % 26

shift\_values.append(shift)

# If all shifts are the same, return that shift value

if all(s == shift\_values[0] for s in shift\_values):

return shift\_values[0]

else:

return "Inconsistent shifts found!"

# Example usage

plain\_text = input("Enter the plain text: ")

cipher\_text = input("Enter the cipher text: ")

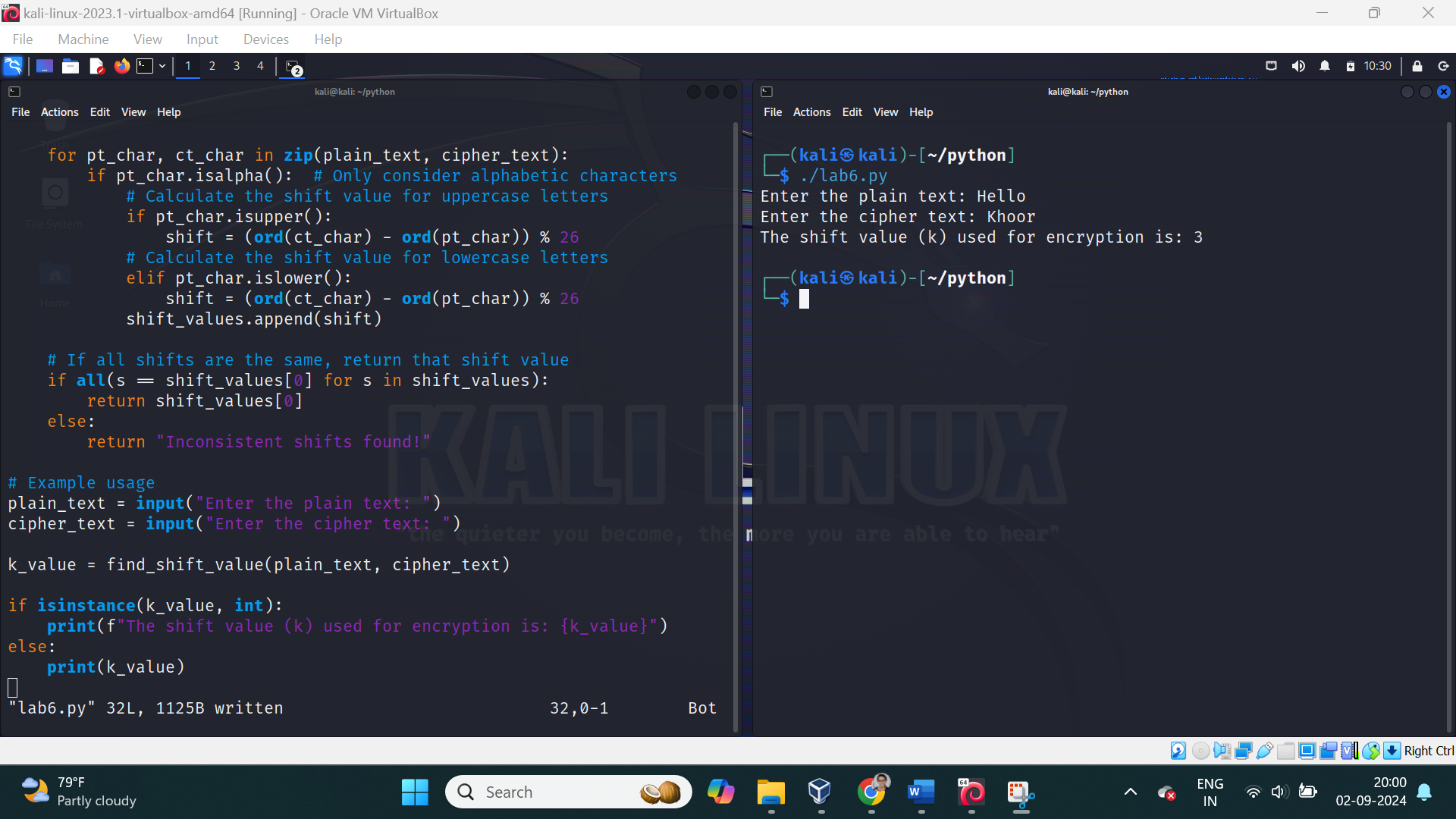
k\_value = find\_shift\_value(plain\_text, cipher\_text)

if isinstance(k\_value, int):

print(f"The shift value (k) used for encryption is: {k\_value}")

else:

print(k\_value)



6. Encrypt and decrypt the string using Atbash cipher

def atbash\_cipher(text):

encrypted\_decrypted\_text = ""

for char in text:

if char.isalpha():

# For uppercase letters

if char.isupper():

encrypted\_decrypted\_text += chr(65 + (25 - (ord(char) - 65)))

# For lowercase letters

else:

encrypted\_decrypted\_text += chr(97 + (25 - (ord(char) - 97)))

else:

# Non-alphabetic characters remain the same

encrypted\_decrypted\_text += char

return encrypted\_decrypted\_text

# Example usage

text = input("Enter the text to encrypt/decrypt using Atbash cipher: ")

result = atbash\_cipher(text)

print("Result:", result)



7. Encrypt and decrypt using Affine cipher

add validation

def mod\_inverse(a, m):

for i in range(1, m):

if (a \* i) % m == 1:

return i

return None

def affine\_encrypt(plain\_text, a, b):

if mod\_inverse(a, 26) is None:

raise ValueError("The key 'a' must be coprime with 26 for the Affine cipher to work.")

encrypted\_text = ""

for char in plain\_text:

if char.isalpha():

if char.isupper():

x = ord(char) - 65

encrypted\_char = chr(((a \* x + b) % 26) + 65)

else:

x = ord(char) - 97

encrypted\_char = chr(((a \* x + b) % 26) + 97)

encrypted\_text += encrypted\_char

else:

encrypted\_text += char

return encrypted\_text

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

a\_inv = mod\_inverse(a, 26)

if a\_inv is None:

raise ValueError("The key 'a' must be coprime with 26 for the Affine cipher to work.")

decrypted\_text = ""

for char in cipher\_text:

if char.isalpha():

if char.isupper():

y = ord(char) - 65

decrypted\_char = chr(((a\_inv \* (y - b)) % 26) + 65)

else:

y = ord(char) - 97

decrypted\_char = chr(((a\_inv \* (y - b)) % 26) + 97)

decrypted\_text += decrypted\_char

else:

decrypted\_text += char

return decrypted\_text

# Validate and get inputs

while True:

try:

a = int(input("Enter the key 'a' (must be coprime with 26): "))

b = int(input("Enter the key 'b': "))

if mod\_inverse(a, 26) is None:

raise ValueError("Invalid 'a' value. It must be coprime with 26.")

break

except ValueError as ve:

print(ve)

# Get the text to encrypt/decrypt

plain\_text = input("Enter the text to encrypt: ")

# Encryption

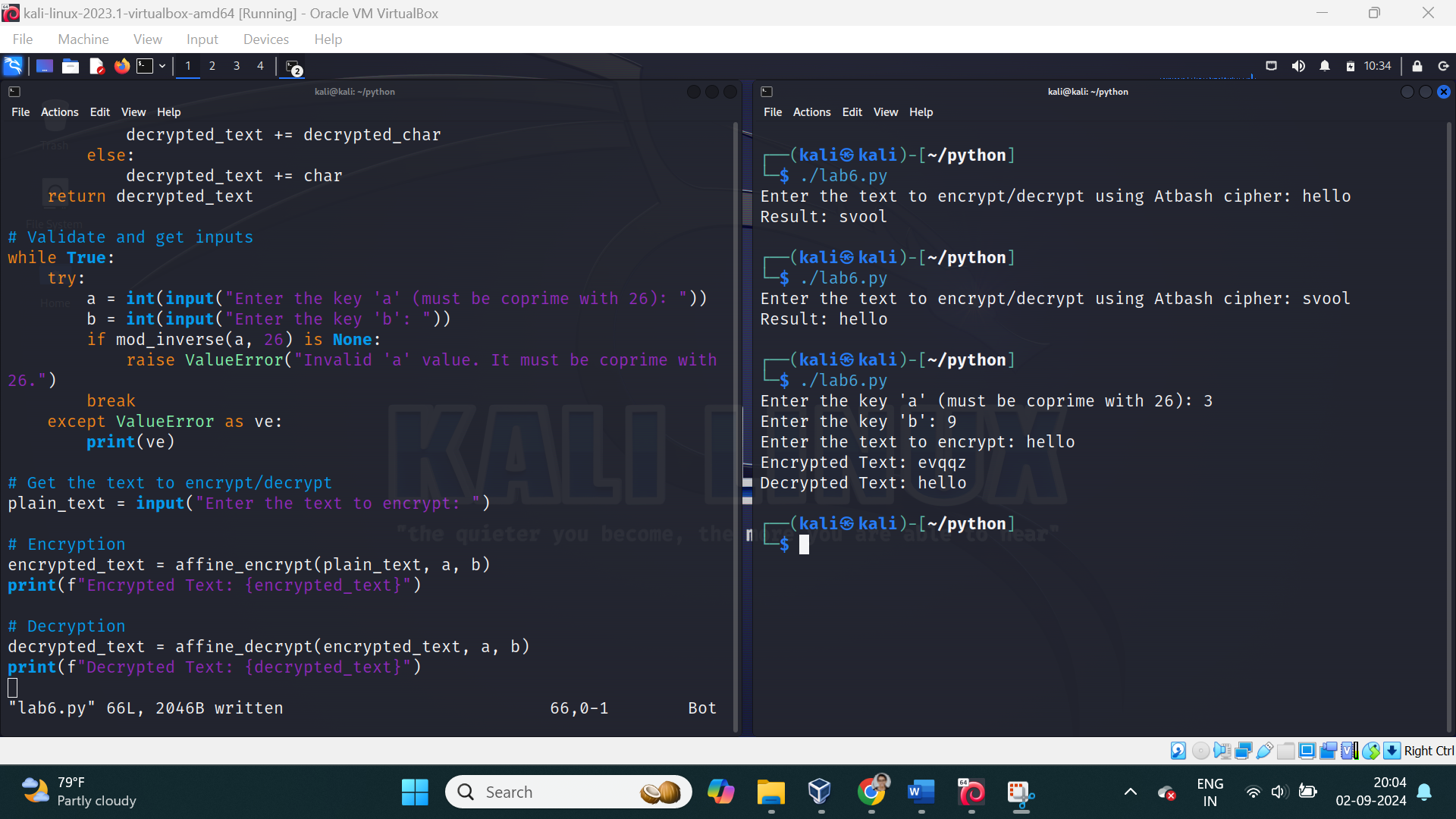
encrypted\_text = affine\_encrypt(plain\_text, a, b)

print(f"Encrypted Text: {encrypted\_text}")

# Decryption

decrypted\_text = affine\_decrypt(encrypted\_text, a, b)

print(f"Decrypted Text: {decrypted\_text}")



The harder you work for something, the greater you will feel when you achieve it.

Do not limit your challenges challenge your limit