* **Substitution Ciphers**

**Additive Cipher**

**#encryption and decryption**

text=input("Enter your text:")

s=int(input("Enter your Key:"))

def encrypt(text,s): **#function for Encryption**

result = ""

for i in range(len(text)):

char = text[i]

if (char.isupper()):

result += chr((ord(char) + s-65) % 26 + 65)

else:

result += chr((ord(char) + s-97) % 26 + 97)

return result

def decrypt(text,s): **#function for Decryption**

result = ""

for i in range(len(text)):

char = text[i]

if (char.isupper()):

result += chr((ord(char) - s-65) % 26 + 65)

else:

result += chr((ord(char) - s-97) % 26 + 97)

return result

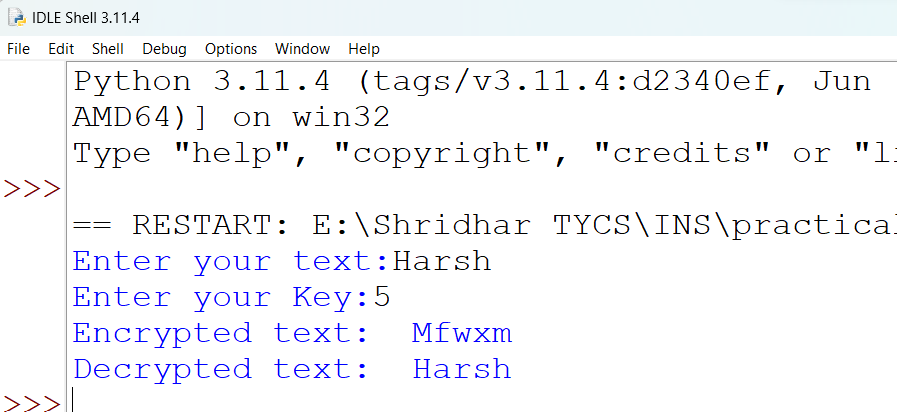
Etext=encrypt(text,s)

Dtext=decrypt(Etext,s)

print('Encrypted text: ',Etext)

print('Decrypted text: ',Dtext)

**Output**

****

**Multiplicative Cipher**

**Code**

def multiplicative\_cipher(text, key):

return "".join(chr((ord(c) - 65) \* key % 26 + 65) if c.isalpha() else c for c in text.upper())

def mod\_inverse(a, m):

for x in range(1, m):

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

return x

return None

def multiplicative\_decipher(text, key):

inverse\_key = mod\_inverse(key, 26)

if inverse\_key is None:

return "Error: Invalid key. The modular inverse does not exist."

return "".join(chr((ord(c) - 65) \* inverse\_key % 26 + 65) if c.isalpha() else c for c in text)

# Example usage

plaintext = input("enter your plain text")

key = int(input("enter your key value "))

encrypted\_text = multiplicative\_cipher(plaintext, key)

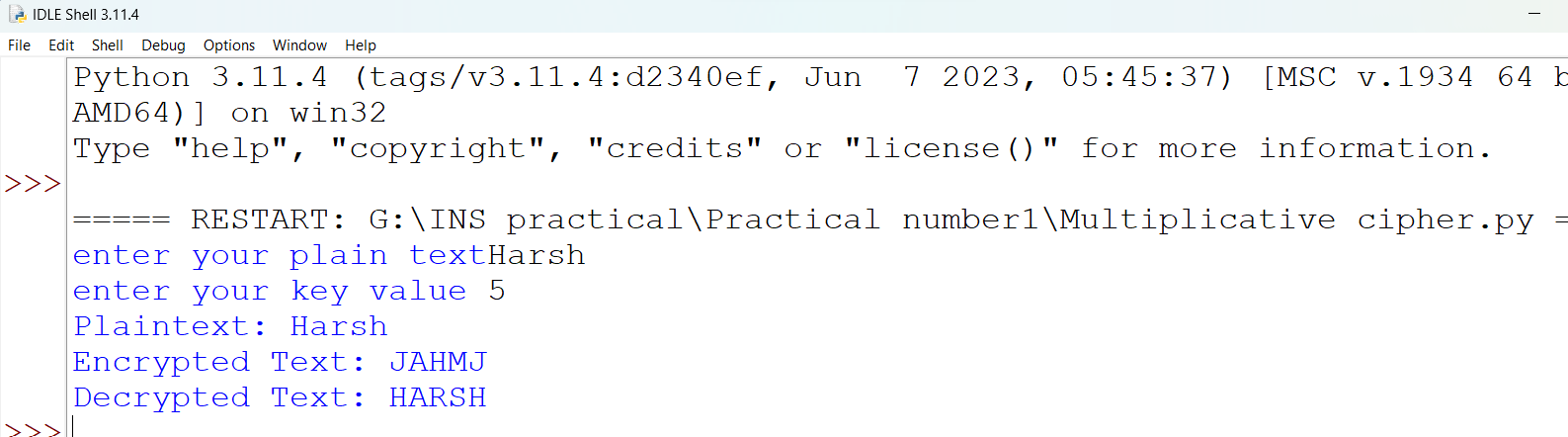
decrypted\_text = multiplicative\_decipher(encrypted\_text, key)

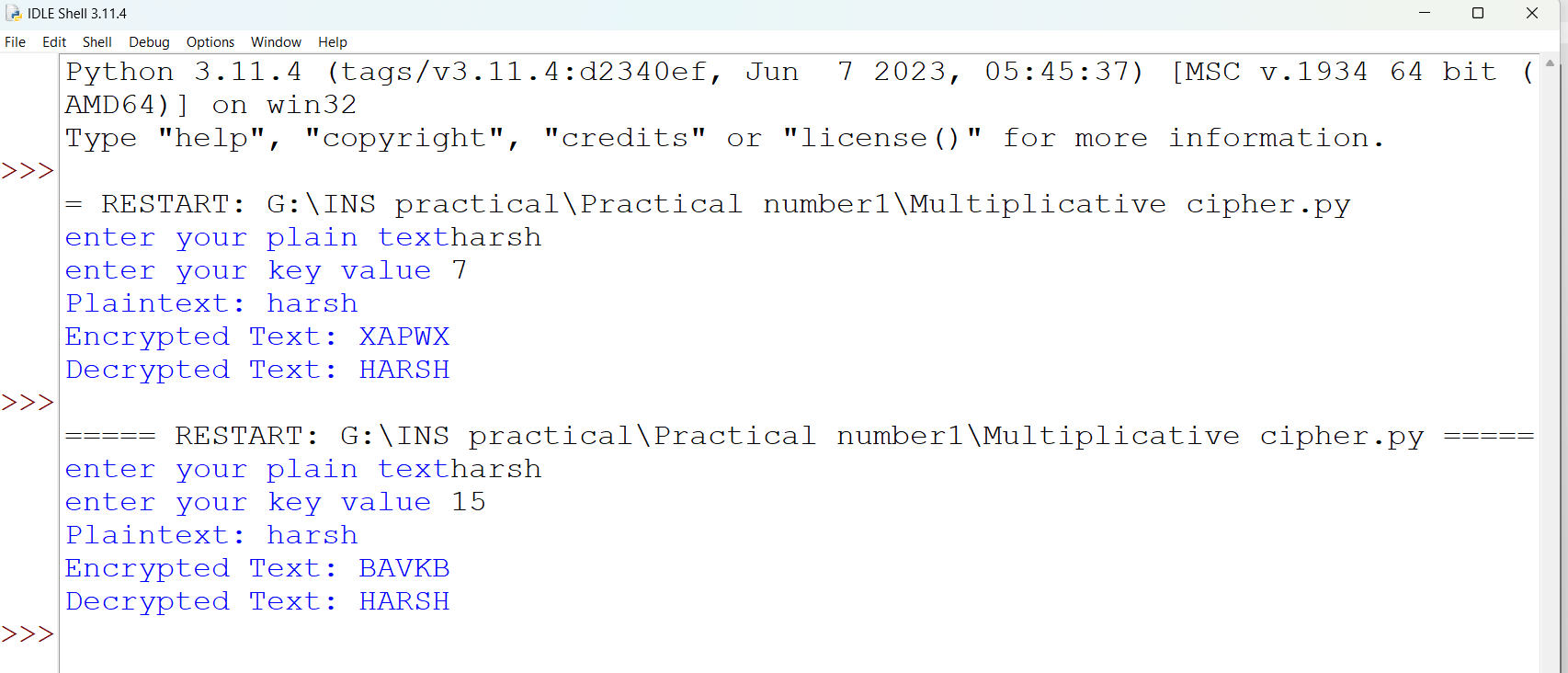
print("Plaintext:", plaintext)

print("Encrypted Text:", encrypted\_text)

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

**Output**

****

****

* **Transposition ciphers**

**Rail fence**

print("Rail\_fence algorithms:")

string=input("Enter the string:")

t=[]

def encry(string):

tt=len(string)

if (tt%2) == 0:

for i in string:

if (string.index(i)%2) == 0:

t.append(i)

for i in string:

if (string.index(i)%2)!= 0:

t.append(i)

else:

for i in string:

if (string.index(i)%2) == 0:

t.append(i)

for i in string:

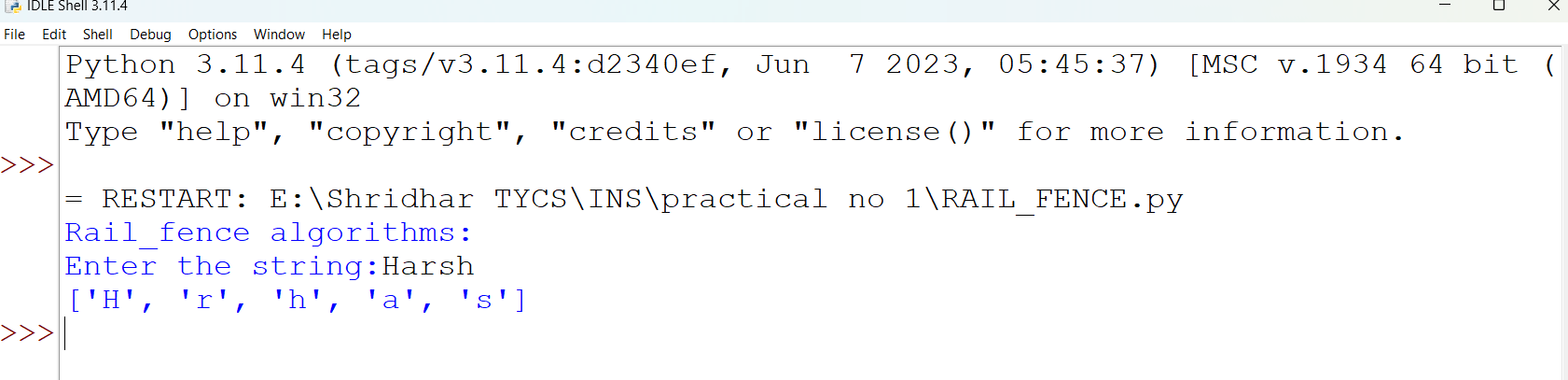
if (string.index(i)%2)!= 0:

t.append(i)

encry(string)

print(t)

**Output**

****

**Columner Cipher**

**Code**

import math

key = "ABC"

**# Encryption**

def encryptMessage(msg):

cipher = ""

k\_indx = 0

msg\_len = float(len(msg))

msg\_lst = list(msg)

key\_lst = sorted(list(key))

col = len(key)

row = int(math.ceil(msg\_len / col))

fill\_null = int((row \* col) - msg\_len)

msg\_lst.extend('\_' \* fill\_null)

matrix = [msg\_lst[i: i + col]

for i in range(0, len(msg\_lst), col)]

for \_ in range(col):

curr\_idx = key.index(key\_lst[k\_indx])

cipher += ''.join([row[curr\_idx]

for row in matrix])

k\_indx += 1

return cipher

**# Decryption**

def decryptMessage(cipher):

msg = ""

k\_indx = 0

msg\_indx = 0

msg\_len = float(len(cipher))

msg\_lst = list(cipher)

col = len(key)

row = int(math.ceil(msg\_len / col))

key\_lst = sorted(list(key))

dec\_cipher = []

for \_ in range(row):

dec\_cipher += [[None] \* col]

for \_ in range(col):

curr\_idx = key.index(key\_lst[k\_indx])

for j in range(row):

dec\_cipher[j][curr\_idx] = msg\_lst[msg\_indx]

msg\_indx += 1

k\_indx += 1

try:

msg = ''.join(sum(dec\_cipher, []))

except TypeError:

raise TypeError("This program cannot","handle repeating words.")

null\_count = msg.count('\_')

if null\_count > 0:

return msg[: -null\_count]

return msg

**# Driver Code**

msg = input("Enter the Message : ")

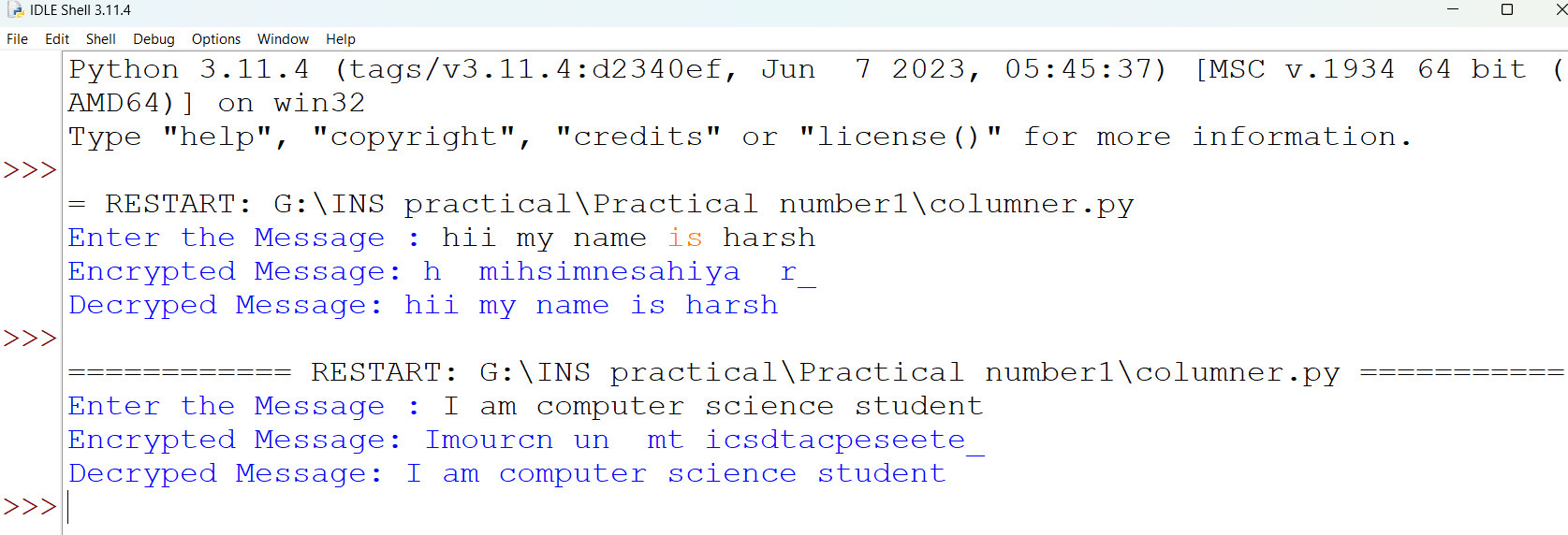
cipher = encryptMessage(msg)

print("Encrypted Message: {}".

format(cipher))

print("Decryped Message: {}".

format(decryptMessage(cipher)))

****