**School: Engineering And Technology.**

**Dept: Computer Science & Engineering.**

**Subject: ELECTIVE -2(CRYPTOGRAPHY AND CYBER SECURITY ) LAB….**

**((( Assignment )))**

**Name: Abir Paul**

**Course: B.Tech**

**Roll No: UG/02/BTCSE/2018/061**

**Enrolment No: AU/2018/02/0002552**

**Sec: B**

**Year: 3rd**

**Semester: 6th**

**ASSIGMNENT : 1**

**1.**

**Program Code:**

//1

#include <math.h>

#include <stdio.h>

int main() {

double base, exp, result;

printf("Enter a base number: ");

scanf("%lf", &base);

printf("Enter an exponent: ");

scanf("%lf", &exp);

// calculates the power

result = pow(base, exp);

printf("%.1lf^%.1lf = %.2lf", base, exp, result);

return 0;

}

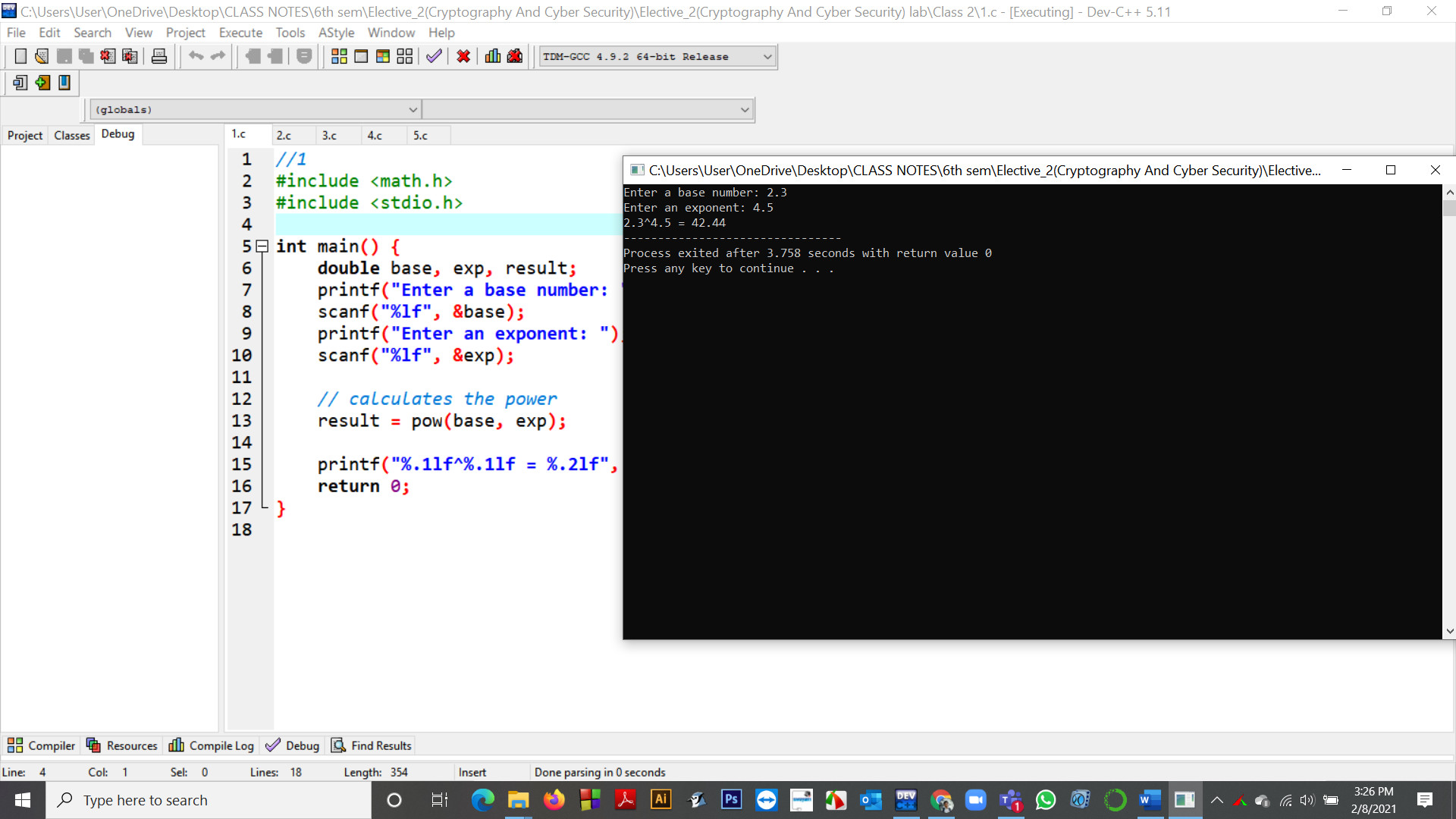
**Output:**

**Enter a base number: 2.3**

**Enter an exponent: 4.5**

**2.3^4.5 = 42.44**

Screenshot of Output:



**2.**

**Program Code:**

//2

#include <stdio.h>

#include <math.h>

int main()

{

double num = 5.6, result;

result = log(num);

printf("log(%.1f) = %.2f", num, result);

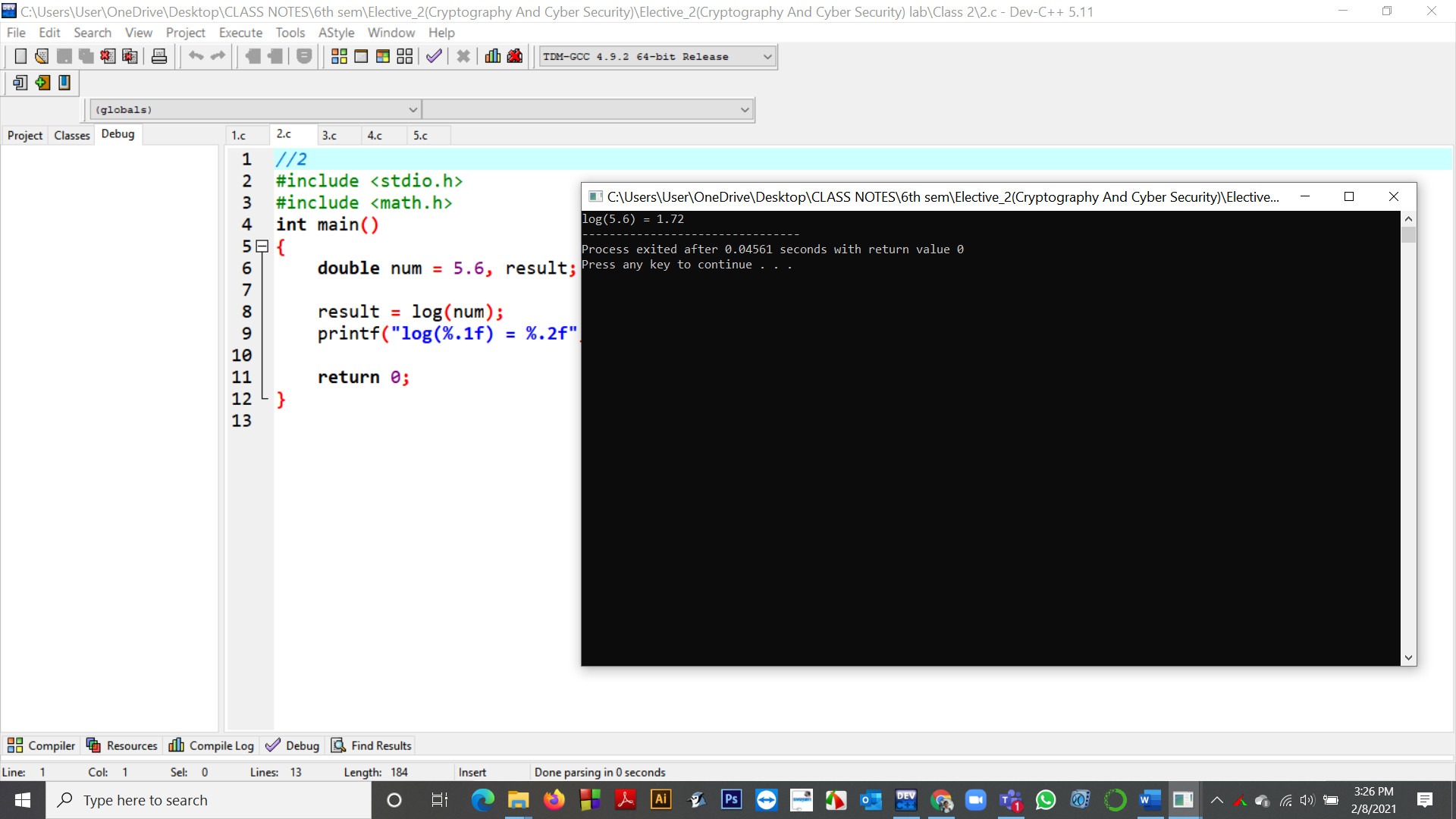
return 0;

}

**Output:**

log(5.6) = 1.72

Screenshot of Output:



**3.**

**Program Code:**

//3

#include <stdio.h>

#include <math.h>

int main()

{

double x;

double result;

x = 2.3;

result = sin(x);

printf("sin(%.2lf) = %.2lf\n", x, result);

x = -2.3;

result = sin(x);

printf("sin(%.2lf) = %.2lf\n", x, result);

x = 0;

result = sin(x);

printf("sin(%.2lf) = %.2lf\n", x, result);

return 0;

}

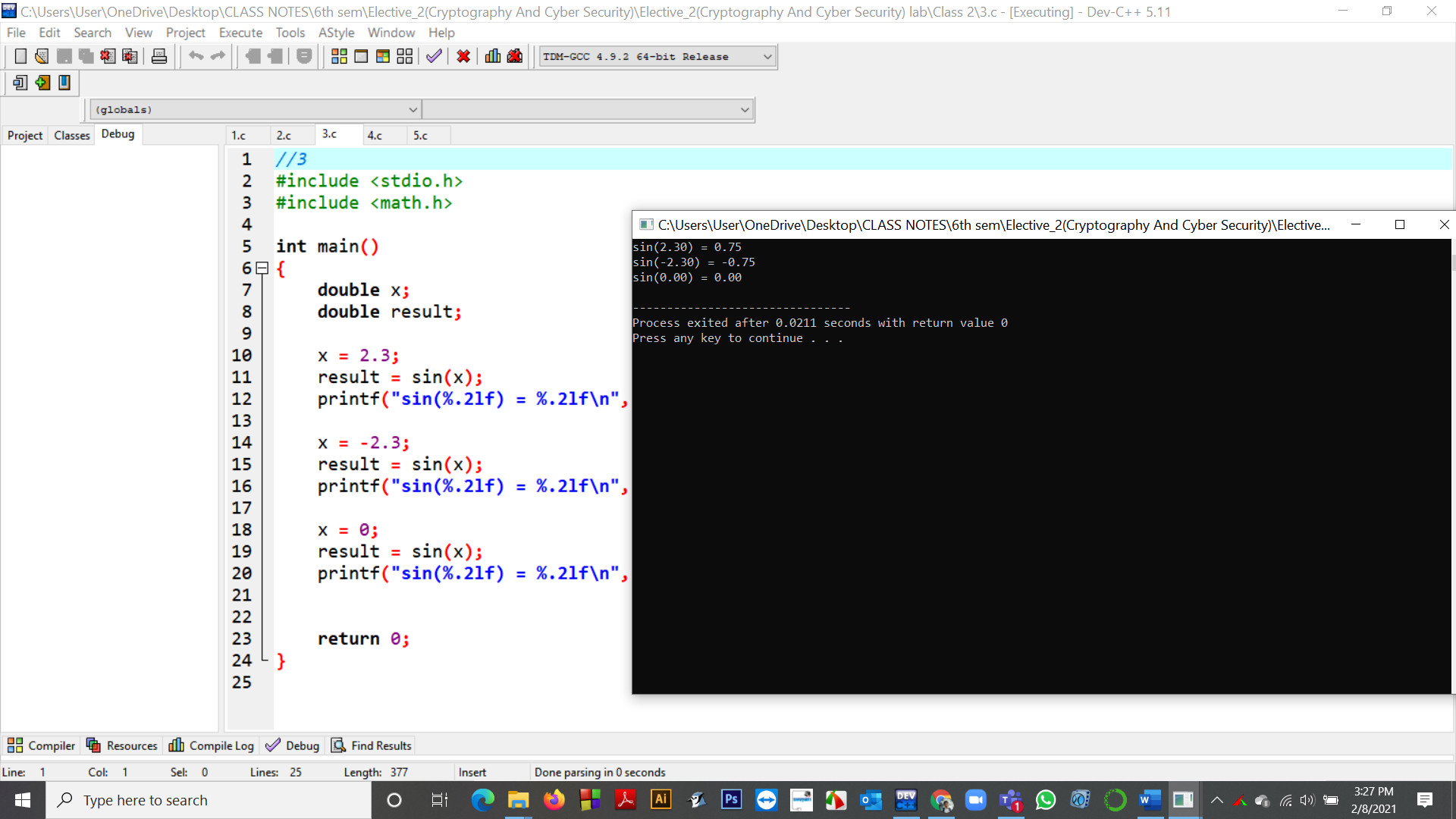
**Output:**

sin(2.30) = 0.75

sin(-2.30) = -0.75

sin(0.00) = 0.00

Screenshot of Output:



**4. C program to demonstrate Basic Euclidean Algorithm**

**Program Code:**

//4 .C program to demonstrate Basic Euclidean Algorithm

#include <stdio.h>

// Function to return gcd of a and b

int gcd(int a, int b)

{

if (a == 0)

return b;

return gcd(b % a, a);

}

// Driver program to test above function

int main()

{

int a = 10, b = 15;

printf("GCD(%d, %d) = %d\n", a, b, gcd(a, b));

a = 35, b = 10;

printf("GCD(%d, %d) = %d\n", a, b, gcd(a, b));

a = 31, b = 2;

printf("GCD(%d, %d) = %d\n", a, b, gcd(a, b));

return 0;

}

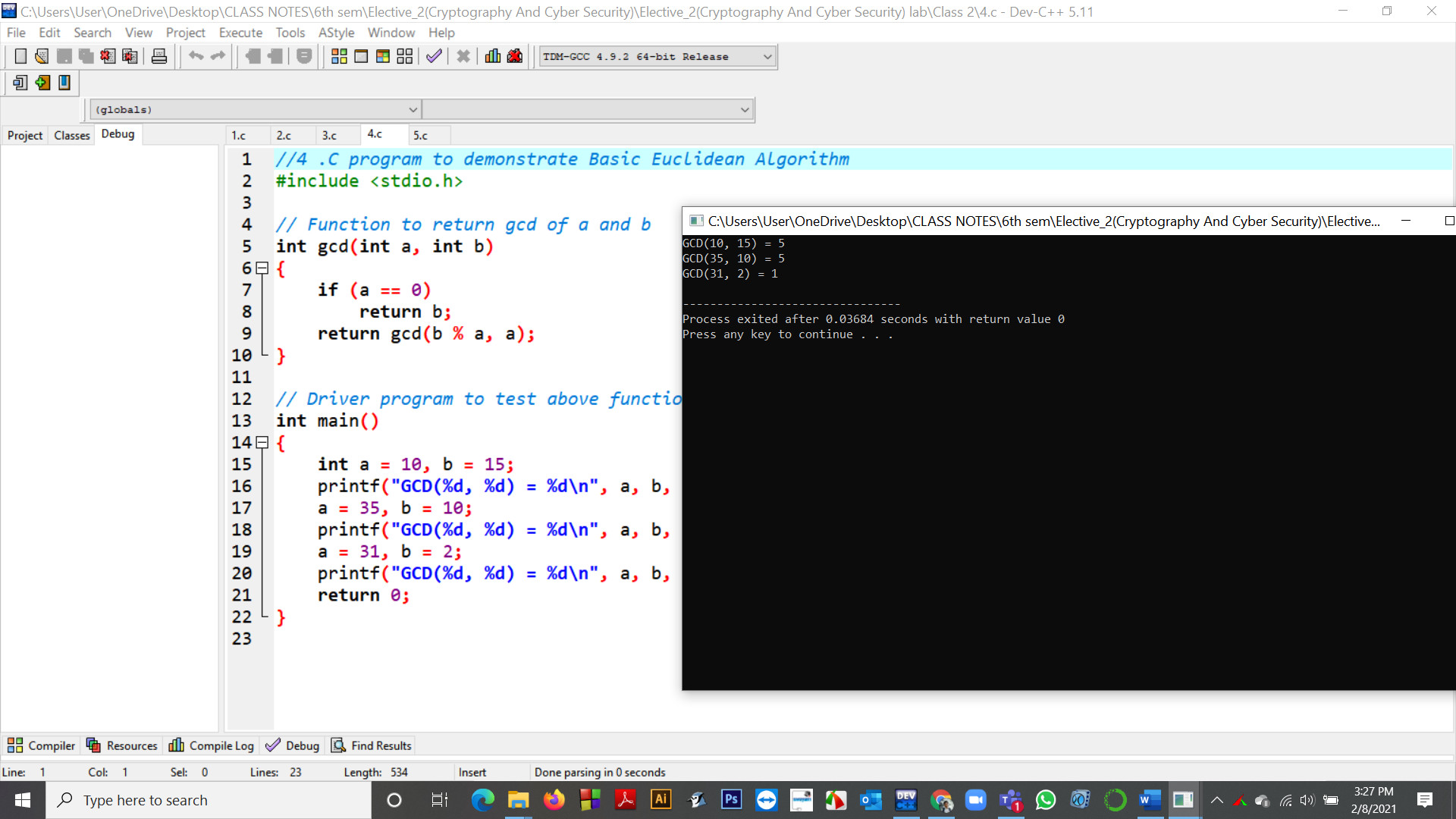
**Output:**

GCD(10, 15) = 5

GCD(35, 10) = 5

GCD(31, 2) = 1

Screenshot of Output:



**5.** **C function for extended Euclidean Algorithm**

**Program Code:**

//5

#include <stdio.h>

// C function for extended Euclidean Algorithm

int gcdExtended(int a, int b, int\* x, int\* y)

{

// Base Case

if (a == 0) {

\*x = 0;

\*y = 1;

return b;

}

int x1, y1; // To store results of recursive call

int gcd = gcdExtended(b % a, a, &x1, &y1);

// Update x and y using results of recursive

// call

\*x = y1 - (b / a) \* x1;

\*y = x1;

return gcd;

}

// Driver Program

int main()

{

int x, y;

int a = 35, b = 15;

int g = gcdExtended(a, b, &x, &y);

printf("gcd(%d, %d) = %d", a, b, g);

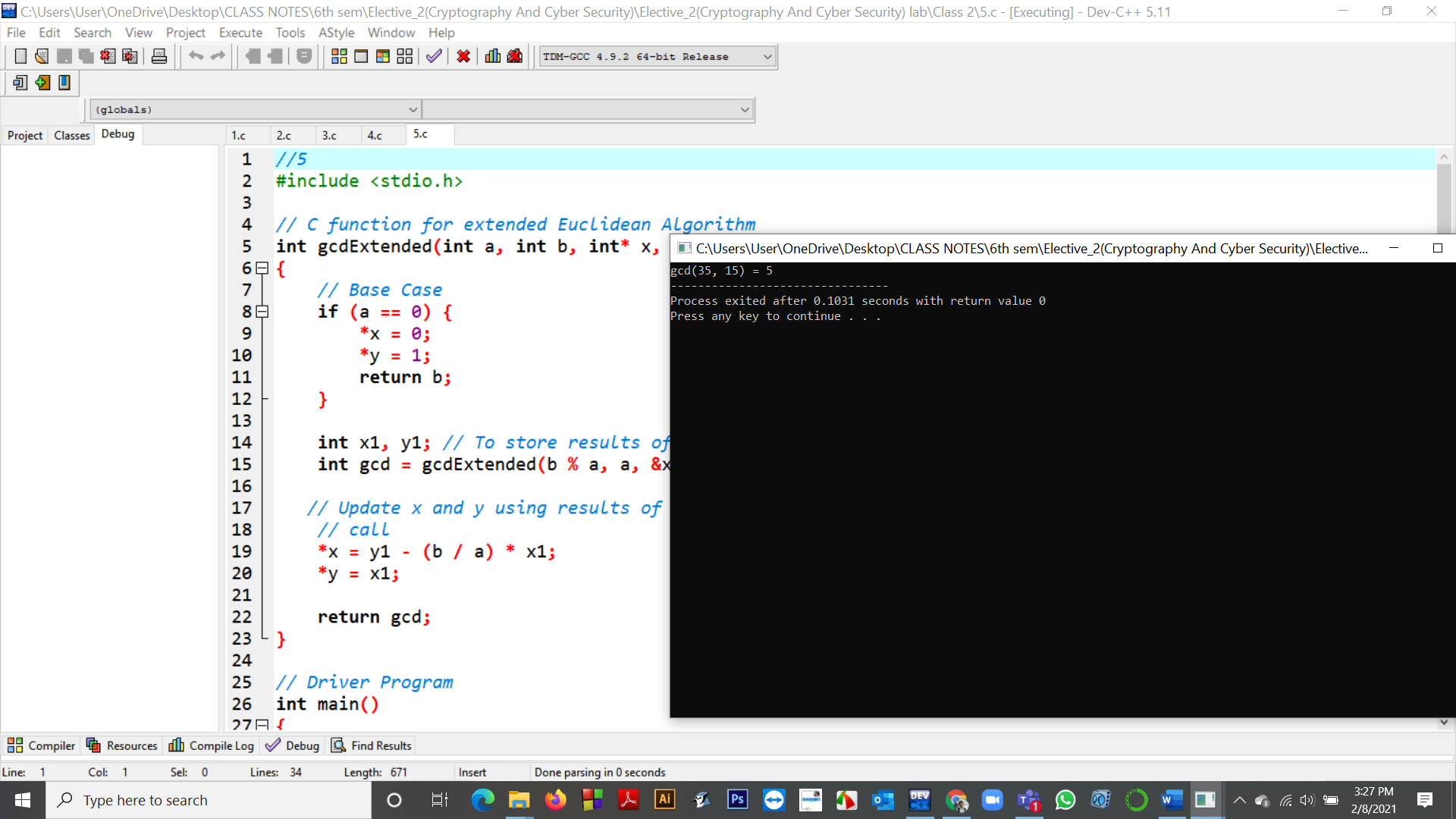
return 0;

}

**Output:**

gcd(35, 15) = 5

Screenshot of Output:



**ASSIGMNENT : 3**

1. **Find multiplication inverse of 17 in z-25 domain**

**Program Code:**

**def modInverse (a, m):**

**for x in range(1, m):**

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

**return x**

**return 1**

**# Driver Code**

**a = 17**

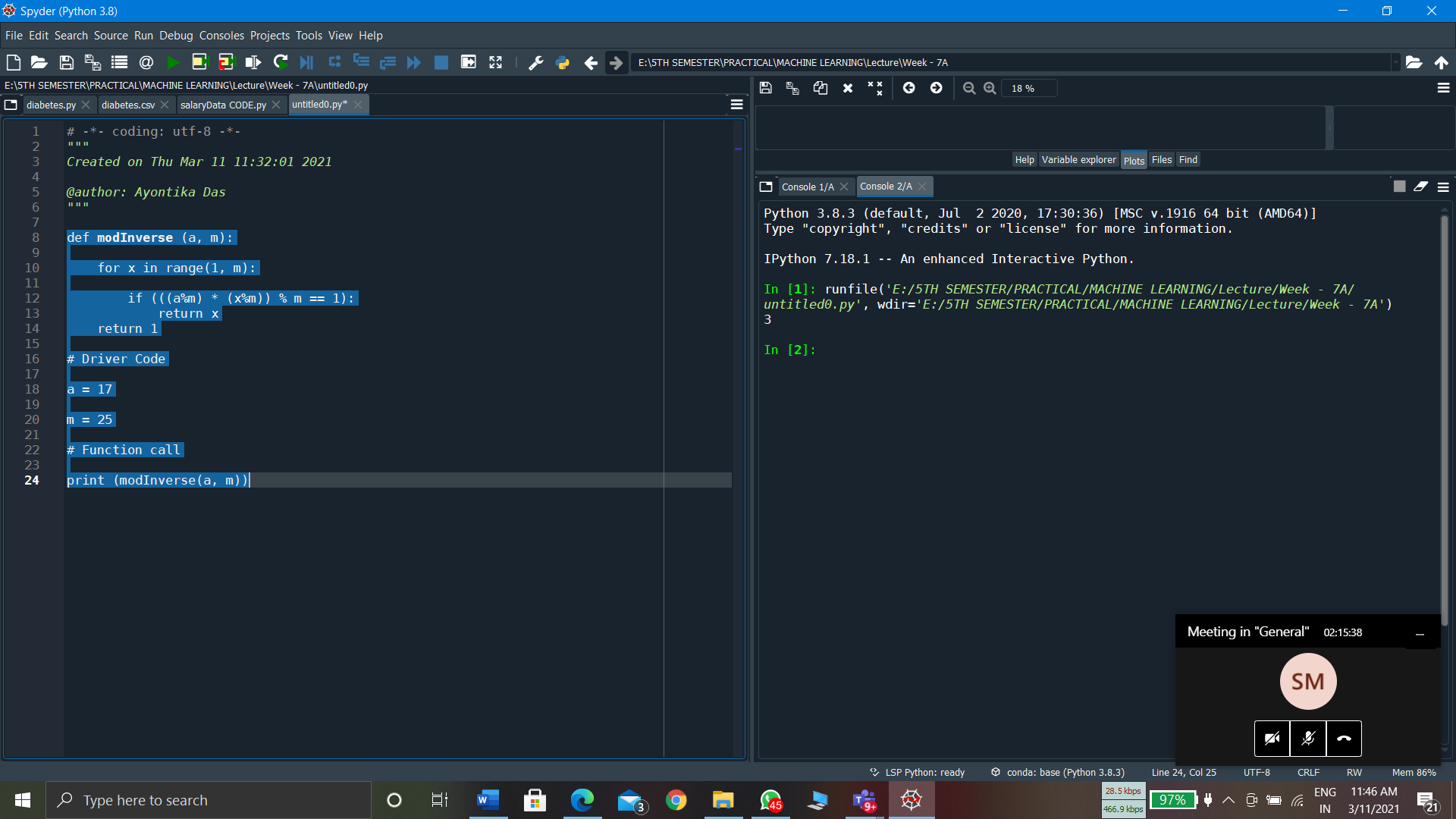
**m = 25**

**# Function call**

**print (modInverse(a, m))**

**Output: 3**

Screenshot of Output:



1. **Find multiplication inverse of 16 in z-21 domain**

**Program Code:**

def modInverse (a, m):

for x in range(1, m):

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

return x

return 1

# Driver Code

a = 16

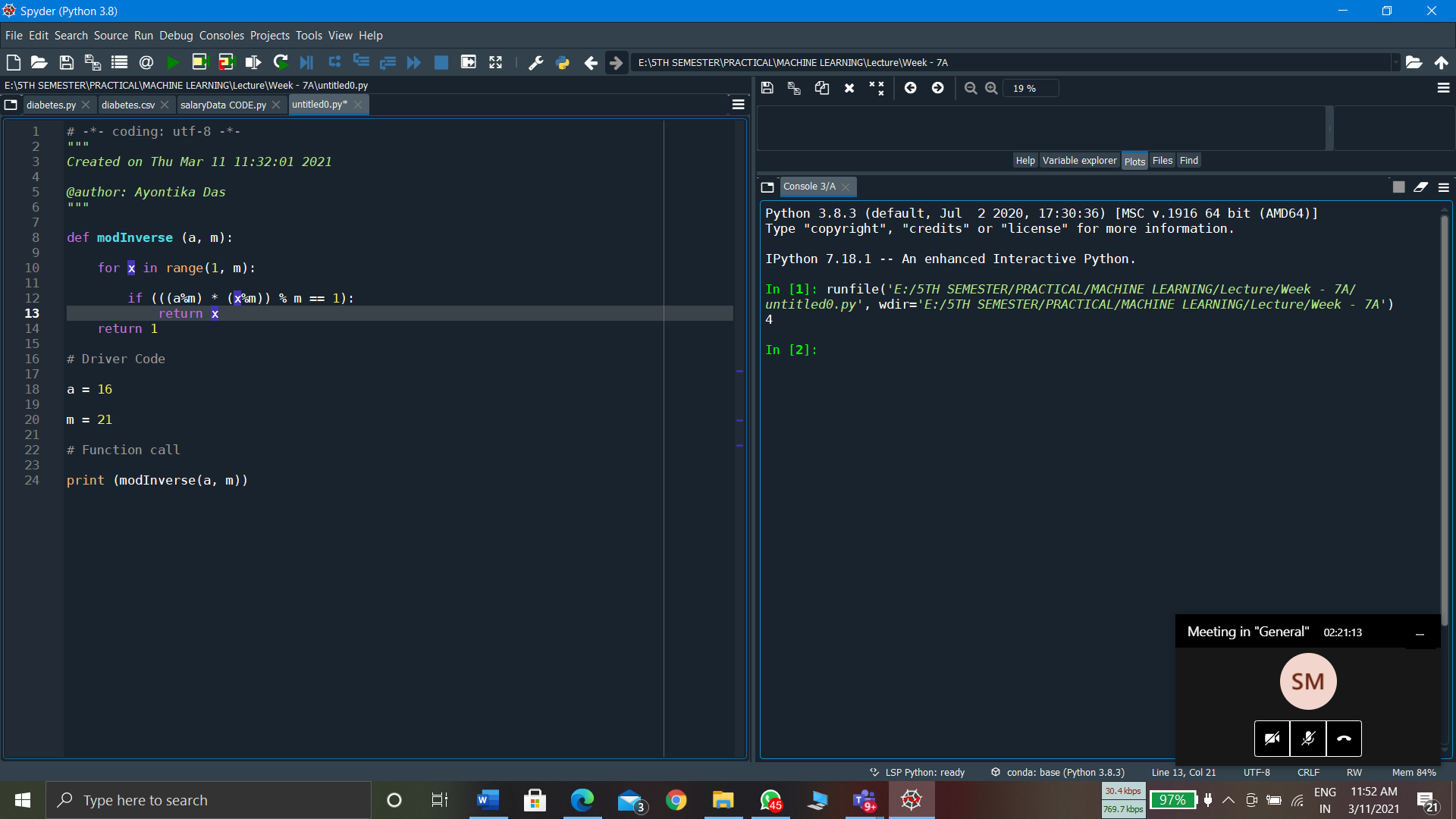
m = 21

# Function call

print (modInverse(a, m))

**Output: 4**

Screenshot of Output:



**Rail Fence Cipher Encryption and Decryption**

**INPUT:**

def encryptRailFence(text, key):

rail = [['\n' for i in range(len(text))]

for j in range(key)]

dir\_down = False

row, col = 0, 0

for i in range(len(text)):

if (row == 0) or (row == key - 1):

dir\_down = not dir\_down

rail[row][col] = text[i]

col += 1

if dir\_down:

row += 1

else:

row -= 1

result = []

for i in range(key):

for j in range(len(text)):

if rail[i][j] != '\n':

result.append(rail[i][j])

return("" . join(result))

def decryptRailFence(cipher, key):

rail = [['\n' for i in range(len(cipher))]

for j in range(key)]

dir\_down = None

row, col = 0, 0

for i in range(len(cipher)):

if row == 0:

dir\_down = True

if row == key - 1:

dir\_down = False

rail[row][col] = '\*'

col += 1

if dir\_down:

row += 1

else:

row -= 1

index = 0

for i in range(key):

for j in range(len(cipher)):

if ((rail[i][j] == '\*') and

(index < len(cipher))):

rail[i][j] = cipher[index]

index += 1

result = []

row, col = 0, 0

for i in range(len(cipher)):

if row == 0:

dir\_down = True

if row == key-1:

dir\_down = False

if (rail[row][col] != '\*'):

result.append(rail[row][col])

col += 1

if dir\_down:

row += 1

else:

row -= 1

return("".join(result))

if \_\_name\_\_ == "\_\_main\_\_":

print(encryptRailFence("attack at once", 2))

print(encryptRailFence("GeeksforGeeks ", 3))

print(encryptRailFence("defend the east wall", 3))

print(decryptRailFence("GsGsekfrek eoe", 3))

print(decryptRailFence("atc toctaka ne", 2))

print(decryptRailFence("dnhaweedtees alf tl", 3))

**OUTPUT:**

atc toctaka ne

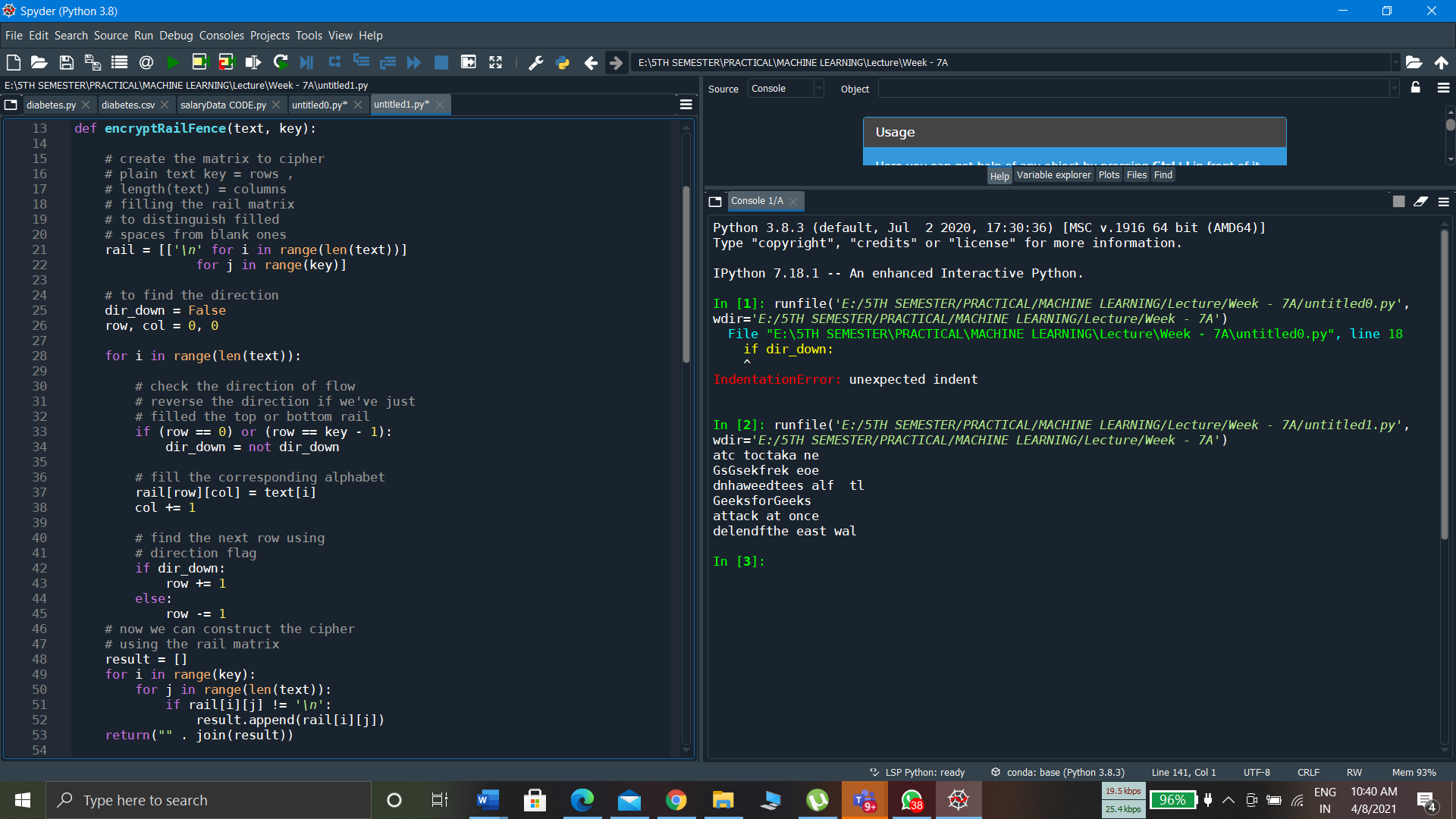
GsGsekfrek eoe

dnhaweedtees alf tl

GeeksforGeeks

attack at once

delendfthe east wal

****

**ASSIGMNENT : 6 (25.03.2021)**

**Program Name:**

**1.Columnar Transposition Chiper in Python..**

**Program Code:**

# Python3 implementation of

# Columnar Transposition

import math

key = "HACK"

# Encryption

def encryptMessage(msg):

cipher = ""

# track key indices

k\_indx = 0

msg\_len = float(len(msg))

msg\_lst = list(msg)

key\_lst = sorted(list(key))

# calculate column of the matrix

col = len(key)

# calculate maximum row of the matrix

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

# add the padding character '\_' in empty

# the empty cell of the matix

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

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

# create Matrix and insert message and

# padding characters row-wise

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

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

# read matrix column-wise using key

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 = ""

# track key indices

k\_indx = 0

# track msg indices

msg\_indx = 0

msg\_len = float(len(cipher))

msg\_lst = list(cipher)

# calculate column of the matrix

col = len(key)

# calculate maximum row of the matrix

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

# convert key into list and sort

# alphabetically so we can access

# each character by its alphabetical position.

key\_lst = sorted(list(key))

# create an empty matrix to

# store deciphered message

dec\_cipher = []

for \_ in range(row):

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

# Arrange the matrix column wise according

# to permutation order by adding into new matrix

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

# convert decrypted msg matrix into a string

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 = "Enemy attacks to night"

cipher = encryptMessage(msg)

print("Encrypted Message: {}".

format(cipher))

print("Decryped Message: {}".

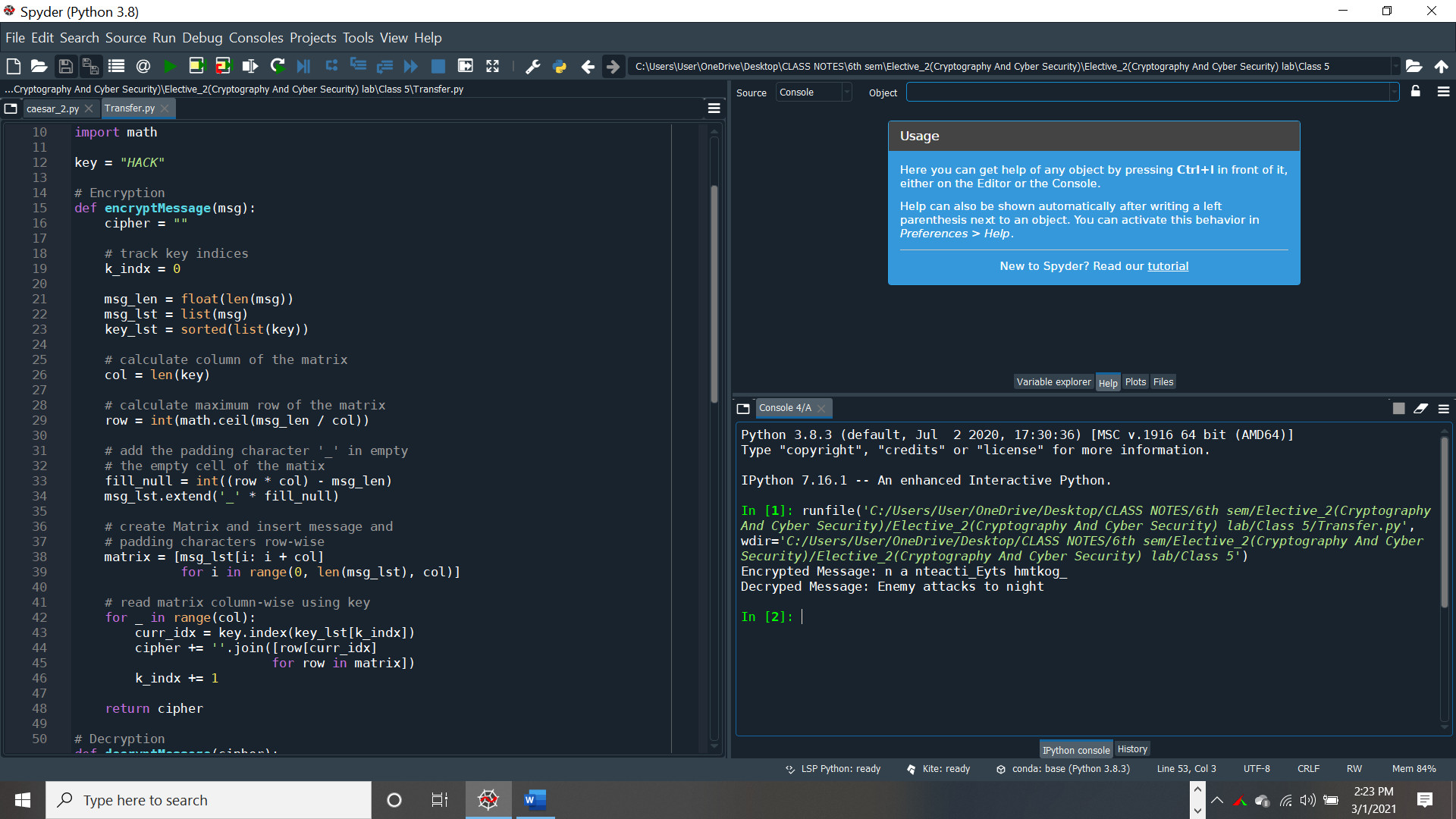
format(decryptMessage(cipher)))

**Output:**

**Encrypted Message: n a nteacti\_Eyts hmtkog\_**

**Decryped Message: Enemy attacks to night**

**Screenshot of Output:**



**ASSIGMNENT : 2**

**Program Name:** Autokey Cipher

**Program Code:**

def encrypt(message, key):

cipher = ""

for i in message:

if i.isupper():

cipher += chr((ord(i) + key - 65) % 26 + 65)

elif i.islower():

cipher += chr((ord(i) + key - 97) % 26 + 97)

else:

cipher+=" "

return cipher

message = input("Enter the message:")

key = input("Enter the key numeric value or any alphabet:")

if key.isupper():

key = ord(key) - 65

elif key.islower():

key = ord(key) - 97

else:

key = int(key)

print("Cipher:", encrypt(message, key))

# Decryption part

def decrypt(cipher, key):

message = ""

for i in cipher:

if i.isupper():

message += chr((ord(i) - key - 65) % 26 + 65)

elif i.islower():

message += chr((ord(i) - key - 97) % 26 + 97)

else:

message+=" "

return message

cipher = input("Enter the cipher:")

key = input("Enter the key numeric value or any alphabet:")

if key.isupper():

key = ord(key) - 65

elif key.islower():

key = ord(key) - 97

else:

key = int(key)

print("Message", decrypt(cipher, key))

**Output:**

Enter the message:Anindya

Enter the key numeric value or any alphabet: 56

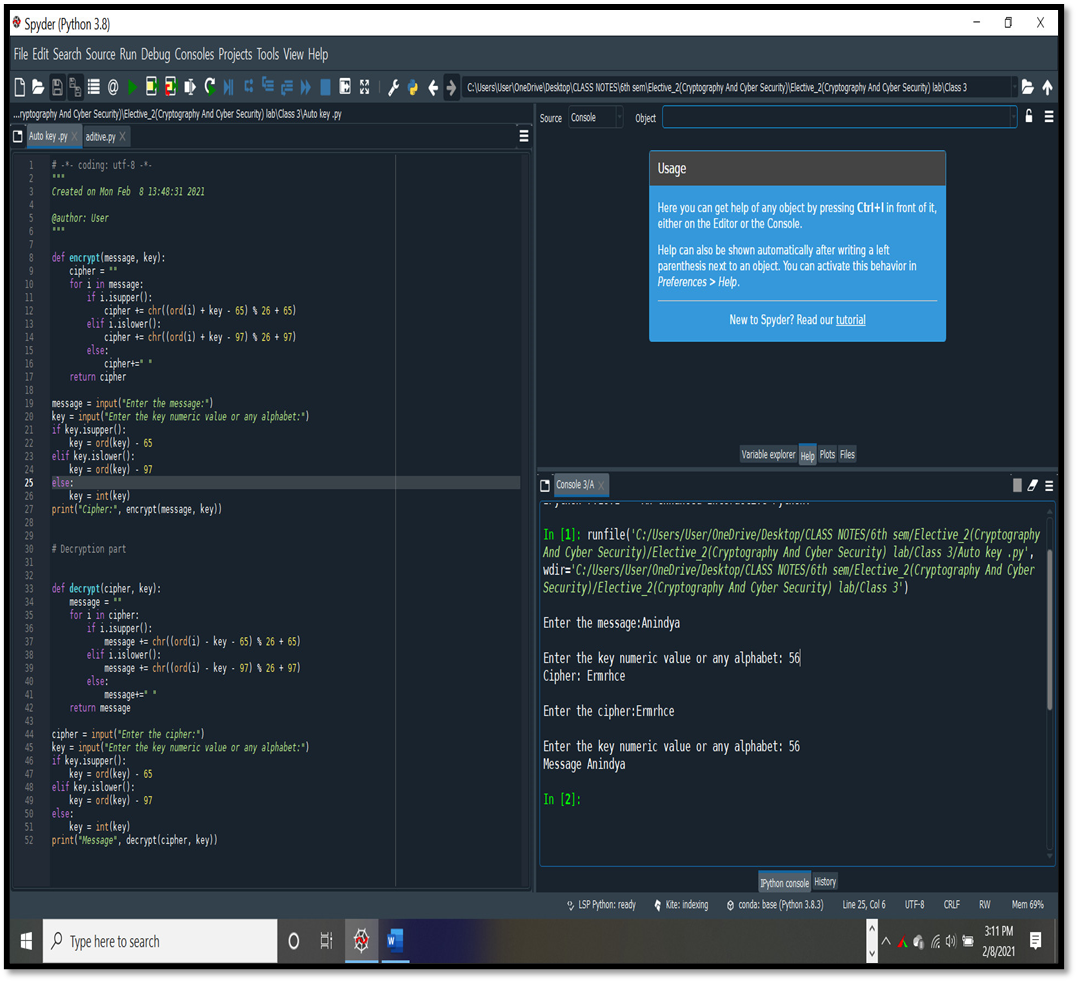
Cipher: Ermrhce

Enter the cipher:Ermrhce

Enter the key numeric value or any alphabet: 56

Message Anindya

Screenshot of Output:

****