Assignment 01 (AND and XOR operation)

Roll No.317

Code:

def process\_string(string, operation):

result = "" for char in string: if operation == 'AND':

result += chr(ord(char) & 127) elif operation == 'XOR':

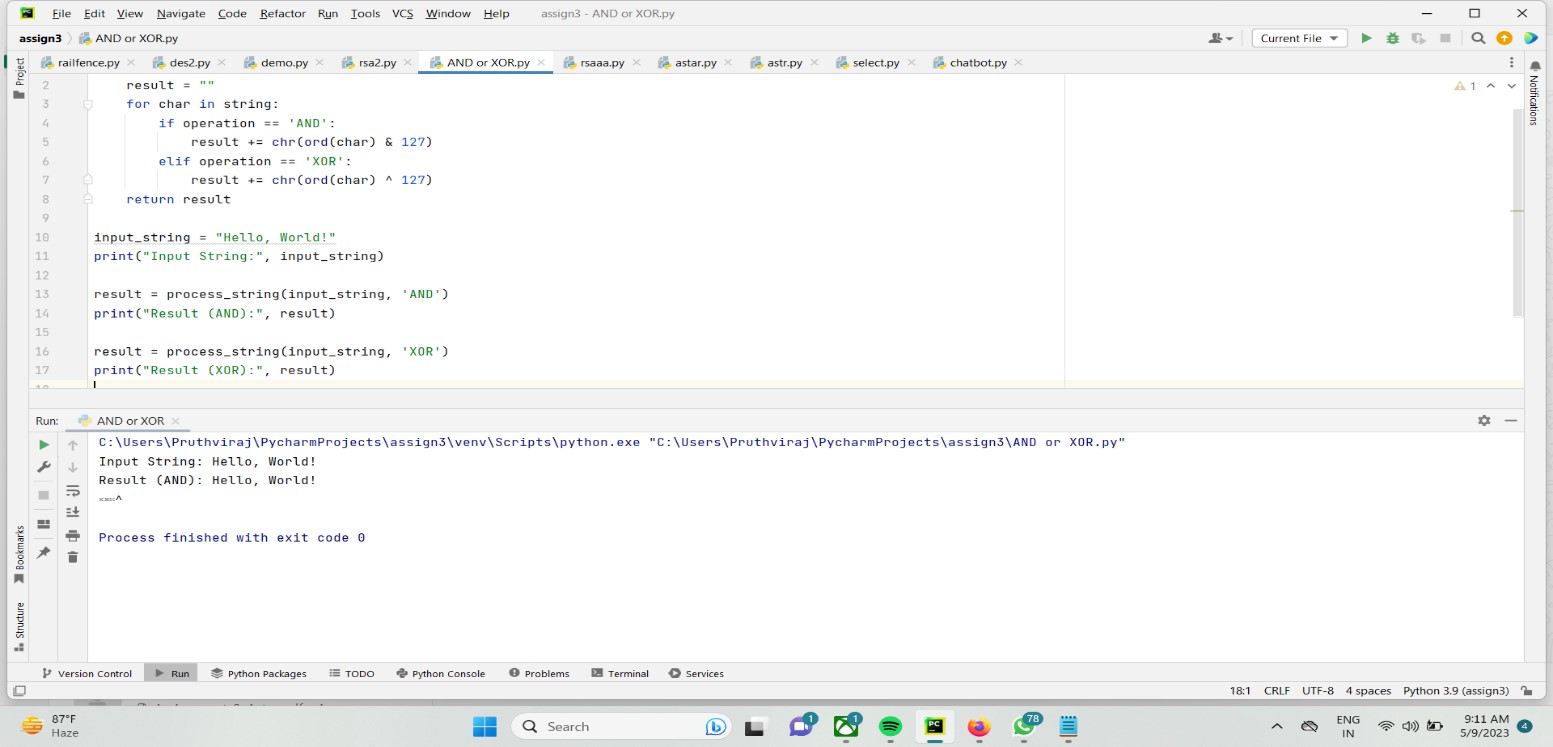
result += chr(ord(char) ^ 127) return result

input\_string = "Hello, World!" print("Input String:", input\_string)

result = process\_string(input\_string, 'AND') print("Result (AND):", result)

result = process\_string(input\_string, 'XOR') print("Result (XOR):", result)

output:



Assignment No. 02(Rail Fence Cipher)

Roll No.317 Code:

import math

*# Encryption function for Rail Fence Cipher* def encryptRailFence(message, key):

message = message.replace(" ", "") *# remove spaces from message* num\_rows = key

num\_cols = math.ceil(len(message) / num\_rows)

arr = [[' ' for j in range(num\_cols)] for i in range(num\_rows)] k = 0 for j in range(num\_cols): for i in range(num\_rows): if k < len(message):

arr[i][j] = message[k] k += 1 else: break ciphertext = "" for i in range(num\_rows): for j in range(num\_cols): ciphertext += arr[i][j] return ciphertext

*# Decryption function for Rail Fence Cipher* def decryptRailFence(ciphertext, key):

ciphertext = ciphertext.replace(" ", "") *# remove spaces from ciphertext* num\_rows = key

num\_cols = math.ceil(len(ciphertext) / num\_rows)

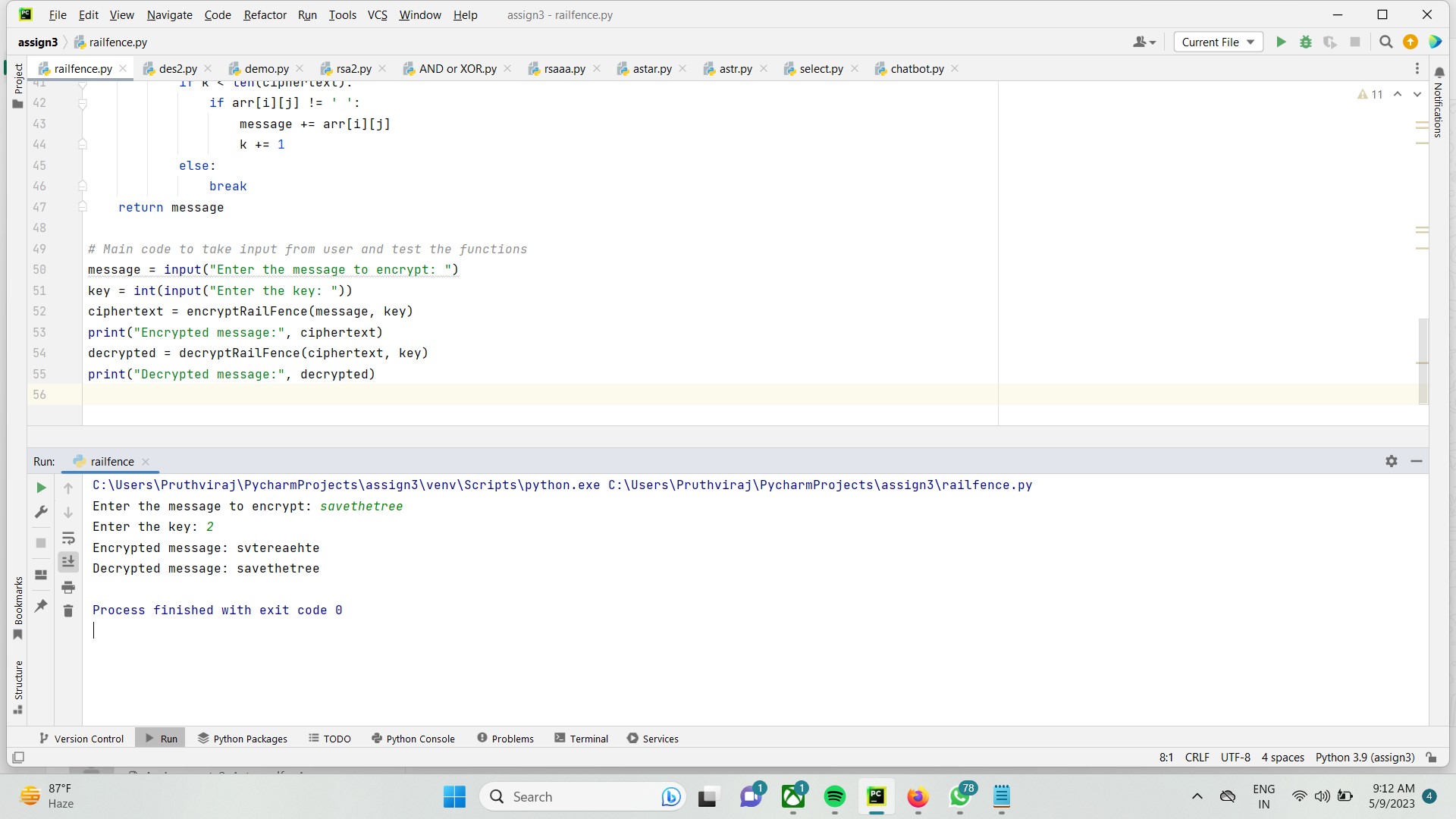
arr = [[' ' for j in range(num\_cols)] for i in range(num\_rows)] k = 0 for i in range(num\_rows): for j in range(num\_cols): if k < len(ciphertext):

arr[i][j] = ciphertext[k] k += 1 else: break message = "" k = 0 for j in range(num\_cols): for i in range(num\_rows): if k < len(ciphertext): if arr[i][j] != ' ': message += arr[i][j] k += 1 else: break return message

*# Main code to take input from user and test the functions* message = input("Enter the message to encrypt: ")

key = int(input("Enter the key: ")) ciphertext = encryptRailFence(message, key) print("Encrypted message:", ciphertext) decrypted = decryptRailFence(ciphertext, key) print("Decrypted message:", decrypted)

Output:



Assignment No. 04(RSA algorithm)

Roll No.317 Code:

import math

message = int(input("Enter the message to be encrypted: "))

p = 7 q = 17 n = p\*q m = (p-1)\*(q-1)

for i in range(2,m): if math.gcd(i,m) == 1:

e = i break for i in range(m): if (e\*i) % m == 1:

d = i

break

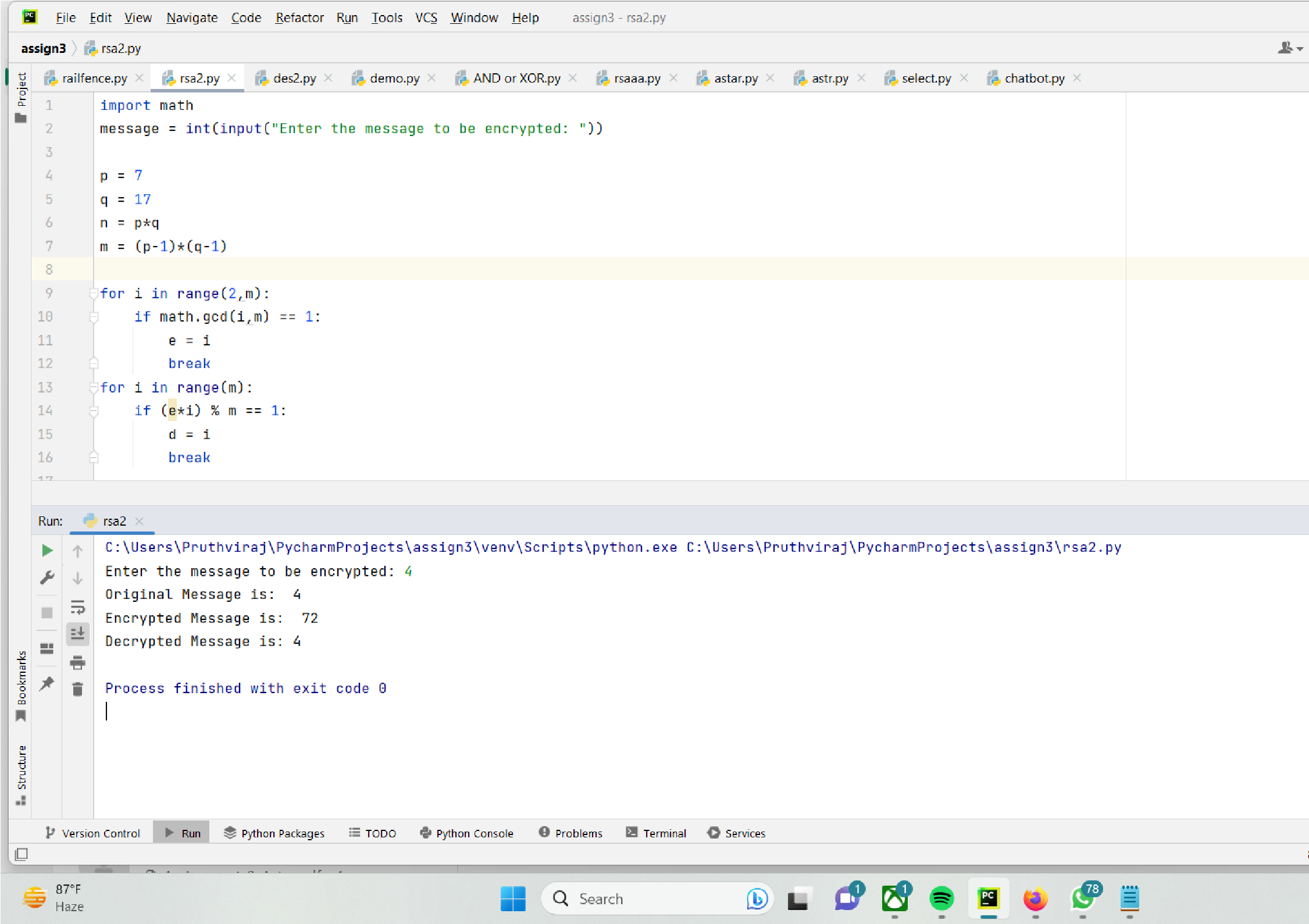
def encrypt(me): c = pow(message, e, n) return c def decrypt(ct): p = pow(ct, d, n) return p

print("Original Message is: ", message) CT = encrypt(message)

print("Encrypted Message is: ", CT) PT = decrypt(CT)

print("Decrypted Message is:", PT)

Output:



Assignment No.5(Diffi-Hellman Algorithm)

Roll No. 317 Code:

HTML File:

<!DOCTYPE html>

<html>

<head>

<title>Diffie-Hellman Key Exchange</title>

</head>

<body>

<h1>Diffie-Hellman Key Exchange</h1>

<p>Enter a prime number and a base value:</p>

<form>

<label for="prime">Prime number:</label>

<input type="number" id="prime" name="prime"><br><br>

<label for="base">Base value:</label>

<input type="number" id="base" name="base"><br><br>

<button type="button" onclick="generateKeys()">Generate Keys</button>

</form>

<p>Public keys:</p>

<p>Alice: <span id="alicePubKey"></span></p>

<p>Bob: <span id="bobPubKey"></span></p>

<p>Shared secret:</p>

<p><span id="sharedSecret"></span></p>

<script src="script.js"></script>

</body>

</html>

Jscript File:

function isPrime(n) { if (n < 2) return false;

for (let i = 2; i <= Math.sqrt(n); i++) { if (n % i === 0) return false;

}

return true;

} function generateKeys() { const prime = parseInt(document.getElementById("prime").value); const base = parseInt(document.getElementById("base").value); if (!isPrime(prime)) { alert("Please enter a prime number."); return;

} const alicePrivateKey = Math.floor(Math.random() \* (prime - 2)) + 2; const bobPrivateKey = Math.floor(Math.random() \* (prime - 2)) + 2;

const alicePublicKey = modExp(base, alicePrivateKey, prime); const bobPublicKey = modExp(base, bobPrivateKey, prime);

const sharedSecret = modExp(alicePublicKey, bobPrivateKey, prime);

document.getElementById("alicePubKey").textContent = alicePublicKey; document.getElementById("bobPubKey").textContent = bobPublicKey; document.getElementById("sharedSecret").textContent = sharedSecret;

} function modExp(base, exponent, modulus) { if (modulus === 1) return 0;

let result = 1; base = base % modulus; while (exponent > 0) { if (exponent % 2 === 1) {

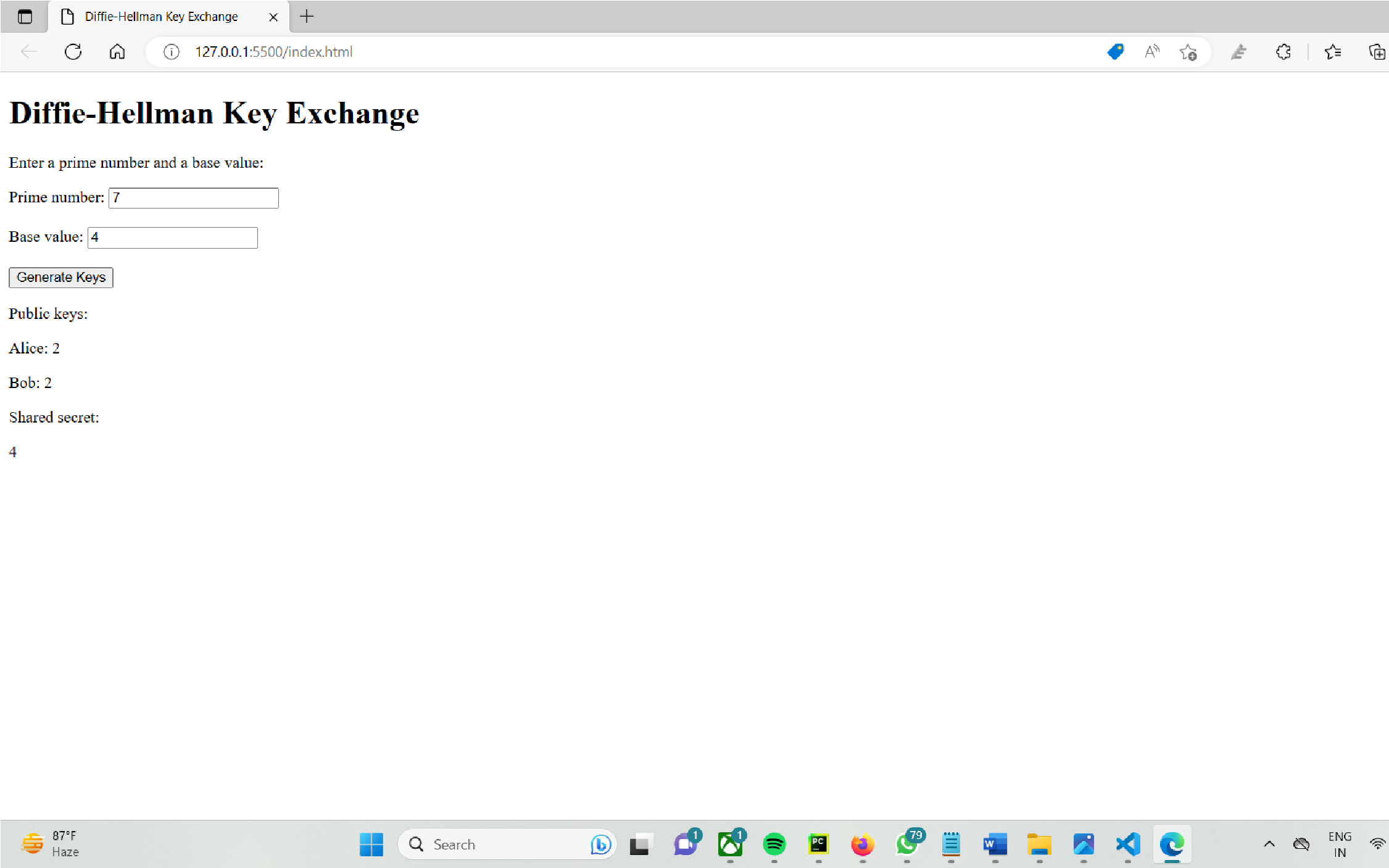
result = (result \* base) % modulus;

} exponent = Math.floor(exponent / 2); base = (base \* base) % modulus;

}

return result;

}

Output:

Assignment No. 01(BFS & DFS )

Roll No. 317 Code:

graph = { '1' : ['2','5'],

'2' : ['3', '4'],

'5' : ['6'],

'3' : [],

'4' : ['6'],

'6' : []

} visited = [] queue = [] def breadthFirstSearch(visited, graph, node):

visited.append(node) queue.append(node)

while queue: m = queue.pop(0) print (m, end = " ")

for neighbour in graph[m]: if neighbour not in visited: visited.append(neighbour) queue.append(neighbour)

print("Breadth-First Search: ") breadthFirstSearch(visited, graph, '1')

*# Program Output:-*

*# Breadth-First Search:*

*# 1 2 5 3 4 6*

*# Depth First Search:*

graph = { '1' : ['2','5'],

'2' : ['3', '4'],

'5' : ['6'], '3' : [],

'4' : ['6'],

'6' : []

} visited = set() def depthFirstSearch(visited, graph, node): if node not in visited:

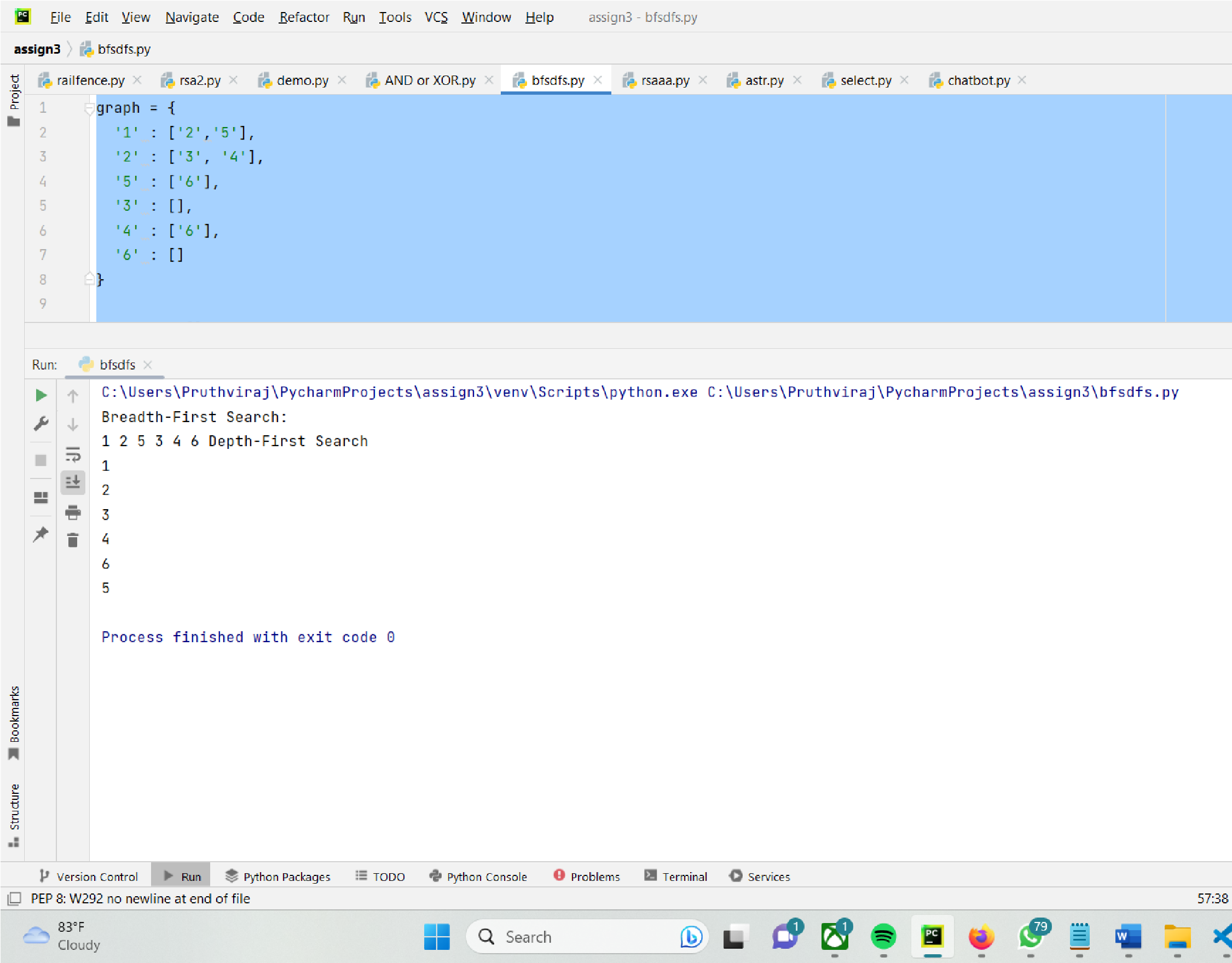
print (node) visited.add(node) for neighbour in graph[node]:

depthFirstSearch(visited, graph, neighbour)

print("Depth-First Search")

depthFirstSearch(visited, graph, '1')

Output:



Assignment No.02 (Selection Sort)

Roll No. 317

Code:

def selection\_sort\_greedy(arr):

n = len(arr)

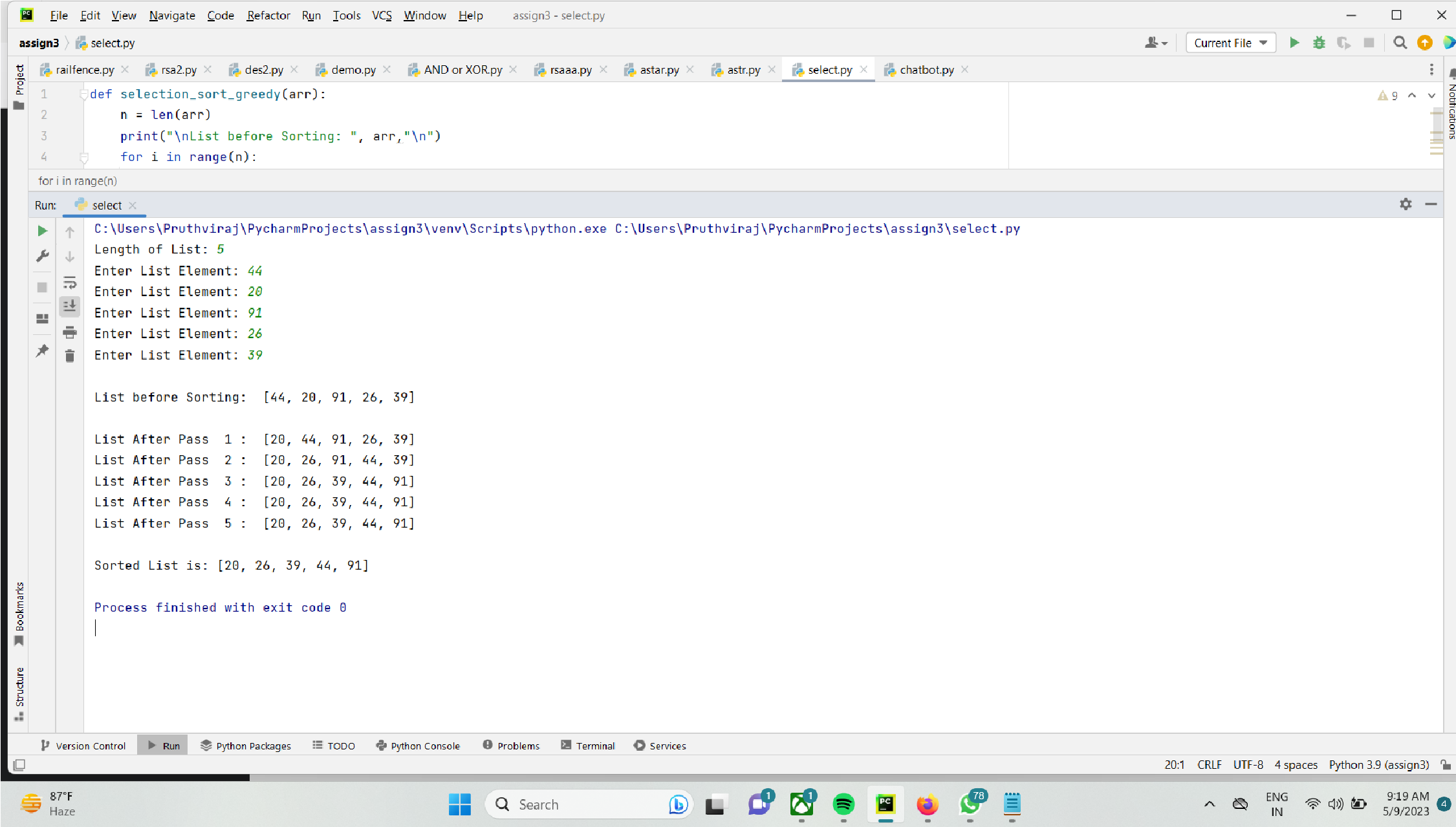
print("\nList before Sorting: ", arr,"\n") for i in range(n): min\_idx = i for j in range(i+1, n): if arr[j] < arr[min\_idx]:

min\_idx = j

arr[i], arr[min\_idx] = arr[min\_idx], arr[i] print("List After Pass ",i+1,": ",arr) return arr

n=int(input("Length of List: ")) arr=[] for i in range(n): element=int(input("Enter List Element: ")) arr.append(element)

print("\nSorted List is:", selection\_sort\_greedy(arr))

Output:

Assignment No. 03(A star Algorithm)

Roll No. 317 Code:

class box():

"""A box class for A\* Pathfinding""" def \_\_init\_\_(self, parent=None, position=None): self.parent = parent self.position = position

self.g = 0 self.h =

0 self.f = 0

def \_\_eq\_\_(self, other): return self.position == other.position

def astar(maze, start, end):

"""Returns a list of tuples as a path from the given start to the given end in the given board"""

# Create start and end node start\_node = box(None, start) start\_node.g = start\_node.h = start\_node.f = 0 end\_node = box(None, end) end\_node.g = end\_node.h = end\_node.f = 0

# Initialize both open and closed list open\_list = [] closed\_list = []

# Add the start node open\_list.append(start\_node)

# Loop until you find the end while len(open\_list) > 0:

# Get the current node current\_node = open\_list[0] current\_index = 0 for index, item in enumerate(open\_list):

if item.f < current\_node.f: current\_node = item current\_index = index

# Pop current off open list, add to closed list open\_list.pop(current\_index) closed\_list.append(current\_node)

# Found the goal if current\_node == end\_node: path = [] current = current\_node while current is not None: path.append(current.position) current = current.parent

return path[::-1] # Return reversed path

# Generate children children = [] for new\_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]: # Adjacent squares

# Get node position node\_position = (current\_node.position[0] + new\_position[0], current\_node.position[1] + new\_position[1])

# Make sure within range if node\_position[0] > (len(maze) - 1) or node\_position[0] < 0 or node\_position[1] > (len(maze[len(maze)-1]) -1) or node\_position[1] < 0:

# continue

# Make sure walkable terrain if

maze[node\_position[0]][node\_position[1]] != 0:

# continue

# Create new node new\_node = box(current\_node, node\_position)

# Append

children.append(new\_node)

# Loop through children for child in children:

# Child is on the closed list for closed\_child in closed\_list: if child == closed\_child:

# continue

# Create the f, g, and h values child.g = current\_node.g + 1 child.h =

((child.position[0] - end\_node.position[0]) \*\* 2) +

((child.position[1] - end\_node.position[1]) \*\* 2)

child.f = child.g + child.h

# Child is already in the open list for open\_node in open\_list: if child == open\_node and child.g > open\_node.g:

# continue

# Add the child to the open list open\_list.append(child)

def main(): board = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]]

start = (0, 0) end = (6,

6) path = astar(board, start, end) print(path)

if \_\_name\_\_ == '\_\_main\_\_': main()

Output:

