DATA STRUCTURE USING PYTHON PRACTICUM 1052234230

EX.NO:1

AIM :

To write a Python program that demonstrates basic operations on a Python List including creation, addition, accessing, and removal of elements.

ALGORITHM :

1. Start the program.
2. Create a list with some initial elements
3. Add an element using append()
4. Access an element using its index
5. Remove an element using remove() or pop()
6. Display the results
7. Stop the program.

PROGRAM:

```
numbers= []
print(" Initial List:", numbers)
numbers.append (10)
numbers append (20)
numbers.append (30)
numbers. insert (1,15)
print("List after adding eletment:"numbers)
print ("First element:", numbers[0])
print(Last element", numbers [-1])
print ("All element:", numbers)
for num in numbers:
```

```
print (num, end="")

print()

numbers.remove (20)

remove _item= numbers.pop()

print ("Remove item using pop():",remove_item)

print("List after removing elements:", numbers)
```

OUTPUT:

Initial list[]

list after adding element [10,15,20,30]

First element: 10

Last element: 30

All element : [10 15 20 30]

Remove item using pop(): 30

Left after removing element: [10, 15]

EX.NO:2

AIM:

To implement a singly linked list in Python with operations to a) create a singly linked list b) add elements c) remove elements.

ALGORITHM :

1. Start the program.

2. Create Singly Linked List

a) Define a Node class with data and next.

b) Define a SinglyLinkedList class with head = None.

3. Add Element

a) Create a new node.

b) If list is empty → new node becomes head.

c) Else traverse to last node.

d) Attach new node at the end.

4. Remove Element

a) If list empty → stop.

b) If head has the value → move head to next node.

c) Else traverse to find the node before the target.

5. Display the result.

6.Stop the program.

PROGRAM:

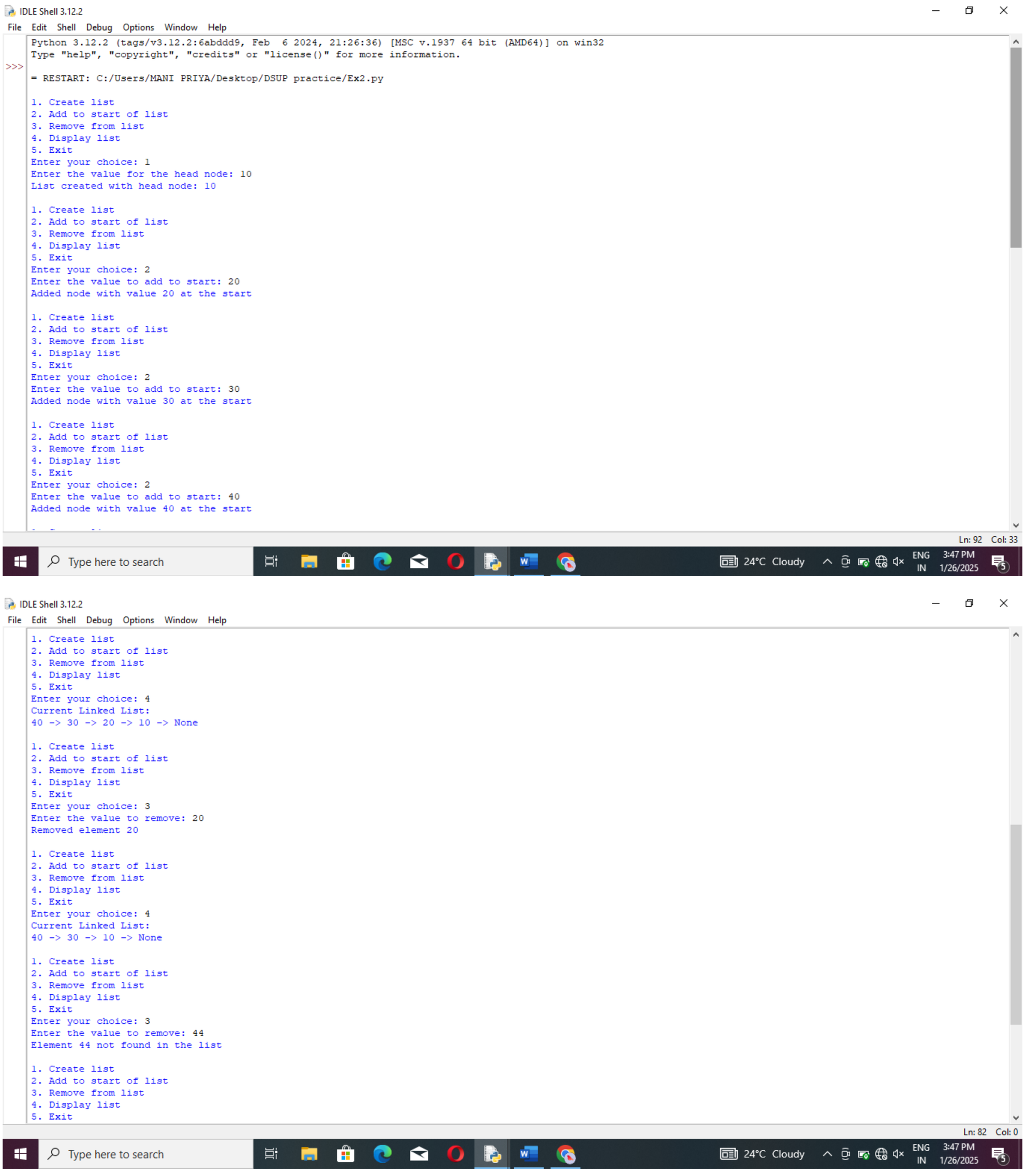
```
class Node:
        def __init__(self, data):
                self.data = data
                self.next = None
class SinglyLinkedList:
        def __init__(self):
                self.head = None
    def create(self, data):
                self.head = Node(data)
                print(f"List created with head node: {data}")
  def addstart(self, data):
                new_node = Node(data)
                new_node.next = self.head
                self.head = new_node
               print(f"Added node with value {data} at the start")
    def remove(self, data):
                current = self.head
```

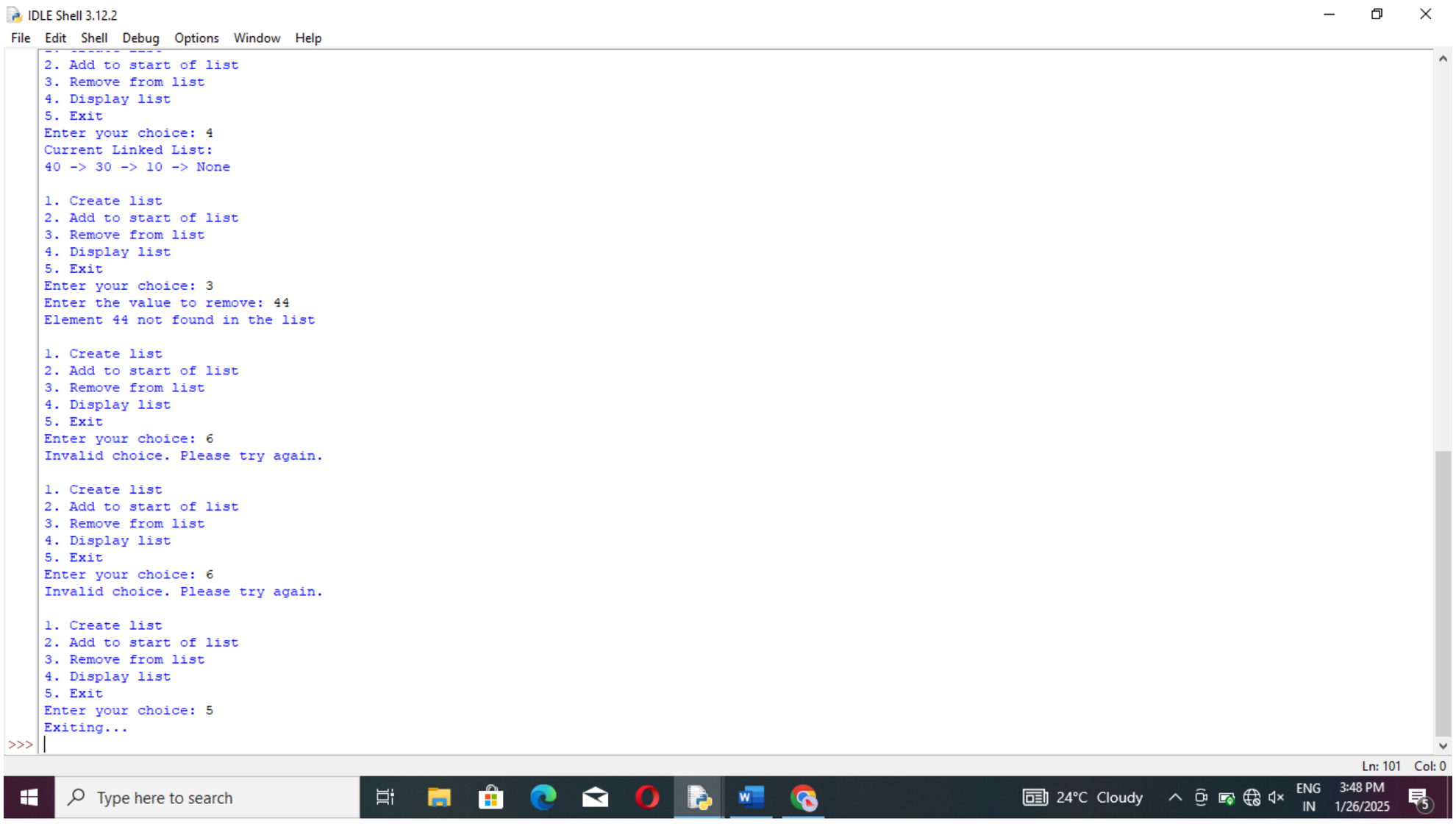
```
        if not current:
            print("List is empty")
            return
    # If the head node is to be removed
        if current.data == data:
            self.head = current.next
            print(f"Removed element {data} from head")
            return
        # Traverse to find the node to remove
            prev = None
            while current and current.data != data:
                prev = current
                current = current.next
    if not current:
                print(f"Element {data} not found in the list")
                return
    # Unlink the node to remove it
            prev.next = current.next
            print(f"Removed element {data}")
    def display(self):
        if not self.head:
            print("The list is empty")
            retur
        current = self.head
        while current:
            print(current.data, end=" -&gt; ")
```

```
                current = current.next
            print("None")
# Menu-driven program
sll = SinglyLinkedList()
while True:
    print("\n1. Create list")
    print("2. Add to start of list")
    print("3. Remove from list")
    print("4. Display list")
    print("5. Exit")
    choice = int(input("Enter your choice: "))
    if choice == 1:
        data = int(input("Enter the value for the head node: "))
        sll.create(data)
    elif choice == 2:
        data = int(input("Enter the value to add to start: "))
        sll.addstart(data)
    elif choice == 3:
        data = int(input("Enter the value to remove: "))
        sll.remove(data)
    elif choice == 4:
        print("Current Linked List:")
        sll.display()
    elif choice == 5:
        print("Exiting...")
        break
```

```
        else:
            print("Invalid choice. Please try again.")
```

OUTPUT:





EX.NO: 3

AIM:

To write a Python program to implement a Stack with basic operations such as push, pop, and display.

ALGORITHM :

1. Start the program.
2. Create an empty list to represent the stack.
3. Define a push() function to add an element to the top of the stack.
4. Define a pop() function to remove the top element from the stack.
5. Define a display() function to show all stack elements.
6. Use a loop to allow user to choose operations.
7. Perform the selected stack operation.
8. Stop the program.

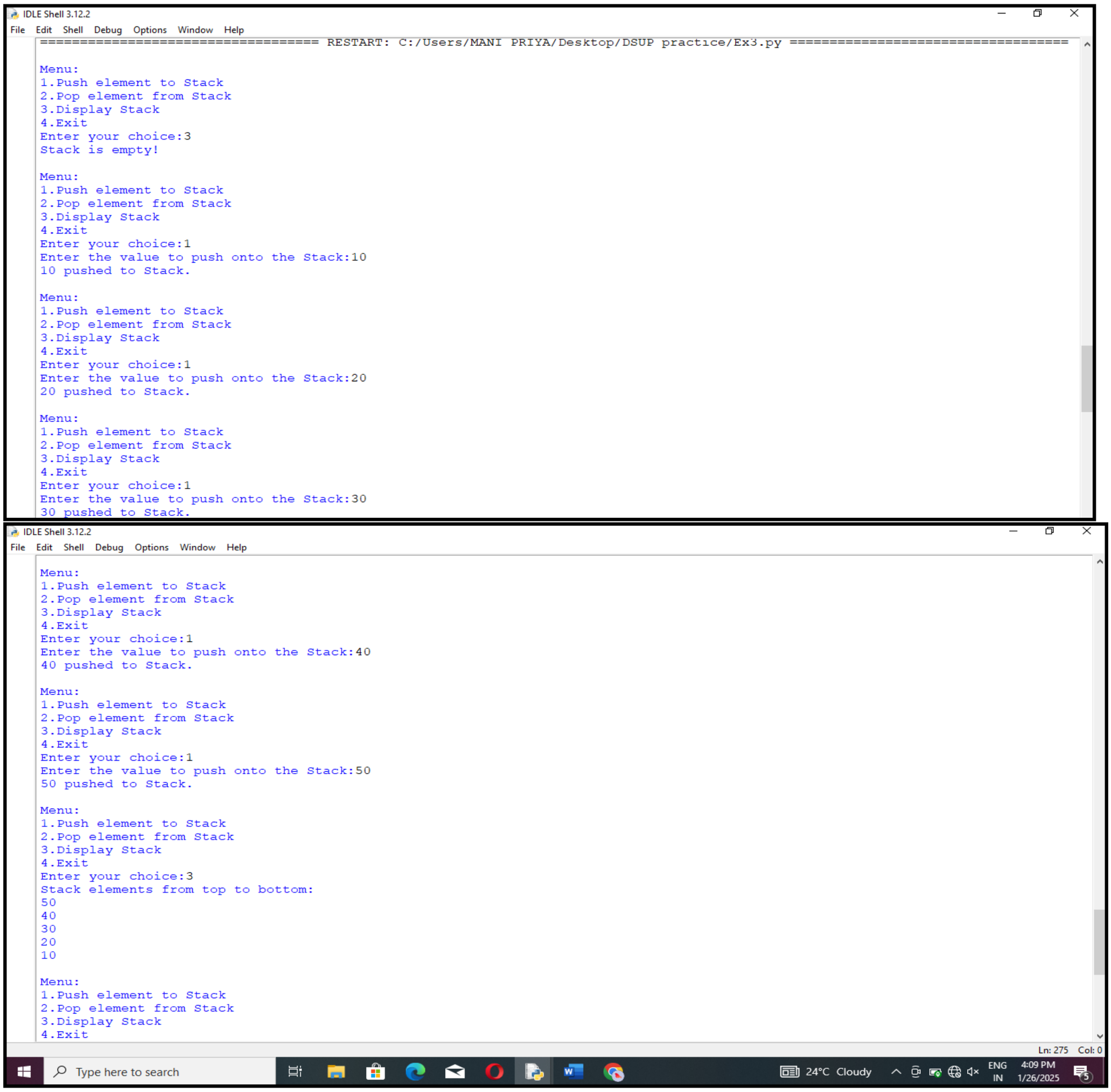
PROGRAM :

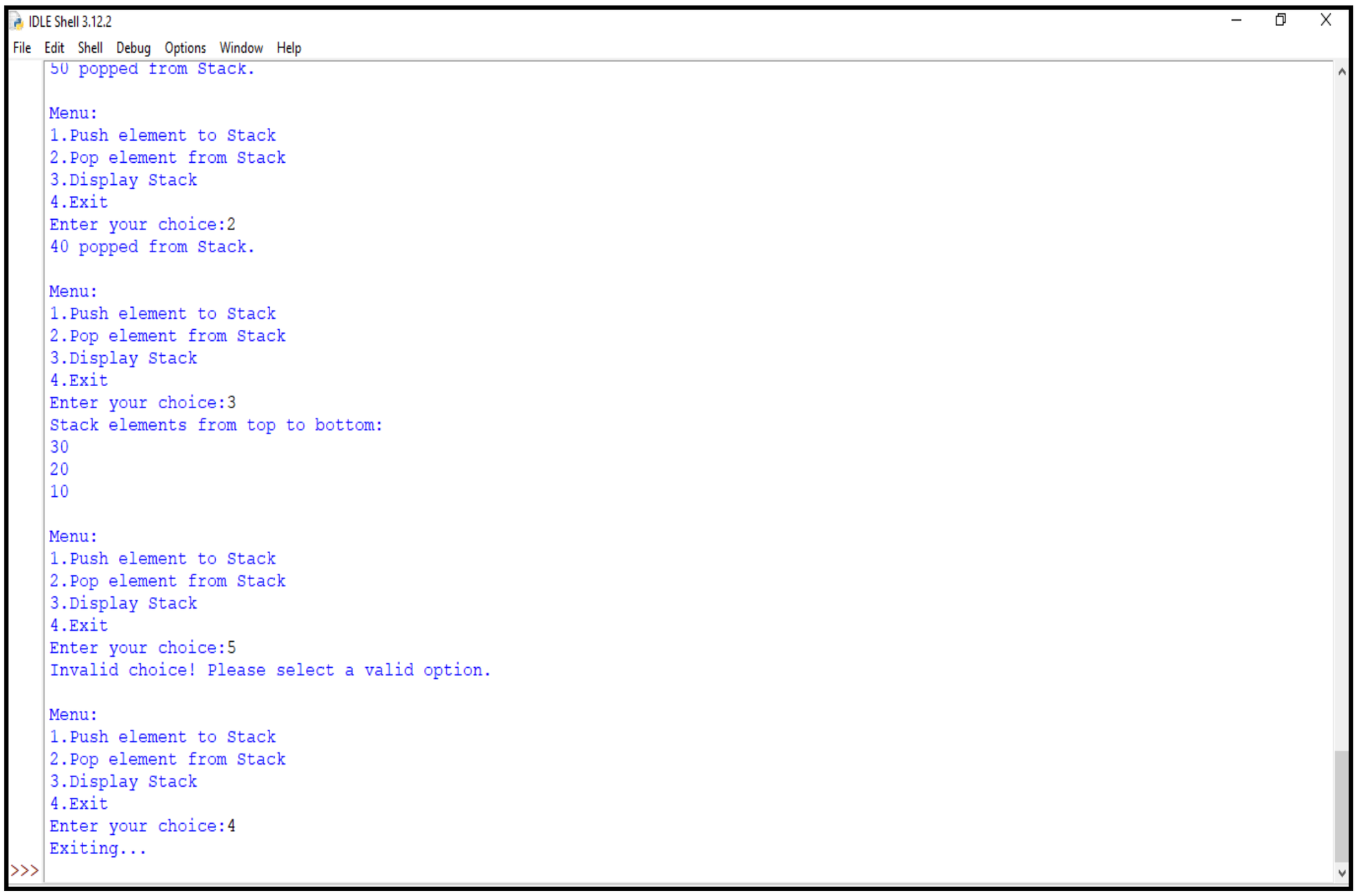
class Stack:

```
def __init__(self):
        self.stack=[]
def push(self,item):
        self.stack.append(item)
        print(f"{item} pushed to Stack.")
def pop(self):
        if not self.isempty():
                ritem=self.stack.pop()
                print(f"{ritem} popped from Stack.")
                return ritem
        else:
                print("Stack is empty! Cannot pop an element.")
                return None
def isempty(self):
        return len(self.stack)==0
def display(self):
        if not self.isempty():
                print("Stack elements from top to bottom:")
                for item in reversed(self.stack):
                        print(item)
                else:
                print("Stack is empty!")
s=Stack()
while True:
        print("\nMenu:")
        print("1.Push element to Stack")
```

```
print("2.Pop element from Stack")
print("3.Display Stack")
print("4.Exit")
choice = int(input("Enter your choice:"))
if choice == 1:
        item=int(input("Enter the value to push onto the Stack:"))
        s.push(item)
elif choice == 2:
        s.pop()
elif choice == 3:
        s.display()
elif choice == 4:
        print("Exiting...")
        break
else:
        print("Invalid choice! Please select a valid option.")
```

OUTPUT:


IDLE Shell 3.12.2
File Edit Shell Debug Options Window Help
========== RESTART: C:/Users/MANI PRIYA/Desktop/DSUP practice/Ex3.py ==========
Menu:
1.Push element to Stack
2.Pop element from Stack
3.Display Stack
4.Exit
Enter your choice:3
Stack is empty!
Menu:
1.Push element to Stack
2.Pop element from Stack
3.Display Stack
4.Exit
Enter your choice:1
Enter the value to push onto the Stack:10
10 pushed to Stack.
Menu:
1.Push element to Stack
2.Pop element from Stack
3.Display Stack
4.Exit
Enter your choice:1
Enter the value to push onto the Stack:20
20 pushed to Stack.
Menu:
1.Push element to Stack
2.Pop element from Stack
3.Display Stack
4.Exit
Enter your choice:1
Enter the value to push onto the Stack:30
30 pushed to Stack.
IDLE Shell 3.12.2
File Edit Shell Debug Options Window Help
Menu:
1.Push element to Stack
2.Pop element from Stack
3.Display Stack
4.Exit
Enter your choice:1
Enter the value to push onto the Stack:40
40 pushed to Stack.
Menu:
1.Push element to Stack
2.Pop element from Stack
3.Display Stack
4.Exit
Enter your choice:1
Enter the value to push onto the Stack:50
50 pushed to Stack.
Menu:
1.Push element to Stack
2.Pop element from Stack
3.Display Stack
4.Exit
Enter your choice:3
Stack elements from top to bottom:
50
40
30
20
10
Menu:
1.Push element to Stack
2.Pop element from Stack
3.Display Stack
4.Exit
Ln: 275 Col: 0
Type here to search
24°C Cloudy
ENG IN
4:09 PM 1/26/2025



EX.NO:4

AIM :

To write a Python program to implement a Queue with basic operations such as enqueue, dequeue, and display.

ALGORITHM :

1. Start the program.
2. Create an empty list to represent the queue.
3. Define enqueue() to insert an element at the rear of the queue.
4. Define dequeue() to remove an element from the front of the queue.
5. Define display() to show all elements in the queue.
6. Use a loop to allow the user to choose operations.
7. Perform the selected queue operation.
8. Stop the program.

PROGRAM :

```
class Queue:
    def __init__(self):
        self.queue=[]
    def enqueue(self,item):
        self.queue.append(item)
        print(f"{item} enqueued to queue.")
    def dequeue(self):
        if not self.isempty():
            ritem=self.queue.pop()
            print(f"{ritem} dequeued from queue.")
            return ritem
        else:
            print("Queue is empty! Cannot dequeued an element.")
            return None
def isempty(self):
    return len(self.queue)==0
def display(self):
        if not self.isempty():
            print("Queue elements from front to rear:")
            for item in self.queue:
                print(item, end="")
            print()
        else:
            print("Queue is empty!")
q=Queue()
```

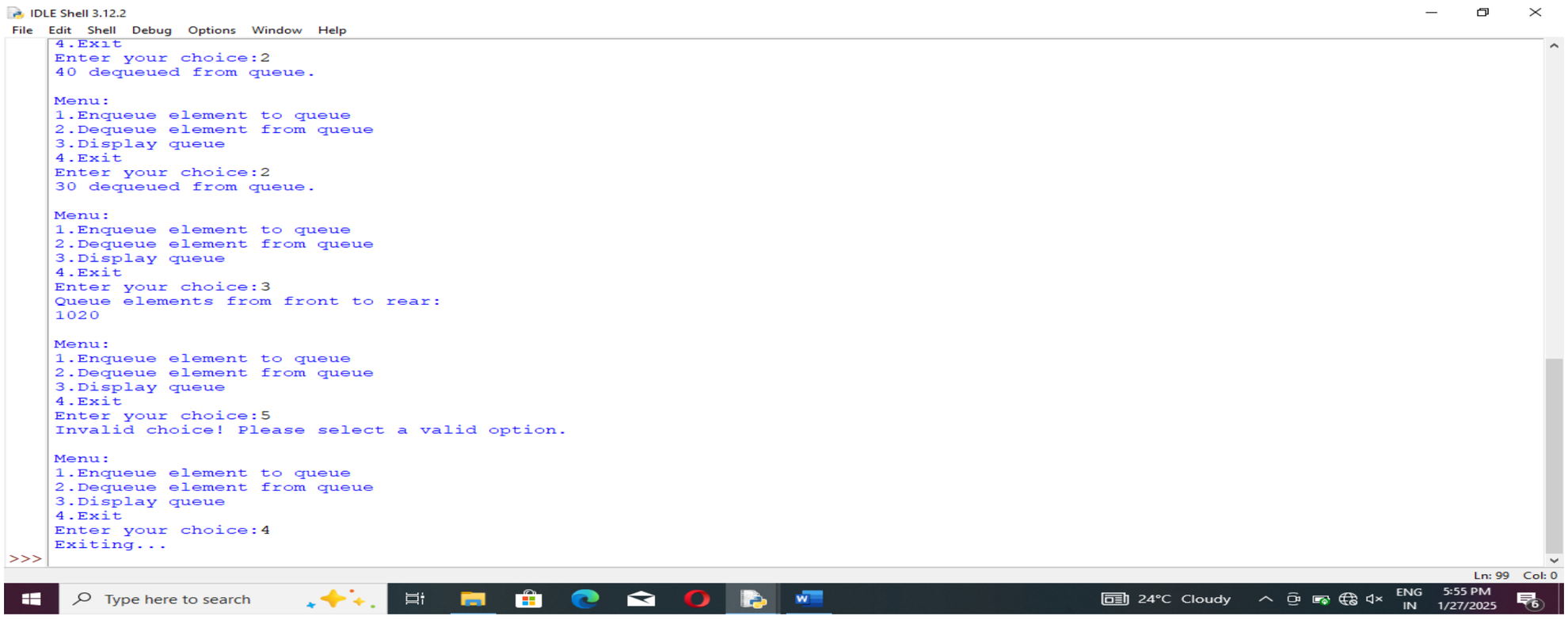
```
while True:
        print("\nMenu:")
        print("1.Enqueue element to queue")
        print("2.Dequeue element from queue")
        print("3.Display queue")
        print("4.Exit")
        choice = int(input("Enter your choice:"))
        if choice == 1:
            item=int(input("Enter the value to enqueue into the queue:"))
            q.enqueue(item)
    elif choice == 2:
        q.dequeue()
    elif choice == 3:
        q.display()
    elif choice == 4:
        print("Exiting...")
        break
    else:
        print("Invalid choice! Please select a valid option.")
```

OUTPUT:

```
IDLE Shell 3.12.2
File  Edit  Shell  Debug  Options  Window  Help
Python 3.12.2 (tags/v3.12.2:6abddd9, Feb  6 2024, 21:26:36) [MSC v.1937 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/MANI PRIYA/Desktop/DSUP practice/Ex4.py

Menu:
1.Enqueue element to queue
2.Dequeue element from queue
3.Display queue
4.Exit
Enter your choice:3
Queue is empty!

Menu:
1.Enqueue element to queue
2.Dequeue element from queue
3.Display queue
4.Exit
Enter your choice:1
Enter the value to enqueue into the queue:10
10 enqueued to queue.

Menu:
1.Enqueue element to queue
2.Dequeue element from queue
3.Display queue
4.Exit
Enter your choice:1
Enter the value to enqueue into the queue:20
20 enqueued to queue.

Menu:
1.Enqueue element to queue
2.Dequeue element from queue
3.Display queue
4.Exit
Enter your choice:1
```

```
Ln: 99  Col: 0
Type here to search   24°C Cloudy   ENG IN   5:53 PM 1/27/2025
IDLE Shell 3.12.2
File  Edit  Shell  Debug  Options  Window  Help
Enter the value to enqueue into the queue:30
30 enqueued to queue.

Menu:
1.Enqueue element to queue
2.Dequeue element from queue
3.Display queue
4.Exit
Enter your choice:1
Enter the value to enqueue into the queue:40
40 enqueued to queue.

Menu:
1.Enqueue element to queue
2.Dequeue element from queue
3.Display queue
4.Exit
Enter your choice:3
Queue elements from front to rear:
10203040

Menu:
1.Enqueue element to queue
2.Dequeue element from queue
3.Display queue
4.Exit
Enter your choice:2
40 dequeued from queue.

Menu:
1.Enqueue element to queue
2.Dequeue element from queue
3.Display queue
4.Exit
Enter your choice:2
30 dequeued from queue.
```

```
Ln: 74  Col: 0
Type here to search   24°C Cloudy   ENG IN   5:54 PM 1/27/2025
```



EX.NO:5

To write a Python program to implement a Binary Tree and perform

Preorder Traversal.

ALGORITHM :

1. Start the program.

2. Create a Node class with:

a) data

b) left child

c) right child

3. Create a function preorder(node):

- If node is not null:

- Visit (print) the node’ s data

- Recursively traverse the left subtree

- Recursively traverse the right subtree

4. Call the preorder function and display output.

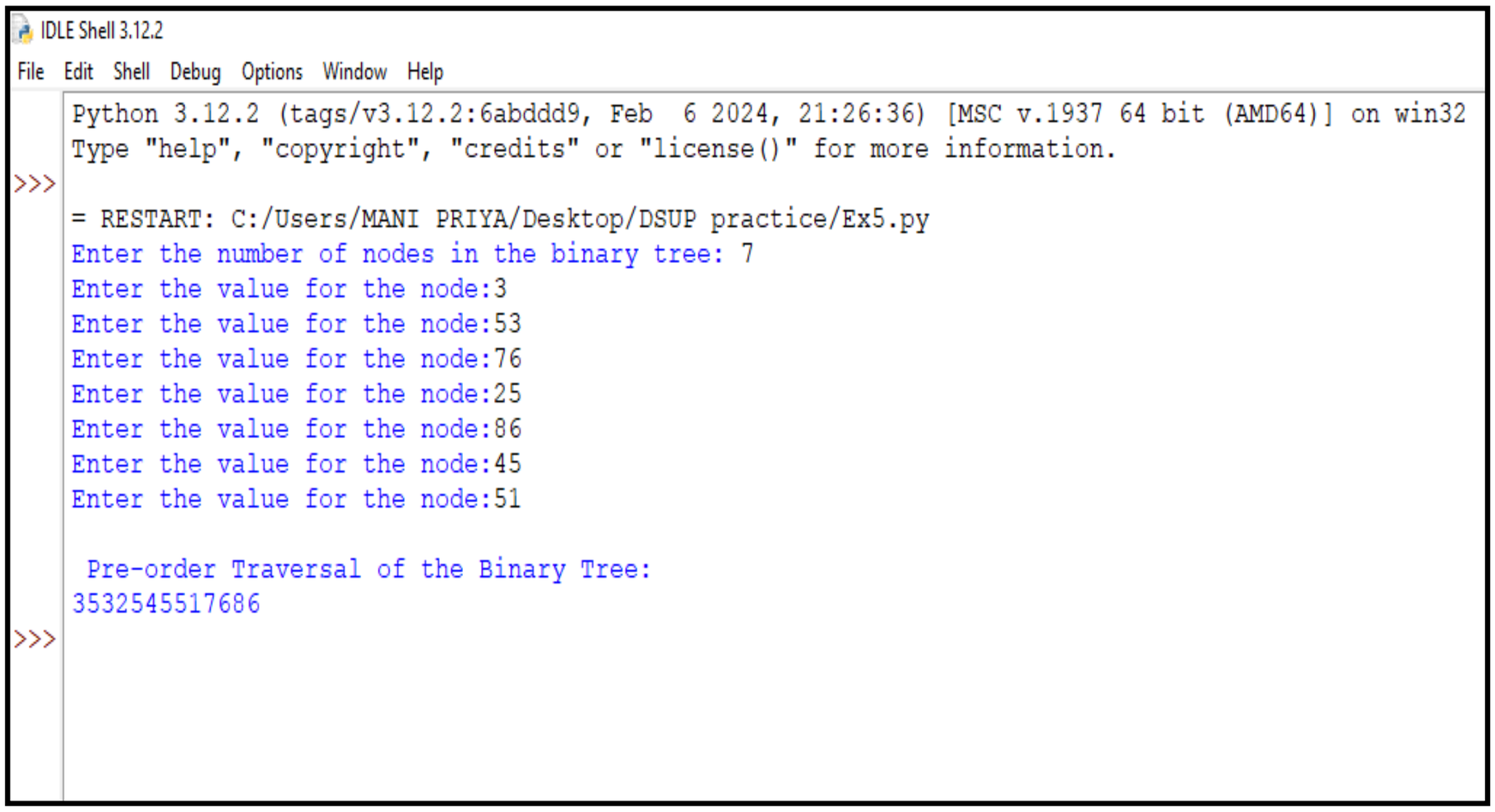
5. Display the result.

6. End the program.

PROGRAM :

```
class TreeNode:
    def __init__(self,value):
        self.value=value
        self.left=None
        self.right=None
def insert_tree(root,value):
    if root is None:
        return TreeNode(value)
    if value &lt; root.value:
        root.left=insert_tree(root.left, value)
    else:
        root.right=insert_tree(root.right, value)
    return root
def preorder(root):
    if root:
        print(root.value,end='')
        preorder(root.left)
        preorder(root.right)
root=None
n=int(input("Enter the number of nodes in the binary tree:"))
for i in range(n):
    value=int(input("Enter the value for the node:"))
    root=insert_tree(root,value)
print("\n Pre-order Traversal of the Binary Tree:")
preorder(root)
```

OUTPUT:



EX.NO:6

AIM :

To write a Python program to sort a list of elements using the Bubble

Sort algorithm.

ALGORITHM :

1. Start the program.

2. Take a list of elements.

3. Repeat for each element in the list:

4. Compare each pair of adjacent elements.

5. If the first is greater than the second, swap them.

6. Continue until the entire list is sorted.

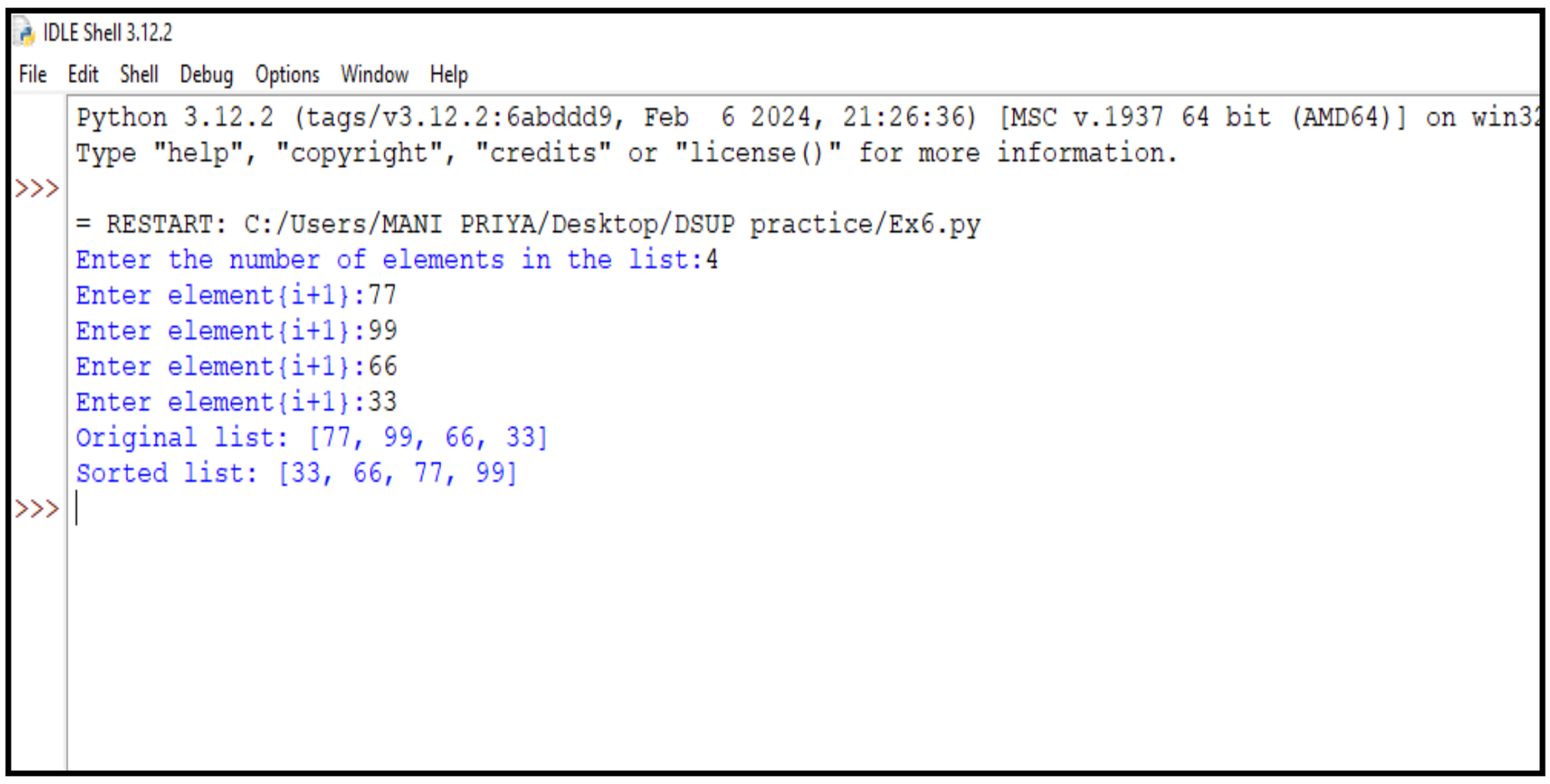
7. Print the sorted list.

8. Stop the program.

PROGRAM :

```
def bubble_sort(a):
    n=len(a)
    for i in range(n):
        for j in range(0, n - i - 1):
            if a[j]&gt;a[j+1]:
                a[j], a[j+1] = a[j+1], a[j]
n=int(input("Enter the number of elements in the list:"))
a=[]
for i in range(n):
    element=int(input("Enter element{i+1}:"))
    a.append(element)
    print("Original list:",a)
bubble_sort(a)
print("Sorted list:",a)
```

OUTPUT:



EX.NO:7

AIM :

To write a Python program to search for an element in a list using the

Linear Search technique.

ALGORITHM:

1. Start

2. Create a list of elements

3. Input the element to be searched

4. Traverse the list from the first item to the last

5. If the current element matches the search element:

- Print “Element found”

- Stop the search

6. Display the result.

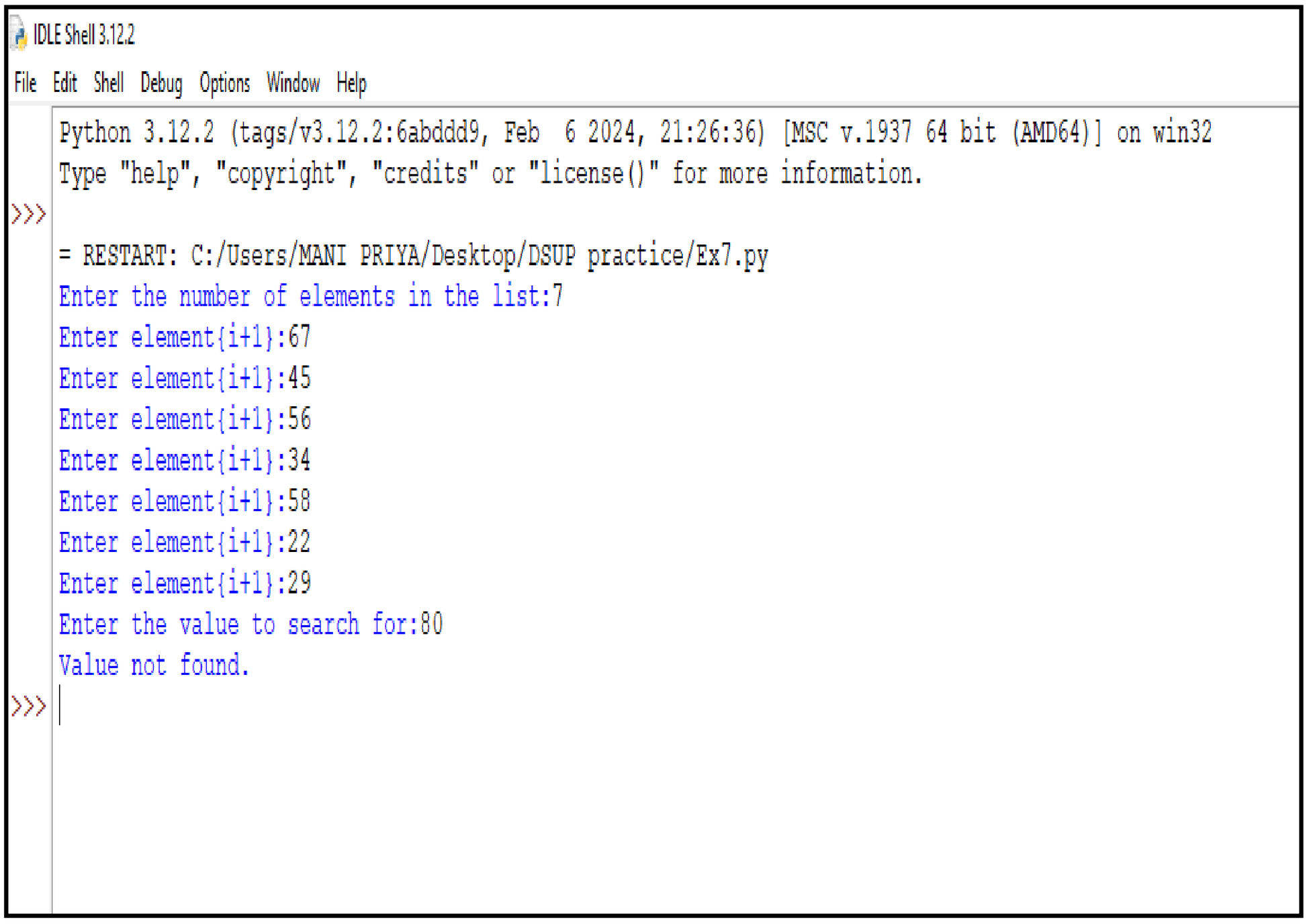
7. Stop the program.

PROGRAM :

```
def linear_search(a,x):
        for i in range(len(a)):
                if a[i] == x:
                        return i
        return -1
n=int(input("Enter the number of elements in the list:"))


ll=[]
for i in range(n):
        element=int(input("Enter element{i+1}:"))
        ll.append(element)
```

```
value=int(input("Enter the value to search for:"))
pos=linear_search(ll, value)
if pos != -1:
        print("Value found at position{pos}.")
else:
        print("Value not found.")
```

OUTPUT:



EX.NO:8

AIM :

To write a Python program to implement Binary Search on a sorted list.

ALGORITHM :

1. Start the program.

2. Start with two pointers:

- low = 0

- high = n - 1

3. Repeat until low <= high:

- Find middle index: mid = (low + high) // 2

- If element at mid equals the key → element found.

- If key is smaller → search left half (high = mid - 1)

- If key is larger → search right half (low = mid + 1)

4. If the loop ends without finding → element not found.

5. Display the result.

6. Stop the program.

PROGRAM :

```
def binary_search(a,x):
    left=0
    right=len(a)-1
    while left&amp;lt;=right:
        mid=(left+right)//2
        midvalue=a[mid]
        if midvalue == x:
            return mid
        elif midvalue &amp;lt; x:
            left=mid + 1
        else:
            right=mid - 1
    return -1
n = int(input("Enter the number of elements in the sorted list:"))
```

```
sortlist=[]
for i in range(n):
    element=int(input("Enter element{i+1}:"))
    sortlist.append(element)
x=int(input("Enter the target value to search for:"))

pos=binary_search(sortlist,x)
if pos!=-1:
    print("Value found at index{pos}.")
else:
    print("Value not found.")
```

OUTPUT: