|  |
| --- |
| Experiment No. 7 |
| Program for data structure using built in function for link list, stack and queues |
| Date of Performance: 11/03/2024 |
| Date of Submission: 18/03/2024 |

**Experiment No. 7**

**Title:** Program for data structure using built in function for link list, stack and queues

**Aim:** To study and implement data structure using built in function for link list, stack and queues

**Objective:** To introduce data structures in python

**Theory:**

Stacks -the simplest of all data structures, but also the most important. A stack is a collection of objects that are inserted and removed using the LIFO principle. LIFO stands for “Last In First Out”. Because of the way stacks are structured, the last item added is the first to be removed, and vice-versa: the first item added is the last to be removed.

Queues – essentially a modified stack. It is a collection of objects that are inserted and removed according to the FIFO (First In First Out) principle. Queues are analogous to a line at the grocery store: people are added to the line from the back, and the first in line is the first that gets checked out – BOOM, FIFO!

Linked Lists

The Stack and Queue representations I just shared with you employ the python-based list to store their elements. A python list is nothing more than a dynamic array, which has some disadvantages.

The length of the dynamic array may be longer than the number of elements it stores, taking up precious free space.

Insertion and deletion from arrays are expensive since you must move the items next to them over

Using Linked Lists to implement a stack and a queue (instead of a dynamic array) solve both of these issues; addition and removal from both of these data structures (when implemented with a linked list) can be accomplished in constant O(1) time. This is a HUGE advantage when dealing with lists of millions of items.

Linked Lists – comprised of ‘Nodes’. Each node stores a piece of data and a reference to its next and/or previous node. This builds a linear sequence of nodes. All Linked Lists store a head, which is a reference to the first node. Some Linked Lists also store a tail, a reference to the last node in the list.

**1] Stack Program:**

class Stack:

def \_\_init\_\_(self):

self.stack = []

def push(self, item):

self.stack.append(item)

def pop(self):

if len(self.stack) == 0:

return "Stack is empty"

else:

return self.stack.pop()

def peek(self):

if len(self.stack) == 0:

return "Stack is empty"

else:

return self.stack[-1]

def display(self):

if len(self.stack) == 0:

return "Stack is empty"

else:

return self.stack

# Initialize the stack

stack = Stack()

print("\nStack!!!\n")

while True:

print("\nOptions:")

print("1. Push")

print("2. Pop")

print("3. Peek")

print("4. Stack all elements")

print("5. Exit")

choice = input("Enter your choice: ")

if choice == '1':

item = input("Enter the element to push: ")

stack.push(item)

print(f"'{item}' pushed into the stack.")

elif choice == '2':

item = stack.pop()

print(f"Popped element: {item}")

elif choice == '3':

item = stack.peek()

print(f"Peeked element: {item}")

elif choice == '4':

elements = stack.display()

print(f"All elements in the stack: {elements}")

elif choice == '5':

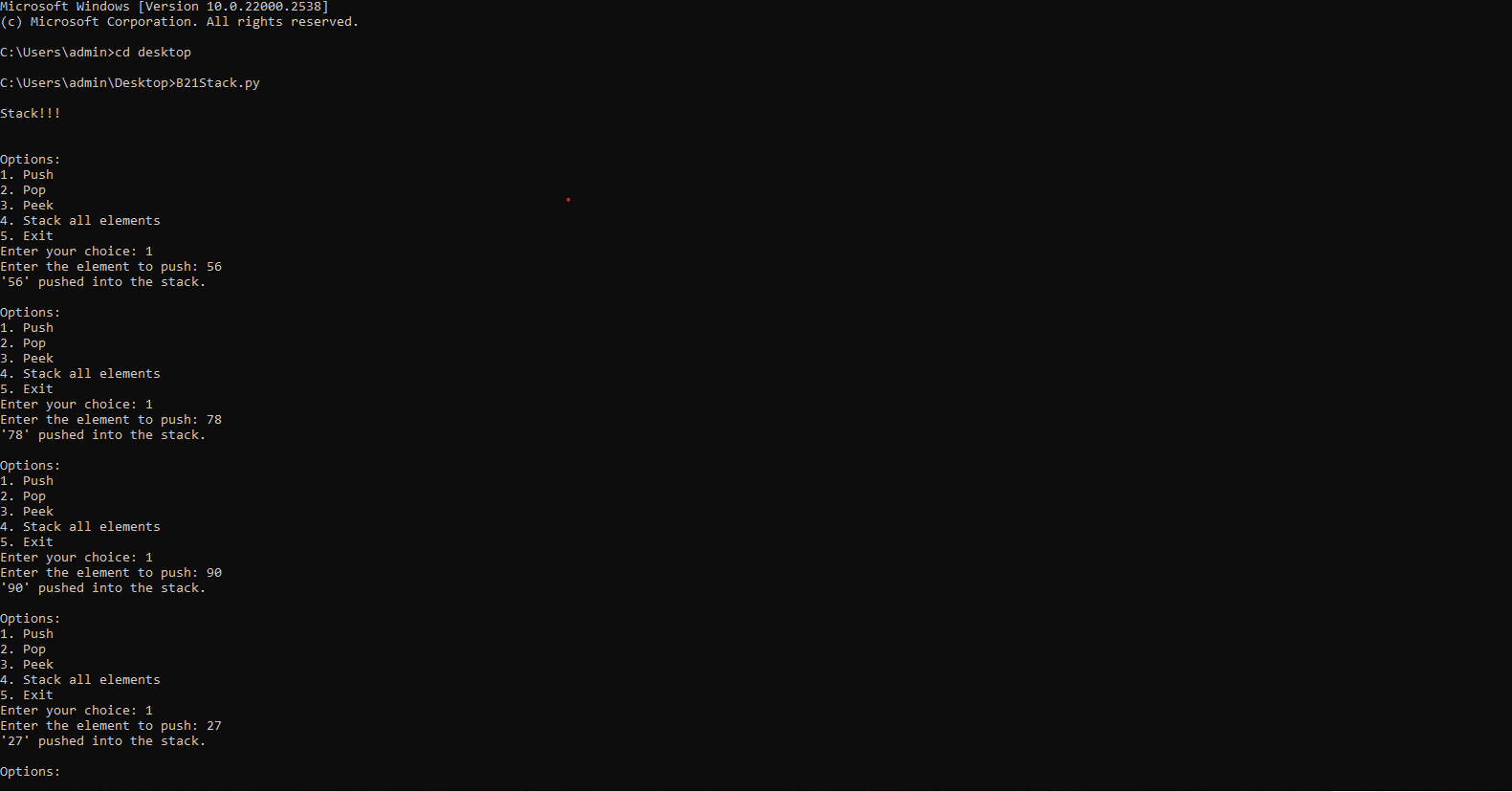
print("Exiting...")

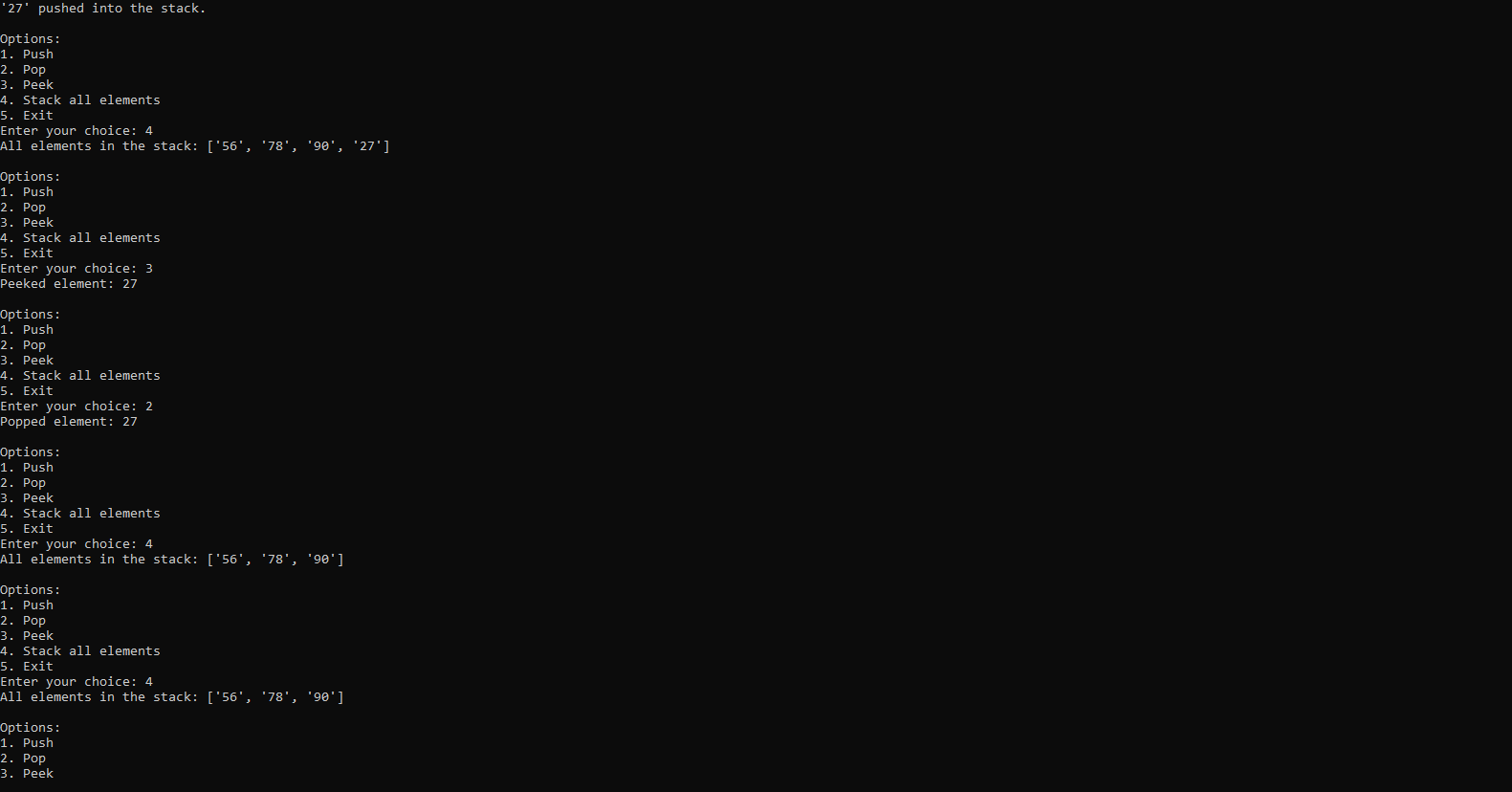
break

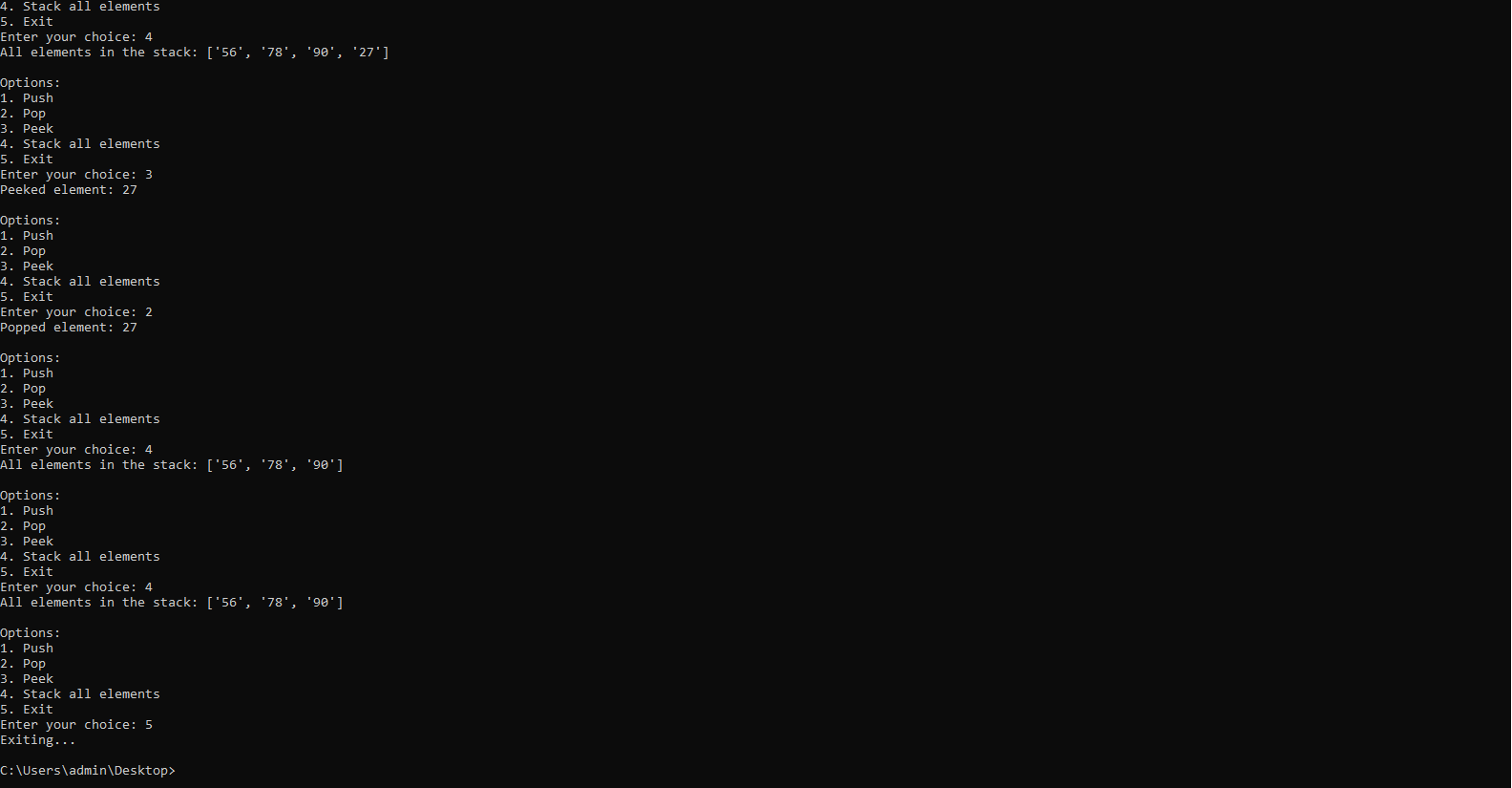
else:

print("Invalid choice! Please enter a valid option.")

**Output:**







**2] Linked List Program:**

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.next = None

class LinkedList:

def \_\_init\_\_(self):

self.head = None

def insertAtBegin(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

else:

new\_node.next = self.head

self.head = new\_node

def insertAtIndex(self, data, index):

new\_node = Node(data)

if index == 0:

self.insertAtBegin(data)

else:

current\_node = self.head

position = 0

while current\_node is not None and position + 1 != index:

position += 1

current\_node = current\_node.next

if current\_node is not None:

new\_node.next = current\_node.next

current\_node.next = new\_node

else:

print("Index not present")

def insertAtEnd(self, data):

new\_node = Node(data)

if self.head is None:

self.head = new\_node

else:

current\_node = self.head

while current\_node.next:

current\_node = current\_node.next

current\_node.next = new\_node

def updateNode(self, val, index):

current\_node = self.head

position = 0

while current\_node is not None and position != index:

position += 1

current\_node = current\_node.next

if current\_node is not None:

current\_node.data = val

else:

print("Index not present")

def remove\_first\_node(self):

if self.head is not None:

self.head = self.head.next

def remove\_last\_node(self):

if self.head is None:

return

current\_node = self.head

while current\_node.next and current\_node.next.next:

current\_node = current\_node.next

current\_node.next = None

def remove\_at\_index(self, index):

if self.head is None:

return

if index == 0:

self.remove\_first\_node()

else:

current\_node = self.head

position = 0

while current\_node is not None and position + 1 != index:

position += 1

current\_node = current\_node.next

if current\_node is not None:

current\_node.next = current\_node.next.next

else:

print("Index not present")

def remove\_node(self, data):

current\_node = self.head

if current\_node.data == data:

self.remove\_first\_node()

return

while current\_node.next is not None and current\_node.next.data != data:

current\_node = current\_node.next

if current\_node.next is not None:

current\_node.next = current\_node.next.next

def sizeOfLL(self):

size = 0

current\_node = self.head

while current\_node:

size += 1

current\_node = current\_node.next

return size

def printLL(self):

elements = []

current\_node = self.head

while current\_node:

elements.append(current\_node.data)

current\_node = current\_node.next

print(elements)

llist = LinkedList()

while True:

print("\nOptions:")

print("1. Insert at Beginning")

print("2. Insert at End")

print("3. Insert at Index")

print("4. Update Node at Index")

print("5. Remove First Node")

print("6. Remove Last Node")

print("7. Remove Node at Index")

print("8. Remove Node by Value")

print("9. Print Linked List")

print("10. Size of Linked List")

print("11. Exit")

choice = input("Enter your choice: ")

if choice == '1':

data = input("Enter the data to insert at beginning: ")

llist.insertAtBegin(data)

elif choice == '2':

data = input("Enter the data to insert at end: ")

llist.insertAtEnd(data)

elif choice == '3':

data = input("Enter the data to insert: ")

index = int(input("Enter the index to insert at: "))

llist.insertAtIndex(data, index)

elif choice == '4':

index = int(input("Enter the index of the node to update: "))

data = input("Enter the new data: ")

llist.updateNode(data, index)

elif choice == '5':

llist.remove\_first\_node()

elif choice == '6':

llist.remove\_last\_node()

elif choice == '7':

index = int(input("Enter the index of the node to remove: "))

llist.remove\_at\_index(index)

elif choice == '8':

data = input("Enter the data of the node to remove: ")

llist.remove\_node(data)

elif choice == '9':

print("Linked List:")

llist.printLL()

elif choice == '10':

print("Size of Linked List:", llist.sizeOfLL())

elif choice == '11':

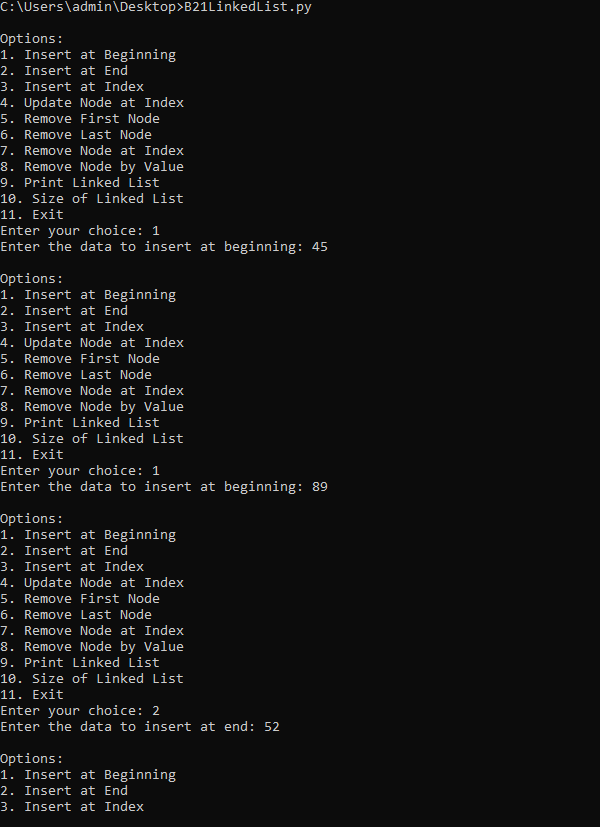
print("Exiting...")

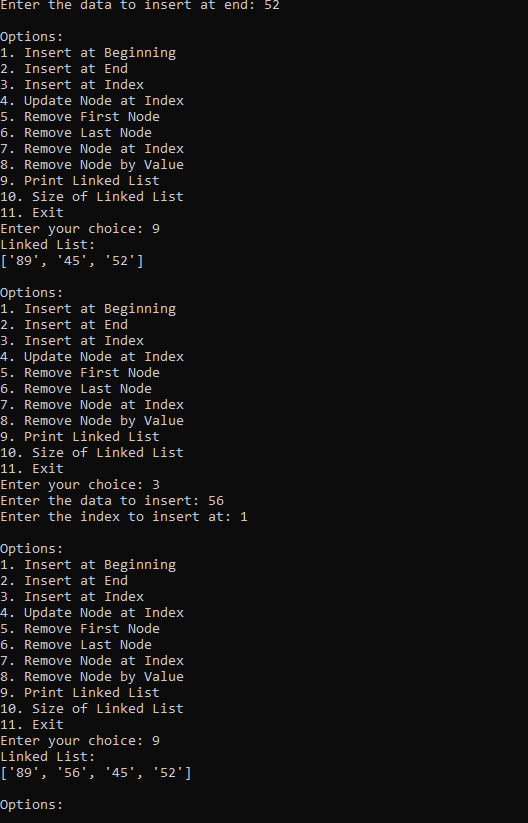
break

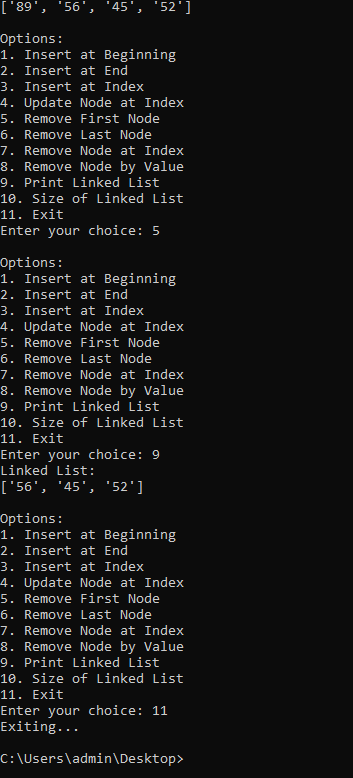
else:

print("Invalid choice! Please enter a valid option.")

**Output:**

****

****

****

**3] Queue Program:**

queue = []

print("\nQUEUE!!!\n")

while True:

print("\nOptions:")

print("1. Enqueue")

print("2. Dequeue")

print("3. Queue List")

print("4. Exit")

choice = input("Enter your choice: ")

if choice == '1':

element = input("Enter the element to enqueue: ")

queue.append(element)

print(f"'{element}' enqueued into the queue.")

elif choice == '2':

if len(queue) == 0:

print("Queue is empty. Cannot remove from an empty queue.")

else:

element = queue.pop(0)

print(f"Element '{element}' dequeued from queue.")

elif choice == '3':

print("Queue elements:", queue)

elif choice == '4':

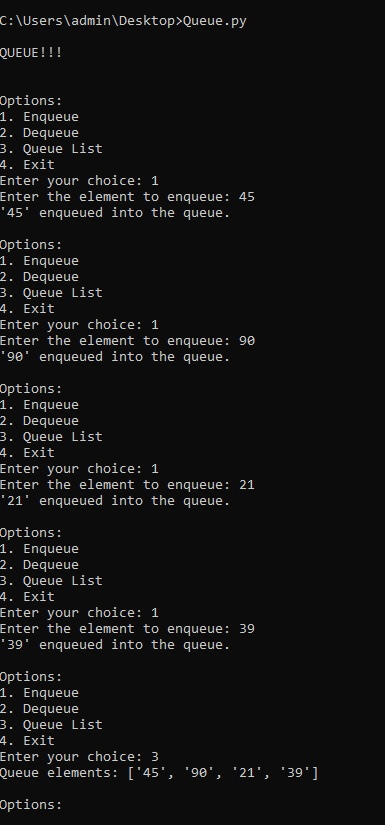
print("Exiting...")

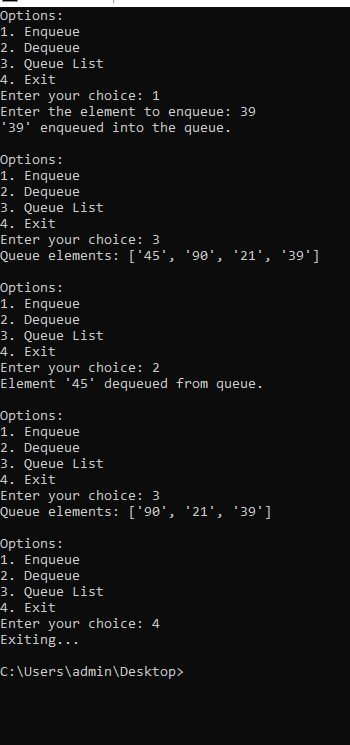
break

else:

print("Invalid choice! Please enter a valid option.")

**Output:**





**Conclusion:**

Through this experiment, various fundamental data structures such as stacks, queues, and linked lists were explored and implemented using built-in functions in Python. These structures follow distinct principles - LIFO for stacks, FIFO for queues, and node-based sequencing for linked lists, showcasing their diverse applications. By understanding and utilizing these structures, efficient data organization and manipulation can be achieved, enhancing programming proficiency and problem-solving skills in Python.