9

"""

Write a program for creating GUI with python containing widgets such as labels, textbox, radio,

checkboxes, and custom dialog boxes etc.

Hanif 231P044 / 01

"""

```
class Stack:
  def __init__(self):
    self.stack = []
  def push(self, item):
    self.stack.append(item)
    print(f"{item} pushed into stack.")
  def pop(self):
    if not self.is_empty():
      print(f"Popped item: {self.stack.pop()}")
    else:
      print("Stack is empty!")
  def display(self):
    if self.is_empty():
      print("Stack is empty!")
    else:
      print("Stack elements:", self.stack)
  def is_empty(self):
    return len(self.stack) == 0
class Queue:
  def __init__(self):
    self.queue = []
```

```
    def enqueue(self, item):
        self.queue.append(item)
        print(f"{item} enqueued into queue.")
    def dequeue(self):
        if not self.is_empty():
            print(f"Dequeued item: {self.queue.pop(0)}")
        else:
            print("Queue is empty!")
    def display(self):
        if self.is_empty():
            print("Queue is empty!")
        else:
            print("Queue elements:", self.queue)
    def is_empty(self):
        return len(self.queue) == 0
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
class LinkedList:
    def __init__(self):
        self.head = None
    def insert(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            temp = self.head
```

```
        while temp.next:
            temp = temp.next
        temp.next = new_node
    print(f"{data} inserted into linked list.")
def delete(self, key):
    if self.head is None:
        print("Linked List is empty!")
        return
    if self.head.data == key:
        self.head = self.head.next
        print(f"Deleted node with value {key}.")
        return
    temp = self.head
    prev = None
    while temp and temp.data != key:
        prev = temp
        temp = temp.next
    if temp is None:
        print("Element not found in linked list!")
        return
    prev.next = temp.next
    print(f"Deleted node with value {key}.")
def display(self):
    if self.head is None:
        print("Linked List is empty!")
        return
    temp = self.head
    print("Linked List elements:", end=" ")
```

```
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
def menu():
    stack = Stack()
    queue = Queue()
    linked_list = LinkedList()
    while True:
        print("\nMenu:")
        print("1. Stack Operations")
        print("2. Queue Operations")
        print("3. Linked List Operations")
        print("4. Exit")
        choice = int(input("Enter your choice: "))
        if choice == 1:
            while True:
                print("\nStack Operations:")
                print("1. Push")
                print("2. Pop")
                print("3. Display")
                print("4. Back to Main Menu")
                op = int(input("Enter operation: "))
                if op == 1:
                    item = input("Enter element to push: ")
                    stack.push(item)
                elif op == 2:
                    stack.pop()
```

```
        elif op == 3:
            stack.display()
        elif op == 4:
            break
        else:
            print("Invalid choice! Try again.")
    elif choice == 2:
        while True:
            print("\nQueue Operations:")
            print("1. Enqueue")
            print("2. Dequeue")
            print("3. Display")
            print("4. Back to Main Menu")
            op = int(input("Enter operation: "))
            if op == 1:
                item = input("Enter element to enqueue: ")
                queue.enqueue(item)
            elif op == 2:
                queue.dequeue()
            elif op == 3:
                queue.display()
            elif op == 4:
                break
            else:
                print("Invalid choice! Try again.")
    elif choice == 3:
        while True:
            print("\nLinked List Operations:")
```

```
            print("1. Insert")
            print("2. Delete")
            print("3. Display")
            print("4. Back to Main Menu")
            op = int(input("Enter operation: "))
            if op == 1:
                item = input("Enter element to insert: ")
                linked_list.insert(item)
            elif op == 2:
                item = input("Enter element to delete: ")
                linked_list.delete(item)
            elif op == 3:
                linked_list.display()
            elif op == 4:
                break
            else:
                print("Invalid choice! Try again.")
        elif choice == 4:
            print("Exiting program. Goodbye!")
            break
        else:
            print("Invalid choice! Try again.")
# Run the menu
menu()
```

OUTPUT:

Menu:

1. Stack Operations

2. Queue Operations

3. Linked List Operations

4. Exit

Enter your choice: 1

Stack Operations:

1. Push

2. Pop

3. Display

4. Back to Main Menu

Enter operation: 1

Enter element to push: 10

10 pushed into stack.

Stack Operations:

1. Push

2. Pop

3. Display

4. Back to Main Menu

Enter operation: 1

Enter element to push: 20

20 pushed into stack.

Stack Operations:

1. Push

2. Pop

3. Display

4. Back to Main Menu

Enter operation: 3

Stack elements: ['10', '20']

Stack Operations:

1. Push

2. Pop

3. Display

4. Back to Main Menu

Enter operation: 2

Popped item: 20

Stack Operations:

1. Push

2. Pop

3. Display

4. Back to Main Menu

Enter operation: 4

Menu:

1. Stack Operations

2. Queue Operations

3. Linked List Operations

4. Exit

Enter your choice: 2

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Back to Main Menu

Enter operation: 1

Enter element to enqueue: 30

30 enqueued into queue.

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Back to Main Menu

Enter operation: 1

Enter element to enqueue: 40

40 enqueued into queue.

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Back to Main Menu

Enter operation: 3

Queue elements: ['30', '40']

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Back to Main Menu

Enter operation: 2

Dequeued item: 30

Queue Operations:

1. Enqueue

2. Dequeue

3. Display

4. Back to Main Menu

Enter operation: 4

Menu:

1. Stack Operations

2. Queue Operations

3. Linked List Operations

4. Exit

Enter your choice: 3

Linked List Operations:

1. Insert

2. Delete

3. Display

4. Back to Main Menu

Enter operation: 1

Enter element to insert: 50

50 inserted into linked list.

Linked List Operations:

1. Insert

2. Delete

3. Display

4. Back to Main Menu

Enter operation: 1

Enter element to insert: 60

60 inserted into linked list.

Linked List Operations:

1. Insert

2. Delete

3. Display

4. Back to Main Menu

Enter operation: 3

Linked List elements: 50 -> 60 -> None

Linked List Operations:

1. Insert

2. Delete

3. Display

4. Back to Main Menu

Enter operation: 2

Enter element to delete: 50

Deleted node with value 50.

Linked List Operations:

1. Insert

2. Delete

3. Display

4. Back to Main Menu

Enter operation: 3

Linked List elements: 60 -> None

Linked List Operations:

1. Insert

2. Delete

3. Display

4. Back to Main Menu

Enter operation: 4

Menu:

1. Stack Operations

2. Queue Operations

3. Linked List Operations

4. Exit

Enter your choice: 4

Exiting program. Goodbye!

9a.

"""

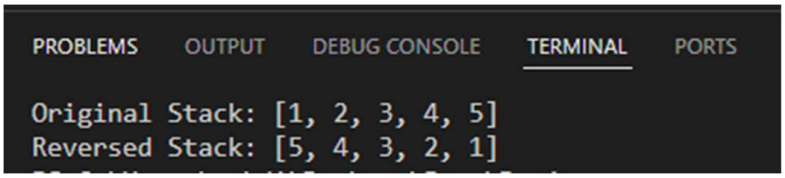
Write a Python Program to Reverse a Stack using Recursion

Hanif 231P044 / 01

"""

```
def insert_at_bottom(stack, item):
    if not stack:
        stack.append(item)
    else:
        temp = stack.pop()
        insert_at_bottom(stack, item)
        stack.append(temp)
def reverse_stack(stack):
    if stack:
        temp = stack.pop()
        reverse_stack(stack)
        insert_at_bottom(stack, temp)
# Example usage
stack = [1,5 ,12 ,14 ,72 ]
print("Original Stack:", stack)
reverse_stack(stack)
print("Reversed Stack:", stack)
```

OUTPUT:



9b.

"""

Write a program to implement circular queue.

Hanif 231P044 / 01

"""

```
class CircularQueue:
    def __init__(self, size):
        self.size = size
        self.queue = [None] * size
        self.front = self.rear = -1

    def enqueue(self, item):
        if (self.rear + 1) % self.size == self.front:
            print("Queue is Full!")
        else:
            if self.front == -1:
                self.front = 0
            self.rear = (self.rear + 1) % self.size
            self.queue[self.rear] = item
            print(f"Inserted {item}")

    def dequeue(self):
        if self.front == -1:
            print("Queue is Empty!")
            return None
        else:
            removed_item = self.queue[self.front]
            if self.front == self.rear:  # Only one element was present
```

```
            self.front = self.rear = -1
        else:
            self.front = (self.front + 1) % self.size
        print(f"Removed {removed_item}")
        return removed_item

    def display(self):
        if self.front == -1:
            print("Queue is Empty!")
        else:
            print("Circular Queue elements:", end=" ")
            i = self.front
            while True:
                print(self.queue[i], end=" ")
                if i == self.rear:
                    break
                i = (i + 1) % self.size
            print()

# Example usage
cq = CircularQueue(5)

cq.enqueue(10)
cq.enqueue(20)
cq.enqueue(30)
cq.enqueue(40)
cq.enqueue(50)  # Queue is full after this
cq.display()
```

cq.dequeue()

cq.dequeue()

cq.display()

cq.enqueue(60)

cq.enqueue(70)

cq.display()

Output: