K. R. Mangalam University

School of Engineering and Technology

Data Structures(ENCS205) Lab File

SUBMITTED BY- **SUBMITTED TO-**

NAME–UDAY KANDPAL Swati Gupta

ROLL NO.-2401420015

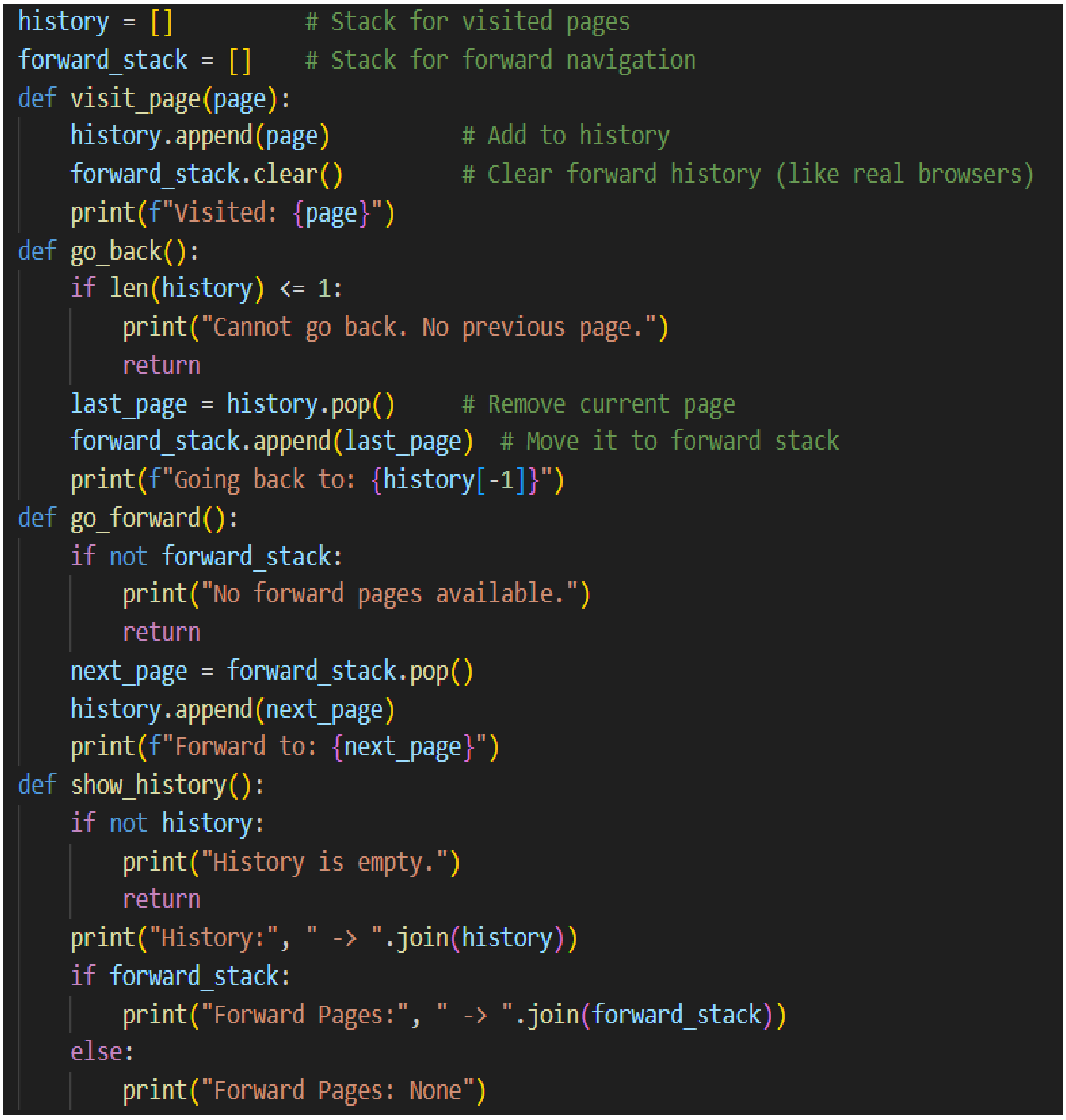
COURSE-BTECH CSE(Data Science)

Semester - 3

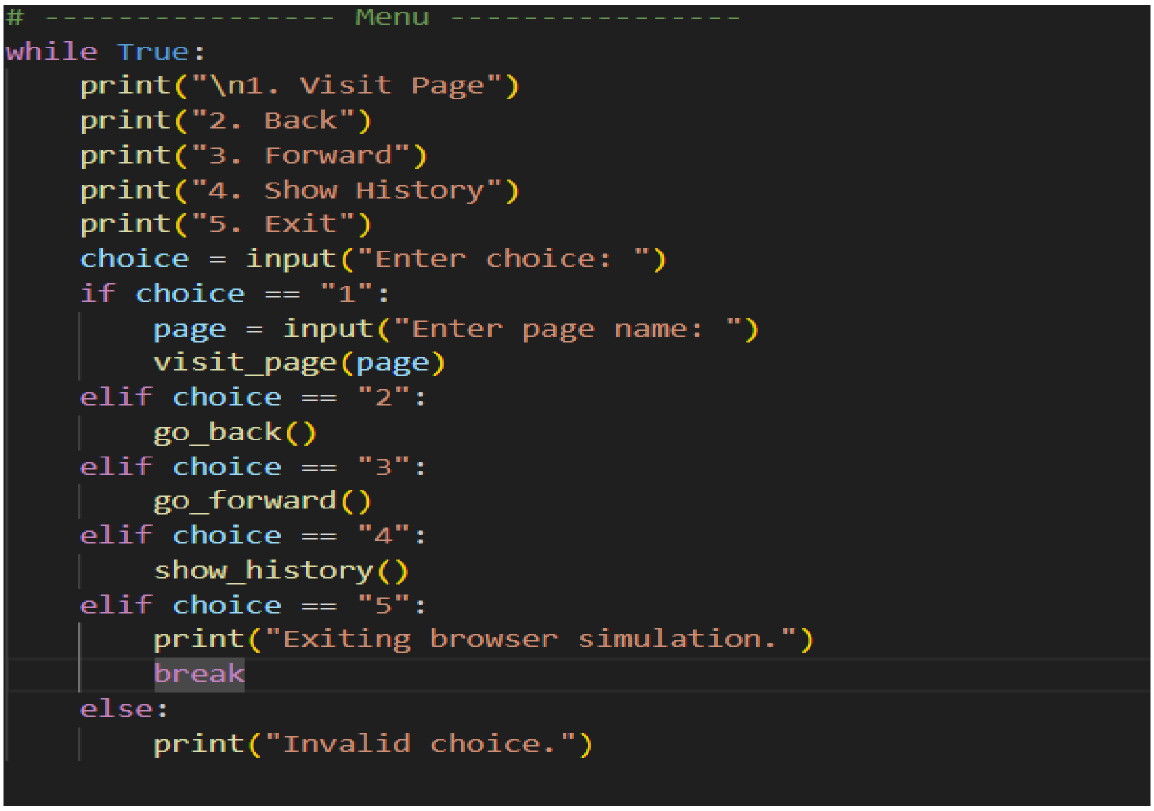
INDEX

1. Browser History Navigation System(Using Stack)

Code:

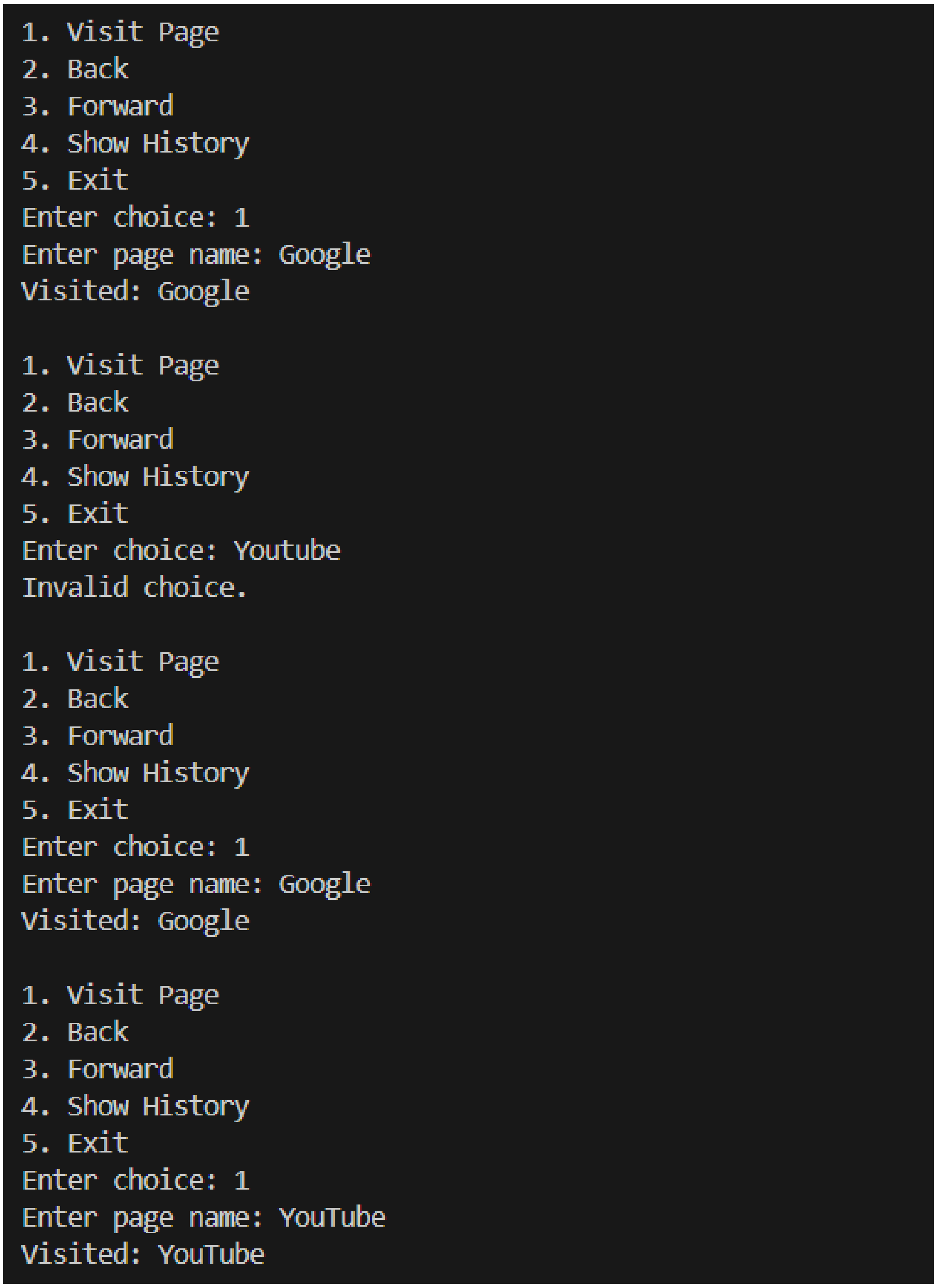


```python
history = []          # Stack for visited pages
forward_stack = []    # Stack for forward navigation
def visit_page(page):
    history.append(page)          # Add to history
    forward_stack.clear()         # Clear forward history (like real browsers)
    print(f"Visited: {page}")
def go_back():
    if len(history) <= 1:
        print("Cannot go back. No previous page.")
        return
    last_page = history.pop()     # Remove current page
    forward_stack.append(last_page)  # Move it to forward stack
    print(f"Going back to: {history[-1]}")
def go_forward():
    if not forward_stack:
        print("No forward pages available.")
        return
    next_page = forward_stack.pop()
    history.append(next_page)
    print(f"Forward to: {next_page}")
def show_history():
    if not history:
        print("History is empty.")
        return
    print("History:", " -> ".join(history))
    if forward_stack:
        print("Forward Pages:", " -> ".join(forward_stack))
    else:
        print("Forward Pages: None")
```

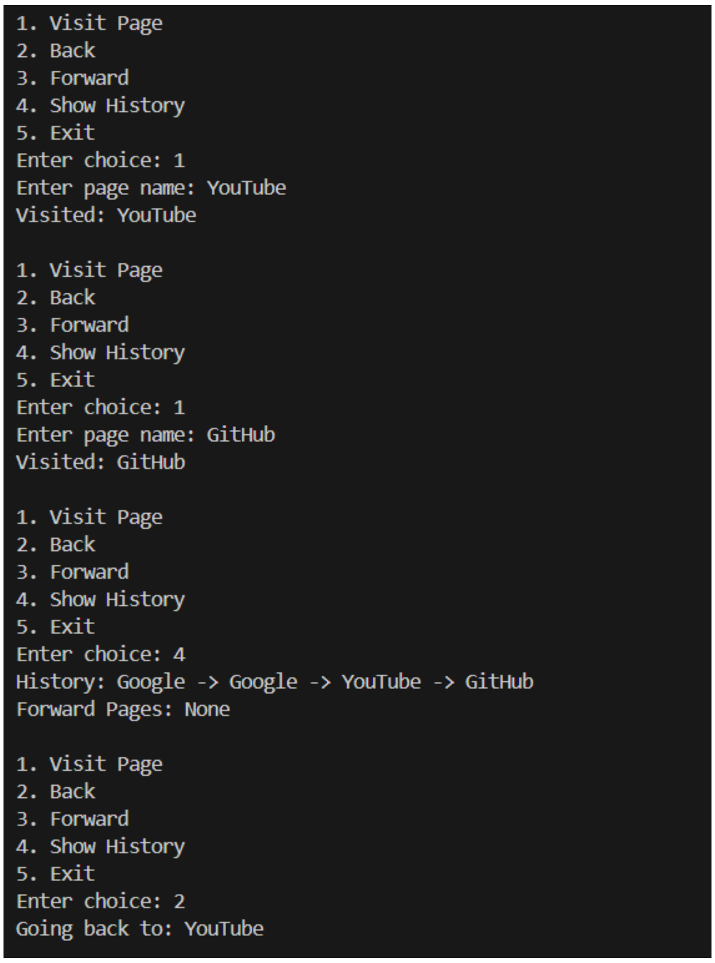


```python
# ---------------- Menu ----------------
while True:
    print("\n1. Visit Page")
    print("2. Back")
    print("3. Forward")
    print("4. Show History")
    print("5. Exit")
    choice = input("Enter choice: ")
    if choice == "1":
        page = input("Enter page name: ")
        visit_page(page)
    elif choice == "2":
        go_back()
    elif choice == "3":
        go_forward()
    elif choice == "4":
        show_history()
    elif choice == "5":
        print("Exiting browser simulation.")
        break
    else:
        print("Invalid choice.")
```

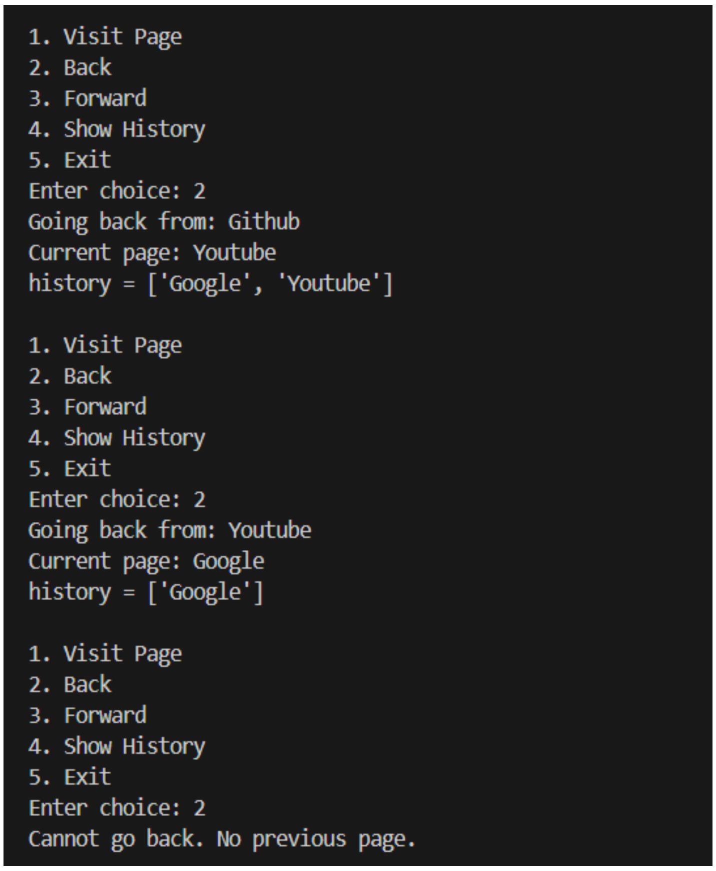
Output:



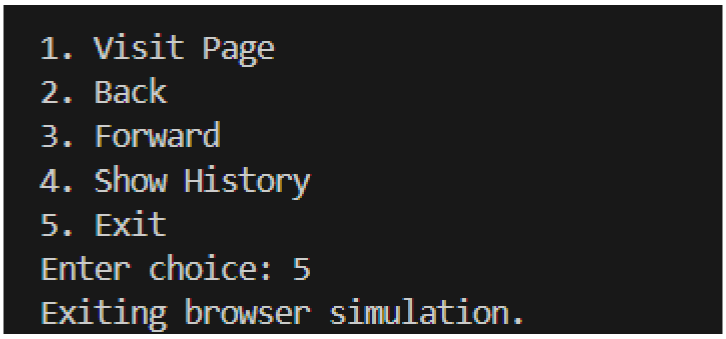
```
1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: 1
Enter page name: Google
Visited: Google

1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: Youtube
Invalid choice.

1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: 1
Enter page name: Google
Visited: Google

1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: 1
Enter page name: YouTube
Visited: YouTube
```



```
1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: 1
Enter page name: YouTube
Visited: YouTube

1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: 1
Enter page name: GitHub
Visited: GitHub

1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: 4
History: Google -> Google -> YouTube -> GitHub
Forward Pages: None

1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: 2
Going back to: YouTube
```



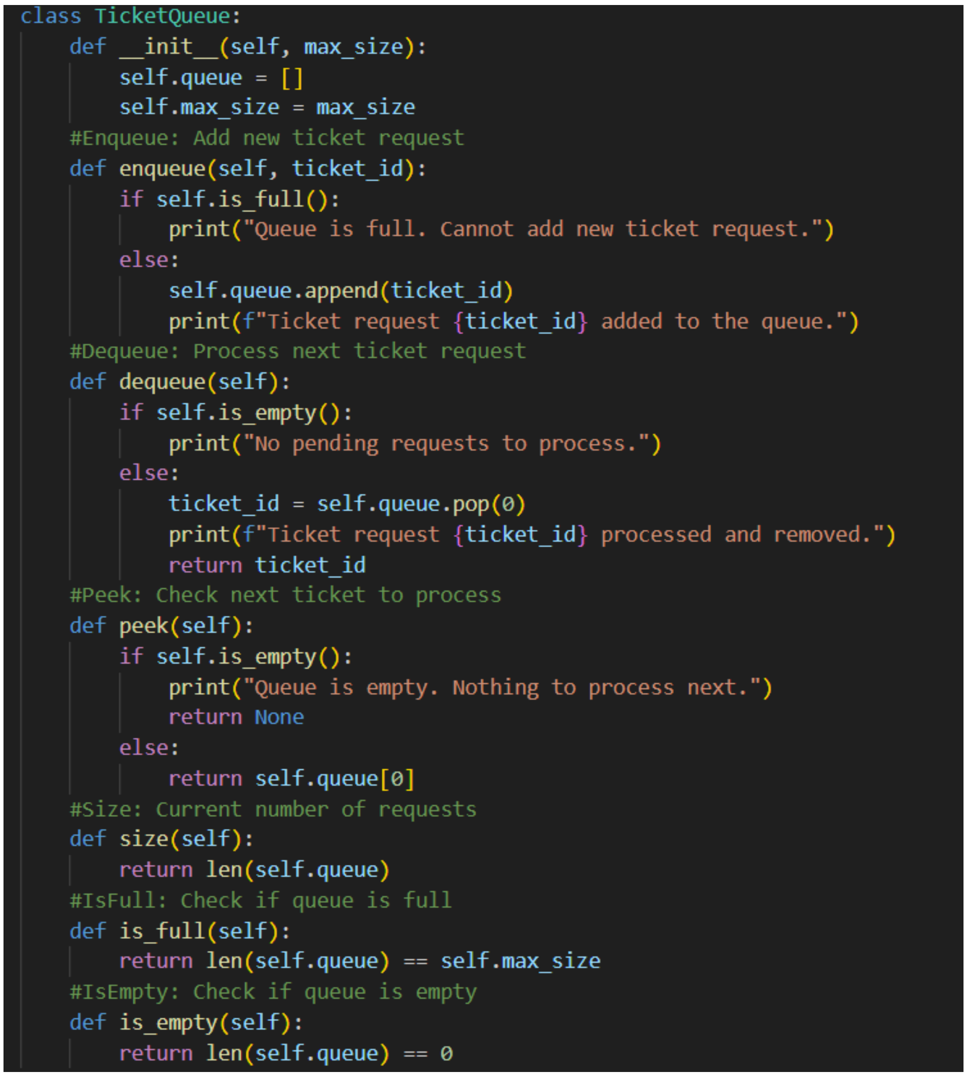
```
1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: 2
Going back from: Github
Current page: Youtube
history = ['Google', 'Youtube']

1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: 2
Going back from: Youtube
Current page: Google
history = ['Google']

1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: 2
Cannot go back. No previous page.
```



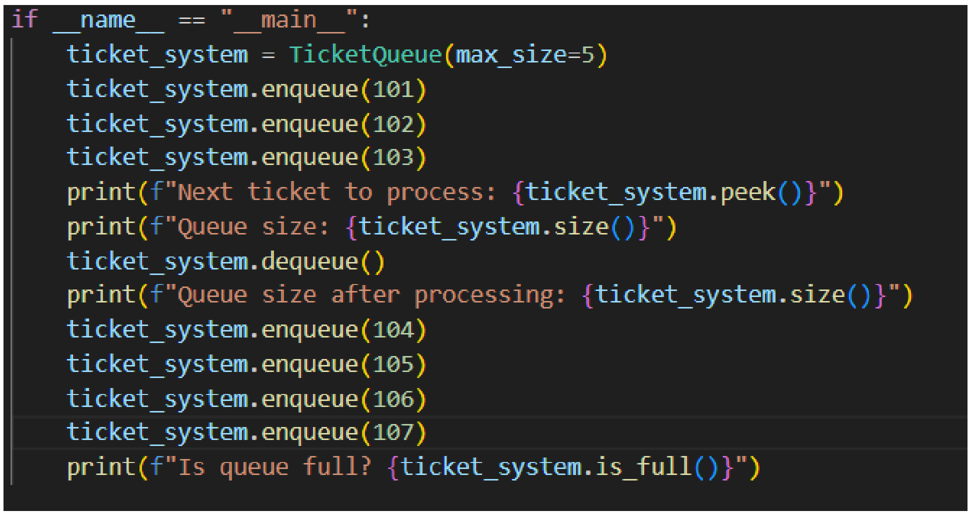
```
1. Visit Page
2. Back
3. Forward
4. Show History
5. Exit
Enter choice: 5
Exiting browser simulation.
```

2. Ticketing System Using Queue(Linear Queue Implementation)

Code:

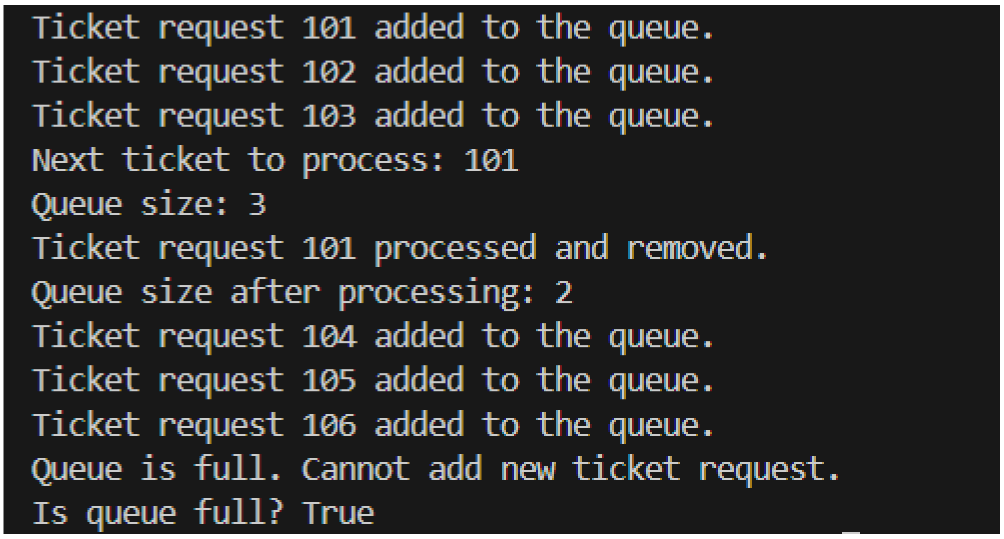


```python
class TicketQueue:
    def __init__(self, max_size):
        self.queue = []
        self.max_size = max_size
    #Enqueue: Add new ticket request
    def enqueue(self, ticket_id):
        if self.is_full():
            print("Queue is full. Cannot add new ticket request.")
        else:
            self.queue.append(ticket_id)
            print(f"Ticket request {ticket_id} added to the queue.")
    #Dequeue: Process next ticket request
    def dequeue(self):
        if self.is_empty():
            print("No pending requests to process.")
        else:
            ticket_id = self.queue.pop(0)
            print(f"Ticket request {ticket_id} processed and removed.")
            return ticket_id
    #Peek: Check next ticket to process
    def peek(self):
        if self.is_empty():
            print("Queue is empty. Nothing to process next.")
            return None
        else:
            return self.queue[0]
    #Size: Current number of requests
    def size(self):
        return len(self.queue)
    #IsFull: Check if queue is full
    def is_full(self):
        return len(self.queue) == self.max_size
    #IsEmpty: Check if queue is empty
    def is_empty(self):
        return len(self.queue) == 0
```



```python
if __name__ == "__main__":
    ticket_system = TicketQueue(max_size=5)
    ticket_system.enqueue(101)
    ticket_system.enqueue(102)
    ticket_system.enqueue(103)
    print(f"Next ticket to process: {ticket_system.peek()}")
    print(f"Queue size: {ticket_system.size()}")
    ticket_system.dequeue()
    print(f"Queue size after processing: {ticket_system.size()}")
    ticket_system.enqueue(104)
    ticket_system.enqueue(105)
    ticket_system.enqueue(106)
    ticket_system.enqueue(107)
    print(f"Is queue full? {ticket_system.is_full()}")
```

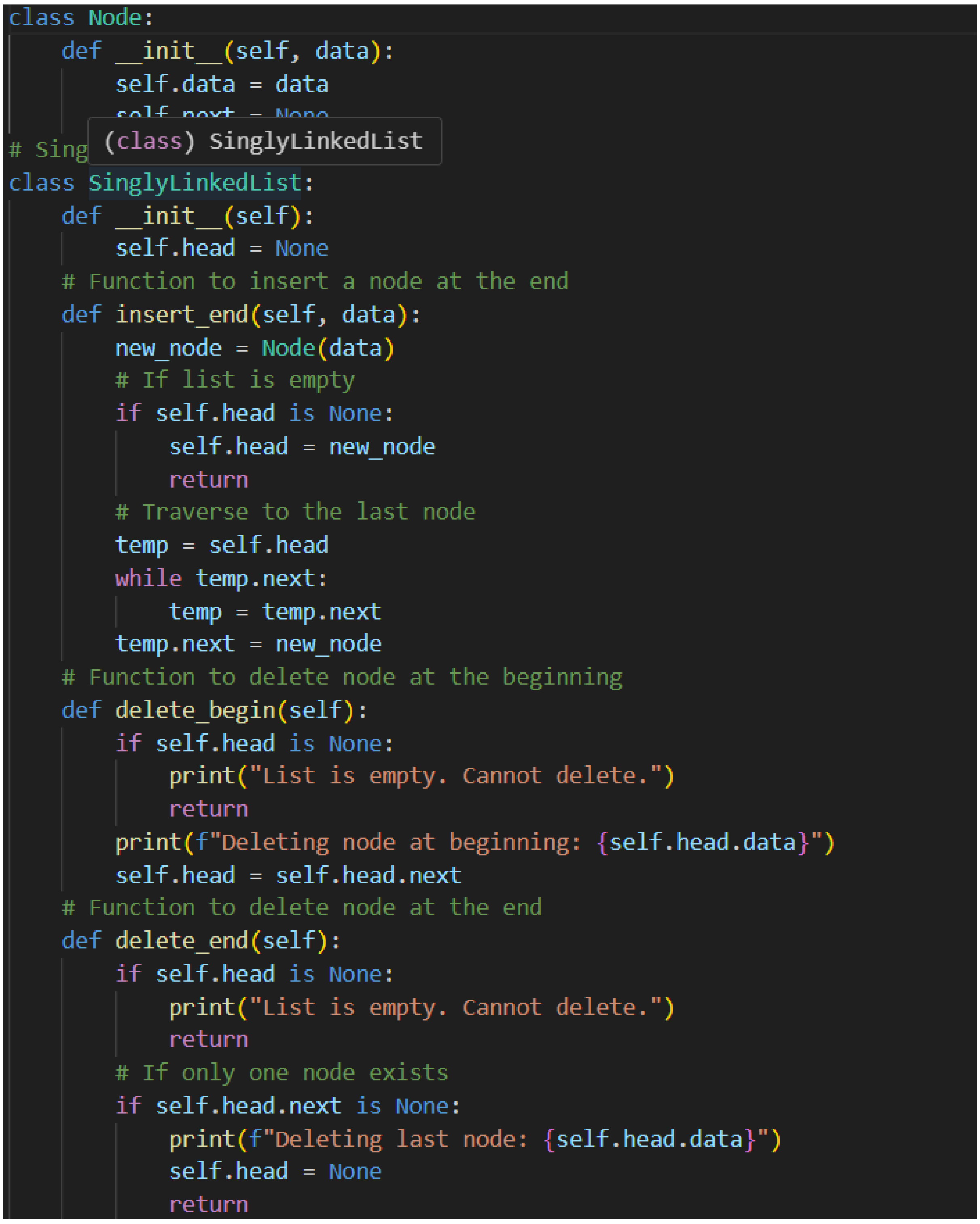
Output:



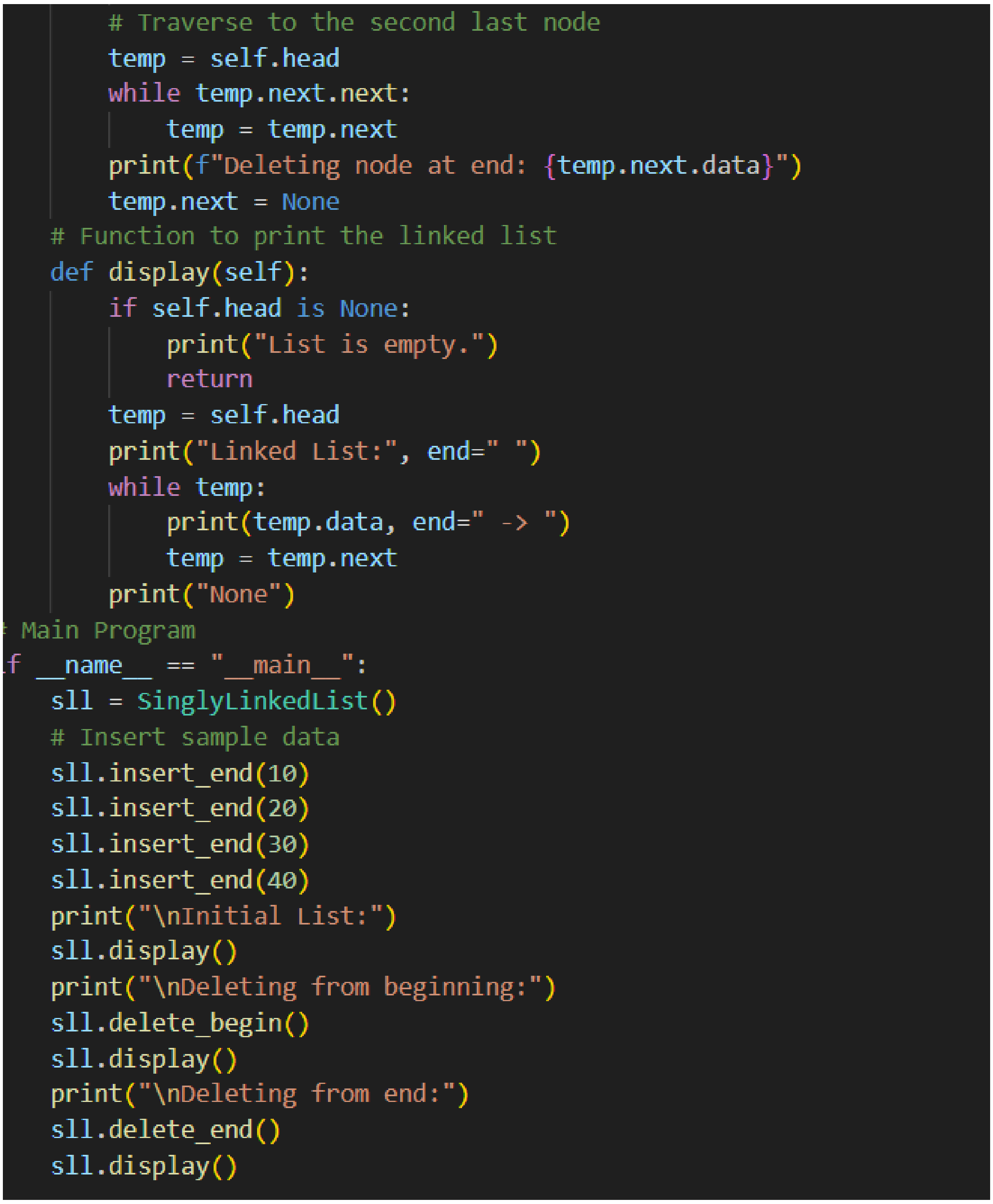
```
Ticket request 101 added to the queue.
Ticket request 102 added to the queue.
Ticket request 103 added to the queue.
Next ticket to process: 101
Queue size: 3
Ticket request 101 processed and removed.
Queue size after processing: 2
Ticket request 104 added to the queue.
Ticket request 105 added to the queue.
Ticket request 106 added to the queue.
Queue is full. Cannot add new ticket request.
Is queue full? True
```

3. Singly Linked List Operations(Insertion ,Deletion ,Search ,Display)

Code:

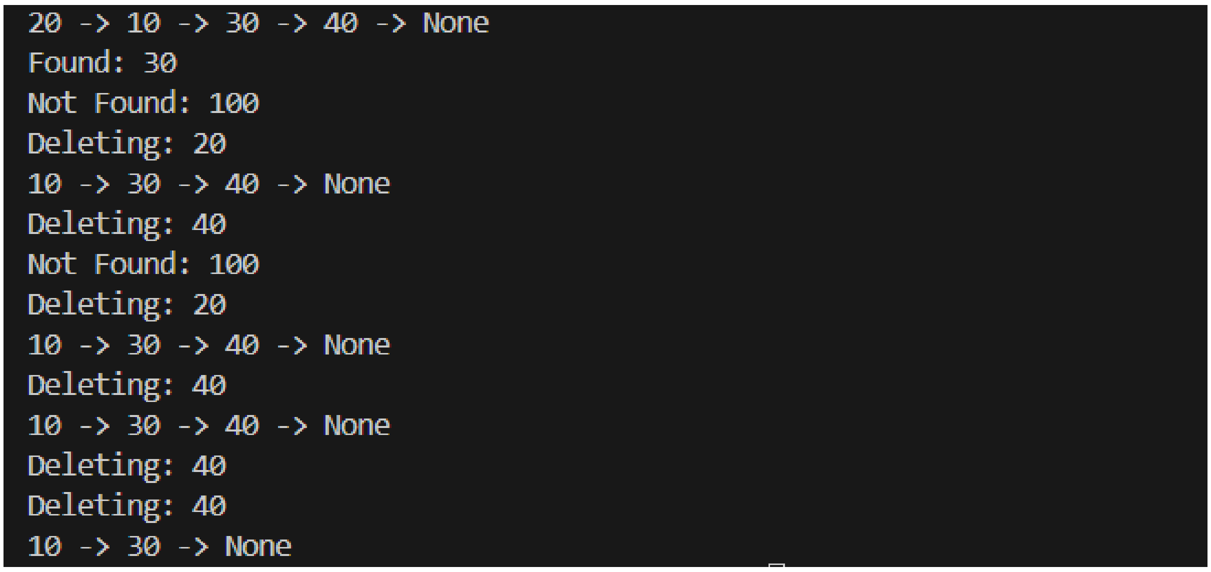


```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
# Sing
class SinglyLinkedList:
    def __init__(self):
        self.head = None
    # Function to insert a node at the end
    def insert_end(self, data):
        new_node = Node(data)
        # If list is empty
        if self.head is None:
            self.head = new_node
            return
        # Traverse to the last node
        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = new_node
    # Function to delete node at the beginning
    def delete_begin(self):
        if self.head is None:
            print("List is empty. Cannot delete.")
            return
        print(f"Deleting node at beginning: {self.head.data}")
        self.head = self.head.next
    # Function to delete node at the end
    def delete_end(self):
        if self.head is None:
            print("List is empty. Cannot delete.")
            return
        # If only one node exists
        if self.head.next is None:
            print(f"Deleting last node: {self.head.data}")
            self.head = None
            return
```



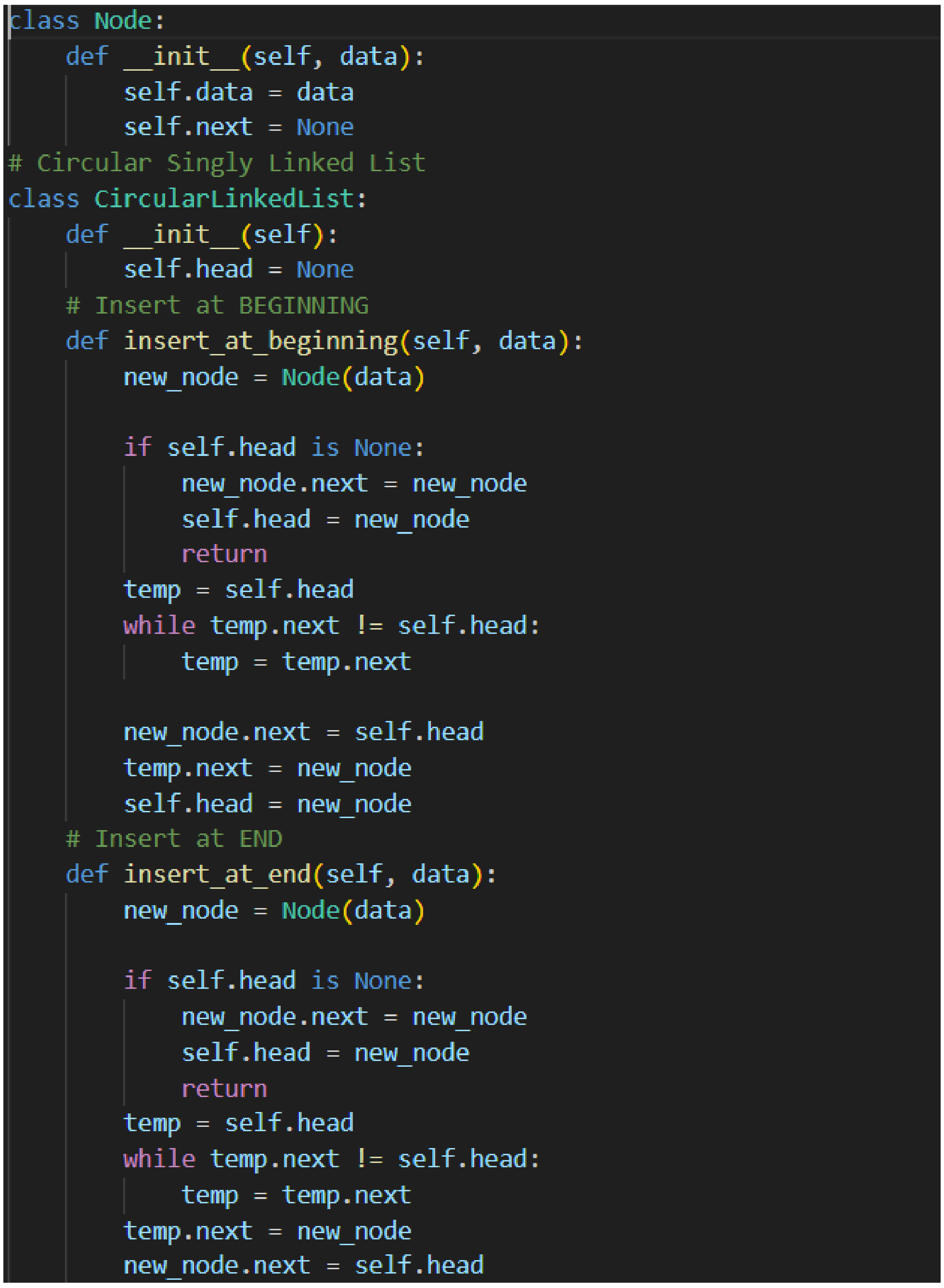
```python
        # Traverse to the second last node
        temp = self.head
        while temp.next.next:
            temp = temp.next
        print(f"Deleting node at end: {temp.next.data}")
        temp.next = None
    # Function to print the linked list
    def display(self):
        if self.head is None:
            print("List is empty.")
            return
        temp = self.head
        print("Linked List:", end=" ")
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
# Main Program
if __name__ == "__main__":
    sll = SinglyLinkedList()
    # Insert sample data
    sll.insert_end(10)
    sll.insert_end(20)
    sll.insert_end(30)
    sll.insert_end(40)
    print("\nInitial List:")
    sll.display()
    print("\nDeleting from beginning:")
    sll.delete_begin()
    sll.display()
    print("\nDeleting from end:")
    sll.delete_end()
    sll.display()
```

Output:

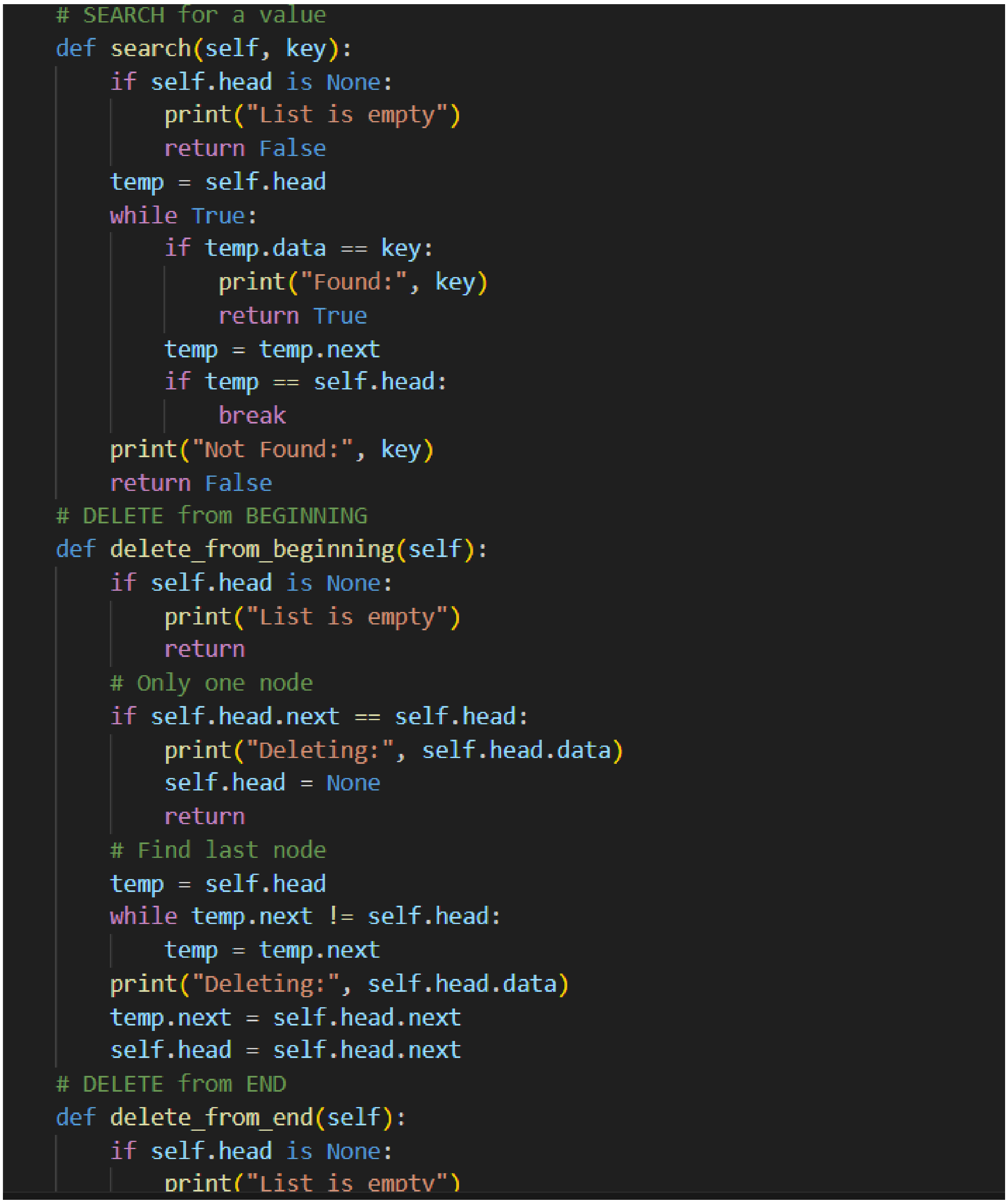

```
20 -> 10 -> 30 -> 40 -> None
Found: 30
Not Found: 100
Deleting: 20
10 -> 30 -> 40 -> None
Deleting: 40
Not Found: 100
Deleting: 20
10 -> 30 -> 40 -> None
Deleting: 40
10 -> 30 -> 40 -> None
Deleting: 40
Deleting: 40
10 -> 30 -> None
```

4. Singly Linked List Operations(Insertion ,Deletion ,Search ,Display)

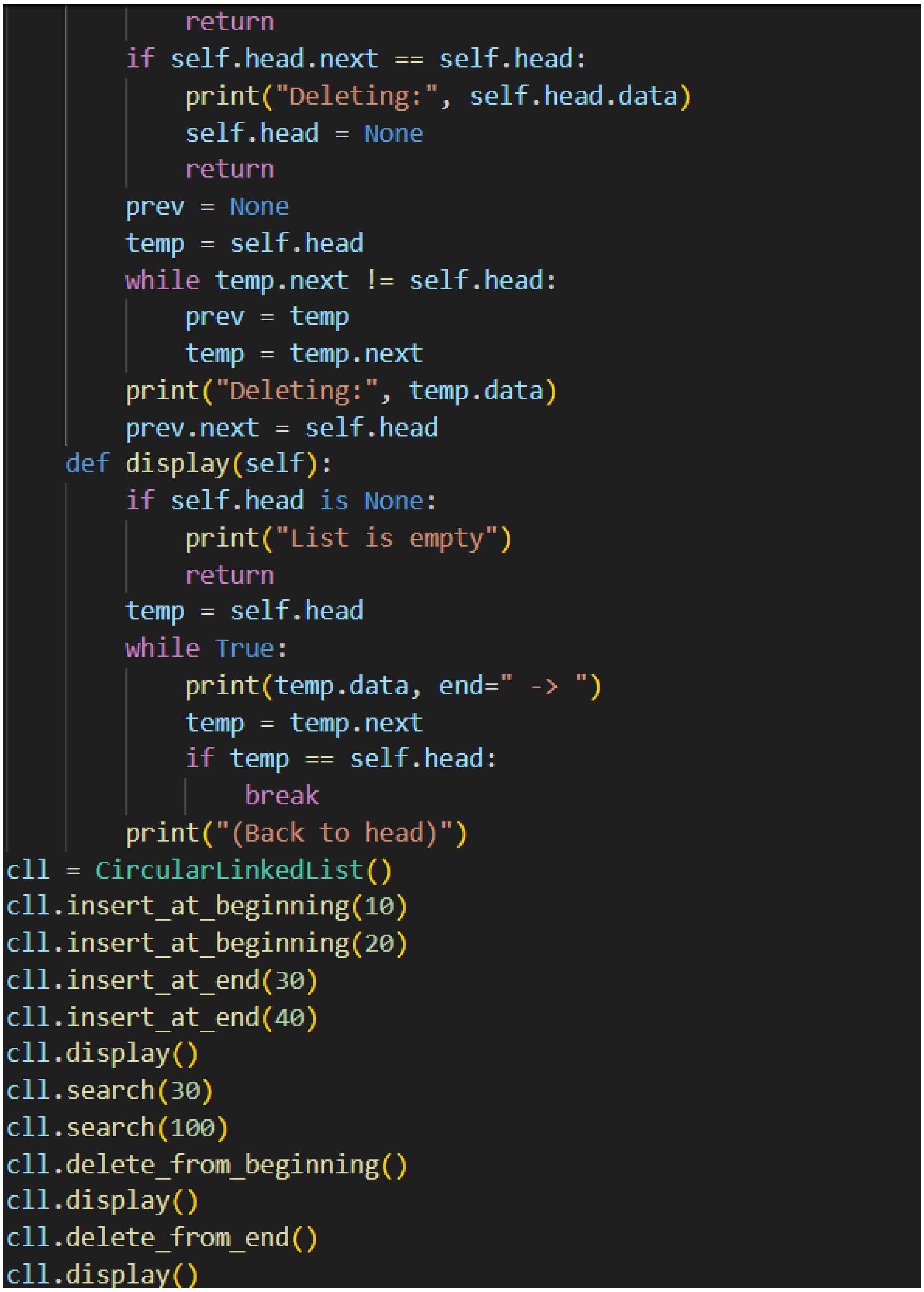
Code:



```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
# Circular Singly Linked List
class CircularLinkedList:
    def __init__(self):
        self.head = None
    # Insert at BEGINNING
    def insert_at_beginning(self, data):
        new_node = Node(data)

        if self.head is None:
            new_node.next = new_node
            self.head = new_node
            return
        temp = self.head
        while temp.next != self.head:
            temp = temp.next

        new_node.next = self.head
        temp.next = new_node
        self.head = new_node
    # Insert at END
    def insert_at_end(self, data):
        new_node = Node(data)

        if self.head is None:
            new_node.next = new_node
            self.head = new_node
            return
        temp = self.head
        while temp.next != self.head:
            temp = temp.next
        temp.next = new_node
        new_node.next = self.head
```

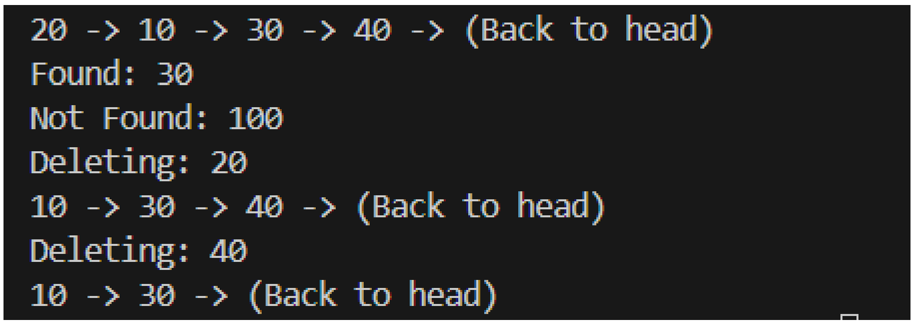


```
# SEARCH for a value
def search(self, key):
    if self.head is None:
        print("List is empty")
        return False
    temp = self.head
    while True:
        if temp.data == key:
            print("Found:", key)
            return True
        temp = temp.next
        if temp == self.head:
            break
    print("Not Found:", key)
    return False
# DELETE from BEGINNING
def delete_from_beginning(self):
    if self.head is None:
        print("List is empty")
        return
    # Only one node
    if self.head.next == self.head:
        print("Deleting:", self.head.data)
        self.head = None
        return
    # Find last node
    temp = self.head
    while temp.next != self.head:
        temp = temp.next
    print("Deleting:", self.head.data)
    temp.next = self.head.next
    self.head = self.head.next
# DELETE from END
def delete_from_end(self):
    if self.head is None:
        print("List is empty")
```



```python
            return
        if self.head.next == self.head:
            print("Deleting:", self.head.data)
            self.head = None
            return
        prev = None
        temp = self.head
        while temp.next != self.head:
            prev = temp
            temp = temp.next
        print("Deleting:", temp.data)
        prev.next = self.head
    def display(self):
        if self.head is None:
            print("List is empty")
            return
        temp = self.head
        while True:
            print(temp.data, end=" -> ")
            temp = temp.next
            if temp == self.head:
                break
        print("(Back to head)")
cll = CircularLinkedList()
cll.insert_at_beginning(10)
cll.insert_at_beginning(20)
cll.insert_at_end(30)
cll.insert_at_end(40)
cll.display()
cll.search(30)
cll.search(100)
cll.delete_from_beginning()
cll.display()
cll.delete_from_end()
cll.display()
```

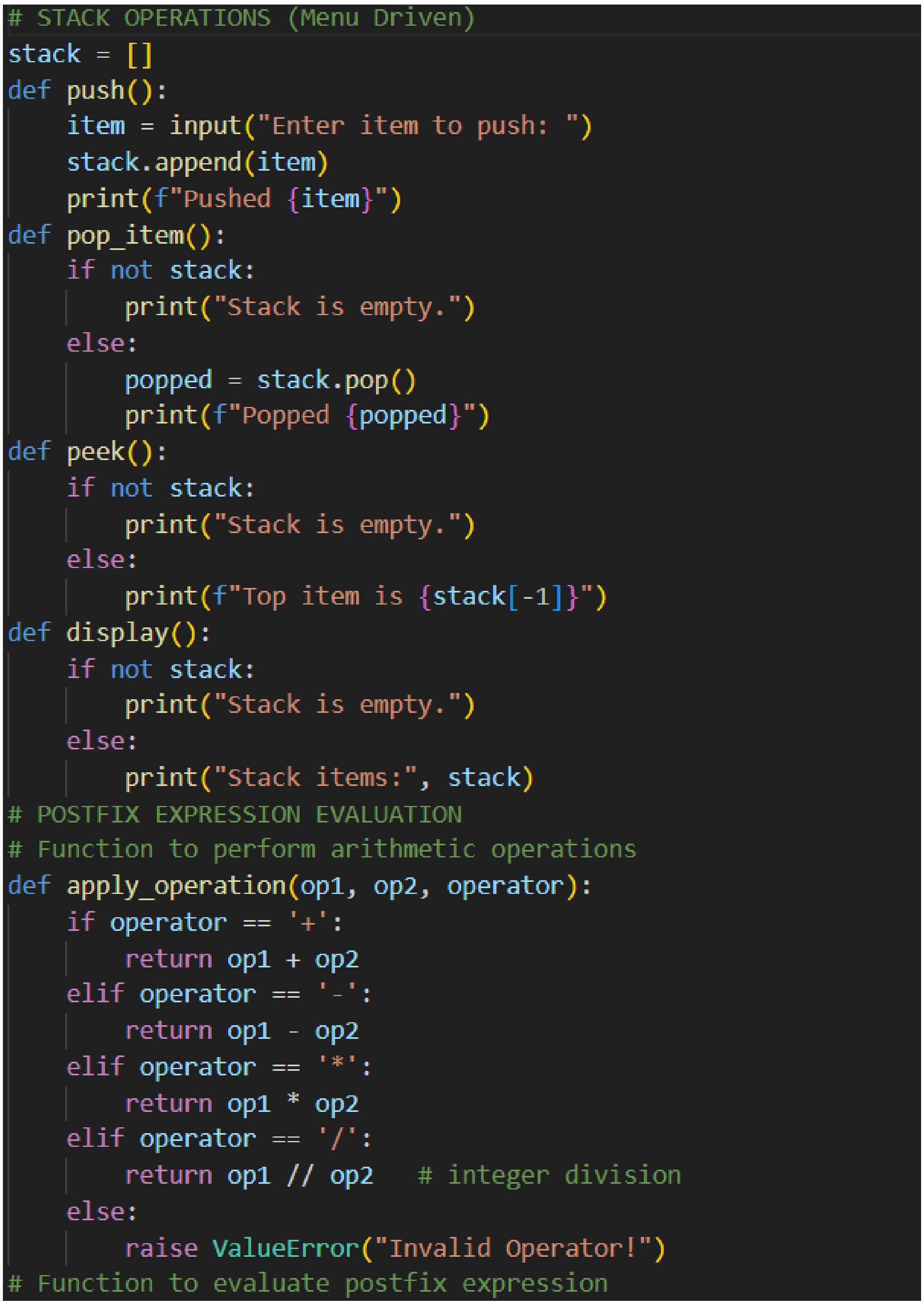
Output:



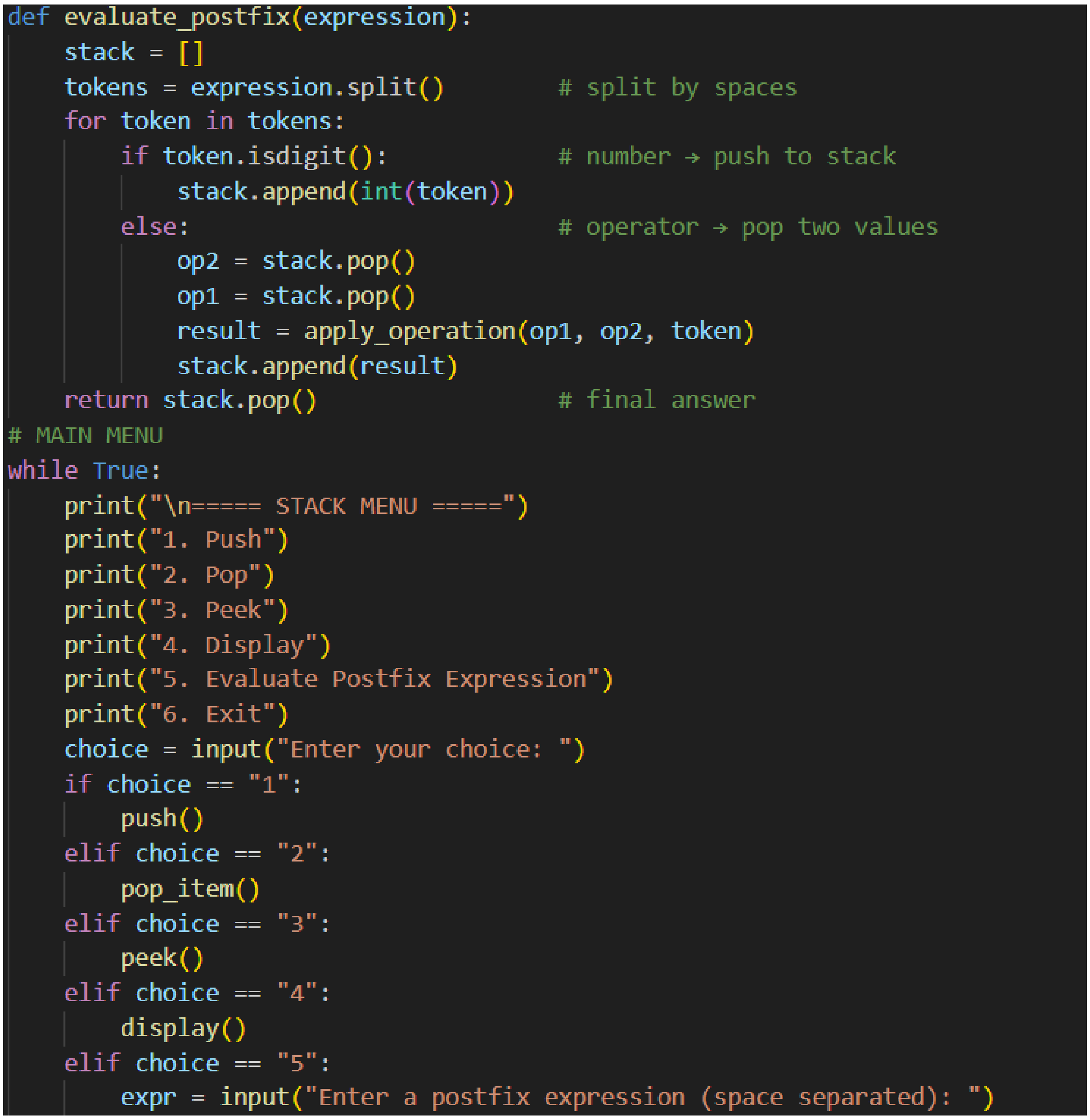
```
20 -> 10 -> 30 -> 40 -> (Back to head)
Found: 30
Not Found: 100
Deleting: 20
10 -> 30 -> 40 -> (Back to head)
Deleting: 40
10 -> 30 -> (Back to head)
```

5. Performing operations on stack (Push, Pop, Peek, Display, Expressing postfix expression)

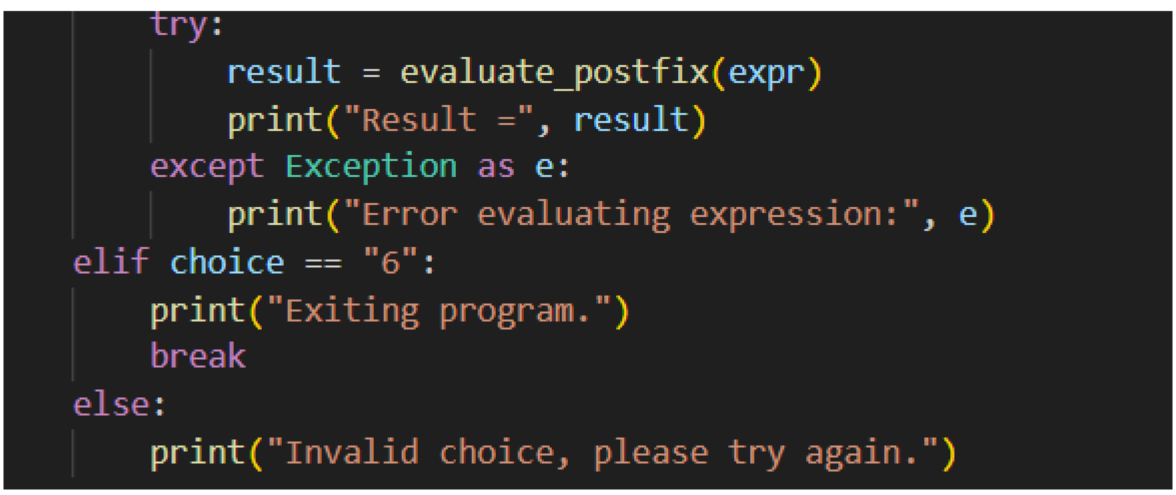
Code:



```python
# STACK OPERATIONS (Menu Driven)
stack = []
def push():
    item = input("Enter item to push: ")
    stack.append(item)
    print(f"Pushed {item}")
def pop_item():
    if not stack:
        print("Stack is empty.")
    else:
        popped = stack.pop()
        print(f"Popped {popped}")
def peek():
    if not stack:
        print("Stack is empty.")
    else:
        print(f"Top item is {stack[-1]}")
def display():
    if not stack:
        print("Stack is empty.")
    else:
        print("Stack items:", stack)
# POSTFIX EXPRESSION EVALUATION
# Function to perform arithmetic operations
def apply_operation(op1, op2, operator):
    if operator == '+':
        return op1 + op2
    elif operator == '-':
        return op1 - op2
    elif operator == '*':
        return op1 * op2
    elif operator == '/':
        return op1 // op2   # integer division
    else:
        raise ValueError("Invalid Operator!")
# Function to evaluate postfix expression
```

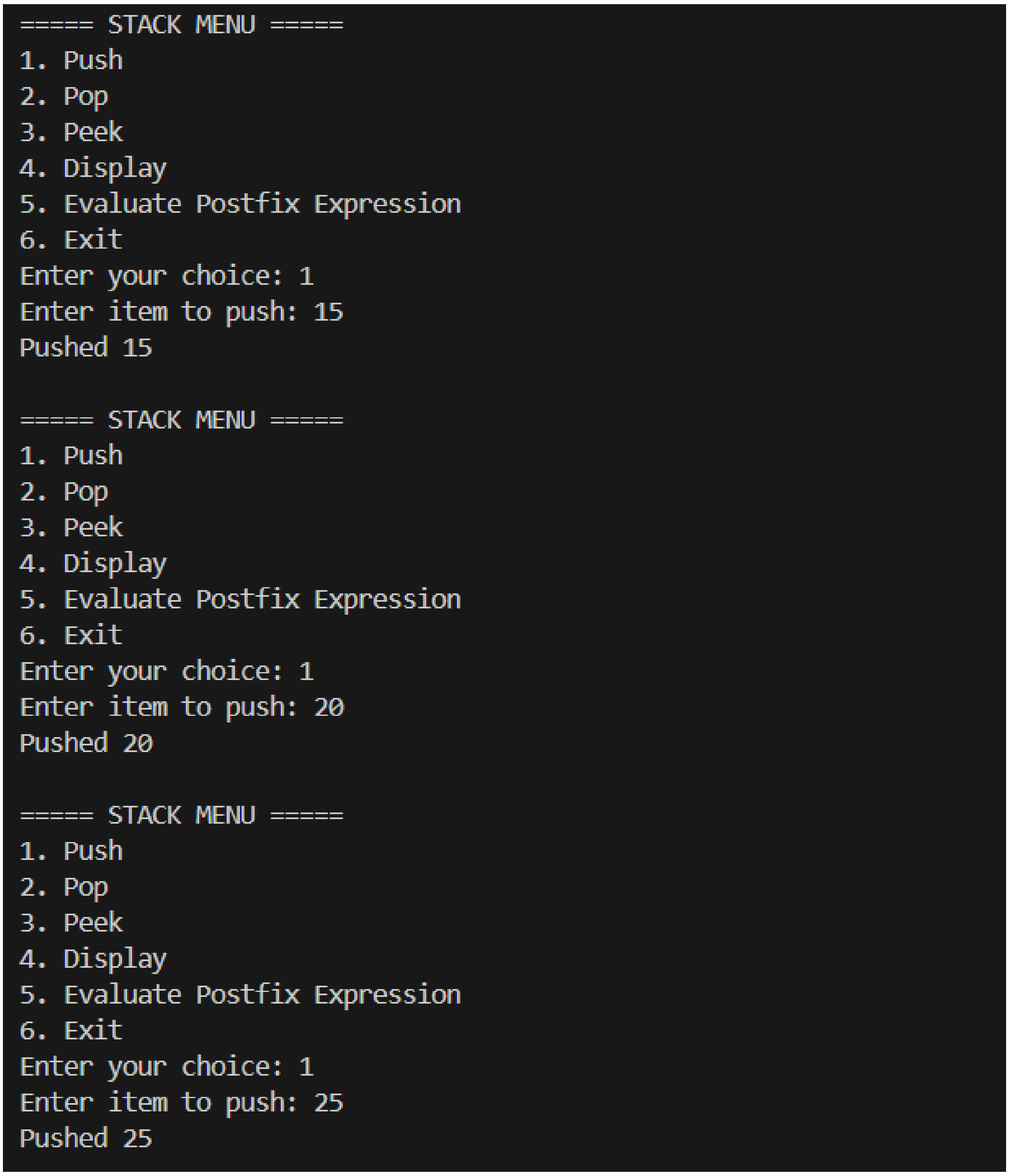


```python
def evaluate_postfix(expression):
    stack = []
    tokens = expression.split()        # split by spaces
    for token in tokens:
        if token.isdigit():            # number → push to stack
            stack.append(int(token))
        else:                          # operator → pop two values
            op2 = stack.pop()
            op1 = stack.pop()
            result = apply_operation(op1, op2, token)
            stack.append(result)
    return stack.pop()                 # final answer
# MAIN MENU
while True:
    print("\n===== STACK MENU =====")
    print("1. Push")
    print("2. Pop")
    print("3. Peek")
    print("4. Display")
    print("5. Evaluate Postfix Expression")
    print("6. Exit")
    choice = input("Enter your choice: ")
    if choice == "1":
        push()
    elif choice == "2":
        pop_item()
    elif choice == "3":
        peek()
    elif choice == "4":
        display()
    elif choice == "5":
        expr = input("Enter a postfix expression (space separated): ")
```

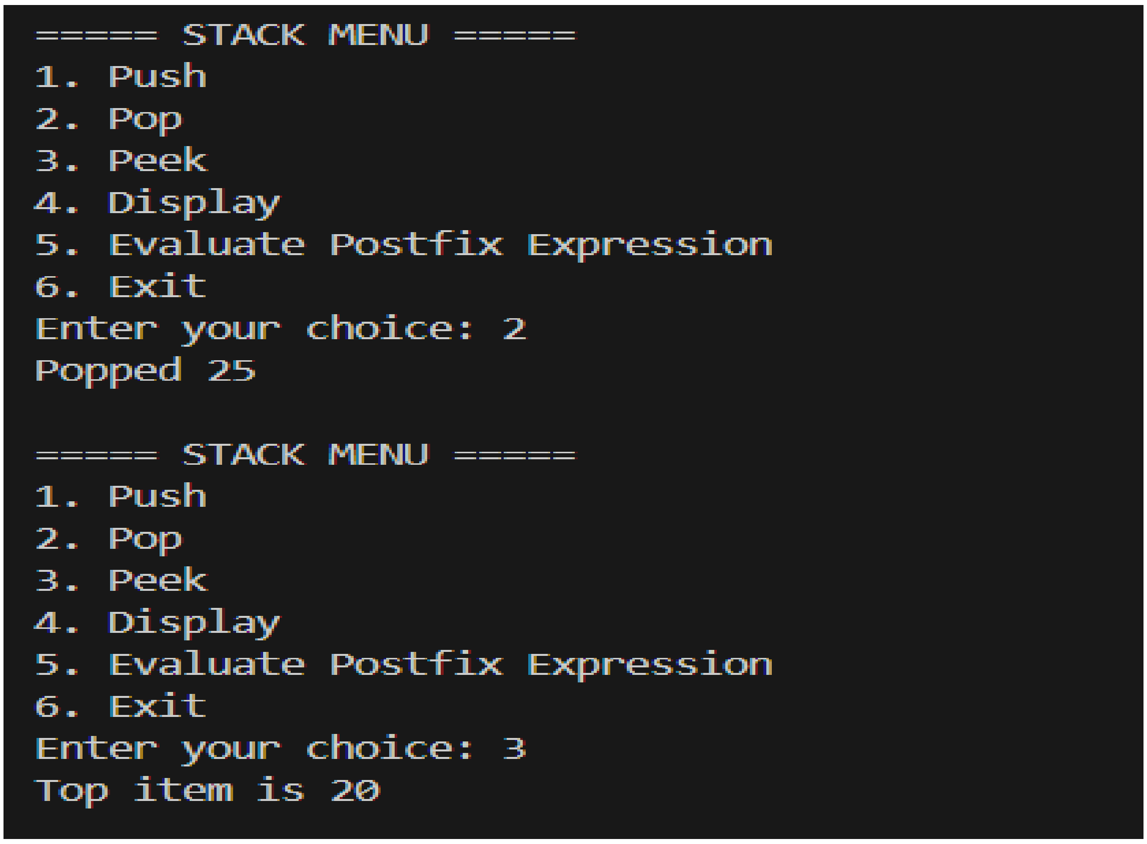
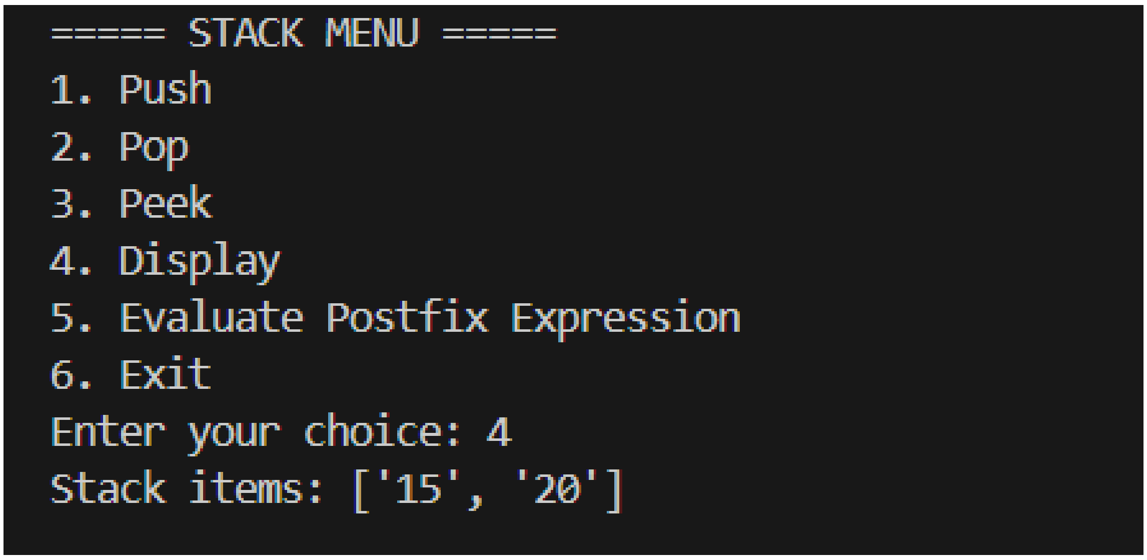


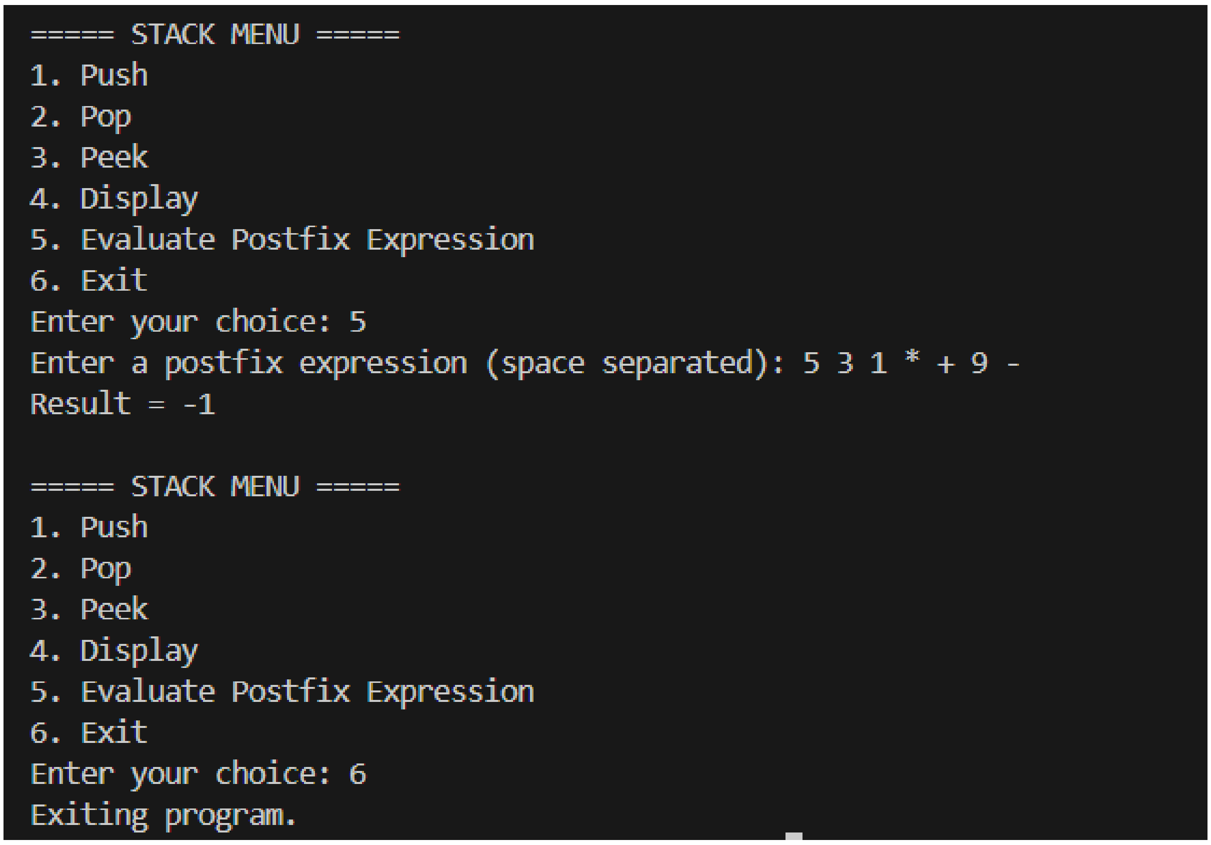
```python
        try:
            result = evaluate_postfix(expr)
            print("Result =", result)
        except Exception as e:
            print("Error evaluating expression:", e)
    elif choice == "6":
        print("Exiting program.")
        break
    else:
        print("Invalid choice, please try again.")
```

Output:



```
===== STACK MENU =====
1. Push
2. Pop
3. Peek
4. Display
5. Evaluate Postfix Expression
6. Exit
Enter your choice: 1
Enter item to push: 15
Pushed 15

===== STACK MENU =====
1. Push
2. Pop
3. Peek
4. Display
5. Evaluate Postfix Expression
6. Exit
Enter your choice: 1
Enter item to push: 20
Pushed 20

===== STACK MENU =====
1. Push
2. Pop
3. Peek
4. Display
5. Evaluate Postfix Expression
6. Exit
Enter your choice: 1
Enter item to push: 25
Pushed 25
```

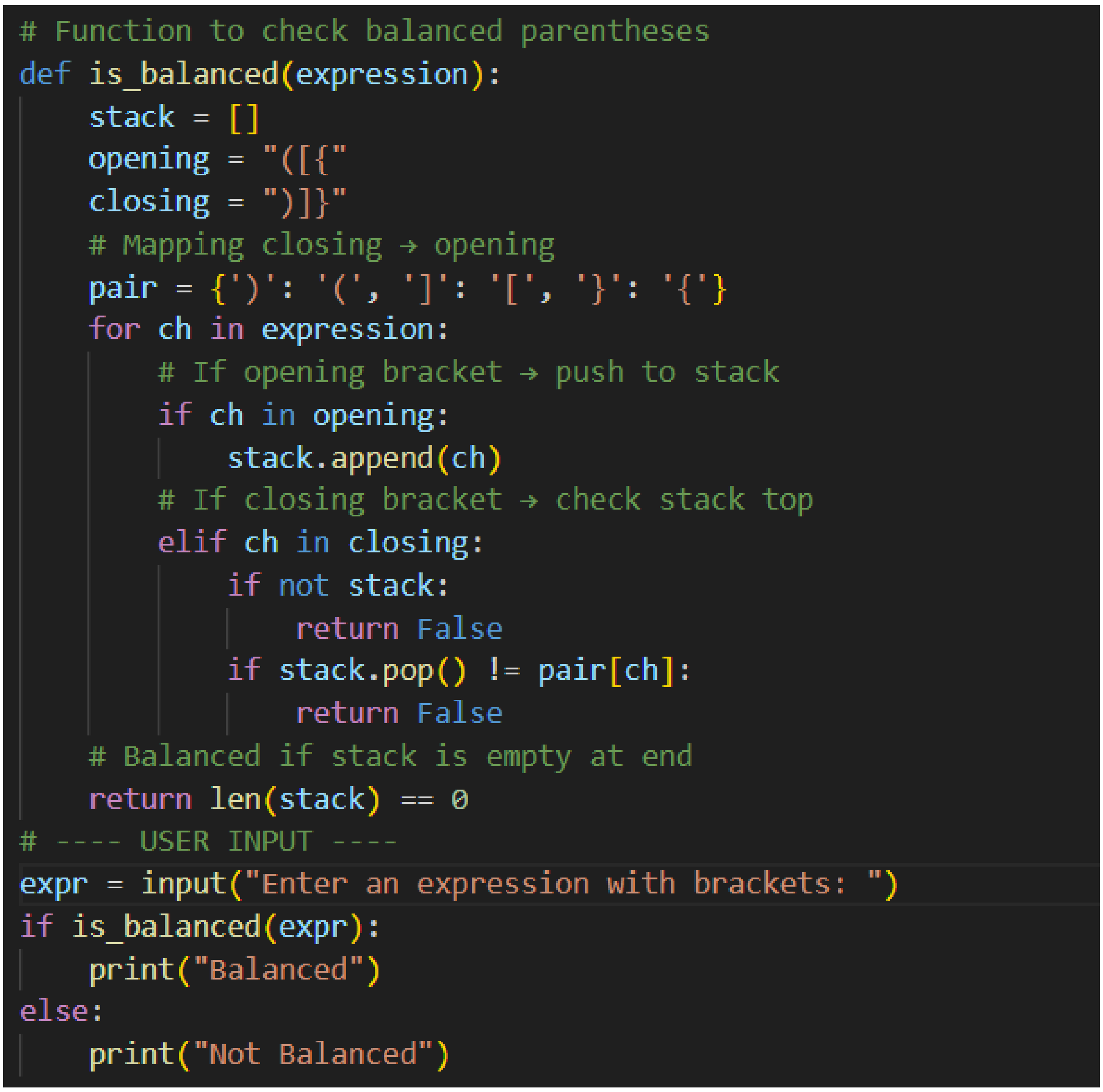

```
===== STACK MENU =====
1. Push
2. Pop
3. Peek
4. Display
5. Evaluate Postfix Expression
6. Exit
Enter your choice: 2
Popped 25

===== STACK MENU =====
1. Push
2. Pop
3. Peek
4. Display
5. Evaluate Postfix Expression
6. Exit
Enter your choice: 3
Top item is 20
```

```
===== STACK MENU =====
1. Push
2. Pop
3. Peek
4. Display
5. Evaluate Postfix Expression
6. Exit
Enter your choice: 4
Stack items: ['15', '20']
```



```
===== STACK MENU =====
1. Push
2. Pop
3. Peek
4. Display
5. Evaluate Postfix Expression
6. Exit
Enter your choice: 5
Enter a postfix expression (space separated): 5 3 1 * + 9 -
Result = -1

===== STACK MENU =====
1. Push
2. Pop
3. Peek
4. Display
5. Evaluate Postfix Expression
6. Exit
Enter your choice: 6
Exiting program.
```

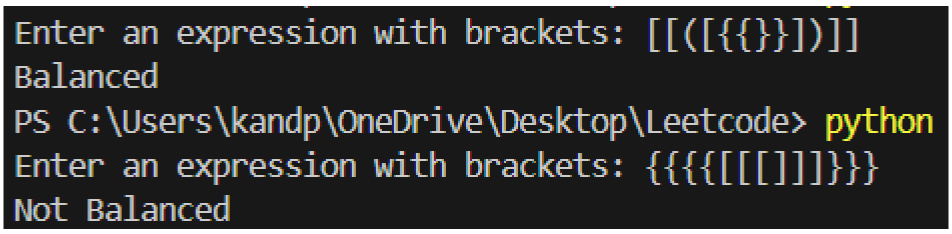
6.Check Balanced Parenthesis using Stack

Code:



```python
# Function to check balanced parentheses
def is_balanced(expression):
    stack = []
    opening = "([{"
    closing = ")]}"
    # Mapping closing → opening
    pair = {')': '(', ']': '[', '}': '{'}
    for ch in expression:
        # If opening bracket → push to stack
        if ch in opening:
            stack.append(ch)
        # If closing bracket → check stack top
        elif ch in closing:
            if not stack:
                return False
            if stack.pop() != pair[ch]:
                return False
    # Balanced if stack is empty at end
    return len(stack) == 0
# ---- USER INPUT ----
expr = input("Enter an expression with brackets: ")
if is_balanced(expr):
    print("Balanced")
else:
    print("Not Balanced")
```

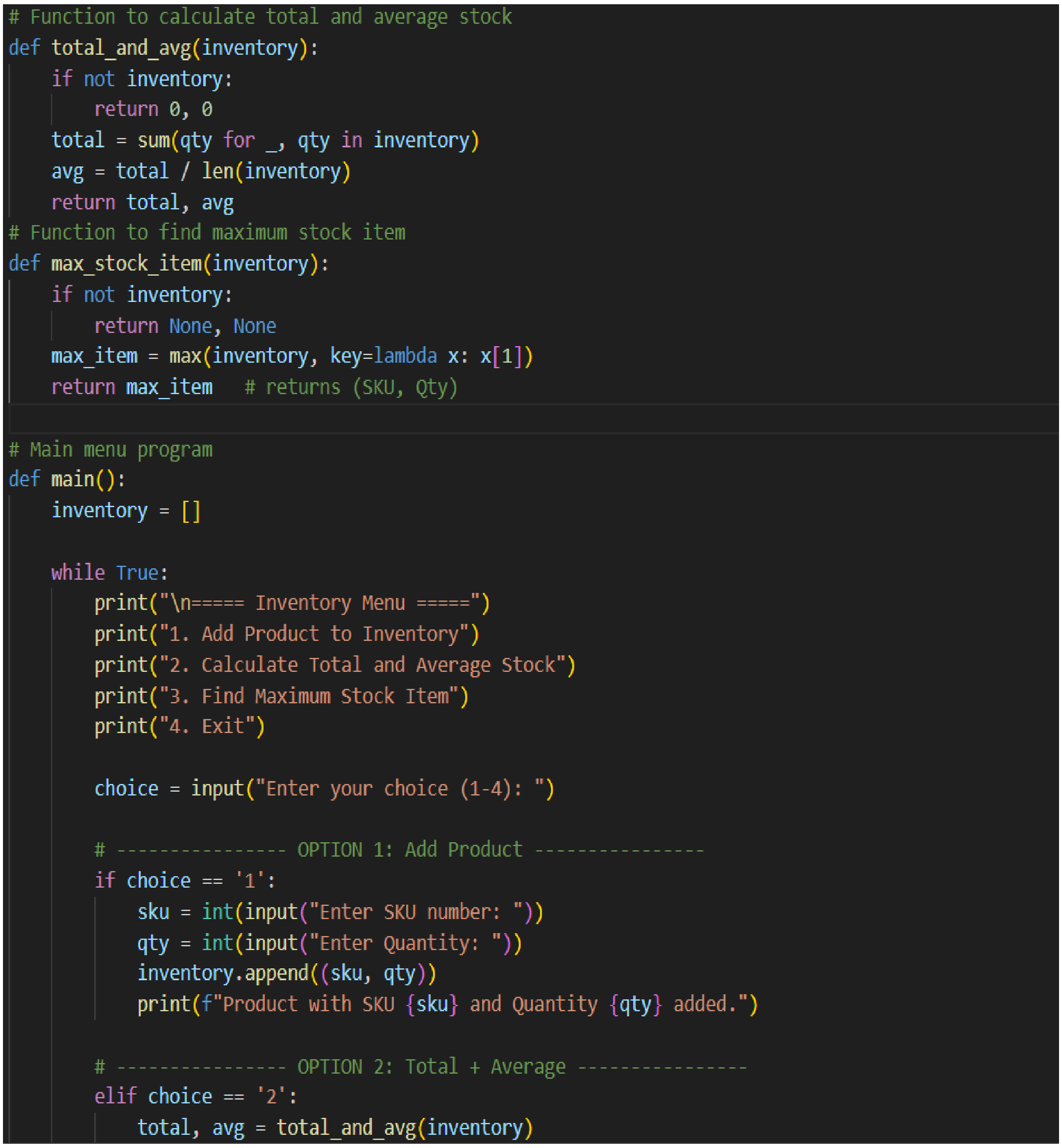
Output:



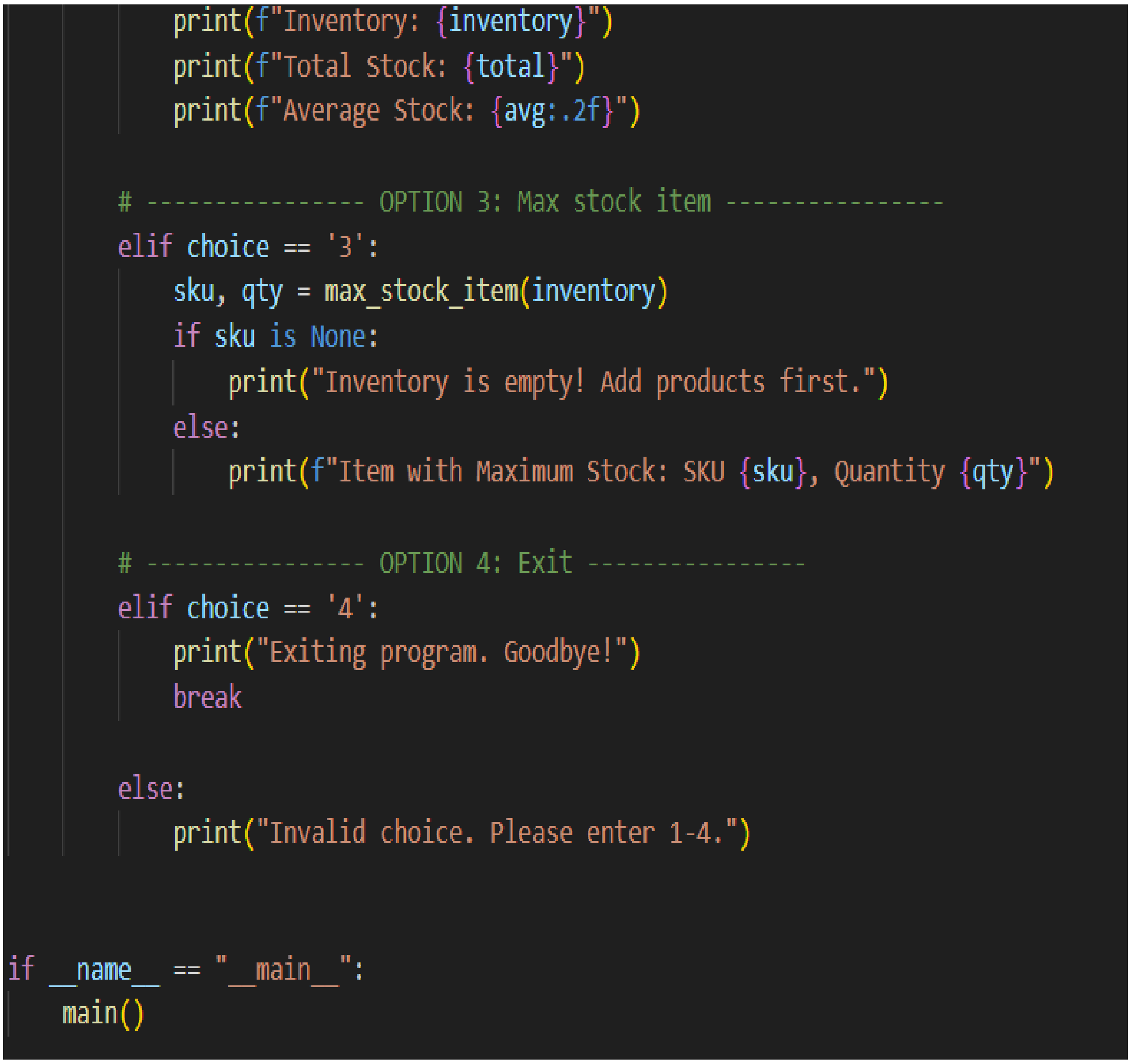
```
Enter an expression with brackets: [[([{{}}])]]
Balanced
PS C:\Users\kandp\OneDrive\Desktop\Leetcode> python
Enter an expression with brackets: {{{{[[[[]]]]}}}
Not Balanced
```

7. Lab Project(Inventory Stock Management System)

Code:

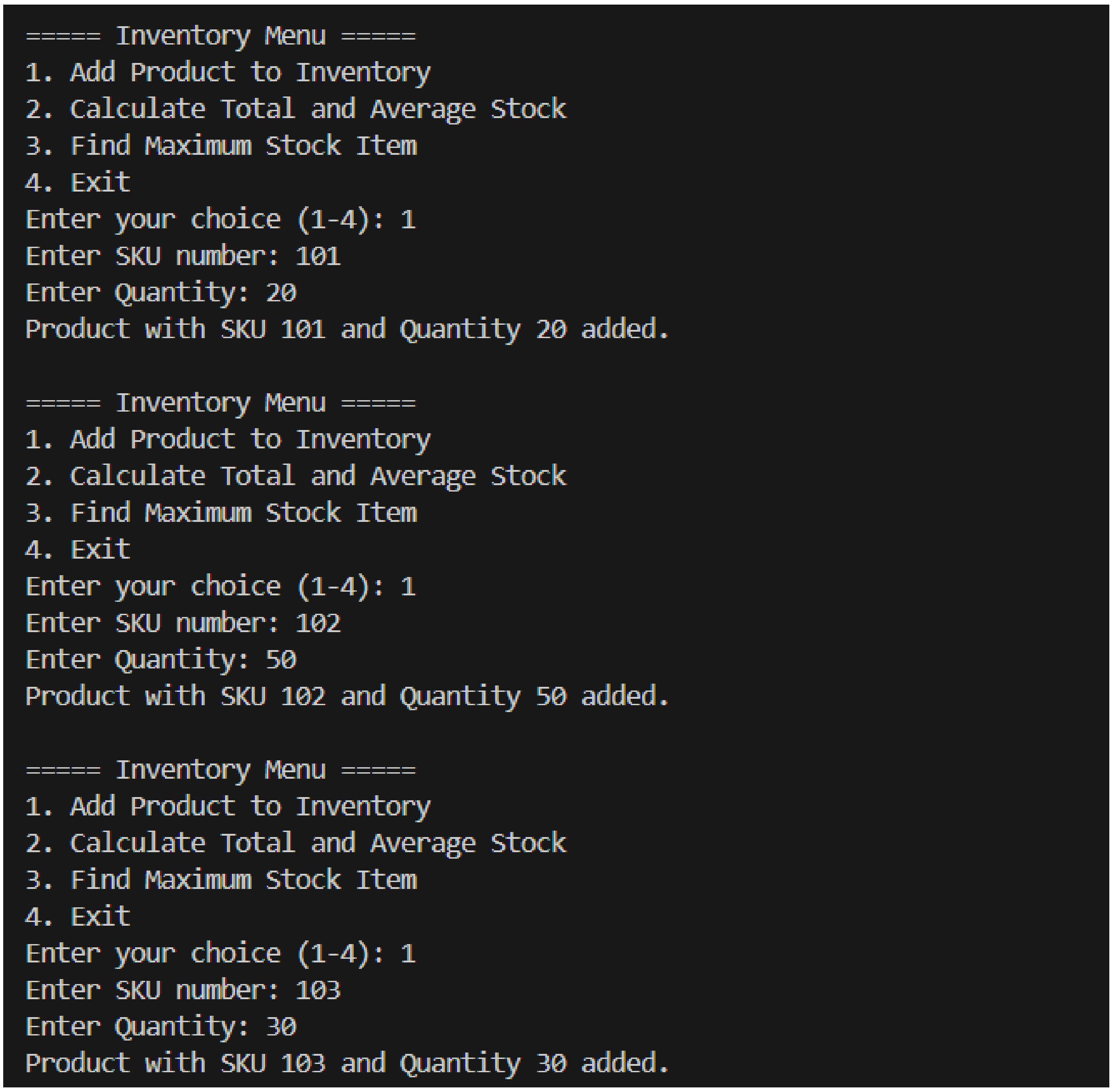


```python
# Function to calculate total and average stock
def total_and_avg(inventory):
    if not inventory:
        return 0, 0
    total = sum(qty for _, qty in inventory)
    avg = total / len(inventory)
    return total, avg
# Function to find maximum stock item
def max_stock_item(inventory):
    if not inventory:
        return None, None
    max_item = max(inventory, key=lambda x: x[1])
    return max_item   # returns (SKU, Qty)

# Main menu program
def main():
    inventory = []

    while True:
        print("\n===== Inventory Menu =====")
        print("1. Add Product to Inventory")
        print("2. Calculate Total and Average Stock")
        print("3. Find Maximum Stock Item")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        # ---------------- OPTION 1: Add Product ----------------
        if choice == '1':
            sku = int(input("Enter SKU number: "))
            qty = int(input("Enter Quantity: "))
            inventory.append((sku, qty))
            print(f"Product with SKU {sku} and Quantity {qty} added.")

        # ---------------- OPTION 2: Total + Average ----------------
        elif choice == '2':
            total, avg = total_and_avg(inventory)
```

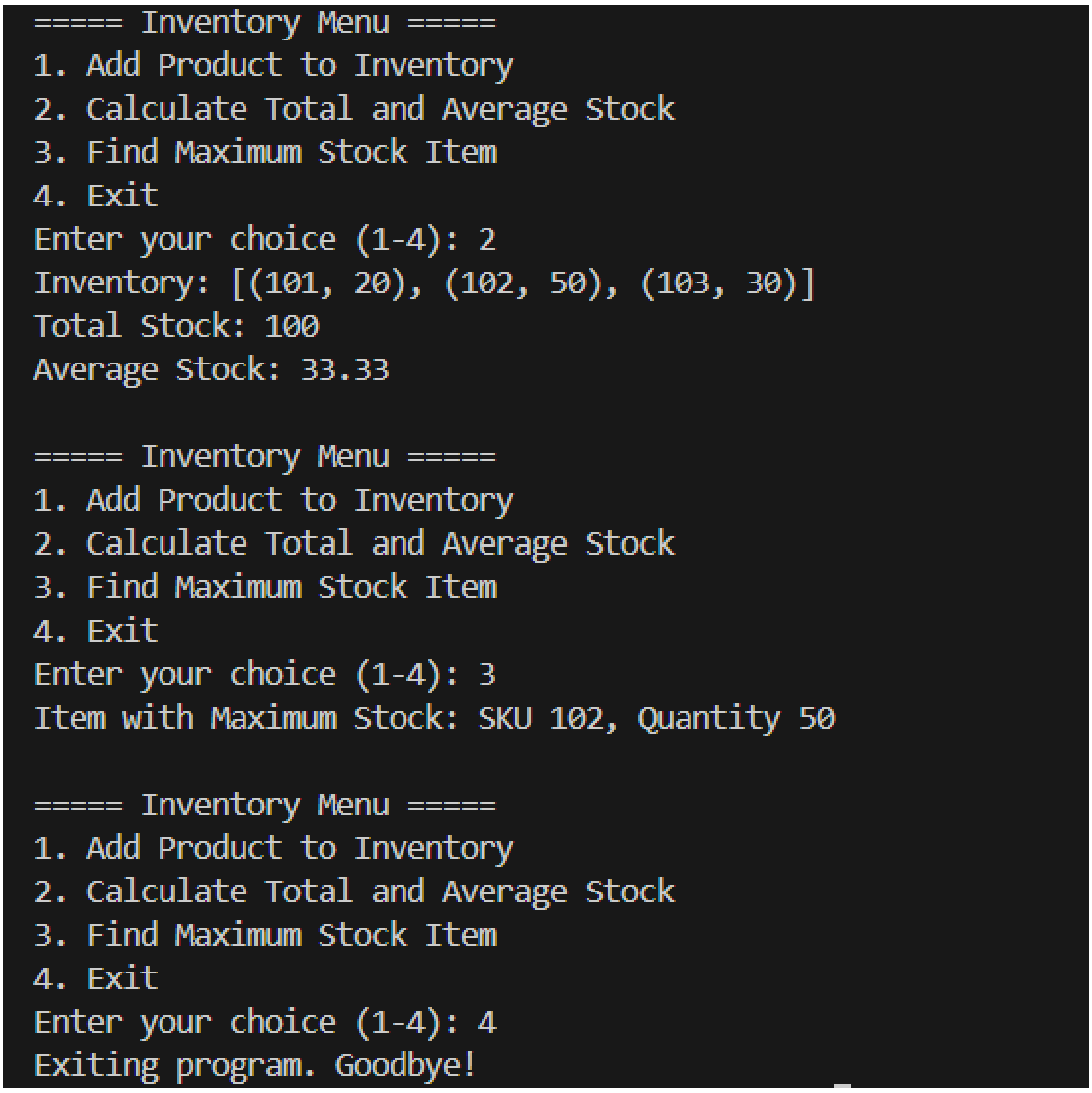


```python
            print(f"Inventory: {inventory}")
            print(f"Total Stock: {total}")
            print(f"Average Stock: {avg:.2f}")

        # ---------------- OPTION 3: Max stock item ----------------
        elif choice == '3':
            sku, qty = max_stock_item(inventory)
            if sku is None:
                print("Inventory is empty! Add products first.")
            else:
                print(f"Item with Maximum Stock: SKU {sku}, Quantity {qty}")

        # ---------------- OPTION 4: Exit ----------------
        elif choice == '4':
            print("Exiting program. Goodbye!")
            break

        else:
            print("Invalid choice. Please enter 1-4.")


if __name__ == "__main__":
    main()
```

Output:



```
===== Inventory Menu =====
1. Add Product to Inventory
2. Calculate Total and Average Stock
3. Find Maximum Stock Item
4. Exit
Enter your choice (1-4): 1
Enter SKU number: 101
Enter Quantity: 20
Product with SKU 101 and Quantity 20 added.

===== Inventory Menu =====
1. Add Product to Inventory
2. Calculate Total and Average Stock
3. Find Maximum Stock Item
4. Exit
Enter your choice (1-4): 1
Enter SKU number: 102
Enter Quantity: 50
Product with SKU 102 and Quantity 50 added.

===== Inventory Menu =====
1. Add Product to Inventory
2. Calculate Total and Average Stock
3. Find Maximum Stock Item
4. Exit
Enter your choice (1-4): 1
Enter SKU number: 103
Enter Quantity: 30
Product with SKU 103 and Quantity 30 added.
```



```
===== Inventory Menu =====
1. Add Product to Inventory
2. Calculate Total and Average Stock
3. Find Maximum Stock Item
4. Exit
Enter your choice (1-4): 2
Inventory: [(101, 20), (102, 50), (103, 30)]
Total Stock: 100
Average Stock: 33.33

===== Inventory Menu =====
1. Add Product to Inventory
2. Calculate Total and Average Stock
3. Find Maximum Stock Item
4. Exit
Enter your choice (1-4): 3
Item with Maximum Stock: SKU 102, Quantity 50

===== Inventory Menu =====
1. Add Product to Inventory
2. Calculate Total and Average Stock
3. Find Maximum Stock Item
4. Exit
Enter your choice (1-4): 4
Exiting program. Goodbye!
```