Assignment_11.1

K.MANOJ KUMAR

2303A52201

B-35

Task-1:

Prompt: Generate a Stack class with push, pop, peek, and is_empty methods.

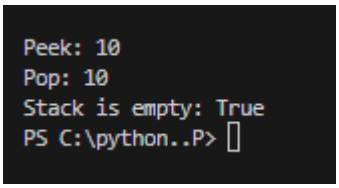
Sample Input :class Stack:pass

Expected Output:• A functional stack implementation with all required methods and docstrings. And print the output

Code:

```python
class Stack:
    """A simple stack implementation using a list."""

    def __init__(self):
        """Initialize an empty stack."""
        self.items = []

    def push(self, item):
        """Add an item to the top of the stack."""
        self.items.append(item)

    def pop(self):
        """Remove and return the top item from the stack."""
        if not self.is_empty():
            return self.items.pop()
        return None

    def peek(self):
        """Return the top item without removing it."""
        if not self.is_empty():
            return self.items[-1]
        return None

    def is_empty(self):
        """Check if the stack is empty."""
        return len(self.items) == 0


# Test the Stack implementation
if __name__ == "__main__":
    stack = Stack()

    print("Stack is empty:", stack.is_empty())

    stack.push(10)
    stack.push(20)
    stack.push(30)
    print("Pushed: 10, 20, 30")

    print("Stack is empty:", stack.is_empty())
    print("Peek:", stack.peek())
    print("Pop:", stack.pop())
    print("Pop:", stack.pop())
    print("Peek:", stack.peek())
    print("Pop:", stack.pop())
    print("Stack is empty:", stack.is_empty())
```

Output:



Task-2:

Prompt: Generate a Queue using Python lists.Sample Input Code:class Queue:pass

Expected Output:• FIFO-based queue class with enqueue, dequeue, peek, and size methods.with output

Code:

```python
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        """Add an item to the rear of the queue"""
        self.items.append(item)

    def dequeue(self):
        """Remove and return the front item from the queue"""
        if not self.is_empty():
            return self.items.pop(0)
        return None

    def peek(self):
        """Return the front item without removing it"""
        if not self.is_empty():
            return self.items[0]
        return None

    def size(self):
        """Return the number of items in the queue"""
        return len(self.items)

    def is_empty(self):
        """Check if the queue is empty"""
        return len(self.items) == 0

    def __str__(self):
        return str(self.items)


# Test the Queue
if __name__ == "__main__":
    q = Queue()

    # Enqueue items
    q.enqueue(10)
    q.enqueue(20)
    q.enqueue(30)
    q.enqueue(40)
    print(f"Queue after enqueuing: {q}")
    print(f"Queue size: {q.size()}")

    # Peek
    print(f"Peek (front element): {q.peek()}")

    # Dequeue items
    print(f"Dequeue: {q.dequeue()}")
    print(f"Queue after dequeue: {q}")
    print(f"Queue size: {q.size()}")
```

Output:

```
Queue: [30, 40]
PS C:\python..P>
```

Task-3:

Prompt: generate a Singly Linked List with insert and display methods.

Sample Input Code:class Node:pass

class LinkedList:pass

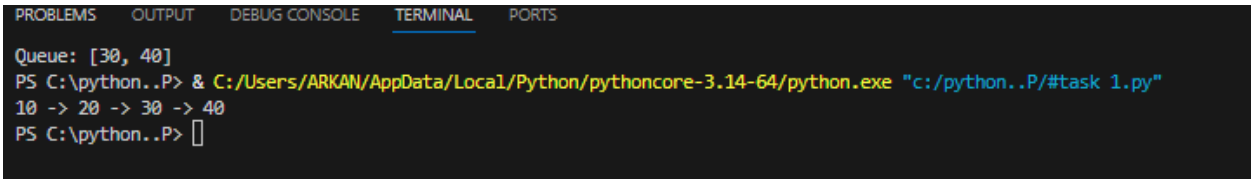
Expected Output:

• A working linked list implementation with clear method documentation.

Code:

```
isk 1.py > ...
class Node:
    """A node in the singly linked list."""
    def __init__(self, data):
        self.data = data
        self.next = None


class LinkedList:
    """A singly linked list implementation."""
    def __init__(self):
        self.head = None

    def insert(self, data):
        """Insert a new node at the end of the linked list."""
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return

        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

    def display(self):
        """Display all elements in the linked list."""
        elements = []
        current = self.head
        while current:
            elements.append(str(current.data))
            current = current.next
        print(" -> ".join(elements) if elements else "Empty List")


# Example usage
if __name__ == "__main__":
    ll = LinkedList()
    ll.insert(10)
    ll.insert(20)
    ll.insert(30)
    ll.insert(40)

    ll.display()  # Output: 10 -> 20 -> 30 -> 40
```

Output:



Task-4:

Generate a python code to create a BST with insert and in-order traversal methods.Sample Input Code:

class BST:pass Expected Output:• BST implementation with recursive insert and traversal methods.

Code:

```python
class BST:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

    def insert(self, value):
        if value < self.value:
            if self.left is None:
                self.left = BST(value)
            else:
                self.left.insert(value)
        else:
            if self.right is None:
                self.right = BST(value)
            else:
                self.right.insert(value)

    def inorder_traversal(self):
        result = []
        if self.left:
            result.extend(self.left.inorder_traversal())
        result.append(self.value)
        if self.right:
            result.extend(self.right.inorder_traversal())
        return result


# Example usage
bst = BST(10)
bst.insert(5)
bst.insert(15)
bst.insert(3)
bst.insert(7)
bst.insert(12)
bst.insert(20)

print(bst.inorder_traversal())  # Output: [3, 5, 7, 10, 12, 15, 20]
```

Output:

```
[3, 5, 7, 10, 12, 15, 20]
PS C:\python..P>
```

Task-5:

Prompt : Generate a python code to implement a hash table with basic insert, search, and delete methods. Sample Input Code :class HashTable : pass

Expected Output:• Collision handling using chaining, with well-commented methods.

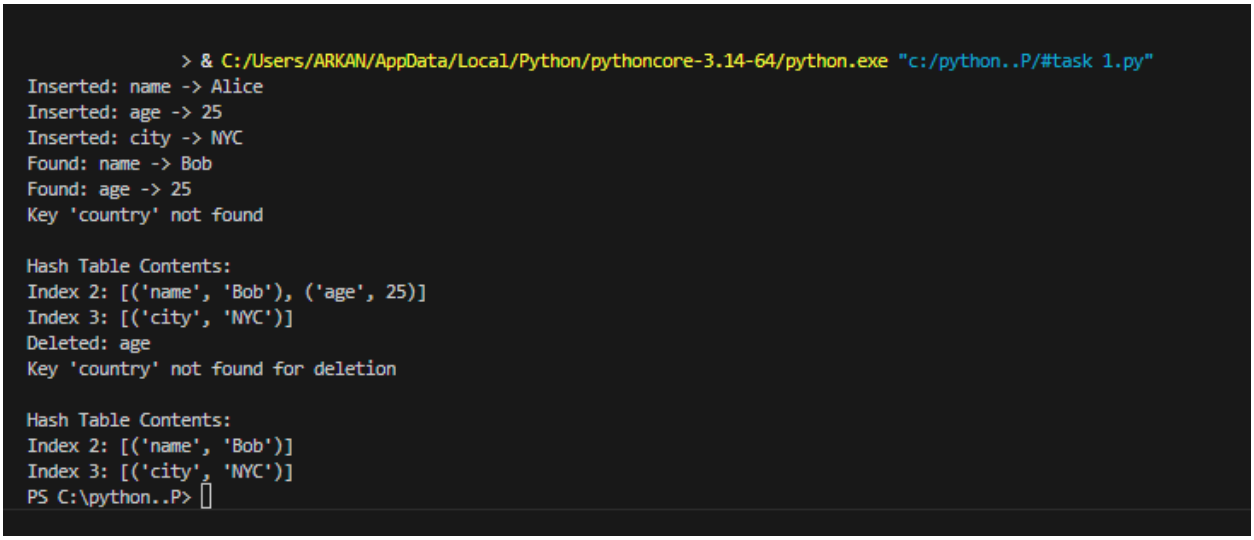
Code:

sk 1.py > ...

```python
class HashTable:
    """A hash table implementation using chaining for collision handling."""

    def __init__(self, size=10):
        """Initialize hash table with given size."""
        self.size = size
        self.table = [[] for _ in range(size)]

    def _hash(self, key):
        """Generate hash value for a given key."""
        return hash(key) % self.size

    def insert(self, key, value):
        """Insert a key-value pair into the hash table."""
        index = self._hash(key)

        # Check if key already exists and update it
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index][i] = (key, value)
                return

        # Add new key-value pair to the chain
        self.table[index].append((key, value))
        print(f"Inserted: {key} -> {value}")

    def search(self, key):
        """Search for a value by key."""
        index = self._hash(key)

        # Traverse the chain at this index
        for k, v in self.table[index]:
            if k == key:
                print(f"Found: {key} -> {v}")
                return v

        print(f"Key '{key}' not found")
        return None

    def delete(self, key):
        """Delete a key-value pair from the hash table."""
        index = self._hash(key)

        # Find and remove the key from the chain
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index].pop(i)
                print(f"Deleted: {key}")
                return True

        print(f"Key '{key}' not found for deletion")
        return False

    def display(self):
        """Display all key-value pairs in the hash table."""
```

```
            if k == key:
                self.table[index].pop(i)
                print(f"Deleted: {key}")
                return True

        print(f"Key '{key}' not found for deletion")
        return False

    def display(self):
        """Display all key-value pairs in the hash table."""
        print("\nHash Table Contents:")
        for i, chain in enumerate(self.table):
            if chain:
                print(f"Index {i}: {chain}")


# Sample Usage
if __name__ == "__main__":
    ht = HashTable(5)

    # Insert operations
    ht.insert("name", "Alice")
    ht.insert("age", 25)
    ht.insert("city", "NYC")
    ht.insert("name", "Bob")  # Update existing key

    # Search operations
    ht.search("name")
    ht.search("age")
    ht.search("country")

    # Display table
    ht.display()

    # Delete operations
    ht.delete("age")
    ht.delete("country")

    ht.display()
```

Output:



Task:6

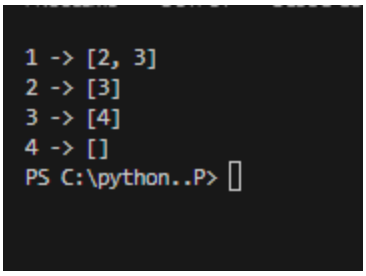
Prompt:implement a graph using an adjacency list.Sample Input Code:class Graph:

Pass Expected Output:• Graph with methods to add vertices, add edges, and display connections.

Code:

```python
class Graph:
    def __init__(self):
        self.graph = {}

    def add_vertex(self, vertex):
        """Add a vertex to the graph"""
        if vertex not in self.graph:
            self.graph[vertex] = []

    def add_edge(self, vertex1, vertex2):
        """Add an edge between two vertices"""
        if vertex1 not in self.graph:
            self.add_vertex(vertex1)
        if vertex2 not in self.graph:
            self.add_vertex(vertex2)

        self.graph[vertex1].append(vertex2)

    def display(self):
        """Display the graph adjacency list"""
        for vertex in self.graph:
            print(f"{vertex} -> {self.graph[vertex]}")


# Example usage
if __name__ == "__main__":
    g = Graph()

    # Add vertices
    g.add_vertex(1)
    g.add_vertex(2)
    g.add_vertex(3)
    g.add_vertex(4)

    # Add edges
    g.add_edge(1, 2)
    g.add_edge(1, 3)
    g.add_edge(2, 3)
    g.add_edge(3, 4)

    # Display graph
    print("Graph Adjacency List:")
    g.display()
```

Output:



Task 6:

Prompt: implement a priority queue using Python’s heapq module.Sample Input Code:class PriorityQueue:pass Expected Output:• Implementation with enqueue (priority), dequeue (highest priority),and display methods.

Code:

sk 1.py > ...

```python
import heapq

class PriorityQueue:
    def __init__(self):
        self.heap = []

    def enqueue(self, item, priority):
        """Add item with priority (lower number = higher priority)"""
        heapq.heappush(self.heap, (priority, item))

    def dequeue(self):
        """Remove and return highest priority item"""
        if self.heap:
            priority, item = heapq.heappop(self.heap)
            return item
        return None

    def display(self):
        """Display all items in priority order"""
        if not self.heap:
            print("Queue is empty")
            return
        sorted_heap = sorted(self.heap)
        for priority, item in sorted_heap:
            print(f"Priority: {priority}, Item: {item}")


# Example usage
pq = PriorityQueue()
pq.enqueue("Task A", 3)
pq.enqueue("Task B", 1)
pq.enqueue("Task C", 2)

print("Queue contents:")
pq.display()

print("\nDequeuing items:")
while pq.heap:
    item = pq.dequeue()
```

Output:

```
PS C:\python..P> & C:/Users/ARKAN/AppDat
Queue contents:
Priority: 1, Item: Task B
Priority: 2, Item: Task C
Priority: 3, Item: Task A

Dequeuing items:
Dequeued: Task B
Dequeued: Task C
Dequeued: Task A
PS C:\python..P>
```

Task 8:

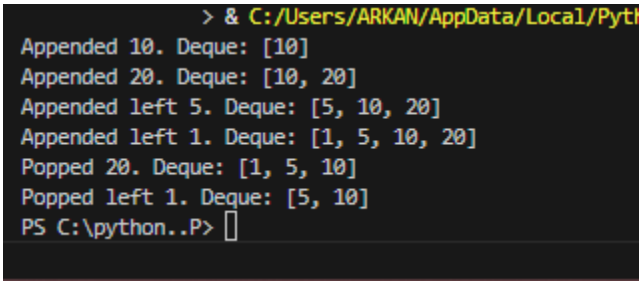
Prompt: implement a double-ended queue using collections.deque.Sample Input Code:

class DequeDS:pass Expected Output:• Insert and remove from both ends with docstrings.

Code:

```python
class DequeDS:

    def __init__(self):
        """Initialize an empty deque."""
        self.items = deque()

    def append(self, item):
        """
        Insert an item at the right end of the deque.

        Args:
            item: The element to insert
        """
        self.items.append(item)
        print(f"Appended {item}. Deque: {list(self.items)}")

    def appendleft(self, item):
        """
        Insert an item at the left end of the deque.

        Args:
            item: The element to insert
        """
        self.items.appendleft(item)
        print(f"Appended left {item}. Deque: {list(self.items)}")

    def pop(self):
        """Remove and return an item from the right end of the deque."""
        if self.items:
            item = self.items.pop()
            print(f"Popped {item}. Deque: {list(self.items)}")
            return item
        print("Deque is empty!")
        return None

    def popleft(self):
        """Remove and return an item from the left end of the deque."""
        if self.items:
            item = self.items.popleft()
            print(f"Popped left {item}. Deque: {list(self.items)}")
            return item
        print("Deque is empty!")
        return None


# Test Output
if __name__ == "__main__":
    dq = DequeDS()
    dq.append(10)
    dq.append(20)
    dq.appendleft(5)
    dq.appendleft(1)
    dq.pop()
    dq.popleft()
```

Output:



Task 9:

Prompt: Create a Campus Resource Management System design in Python.

The system should include these features:

1. Student Attendance Tracking – Daily log of students entering and exiting campus.

2. Event Registration System – Manage event participants with quick search and removal.

3. Library Book Borrowing – Track available books and due dates.

4. Bus Scheduling System – Maintain bus routes and stop connections.

5. Cafeteria Order Queue – Serve students in the order they arrive.

Tasks:

1. For each feature, choose the most appropriate data structure from:

   - Stack

   - Queue

   - Priority Queue

   - Linked List

   - Binary Search Tree (BST)

   - Graph

   - Hash Table

   - Deque

2. Create a table mapping:

   Feature → Chosen Data Structure → 2–3 sentence justification.

3. Implement ONE selected feature as a working Python program.

   - Include comments

   - Include proper docstrings

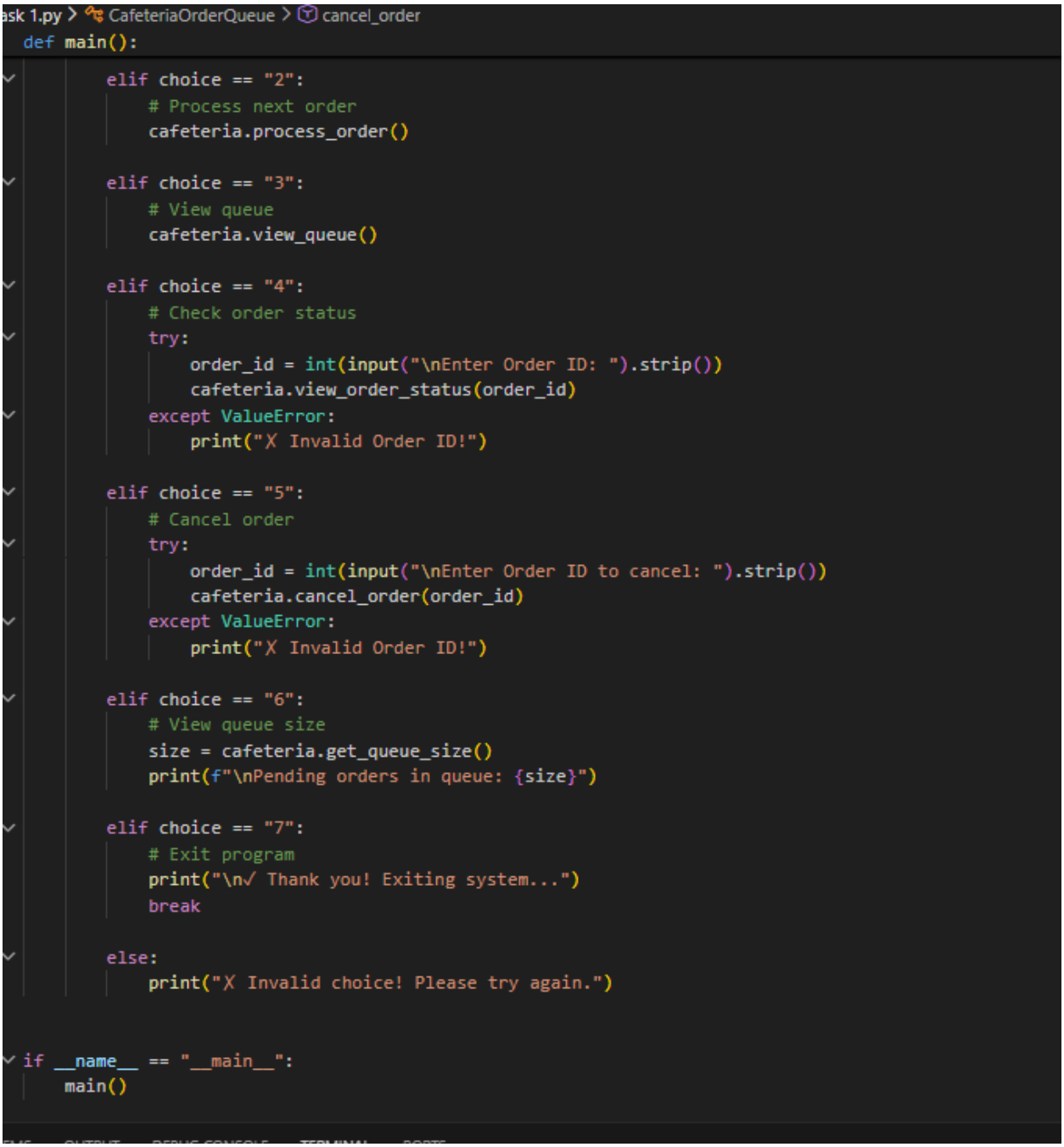
   - Follow clean coding practices

- Make it menu-driven (user input-based)

Output Requirements:

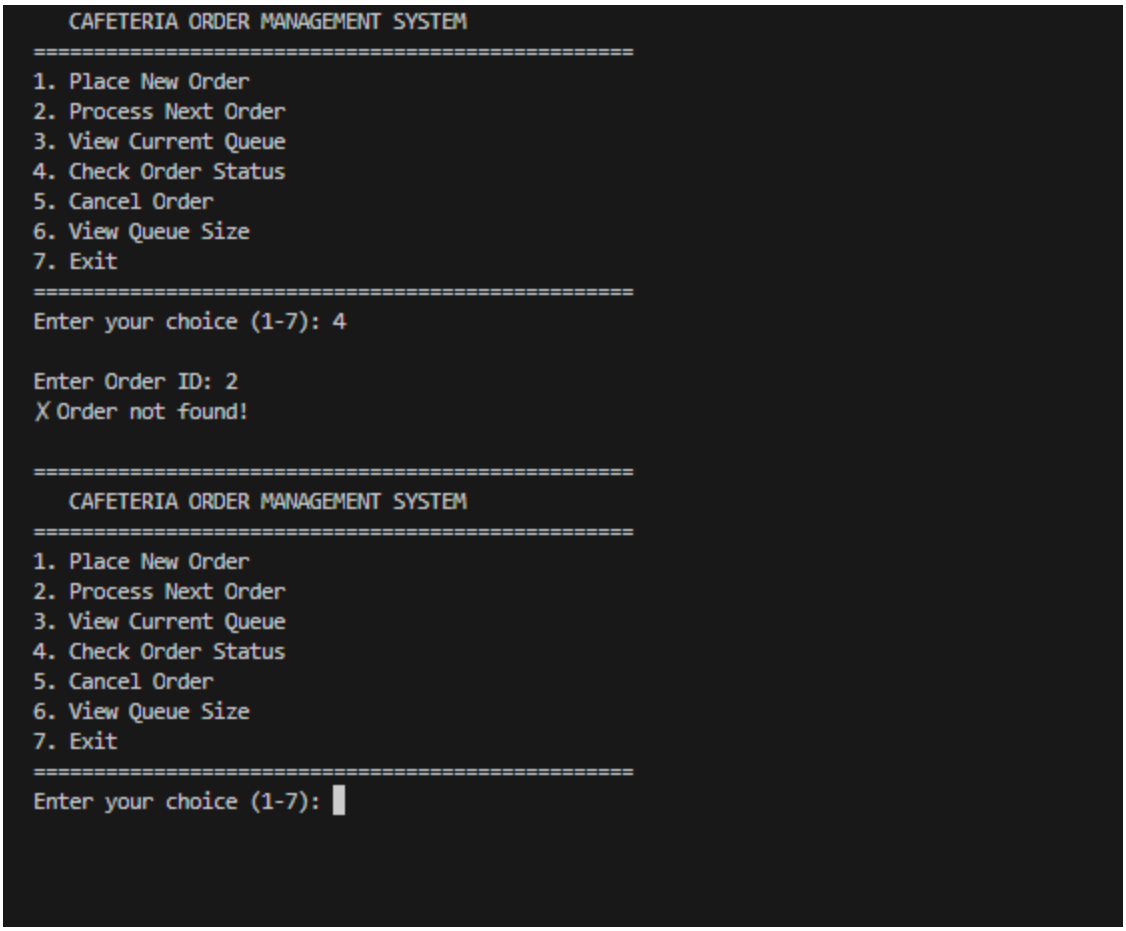
- Show the table first.

- Then provide the complete Python implementation.



```
class CafeteriaOrderQueue:
    def process_order(self):
        order.status = "Completed"
        self.completed_orders.append(order)
        print(f"✓ Order processed: {order}")
        return order

    def view_queue(self):
        """Display all pending orders in the queue."""
        if not self.queue:
            print("✗ Queue is empty!")
            return

        print("\n--- Current Queue ---")
        for idx, order in enumerate(self.queue, 1):
            print(f"{idx}. {order}")
        print(f"Total pending orders: {len(self.queue)}\n")

    def view_order_status(self, order_id):

        # Check pending orders
        for order in self.queue:
            if order.order_id == order_id:
                print(f"✓ {order}")
                return

        # Check completed orders
        for order in self.completed_orders:
            if order.order_id == order_id:
                print(f"✓ {order}")
                return

        print("✗ Order not found!")

    def cancel_order(self, order_id):

        for order in list(self.queue):
            if order.order_id == order_id:
                self.queue.remove(order)
                print(f"✓ Order #{order_id} cancelled!")
                return
```

Output:



Task:10:

Prompt: Design a Smart Online Shopping System in Python.

The system should include these features:

1. Shopping Cart Management – Add and remove products dynamically.

2. Order Processing System – Process orders in the order they are placed.

3. Top-Selling Products Tracker – Rank products by sales count.

4. Product Search Engine – Fast lookup of products using product ID.

5. Delivery Route Planning – Connect warehouses and delivery locations.

Tasks:

1. For each feature, choose the most appropriate data structure from:

- Stack
- Queue
- Priority Queue
- Linked List
- Binary Search Tree (BST)
- Graph
- Hash Table
- Deque

2. Create a table mapping:

Feature → Chosen Data Structure → 2–3 sentence justification.

3. Implement ONE selected feature as a working Python program.

- Include proper comments
- Include docstrings
- Follow clean coding practices
- Make it menu-driven (user input-based)

Output Requirements:

- First show the table.
- Then provide the complete Python implementation.

Code:

```python
from collections import deque

"""
Smart Online Shopping System - Shopping Cart Management
Uses Deque (Double-Ended Queue) for efficient cart operations.
"""


class Product:
    """Represents a product in the shopping system."""

    def __init__(self, product_id, name, price, quantity):
        """
        Initialize a product.

        Args:
            product_id (int): Unique product identifier
            name (str): Product name
            price (float): Product price
            quantity (int): Available quantity
        """
        self.product_id = product_id
        self.name = name
        self.price = price
        self.quantity = quantity

    def __str__(self):
        return f"ID: {self.product_id} | {self.name} | ${self.price:.2f} | Stock: {self.quantity}"


class ShoppingCart:
    """Manages shopping cart using Deque for efficient operations."""

    def __init__(self):
        """Initialize an empty shopping cart."""
        self.cart = deque()

    def add_product(self, product, qty):
        """
        Add a product to the cart.

        Args:
            product (Product): Product object to add
            qty (int): Quantity to add
        """
        if qty > product.quantity:
            print(f"❌ Only {product.quantity} units available!")
            return False

        self.cart.append({"product": product, "qty": qty})
        print(f"✓ Added {qty}x {product.name} to cart")
```

#task 1.py > ...

```python
class ShoppingCart:

    def remove_product(self):
        """Remove the last added product from cart (LIFO behavior)."""
        if not self.cart:
            print("❌ Cart is empty!")
            return

        removed = self.cart.pop()
        print(f"✓ Removed {removed['qty']}x {removed['product'].name} from cart")

    def remove_first_product(self):
        """Remove the first added product from cart (FIFO behavior)."""
        if not self.cart:
            print("❌ Cart is empty!")
            return

        removed = self.cart.popleft()
        print(f"✓ Removed {removed['qty']}x {removed['product'].name} from cart")

    def view_cart(self):
        """Display all items in the cart."""
        if not self.cart:
            print("🛒 Cart is empty!")
            return

        print("\n" + "="*60)
        print("🛒 SHOPPING CART")
        print("="*60)
        total = 0
        for i, item in enumerate(self.cart, 1):
            product = item["product"]
            qty = item["qty"]
            subtotal = product.price * qty
            total += subtotal
            print(f"{i}. {product.name} x{qty} @ ${product.price:.2f} = ${subtotal:.2f}")

        print("="*60)
        print(f"💰 Total: ${total:.2f}")
        print("="*60 + "\n")

    def get_total_price(self):
        """Calculate total cart price."""
        return sum(item["product"].price * item["qty"] for item in self.cart)

    def clear_cart(self):
        """Clear all items from cart."""
        self.cart.clear()
        print("✓ Cart cleared!")
```

#task 1.py > ...

```
def main():
    print("📦 AVAILABLE PRODUCTS:")
    print("="*60)
    for pid, product in products.items():
        print(f"{pid}. {product}")

    while True:
        display_menu()
        choice = input("Enter your choice (1-7): ").strip()

        if choice == "1":
            try:
                pid = int(input("Enter product ID: "))
                qty = int(input("Enter quantity: "))

                if pid in products:
                    cart.add_product(products[pid], qty)
                else:
                    print("❌ Product not found!")
            except ValueError:
                print("❌ Invalid input!")

        elif choice == "2":
            cart.remove_product()

        elif choice == "3":
            cart.remove_first_product()

        elif choice == "4":
            cart.view_cart()

        elif choice == "5":
            cart.view_cart()
            if cart.cart:
                print(f"✓ Order processed! Total: ${cart.get_total_price():.2f}")
                cart.clear_cart()

        elif choice == "6":
            cart.clear_cart()

        elif choice == "7":
            print("\n👋 Thank you for shopping! Goodbye!\n")
            break

        else:
            print("❌ Invalid choice!")

```

Output:

```
5. Cancel Order
6. View Queue Size
7. Exit
==================================================
Enter your choice (1-7): 1

Enter student name: arkan
Enter items (comma-separated): burger pizza
✓ Order placed successfully! Order ID: 1

==================================================
   CAFETERIA ORDER MANAGEMENT SYSTEM
==================================================
1. Place New Order
2. Process Next Order
3. View Current Queue
4. Check Order Status
5. Cancel Order
6. View Queue Size
7. Exit
==================================================
Enter your choice (1-7): 7

✓ Thank you! Exiting system...
PS C:\python..P>
```