School of Computer Science and Artificial Intelligence

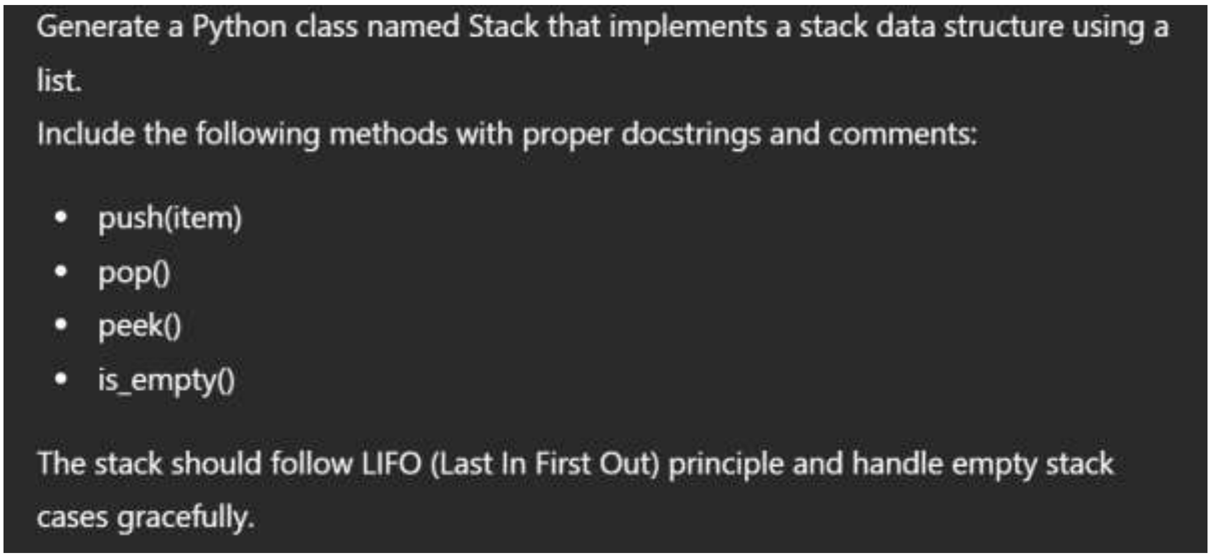
Lab Assignment # 11.5

Program	: B. Tech (CSE)
Specialization	:AIML
Course Title	: AI Assisted Coding
Course Code	: 23CS002PC304
Semester	: VI
Academic Session	: 2025-2026
Name of Student	: Shiva Charan
Enrollment No.	: 2303A52160
Batch No.	: 34
Date	:20/02/26

Lab 11 – Data Structures with AI: Implementing Fundamental Structures

Task 1 – Stack Implementation

Prompt:



Generate a Python class named Stack that implements a stack data structure using a list.
Include the following methods with proper docstrings and comments:

- push(item)
- pop()
- peek()
- is_empty()

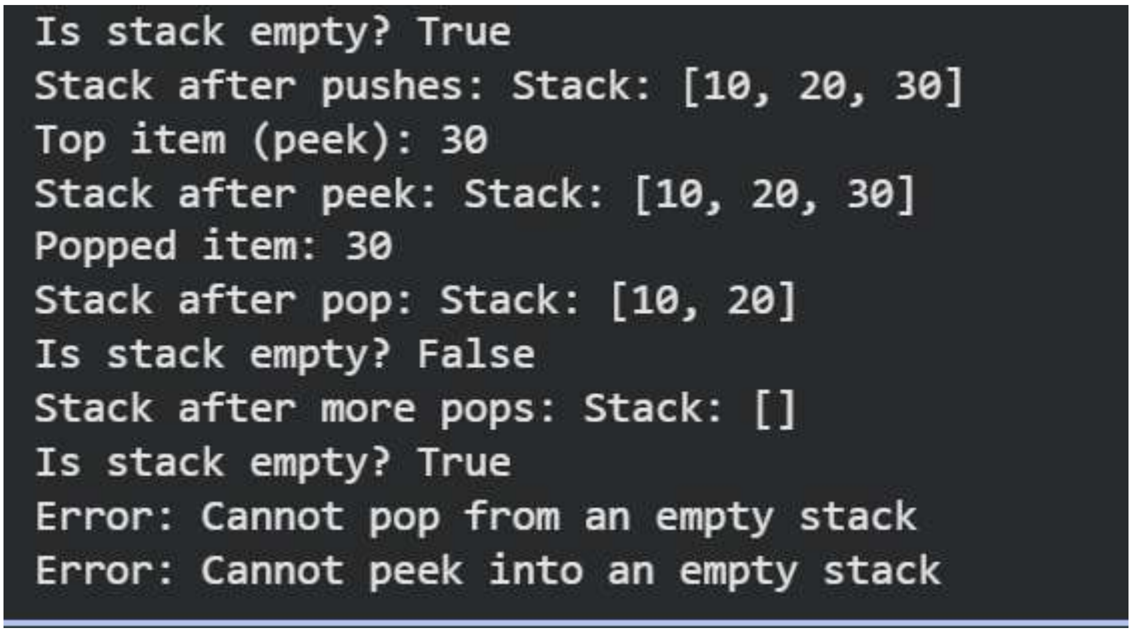
The stack should follow LIFO (Last In First Out) principle and handle empty stack cases gracefully.

Code:

```
class Stack:
    """A simple implementation of a Stack data structure using a Python list."""

    def __init__(self):
        """Initializes an empty Stack."""
        self._items = []  # Internal list to store stack elements

    def push(self, item):
        """Adds an item to the top of the stack (LIFO).

        Args:
            item: The item to be added to the stack.
        """
        self._items.append(item)

    def pop(self):
        """Removes and returns the item from the top of the stack.

        Returns:
            The item removed from the top of the stack.

        Raises:
            IndexError: If the stack is empty when pop is called.
        """
        if self.is_empty():
            raise IndexError("Cannot pop from an empty stack")
        return self._items.pop() # list's pop() removes and returns the last item

    def peek(self):
        """Returns the item from the top of the stack without removing it.

        Returns:
            The item at the top of the stack.

        Raises:
            IndexError: If the stack is empty when peek is called.
        """
        if self.is_empty():
            raise IndexError("Cannot peek into an empty stack")
        return self._items[-1] # Return the last item without removing it

    def is_empty(self):
        """Checks if the stack is empty.

        Returns:
            True if the stack is empty, False otherwise.
        """
        return len(self._items) == 0

    def __len__(self):
        """Returns the number of items in the stack."""
        return len(self._items)

    def __str__(self):
        """Returns a string representation of the stack."""
        return f"Stack: {self._items}"

# Example usage:
my_stack = Stack()
print(f"Is stack empty? {my_stack.is_empty()}") # True

my_stack.push(10)
my_stack.push(20)
my_stack.push(30)
```

```
print(f"Stack after pushes: {my_stack}") # Stack: [10, 20, 30]
print(f"Top item (peek): {my_stack.peek()}") # 30
print(f"Stack after peek: {my_stack}") # Stack: [10, 20, 30]

popped_item = my_stack.pop()
print(f"Popped item: {popped_item}") # 30
print(f"Stack after pop: {my_stack}") # Stack: [10, 20]

print(f"Is stack empty? {my_stack.is_empty()}") # False

my_stack.pop()
my_stack.pop()
print(f"Stack after more pops: {my_stack}") # Stack: []
print(f"Is stack empty? {my_stack.is_empty()}") # True

# Demonstrating error handling for empty stack
try:
    my_stack.pop()
except IndexError as e:
    print(f"Error: {e}") # Error: Cannot pop from an empty stack

try:
    my_stack.peek()
except IndexError as e:
    print(f"Error: {e}") # Error: Cannot peek into an empty stack
```

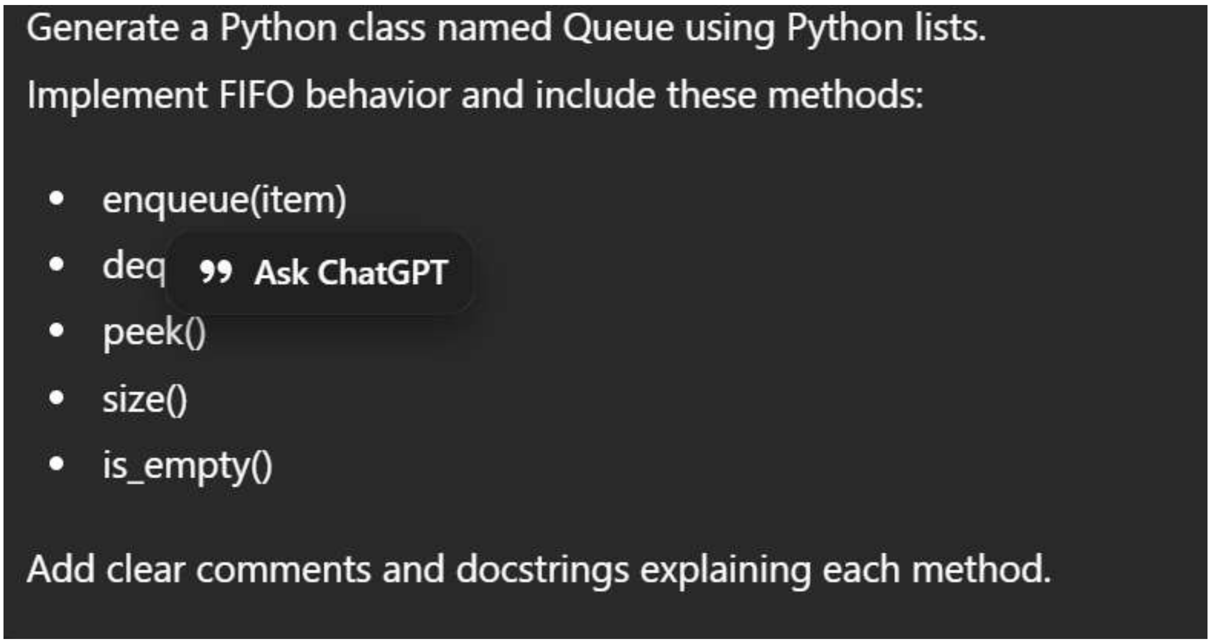
Output:



```
Is stack empty? True
Stack after pushes: Stack: [10, 20, 30]
Top item (peek): 30
Stack after peek: Stack: [10, 20, 30]
Popped item: 30
Stack after pop: Stack: [10, 20]
Is stack empty? False
Stack after more pops: Stack: []
Is stack empty? True
Error: Cannot pop from an empty stack
Error: Cannot peek into an empty stack
```

Task 2 – Queue Implementation

Prompt:



Generate a Python class named Queue using Python lists.

Implement FIFO behavior and include these methods:

- enqueue(item)
- deq
- peek()
- size()
- is_empty()

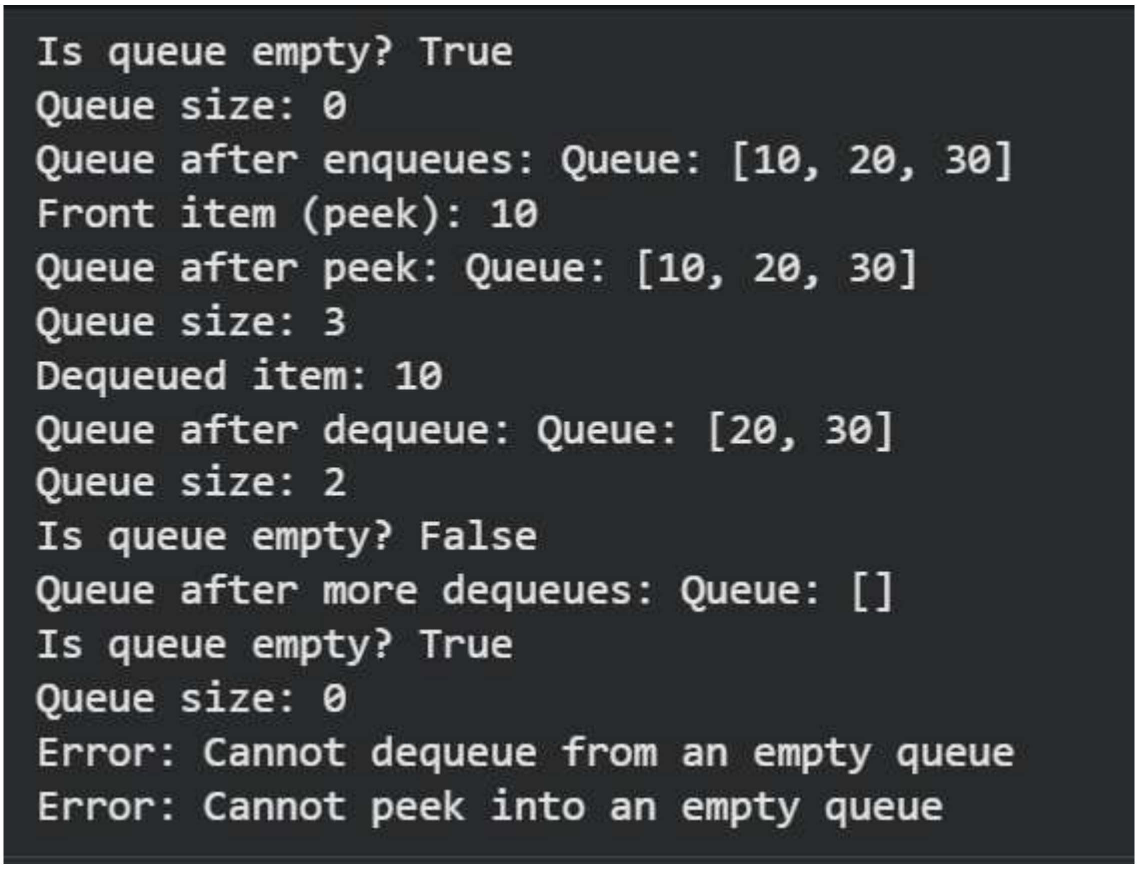
Add clear comments and docstrings explaining each method.

Code:

```
class Queue:
    """A simple implementation of a Queue data structure using a Python list."""
```

```
def __init__(self):
    """Initializes an empty Queue."""
    self._items = []  # Internal list to store queue elements

def enqueue(self, item):
    """Adds an item to the rear of the queue (FIFO).

    Args:
        item: The item to be added to the queue.
    """
    self._items.append(item)

def dequeue(self):
    """Removes and returns the item from the front of the queue.

    Returns:
        The item removed from the front of the queue.

    Raises:
        IndexError: If the queue is empty when dequeue is called.
    """
    if self.is_empty():
        raise IndexError("Cannot dequeue from an empty queue")
    return self._items.pop(0) # list's pop(0) removes and returns the first item

def peek(self):
    """Returns the item from the front of the queue without removing it.

    Returns:
        The item at the front of the queue.

    Raises:
        IndexError: If the queue is empty when peek is called.
    """
    if self.is_empty():
        raise IndexError("Cannot peek into an empty queue")
    return self._items[0] # Return the first item without removing it

def is_empty(self):
    """Checks if the queue is empty.

    Returns:
        True if the queue is empty, False otherwise.
    """
    return len(self._items) == 0

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

def __str__(self):
    """Returns a string representation of the queue."""
    return f"Queue: {self._items}"

# Example usage:
my_queue = Queue()
print(f"Is queue empty? {my_queue.is_empty()}") # True
print(f"Queue size: {my_queue.size()}") # 0

my_queue.enqueue(10)
my_queue.enqueue(20)
my_queue.enqueue(30)
print(f"Queue after enqueues: {my_queue}") # Queue: [10, 20, 30]
print(f"Front item (peek): {my_queue.peek()}") # 10
print(f"Queue after peek: {my_queue}") # Queue: [10, 20, 30]
print(f"Queue size: {my_queue.size()}") # 3

dequeued_item = my_queue.dequeue()
print(f"Dequeued item: {dequeued_item}") # 10
print(f"Queue after dequeue: {my_queue}") # Queue: [20, 30]
print(f"Queue size: {my_queue.size()}") # 2

print(f"Is queue empty? {my_queue.is_empty()}") # False

my_queue.dequeue()
my_queue.dequeue()
print(f"Queue after more dequeues: {my_queue}") # Queue: []
print(f"Is queue empty? {my_queue.is_empty()}") # True
print(f"Queue size: {my_queue.size()}") # 0


# Demonstrating error handling for empty queue
try:
    my_queue.dequeue()
except IndexError as e:
    print(f"Error: {e}") # Error: Cannot dequeue from an empty queue


try:
    my_queue.peek()
except IndexError as e:
    print(f"Error: {e}") # Error: Cannot peek into an empty queue
```

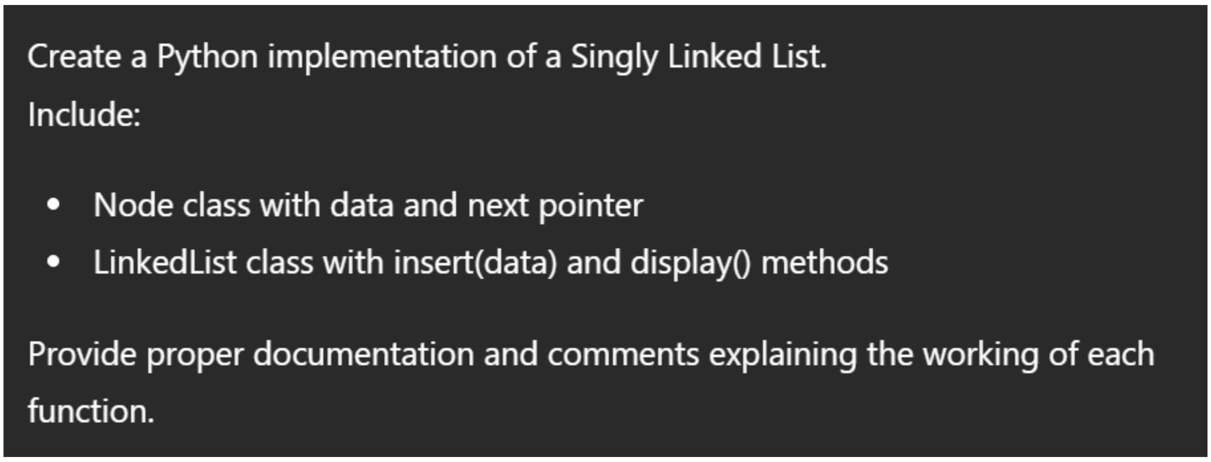
Output:



```
Is queue empty? True
Queue size: 0
Queue after enqueues: Queue: [10, 20, 30]
Front item (peek): 10
Queue after peek: Queue: [10, 20, 30]
Queue size: 3
Dequeued item: 10
Queue after dequeue: Queue: [20, 30]
Queue size: 2
Is queue empty? False
Queue after more dequeues: Queue: []
Is queue empty? True
Queue size: 0
Error: Cannot dequeue from an empty queue
Error: Cannot peek into an empty queue
```

Task 3 – Singly Linked List

Prompt:



Create a Python implementation of a Singly Linked List.
Include:

- Node class with data and next pointer
- LinkedList class with insert(data) and display() methods

Provide proper documentation and comments explaining the working of each function.

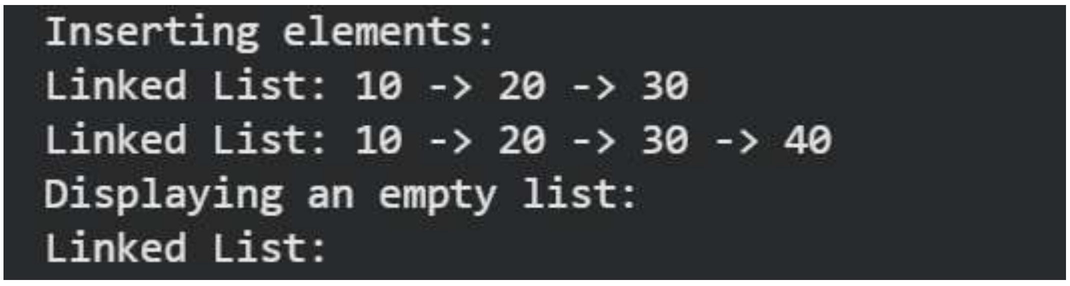
Code:

```
class Node:
    """Represents a node in a singly linked list."""


    def __init__(self, data):
        """Initializes a new node with data and sets the next pointer to None.


        Args:
            data: The data to be stored in the node.
        """
        self.data = data
        self.next = None  # Pointer to the next node, initially None


class LinkedList:
    """Implements a singly linked list data structure."""


    def __init__(self):
```

```
    """Initializes an empty linked list with the head pointer set to None."""
    self.head = None  # The head of the list, initially None

def insert(self, data):
    """Inserts a new node with the given data at the end of the linked list.

    If the list is empty, the new node becomes the head.

    Args:
        data: The data to be inserted into the new node.
    """
    new_node = Node(data)
    if self.head is None:
        self.head = new_node  # If list is empty, new node is the head
    else:
        current = self.head
        while current.next:
            current = current.next  # Traverse to the last node
        current.next = new_node  # Link the new node to the end

def display(self):
    """Prints the data of all nodes in the linked list from head to tail."""
    elements = []
    current = self.head
    while current:
        elements.append(current.data)
```

```
            current = current.next
        print("Linked List: " + " -> ".join(map(str, elements)))

# Example Usage:
my_list = LinkedList()
print("Inserting elements:")
my_list.insert(10)
my_list.insert(20)
my_list.insert(30)
my_list.display()

my_list.insert(40)
my_list.display()

empty_list = LinkedList()
print("Displaying an empty list:")
empty_list.display()
```

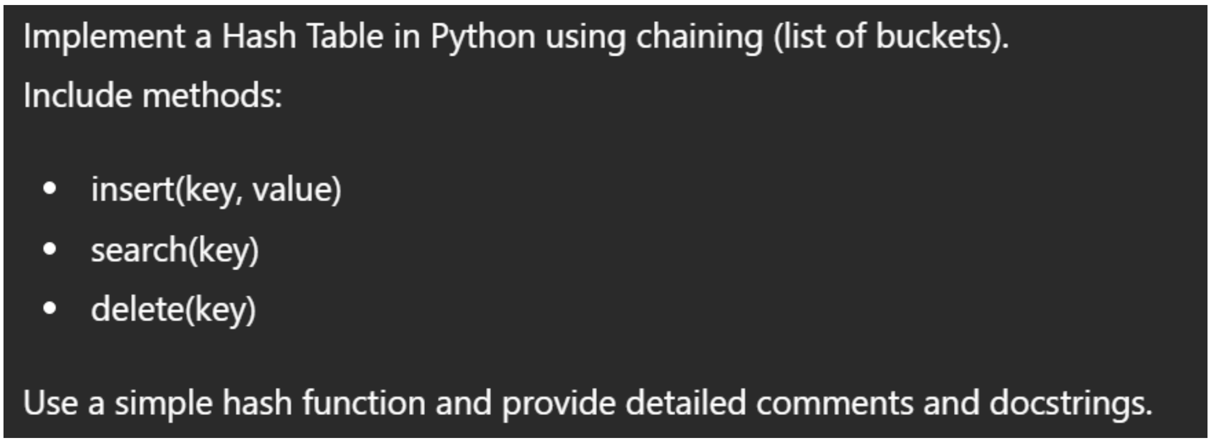
Output:



```
Inserting elements:
Linked List: 10 -> 20 -> 30
Linked List: 10 -> 20 -> 30 -> 40
Displaying an empty list:
Linked List:
```

Task 4 – Hash Table

Prompt:



Implement a Hash Table in Python using chaining (list of buckets).

Include methods:

- insert(key, value)
- search(key)
- delete(key)

Use a simple hash function and provide detailed comments and docstrings.

Code:

```
class HashTable:
    """A simple Hash Table implementation using chaining for collision
resolution."""


    def __init__(self, size=10):
        """Initializes the Hash Table with a given size.


        Args:
            size (int): The number of buckets in the hash table.
        """
        self.size = size
        self.table = [[] for _ in range(self.size)]  # Initialize with empty buckets
(lists)


    def _hash(self, key):
        """A simple hash function that returns an index for the given key.
        Args:
            key: The key to be hashed.

        Returns:
            int: The index within the hash table's buckets.
        """
        return hash(key) % self.size

    def insert(self, key, value):
        """Inserts a key-value pair into the hash table.

        If the key already exists, its value is updated.

        Args:
            key: The key to be inserted.
            value: The value associated with the key.
        """
        index = self._hash(key)
        bucket = self.table[index]

        # Check if the key already exists in the bucket
        for i, (k, v) in enumerate(bucket):
            if k == key:
                bucket[i] = (key, value)  # Update existing key's value
                print(f"Updated key '{key}' at index {index} with new value
'{value}'.")
```

```
            return

        # If key does not exist, add the new key-value pair
        bucket.append((key, value))
        print(f"Inserted ('{key}', '{value}') at index {index}.")

    def search(self, key):
        """Searches for a key in the hash table and returns its associated value.

        Args:
            key: The key to search for.

        Returns:
            any: The value associated with the key, or None if the key is not found.
        """
        index = self._hash(key)
        bucket = self.table[index]

        for k, v in bucket:
            if k == key:
                print(f"Found key '{key}' with value '{v}' at index {index}.")
                return v
        print(f"Key '{key}' not found in hash table.")
        return None

    def delete(self, key):
```

```
    """Deletes a key-value pair from the hash table.

    Args:
        key: The key to be deleted.

    Returns:
        bool: True if the key was deleted, False if not found.
    """
    index = self._hash(key)
    bucket = self.table[index]

    for i, (k, v) in enumerate(bucket):
        if k == key:
            del bucket[i]  # Remove the item from the bucket
            print(f"Deleted key '{key}' from index {index}.")
            return True
    print(f"Key '{key}' not found for deletion.")
    return False

def display(self):
    """Prints the current state of the hash table."""
    print("\n--- Hash Table Contents ---")
    for i, bucket in enumerate(self.table):
        print(f"Bucket {i}: {bucket}")
    print("---------------------------")
```

```
# Example Usage:
ht = HashTable(size=5)
ht.display()

print("\n--- Inserting elements ---")
ht.insert("apple", 10)
ht.insert("banana", 20)
ht.insert("cherry", 30)
ht.insert("date", 40)
ht.insert("elderberry", 50)
ht.insert("fig", 60) # This might cause a collision depending on hash function
ht.display()

print("\n--- Searching for elements ---")
ht.search("banana")
ht.search("cherry")
ht.search("grape") # Not found

print("\n--- Updating an element ---")
ht.insert("apple", 100) # Update existing key
ht.display()

print("\n--- Deleting elements ---")
ht.delete("date")
ht.delete("banana")
ht.delete("kiwi") # Not found
```

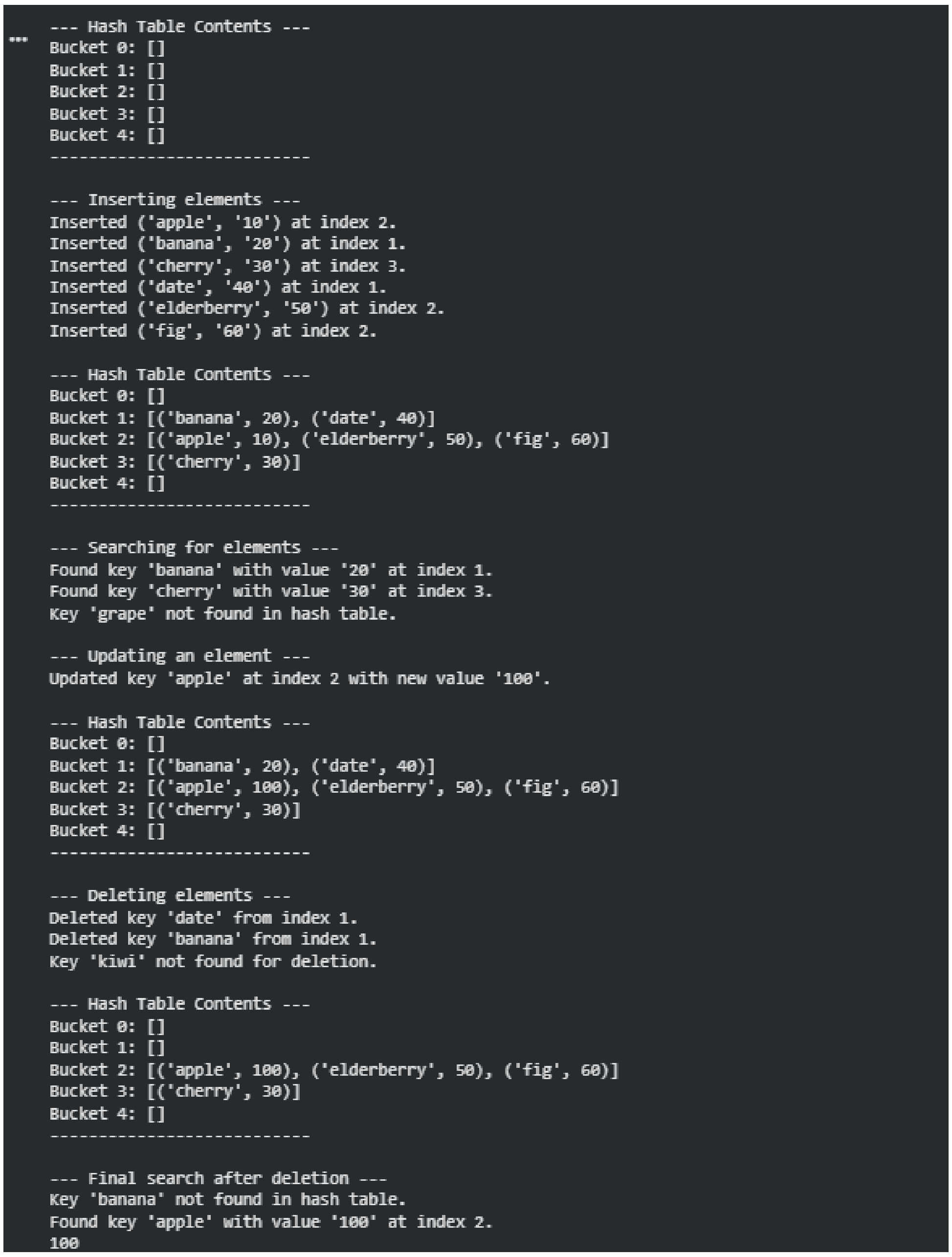
ht.display()

print("\n--- Final search after deletion ---")

ht.search("banana") # Should not be found

ht.search("apple") # Should be found

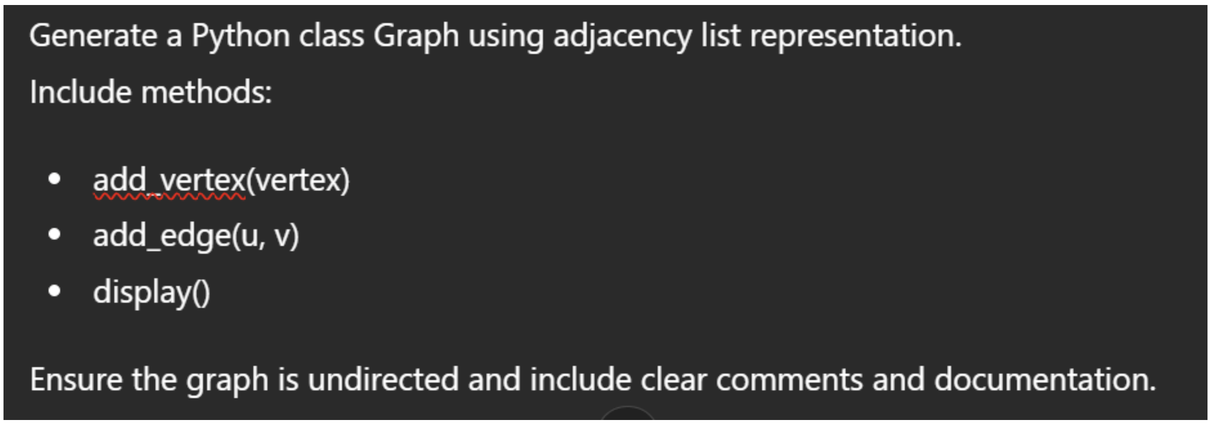
Output:



```
--- Hash Table Contents ---
Bucket 0: []
Bucket 1: []
Bucket 2: []
Bucket 3: []
Bucket 4: []
---------------------------

--- Inserting elements ---
Inserted ('apple', '10') at index 2.
Inserted ('banana', '20') at index 1.
Inserted ('cherry', '30') at index 3.
Inserted ('date', '40') at index 1.
Inserted ('elderberry', '50') at index 2.
Inserted ('fig', '60') at index 2.

--- Hash Table Contents ---
Bucket 0: []
Bucket 1: [('banana', 20), ('date', 40)]
Bucket 2: [('apple', 10), ('elderberry', 50), ('fig', 60)]
Bucket 3: [('cherry', 30)]
Bucket 4: []
---------------------------

--- Searching for elements ---
Found key 'banana' with value '20' at index 1.
Found key 'cherry' with value '30' at index 3.
Key 'grape' not found in hash table.

--- Updating an element ---
Updated key 'apple' at index 2 with new value '100'.

--- Hash Table Contents ---
Bucket 0: []
Bucket 1: [('banana', 20), ('date', 40)]
Bucket 2: [('apple', 100), ('elderberry', 50), ('fig', 60)]
Bucket 3: [('cherry', 30)]
Bucket 4: []
---------------------------

--- Deleting elements ---
Deleted key 'date' from index 1.
Deleted key 'banana' from index 1.
Key 'kiwi' not found for deletion.

--- Hash Table Contents ---
Bucket 0: []
Bucket 1: []
Bucket 2: [('apple', 100), ('elderberry', 50), ('fig', 60)]
Bucket 3: [('cherry', 30)]
Bucket 4: []
---------------------------

--- Final search after deletion ---
Key 'banana' not found in hash table.
Found key 'apple' with value '100' at index 2.
100
```

Task 5 – Graph Representation

Prompt:



Generate a Python class Graph using adjacency list representation.

Include methods:

- add_vertex(vertex)
- add_edge(u, v)
- display()

Ensure the graph is undirected and include clear comments and documentation.

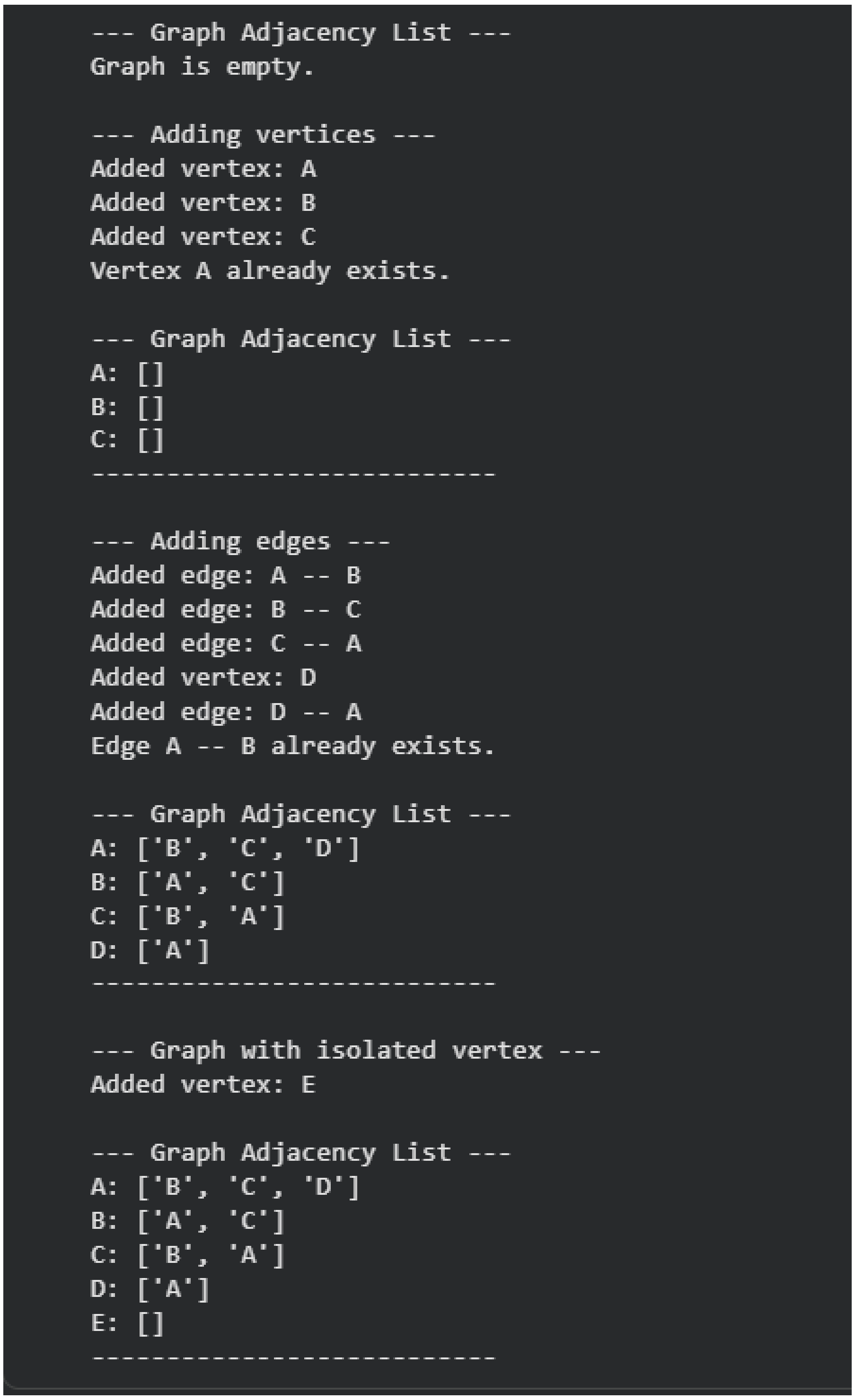
Code:

```
class Graph:
    """A simple undirected Graph implementation using an adjacency list."""


    def __init__(self):
        """Initializes an empty graph with no vertices or edges."""
        self.adj_list = {}


    def add_vertex(self, vertex):
        """Adds a vertex to the graph if it doesn't already exist.


        Args:
            vertex: The vertex to be added.
        """
        if vertex not in self.adj_list:
            self.adj_list[vertex] = []
            print(f"Added vertex: {vertex}")
        else:
            print(f"Vertex {vertex} already exists.")

    def add_edge(self, u, v):
        """Adds an undirected edge between two vertices u and v.

        If u or v do not exist, they are added to the graph first.

        Args:
            u: The first vertex of the edge.
            v: The second vertex of the edge.
        """
        # Ensure both vertices exist in the graph
        if u not in self.adj_list:
            self.add_vertex(u)
        if v not in self.adj_list:
            self.add_vertex(v)

        # Add edge (u, v) and (v, u) for an undirected graph
        if v not in self.adj_list[u]:
            self.adj_list[u].append(v)
            print(f"Added edge: {u} -- {v}")
        else:
            print(f"Edge {u} -- {v} already exists.")

        if u not in self.adj_list[v]: # Ensure the reverse edge is also added
```

```
            self.adj_list[v].append(u)

    def display(self):
        """Prints the adjacency list representation of the graph."""
        print("\n--- Graph Adjacency List ---")
        if not self.adj_list:
            print("Graph is empty.")
            return

        for vertex, neighbors in self.adj_list.items():
            print(f"{vertex}: {neighbors}")
        print("--------------------------")

# Example Usage:
g = Graph()
g.display()

print("\n--- Adding vertices ---")
g.add_vertex('A')
g.add_vertex('B')
g.add_vertex('C')
g.add_vertex('A') # Try adding existing vertex
g.display()

print("\n--- Adding edges ---")
g.add_edge('A', 'B')
```

```
g.add_edge('B', 'C')
g.add_edge('C', 'A')
g.add_edge('D', 'A') # D will be added automatically
g.add_edge('A', 'B') # Try adding existing edge
g.display()

print("\n--- Graph with isolated vertex ---")
g.add_vertex('E')
g.display()
```

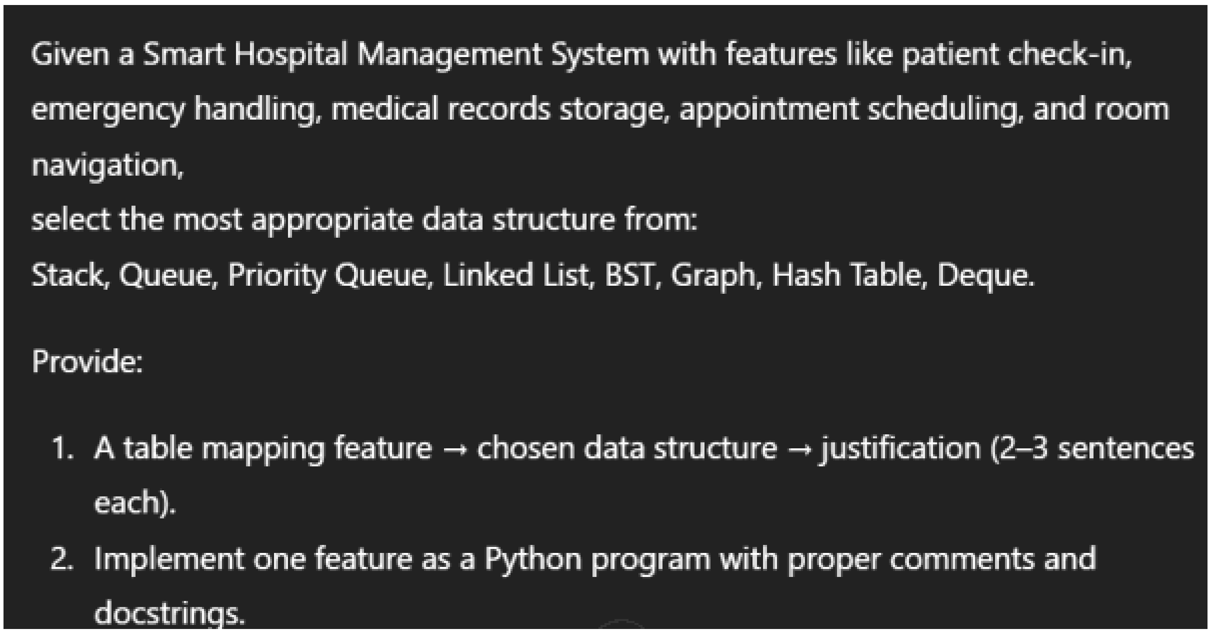
Output:



```
--- Graph Adjacency List ---
Graph is empty.

--- Adding vertices ---
Added vertex: A
Added vertex: B
Added vertex: C
Vertex A already exists.

--- Graph Adjacency List ---
A: []
B: []
C: []
---------------------------

--- Adding edges ---
Added edge: A -- B
Added edge: B -- C
Added edge: C -- A
Added vertex: D
Added edge: D -- A
Edge A -- B already exists.

--- Graph Adjacency List ---
A: ['B', 'C', 'D']
B: ['A', 'C']
C: ['B', 'A']
D: ['A']
---------------------------

--- Graph with isolated vertex ---
Added vertex: E

--- Graph Adjacency List ---
A: ['B', 'C', 'D']
B: ['A', 'C']
C: ['B', 'A']
D: ['A']
E: []
---------------------------
```

Task 6 – Smart Hospital Management System

Prompt:


Given a Smart Hospital Management System with features like patient check-in, emergency handling, medical records storage, appointment scheduling, and room navigation,
select the most appropriate data structure from:
Stack, Queue, Priority Queue, Linked List, BST, Graph, Hash Table, Deque.

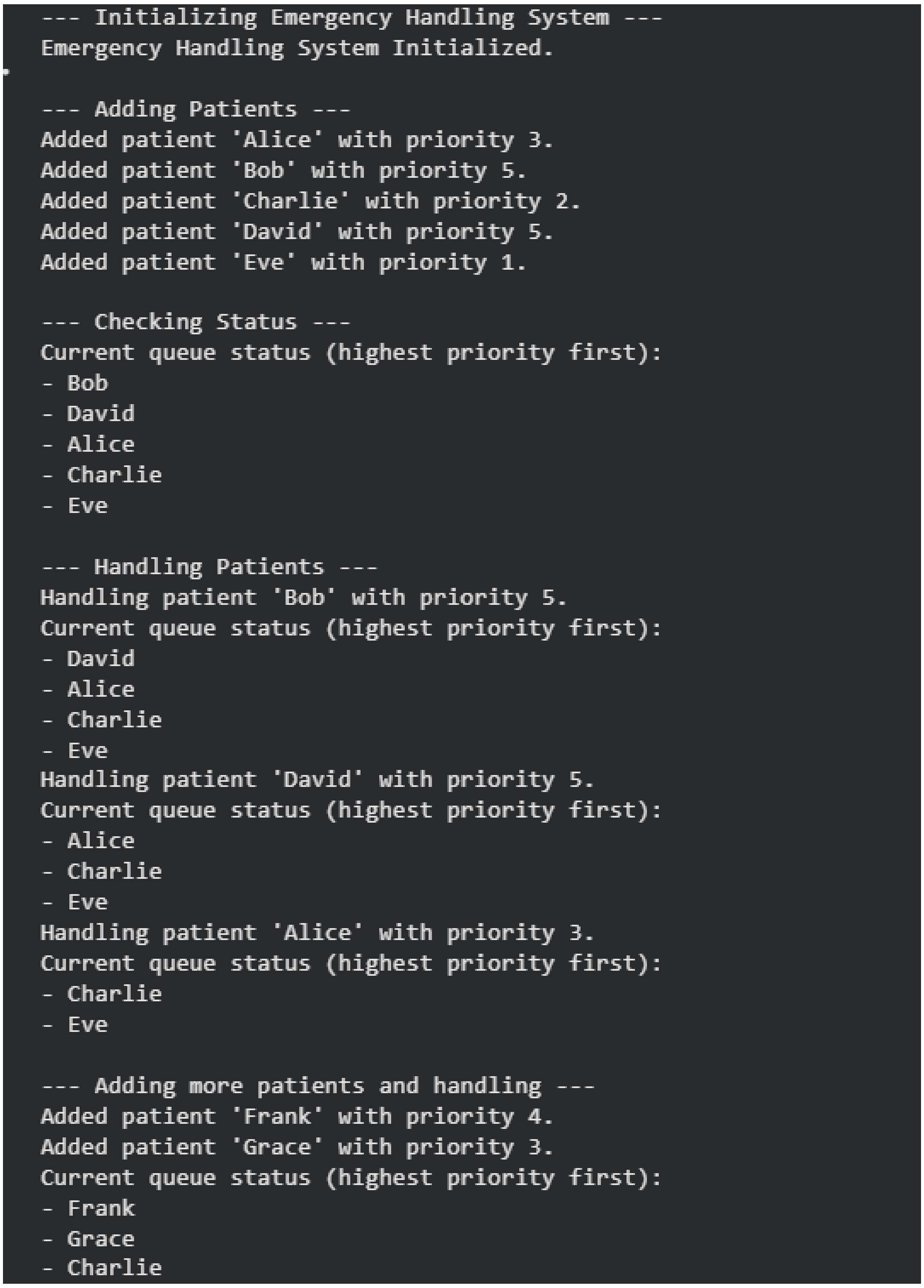
Provide:

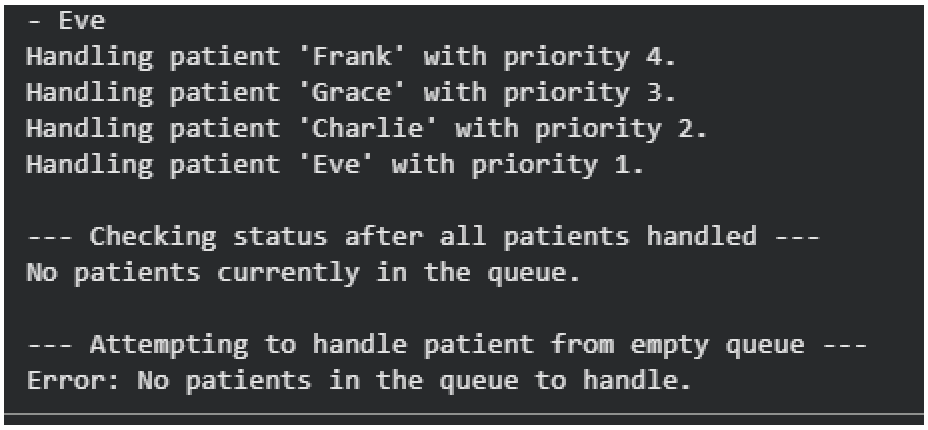
1. A table mapping feature → chosen data structure → justification (2–3 sentences each).
2. Implement one feature as a Python program with proper comments and docstrings.

Code:

```
class EmergencyHandling:
    """Manages emergency patient handling using a priority queue.


    Patients are added with a priority, and the system can handle the next
    highest priority patient or check the current queue status.
    Higher priority numbers indicate a more critical patient.
    """


    def __init__(self):
        """Initializes an empty emergency handling queue."""
        # Stores patients as tuples: (priority, patient_name)
        # Higher numbers mean higher priority
        self._patients = []
        print("Emergency Handling System Initialized.")


    def add_patient(self, patient_name, priority):
        """Adds a new patient to the emergency queue with a specified priority.
        Args:
            patient_name (str): The name of the patient.
            priority (int): The priority level of the patient. Higher numbers
                            indicate higher criticality (e.g., 5 > 1).
        """
        if not isinstance(priority, int) or priority < 0:
            print(f"Warning: Invalid priority '{priority}' for patient '{patient_name}'. Priority
must be a non-negative integer.")
            return
        self._patients.append((priority, patient_name))
        print(f"Added patient '{patient_name}' with priority {priority}.")

    def handle_next_patient(self):
        """Identifies, removes, and returns the patient with the highest priority.

        If the queue is empty, an IndexError is raised.

        Returns:
            str: The name of the patient with the highest priority.

        Raises:
            IndexError: If the emergency queue is empty.
        """
        if not self._patients:
            raise IndexError("No patients in the queue to handle.")

        # Find the patient with the highest priority
        highest_priority = -1
        patient_to_handle_index = -1
        for i, (priority, patient_name) in enumerate(self._patients):
            if priority > highest_priority:
                highest_priority = priority
                patient_to_handle_index = i
            # If priorities are equal, the patient who arrived earlier (lower index) is handled first
            # This is implicitly handled by iterating from start to end and only updating if >
highest_priority

        # Remove and return the patient
        # pop() returns the item at the specified index
        handled_patient = self._patients.pop(patient_to_handle_index)
        print(f"Handling patient '{handled_patient[1]}' with priority {handled_patient[0]}.")
        return handled_patient[1]

    def check_status(self):
        """Returns a list of patients in the queue, ordered by priority (highest first).

        Returns:
            list: A list of patient names, or a message if the queue is empty.
        """
        if not self._patients:
            return "No patients currently in the queue."

        # Sort patients by priority in descending order (highest priority first)
        # If priorities are equal, their original insertion order (stable sort) is maintained.
        sorted_patients = sorted(self._patients, key=lambda x: x[0], reverse=True)

        status_list = [patient_name for priority, patient_name in sorted_patients]
        print("Current queue status (highest priority first):")
        for patient in status_list:
            print(f"- {patient}")
        return status_list

# Example Usage:
print("\n--- Initializing Emergency Handling System ---")
emergency_queue = EmergencyHandling()

print("\n--- Adding Patients ---")
emergency_queue.add_patient("Alice", 3)
emergency_queue.add_patient("Bob", 5) # Bob is more critical
emergency_queue.add_patient("Charlie", 2)
emergency_queue.add_patient("David", 5) # David has same priority as Bob
emergency_queue.add_patient("Eve", 1)

print("\n--- Checking Status ---")
emergency_queue.check_status()

print("\n--- Handling Patients ---")
try:
    emergency_queue.handle_next_patient() # Should be Bob (priority 5)
    emergency_queue.check_status()
    emergency_queue.handle_next_patient() # Should be David (priority 5, arrived after Bob)
    emergency_queue.check_status()
    emergency_queue.handle_next_patient() # Should be Alice (priority 3)
    emergency_queue.check_status()
except IndexError as e:
    print(f"Error: {e}")
print("\n--- Adding more patients and handling ---")
emergency_queue.add_patient("Frank", 4)
emergency_queue.add_patient("Grace", 3)
emergency_queue.check_status()
emergency_queue.handle_next_patient() # Should be Frank (priority 4)
emergency_queue.handle_next_patient() # Should be Grace (priority 3)
emergency_queue.handle_next_patient() # Should be Charlie (priority 2)
emergency_queue.handle_next_patient() # Should be Eve (priority 1)

print("\n--- Checking status after all patients handled ---")
print(emergency_queue.check_status())

print("\n--- Attempting to handle patient from empty queue ---")
try:
    emergency_queue.handle_next_patient()
except IndexError as e:
    print(f"Error: {e}")
```

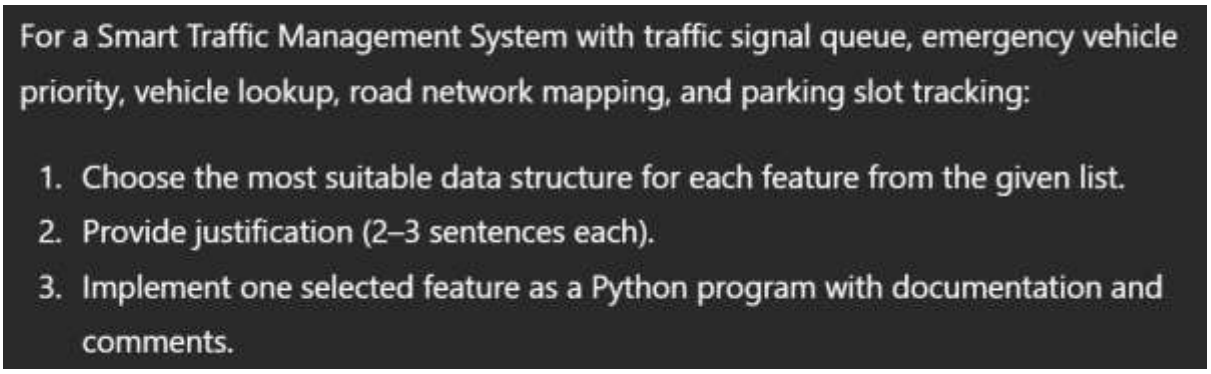
Output:


```
--- Initializing Emergency Handling System ---
Emergency Handling System Initialized.

--- Adding Patients ---
Added patient 'Alice' with priority 3.
Added patient 'Bob' with priority 5.
Added patient 'Charlie' with priority 2.
Added patient 'David' with priority 5.
Added patient 'Eve' with priority 1.

--- Checking Status ---
Current queue status (highest priority first):
- Bob
- David
- Alice
- Charlie
- Eve

--- Handling Patients ---
Handling patient 'Bob' with priority 5.
Current queue status (highest priority first):
- David
- Alice
- Charlie
- Eve
Handling patient 'David' with priority 5.
Current queue status (highest priority first):
- Alice
- Charlie
- Eve
Handling patient 'Alice' with priority 3.
Current queue status (highest priority first):
- Charlie
- Eve

--- Adding more patients and handling ---
Added patient 'Frank' with priority 4.
Added patient 'Grace' with priority 3.
Current queue status (highest priority first):
- Frank
- Grace
- Charlie
```


```
- Eve
Handling patient 'Frank' with priority 4.
Handling patient 'Grace' with priority 3.
Handling patient 'Charlie' with priority 2.
Handling patient 'Eve' with priority 1.

--- Checking status after all patients handled ---
No patients currently in the queue.

--- Attempting to handle patient from empty queue ---
Error: No patients in the queue to handle.
```

Task 7 – Smart City Traffic Control System

Prompt:


For a Smart Traffic Management System with traffic signal queue, emergency vehicle priority, vehicle lookup, road network mapping, and parking slot tracking:

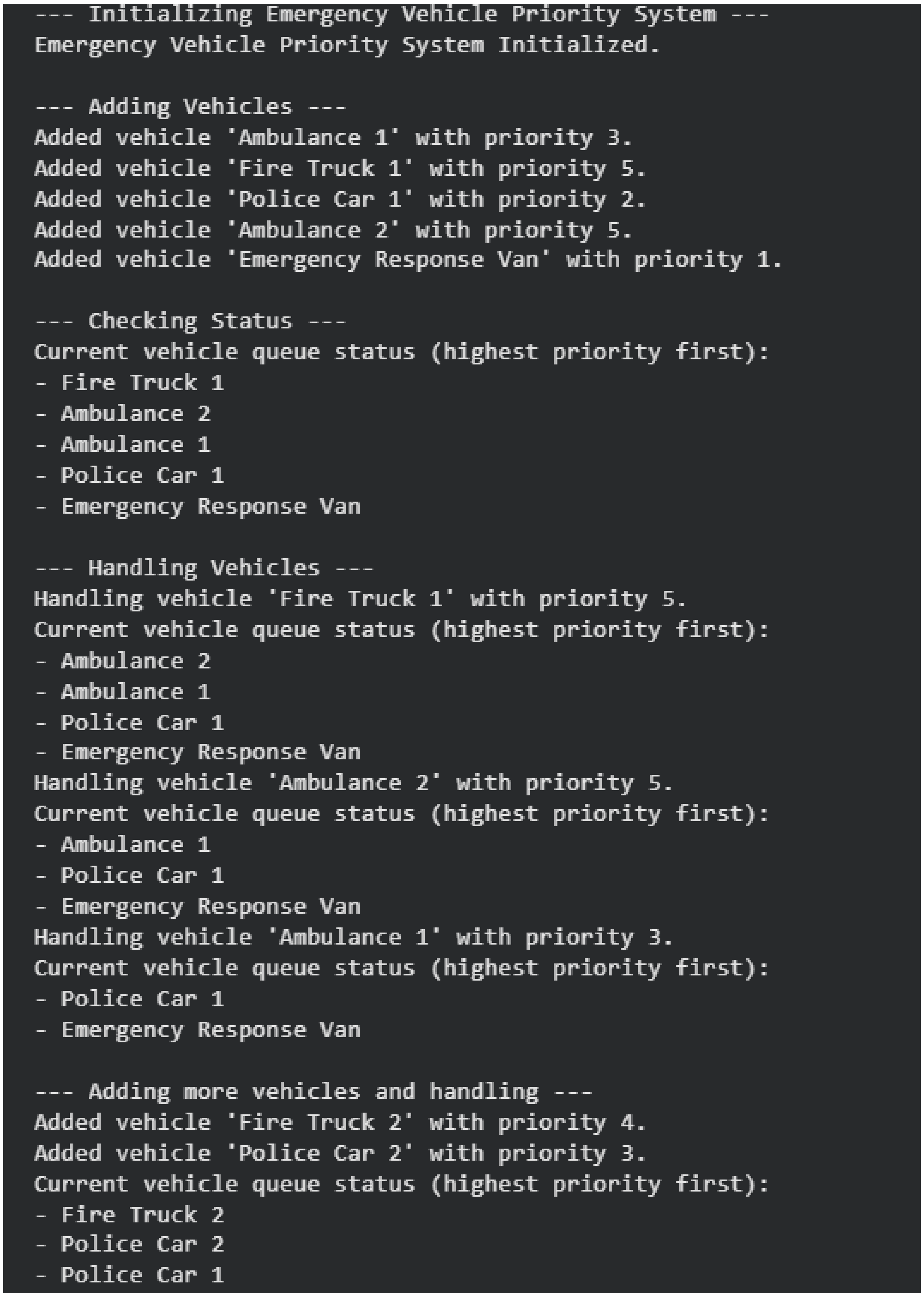
1. Choose the most suitable data structure for each feature from the given list.
2. Provide justification (2–3 sentences each).
3. Implement one selected feature as a Python program with documentation and comments.

Code:

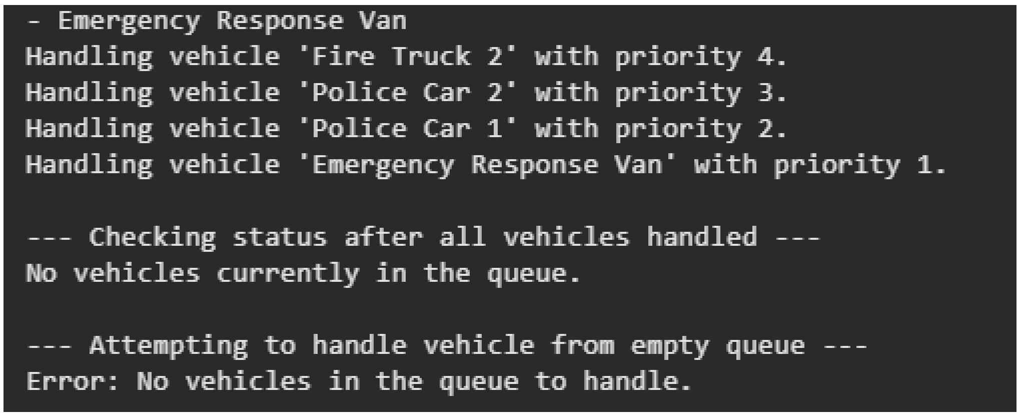
```
class EmergencyVehiclePriority:
    """Manages emergency vehicle priority using a custom priority queue logic.


    Vehicles are added with a priority, and the system can identify and handle
    the next highest priority vehicle, or check the current queue status.
    Higher priority numbers indicate a more critical or urgent vehicle.
    """


    def __init__(self):
        """Initializes an empty emergency vehicle queue."""
        # Stores vehicles as tuples: (priority, vehicle_name)
        # Higher numbers mean higher priority. We use a list to maintain
        # insertion order for tie-breaking.
        self._vehicles = []
        print("Emergency Vehicle Priority System Initialized.")

    def add_vehicle(self, vehicle_name, priority):
        """Adds a new emergency vehicle to the queue with a specified priority.


        Args:
            vehicle_name (str): The name or identifier of the vehicle.
            priority (int): The priority level of the vehicle. Higher numbers
                            indicate higher criticality (e.g., 5 > 1).
        """
        if not isinstance(priority, int) or priority < 0:
            print(f"Warning: Invalid priority '{priority}' for vehicle
'{vehicle_name}'. Priority must be a non-negative integer.")
            return
        self._vehicles.append((priority, vehicle_name))
        print(f"Added vehicle '{vehicle_name}' with priority {priority}.")

    def handle_next_vehicle(self):
        """Identifies, removes, and returns the vehicle with the highest priority.


        In case of a tie in priority, the vehicle that was added earliest (first
occurrence
        in the internal list) is selected.
        Returns:
            str: The name of the vehicle with the highest priority.

        Raises:
            IndexError: If the emergency vehicle queue is empty.
        """
        if not self._vehicles:
            raise IndexError("No vehicles in the queue to handle.")

        highest_priority = -1
        vehicle_to_handle_index = -1

        # Iterate to find the highest priority vehicle, respecting insertion order for
ties.
        # The first vehicle encountered with the highest priority will be chosen.
        for i, (priority, vehicle_name) in enumerate(self._vehicles):
            if priority > highest_priority:
                highest_priority = priority
                vehicle_to_handle_index = i

        # Remove and return the vehicle using its index
        handled_vehicle = self._vehicles.pop(vehicle_to_handle_index)
        print(f"Handling vehicle '{handled_vehicle[1]}' with priority
{handled_vehicle[0]}.")
        return handled_vehicle[1]

    def check_status(self):
        """Returns a list of vehicles currently in the queue, ordered by priority (highest first).

        If priorities are equal, their original insertion order is maintained (stable sort).

        Returns:
            list: A list of vehicle names (strings) currently in the queue, sorted by priority.
                  Returns a descriptive string if the queue is empty.
        """
        if not self._vehicles:
            return "No vehicles currently in the queue."

        # Sort vehicles by priority in descending order (highest priority first).
        # Python's Timsort is stable, so original insertion order is preserved for equal priorities.
        sorted_vehicles = sorted(self._vehicles, key=lambda x: x[0], reverse=True)

        status_list = [vehicle_name for priority, vehicle_name in sorted_vehicles]
        print("Current vehicle queue status (highest priority first):")
        for vehicle in status_list:
            print(f"- {vehicle}")
        return status_list

# Example Usage:
print("\n--- Initializing Emergency Vehicle Priority System ---")
```

```
vehicle_queue = EmergencyVehiclePriority()

print("\n--- Adding Vehicles ---")
vehicle_queue.add_vehicle("Ambulance 1", 3)
vehicle_queue.add_vehicle("Fire Truck 1", 5) # Highest priority
vehicle_queue.add_vehicle("Police Car 1", 2)
vehicle_queue.add_vehicle("Ambulance 2", 5) # Same priority as Fire Truck 1
vehicle_queue.add_vehicle("Emergency Response Van", 1)

print("\n--- Checking Status ---")
vehicle_queue.check_status()

print("\n--- Handling Vehicles ---")
try:
    vehicle_queue.handle_next_vehicle() # Should be Fire Truck 1 (priority 5, 
added earlier)
    vehicle_queue.check_status()
    vehicle_queue.handle_next_vehicle() # Should be Ambulance 2 (priority 5)
    vehicle_queue.check_status()
    vehicle_queue.handle_next_vehicle() # Should be Ambulance 1 (priority 3)
    vehicle_queue.check_status()
except IndexError as e:
    print(f"Error: {e}")

print("\n--- Adding more vehicles and handling ---")
vehicle_queue.add_vehicle("Fire Truck 2", 4)
vehicle_queue.add_vehicle("Police Car 2", 3)

vehicle_queue.check_status()

vehicle_queue.handle_next_vehicle() # Should be Fire Truck 2 (priority 4)

vehicle_queue.handle_next_vehicle() # Should be Police Car 2 (priority 3)

vehicle_queue.handle_next_vehicle() # Should be Police Car 1 (priority 2)

vehicle_queue.handle_next_vehicle() # Should be Emergency Response Van
(priority 1)


print("\n--- Checking status after all vehicles handled ---")

print(vehicle_queue.check_status())


print("\n--- Attempting to handle vehicle from empty queue ---")

try:

    vehicle_queue.handle_next_vehicle()

except IndexError as e:

    print(f"Error: {e}")
```

Output:



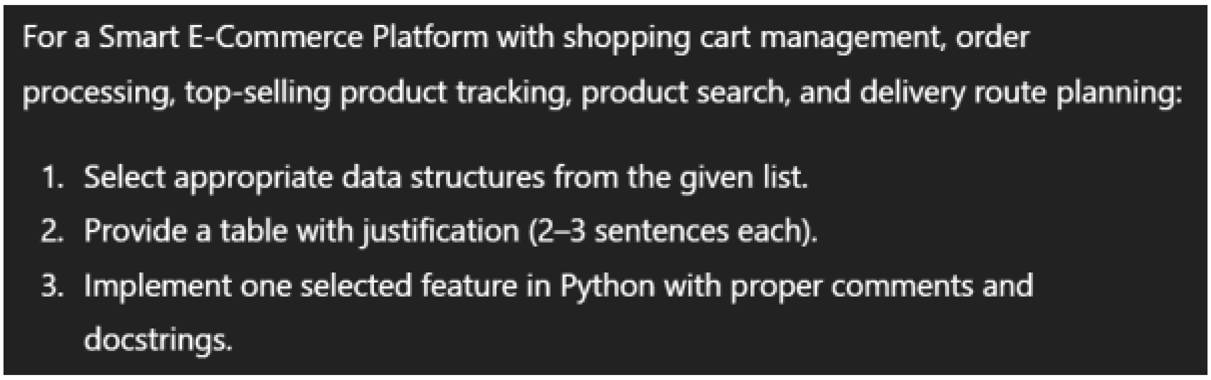
```
--- Initializing Emergency Vehicle Priority System ---
Emergency Vehicle Priority System Initialized.

--- Adding Vehicles ---
Added vehicle 'Ambulance 1' with priority 3.
Added vehicle 'Fire Truck 1' with priority 5.
Added vehicle 'Police Car 1' with priority 2.
Added vehicle 'Ambulance 2' with priority 5.
Added vehicle 'Emergency Response Van' with priority 1.

--- Checking Status ---
Current vehicle queue status (highest priority first):
- Fire Truck 1
- Ambulance 2
- Ambulance 1
- Police Car 1
- Emergency Response Van

--- Handling Vehicles ---
Handling vehicle 'Fire Truck 1' with priority 5.
Current vehicle queue status (highest priority first):
- Ambulance 2
- Ambulance 1
- Police Car 1
- Emergency Response Van
Handling vehicle 'Ambulance 2' with priority 5.
Current vehicle queue status (highest priority first):
- Ambulance 1
- Police Car 1
- Emergency Response Van
Handling vehicle 'Ambulance 1' with priority 3.
Current vehicle queue status (highest priority first):
- Police Car 1
- Emergency Response Van

--- Adding more vehicles and handling ---
Added vehicle 'Fire Truck 2' with priority 4.
Added vehicle 'Police Car 2' with priority 3.
Current vehicle queue status (highest priority first):
- Fire Truck 2
- Police Car 2
- Police Car 1
```



```
 - Emergency Response Van
Handling vehicle 'Fire Truck 2' with priority 4.
Handling vehicle 'Police Car 2' with priority 3.
Handling vehicle 'Police Car 1' with priority 2.
Handling vehicle 'Emergency Response Van' with priority 1.

--- Checking status after all vehicles handled ---
No vehicles currently in the queue.

--- Attempting to handle vehicle from empty queue ---
Error: No vehicles in the queue to handle.
```

Task 8 – Smart E-Commerce Platform

Prompt:



For a Smart E-Commerce Platform with shopping cart management, order processing, top-selling product tracking, product search, and delivery route planning:

1. Select appropriate data structures from the given list.
2. Provide a table with justification (2–3 sentences each).
3. Implement one selected feature in Python with proper comments and docstrings.

Code:

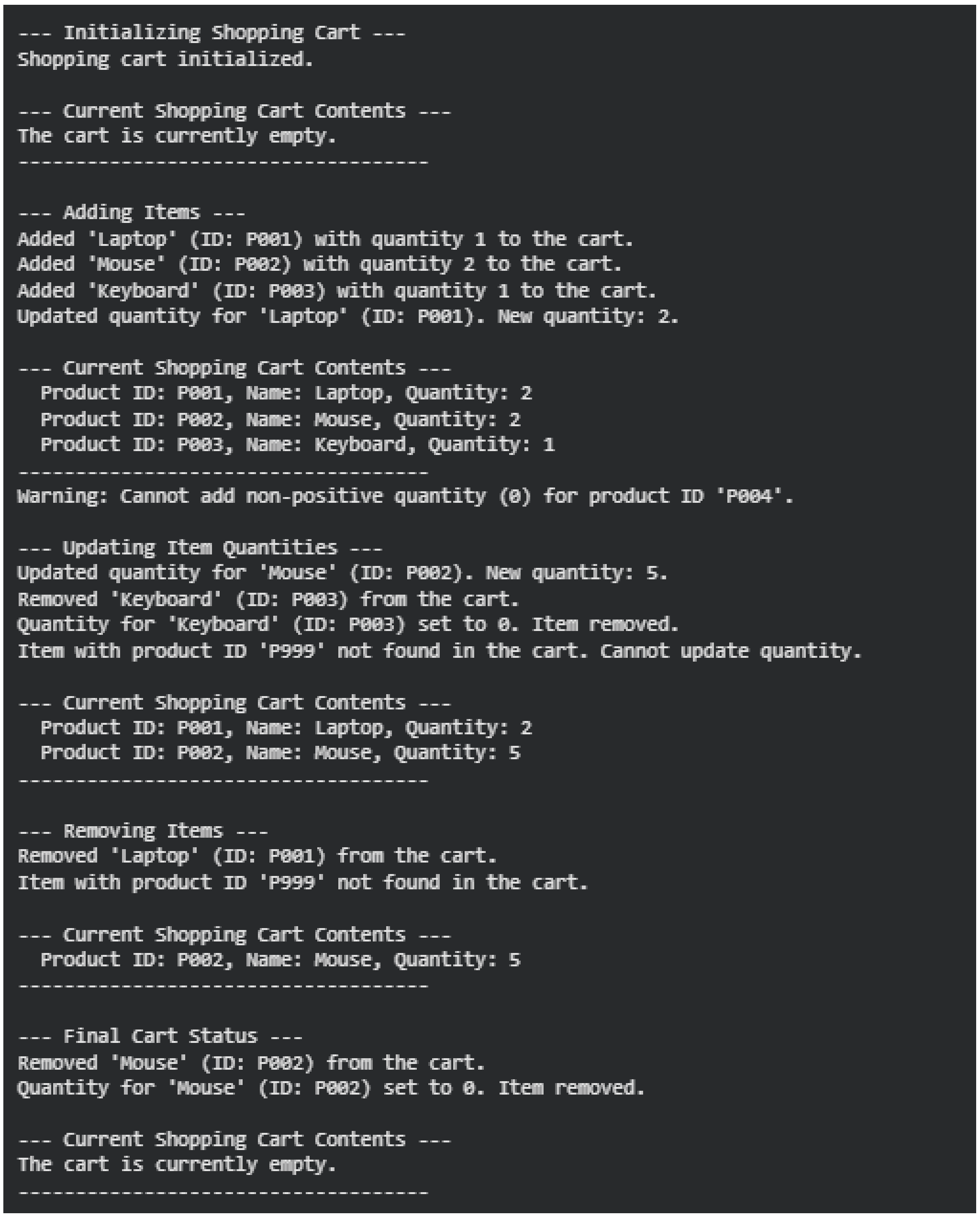
```
class ShoppingCart:
    """Manages a shopping cart using a hash table (Python dictionary).


    Provides methods to add, remove, update quantities, and view items in the
cart.
    """



    def __init__(self):
        """Initializes an empty shopping cart.


        The cart is represented as a dictionary where keys are product_ids
        and values are dictionaries containing 'name' and 'quantity'.
        """
        self._items = {}
        print("Shopping cart initialized.")

    def add_item(self, product_id, name, quantity):
        """Adds an item to the cart or updates its quantity if it already exists.

        Args:
            product_id (str): A unique identifier for the product.
            name (str): The name of the product.
            quantity (int): The quantity of the product to add.
        """
        if quantity <= 0:
            print(f"Warning: Cannot add non-positive quantity ({quantity}) for
product ID '{product_id}'.")
            return

        if product_id in self._items:
            self._items[product_id]['quantity'] += quantity
            print(f"Updated quantity for '{name}' (ID: {product_id}). New quantity:
{self._items[product_id]['quantity']}.")
        else:
            self._items[product_id] = {'name': name, 'quantity': quantity}
            print(f"Added '{name}' (ID: {product_id}) with quantity {quantity} to
the cart.")
```

```
def remove_item(self, product_id):
    """Removes an item completely from the cart.

    Args:
        product_id (str): The unique identifier of the product to remove.
    """
    if product_id in self._items:
        item_name = self._items[product_id]['name']
        del self._items[product_id]
        print(f"Removed '{item_name}' (ID: {product_id}) from the cart.")
    else:
        print(f"Item with product ID '{product_id}' not found in the cart.")

def update_quantity(self, product_id, new_quantity):
    """Updates the quantity of an existing item in the cart.

    If `new_quantity` is 0 or less, the item is removed from the cart.

    Args:
        product_id (str): The unique identifier of the product to update.
        new_quantity (int): The new quantity for the product.
    """
    if product_id in self._items:
        item_name = self._items[product_id]['name']
        if new_quantity <= 0:
            self.remove_item(product_id)
```

```
                print(f"Quantity for '{item_name}' (ID: {product_id}) set to {new_quantity}. Item removed.")
            else:
                self._items[product_id]['quantity'] = new_quantity
                print(f"Updated quantity for '{item_name}' (ID: {product_id}). New quantity: {new_quantity}.")
        else:
            print(f"Item with product ID '{product_id}' not found in the cart. Cannot update quantity.")

    def view_cart(self):
        """Displays the current contents of the shopping cart.

        If the cart is empty, a message indicating this is printed.
        """
        print("\n--- Current Shopping Cart Contents ---")
        if not self._items:
            print("The cart is currently empty.")
        else:
            for product_id, details in self._items.items():
                print(f"  Product ID: {product_id}, Name: {details['name']}, Quantity: {details['quantity']}")
        print("------------------------------------")

# Example Usage:
print("\n--- Initializing Shopping Cart ---")
my_cart = ShoppingCart()
```

```
my_cart.view_cart()

print("\n--- Adding Items ---")
my_cart.add_item("P001", "Laptop", 1)
my_cart.add_item("P002", "Mouse", 2)
my_cart.add_item("P003", "Keyboard", 1)
my_cart.add_item("P001", "Laptop", 1) # Add existing item to increase quantity
my_cart.view_cart()

my_cart.add_item("P004", "Monitor", 0) # Attempt to add with non-positive
quantity

print("\n--- Updating Item Quantities ---")
my_cart.update_quantity("P002", 5) # Increase quantity
my_cart.update_quantity("P003", 0) # Decrease to zero, should remove
my_cart.update_quantity("P999", 3) # Update non-existent item
my_cart.view_cart()

print("\n--- Removing Items ---")
my_cart.remove_item("P001") # Remove an existing item
my_cart.remove_item("P999") # Remove a non-existent item
my_cart.view_cart()

print("\n--- Final Cart Status ---")
my_cart.update_quantity("P002", 0) # Remove last item
my_cart.view_cart()
```

Output:



```
--- Initializing Shopping Cart ---
Shopping cart initialized.

--- Current Shopping Cart Contents ---
The cart is currently empty.
------------------------------------

--- Adding Items ---
Added 'Laptop' (ID: P001) with quantity 1 to the cart.
Added 'Mouse' (ID: P002) with quantity 2 to the cart.
Added 'Keyboard' (ID: P003) with quantity 1 to the cart.
Updated quantity for 'Laptop' (ID: P001). New quantity: 2.

--- Current Shopping Cart Contents ---
  Product ID: P001, Name: Laptop, Quantity: 2
  Product ID: P002, Name: Mouse, Quantity: 2
  Product ID: P003, Name: Keyboard, Quantity: 1
------------------------------------
Warning: Cannot add non-positive quantity (0) for product ID 'P004'.

--- Updating Item Quantities ---
Updated quantity for 'Mouse' (ID: P002). New quantity: 5.
Removed 'Keyboard' (ID: P003) from the cart.
Quantity for 'Keyboard' (ID: P003) set to 0. Item removed.
Item with product ID 'P999' not found in the cart. Cannot update quantity.

--- Current Shopping Cart Contents ---
  Product ID: P001, Name: Laptop, Quantity: 2
  Product ID: P002, Name: Mouse, Quantity: 5
------------------------------------

--- Removing Items ---
Removed 'Laptop' (ID: P001) from the cart.
Item with product ID 'P999' not found in the cart.

--- Current Shopping Cart Contents ---
  Product ID: P002, Name: Mouse, Quantity: 5
------------------------------------

--- Final Cart Status ---
Removed 'Mouse' (ID: P002) from the cart.
Quantity for 'Mouse' (ID: P002) set to 0. Item removed.

--- Current Shopping Cart Contents ---
The cart is currently empty.
------------------------------------
```