

**Name : Sai Venkatesh Paruchuri**  
**H.NO : 2303A53039 - B46**

## Lab 11 – Data Structures with AI: Implementing Fundamental Structures

### Lab Objectives

- Use AI to assist in designing and implementing fundamental data structures in Python.
- Learn how to prompt AI for structure creation, optimization, and documentation.
- Improve understanding of Lists, Stacks, Queues, Linked Lists, Trees, Graphs, and Hash Tables.
- Enhance code quality with AI-generated comments and performance suggestions.

### Task Description #1 – Stack Implementation

Task: Use AI to generate a Stack class with push, pop, peek, and is\_empty methods.

Sample Input Code:

```
class Stack:
```

```
    pass
```

Expected Output:

- A functional stack implementation with all required methods and docstrings.

### PROMPT:

```
#Generate a simple and clean Python Stack class using a
list with methods push, pop, peek, and is_empty, handle
empty stack safely, include short student-style
docstrings, keep code readable for a college lab record,
and show a small example usage.
```

### CODE:

```
class Stack:
    """A simple Stack class using a list to store elements."""

    def __init__(self):
```

```

"""Initialize an empty stack."""
self.stack = []

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

def pop(self):
    """Remove and return the item at the top of the stack.
    Returns None if the stack is empty."""
    if not self.is_empty():
        return self.stack.pop()
    else:
        print("Stack is empty. Cannot pop.")
        return None

def peek(self):
    """Return the item at the top of the stack without removing it.
    Returns None if the stack is empty."""
    if not self.is_empty():
        return self.stack[-1]
    else:
        print("Stack is empty. Cannot peek.")
        return None

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

# Example usage
if __name__ == "__main__":
    my_stack = Stack()
    my_stack.push(1)
    my_stack.push(2)
    my_stack.push(3)

    print("Top item:", my_stack.peek()) # Output: Top item: 3
    print("Popped item:", my_stack.pop()) # Output: Popped item: 3
    print("Is stack empty?", my_stack.is_empty()) # Output: Is stack empty? False
    print("Popped item:", my_stack.pop()) # Output: Popped item: 2
    print("Popped item:", my_stack.pop()) # Output: Popped item: 1
    print("Is stack empty?", my_stack.is_empty()) # Output: Is stack empty? True
    print("Popped item from empty stack:", my_stack.pop()) # Output: Stack is empty.
    Cannot pop. Popped item from empty stack: None
    print("Peek item from empty stack:", my_stack.peek()) # Output: Stack is empty. Cannot
    peek. Peek item from empty stack: None

```

## OUTPUT:

Top item: 3  
 Popped item: 3  
 Is stack empty? False

Popped item: 2  
 Popped item: 1  
 Is stack empty? True  
 Stack is empty. Cannot pop.  
 Popped item from empty stack: None  
 Stack is empty. Cannot peek.  
 Peek item from empty stack: None

```

A11.1.py > ...
2  class Stack:
22     def peek(self):
25         if not self.is_empty():
26             return self.stack[-1]
27         else:
28             print("Stack is empty. Cannot peek.")
29             return None
30
31     def is_empty(self):
32         """Check if the stack is empty."""
33         return len(self.stack) == 0
34
# Example usage
35 if __name__ == "__main__":
36     my_stack = Stack()
37     my_stack.push(1)
38     my_stack.push(2)
39     my_stack.push(3)
40
41     print("Top item:", my_stack.peek()) # Output: Top item: 3
42     print("Popped item:", my_stack.pop()) # Output: Popped item: 3
43     print("Is stack empty?", my_stack.is_empty()) # Output: Is stack empty? False
44     print("Popped item:", my_stack.pop()) # Output: Popped item: 2
45     print("Popped item:", my_stack.pop()) # Output: Popped item: 1
46     print("Is stack empty?", my_stack.is_empty()) # Output: Is stack empty? True
47     print("Popped item from empty stack:", my_stack.pop()) # Output: Stack is empty. Cannot pop. Popped item from empty stack: None
48     print("Peek item from empty stack:", my_stack.peek()) # Output: Stack is empty. Cannot peek. Peek item from empty stack: None
49

```



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
+ × ⋮ | : - [ ] Python [ ] Python
● saivenkatesh@Sais-MacBook-Air AI Coding % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
Top item: 3
Popped item: 3
Is stack empty? False
Popped item: 2
Popped item: 1
Is stack empty? True
Stack is empty. Cannot pop.
Popped item from empty stack: None
Stack is empty. Cannot peek.
Peek item from empty stack: None
saivenkatesh@Sais-MacBook-Air AI Coding %

```

## Task Description #2 – Queue Implementation

Task: Use AI to implement a Queue using Python lists.

Sample Input Code:

```
class Queue:
```

```
    pass
```

Expected Output:

- FIFO-based queue class with enqueue, dequeue, peek, and size methods.

## PROMPT:

```
#Generate a simple and clean Python Queue class using a
list with FIFO behavior and methods enqueue, dequeue,
peek, and size, handle empty queue safely, include short
natural student-style docstrings, keep the code clear and
```

suitable for a college lab record, and provide a small example usage.

### CODE:

```
class Queue:
    """A simple Queue class using a list to store elements with
    FIFO behavior."""

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

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

    def dequeue(self):
        """Remove and return the item at the front of the queue.
        Returns None if the queue is empty."""
        if not self.is_empty():
            return self.queue.pop(0)
        else:
            print("Queue is empty. Cannot dequeue.")
            return None

    def peek(self):
        """Return the item at the front of the queue without
        removing it.
        Returns None if the queue is empty."""
        if not self.is_empty():
            return self.queue[0]
        else:
            print("Queue is empty. Cannot peek.")
            return None

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

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

# Example usage
if __name__ == "__main__":
    my_queue = Queue()
    my_queue.enqueue(1)
    my_queue.enqueue(2)
    my_queue.enqueue(3)
```

```

print("Front item:", my_queue.peek()) # Output: Front item: 1
print("Dequeued item:", my_queue.dequeue()) # Output:
Dequeued item: 1
    print("Queue size:", my_queue.size()) # Output: Queue size: 2
    print("Dequeued item:", my_queue.dequeue()) # Output:
Dequeued item: 2
    print("Dequeued item:", my_queue.dequeue()) # Output:
Dequeued item: 3
    print("Is queue empty?", my_queue.is_empty()) # Output: Is
queue empty? True
    print("Dequeued item from empty queue:", my_queue.dequeue())
# Output: Queue is empty. Cannot dequeue. Dequeued item from empty
queue: None
    print("Peek item from empty queue:", my_queue.peek()) #
Output: Queue is empty. Cannot peek. Peek item from empty queue:
None

```

## OUTPUT:

Front item: 1  
 Dequeued item: 1  
 Queue size: 2  
 Dequeued item: 2  
 Dequeued item: 3  
 Is queue empty? True  
 Queue is empty. Cannot dequeue.  
 Dequeued item from empty queue: None  
 Queue is empty. Cannot peek.  
 Peek item from empty queue: None

## Task Description #3 – Linked List

Task: Use AI to 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.

```

❷ A11.1.py > ...
55     class Queue:
56         def __init__(self):
57             self.queue = []
58
59         def enqueue(self, item):
60             self.queue.append(item)
61
62         def dequeue(self):
63             if len(self.queue) == 0:
64                 return None
65             else:
66                 return self.queue.pop(0)
67
68         def peek(self):
69             if len(self.queue) == 0:
70                 return None
71             else:
72                 return self.queue[0]
73
74         def size(self):
75             return len(self.queue)
76
77     def is_empty(self):
78         """Check if the queue is empty."""
79         return len(self.queue) == 0
80
81     # Example usage
82     if __name__ == "__main__":
83         my_queue = Queue()
84         my_queue.enqueue(1)
85         my_queue.enqueue(2)
86         my_queue.enqueue(3)
87
88         print("Front item:", my_queue.peek()) # Output: Front item: 1
89         print("Dequeued item:", my_queue.dequeue()) # Output: Dequeued item: 1
90         print("Queue size:", my_queue.size()) # Output: Queue size: 2
91         print("Dequeued item:", my_queue.dequeue()) # Output: Dequeued item: 2
92         print("Dequeued item:", my_queue.dequeue()) # Output: Dequeued item: 3
93         print("Is queue empty?", my_queue.is_empty()) # Output: Is queue empty? True
94         print("Dequeued item from empty queue:", my_queue.dequeue()) # Output: Queue is empty. Cannot dequeue. Dequeued item from empty queue: None
95         print("Peek item from empty queue:", my_queue.peek()) # Output: Queue is empty. Cannot peek. Peek item from empty queue: None
96
97
98
99
100
101
102
103
104
105
106

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```

/usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
❸ saivenkatesh@Sais-MacBook-Air AI Coding % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
Front item: 1
Dequeued item: 1
Queue size: 2
Dequeued item: 2
Dequeued item: 3
Is queue empty? True
Queue is empty. Cannot dequeue.
Dequeued item from empty queue: None
Queue is empty. Cannot peek.
Peek item from empty queue: None
❹ saivenkatesh@Sais-MacBook-Air AI Coding %

```

## PROMPT:

#Generate a simple Python implementation of a Singly Linked List with a Node class and LinkedList class, include insert at end and display methods, use clear beginner-friendly docstrings and comments, avoid complex optimizations, keep the code clean for a college lab record, and provide a small example usage.

## CODE:

```

class Node:
    """A Node class to represent each element in the linked list."""

    def __init__(self, data):
        """Initialize a node with data and a pointer to the next node."""
        self.data = data # Store the data
        self.next = None # Initialize the next pointer to None

class LinkedList:
    """A LinkedList class to manage the linked list operations."""

    def __init__(self):
        """Initialize an empty linked list."""

```

```

        self.head = None # Start with an empty list

    def insert_at_end(self, data):
        """Insert a new node with the given data at the end of the
list."""
        new_node = Node(data) # Create a new node
        if self.head is None: # If the list is empty, set the new
node as head
            self.head = new_node
        return
        last_node = self.head # Start from the head
        while last_node.next: # Traverse to the end of the list
            last_node = last_node.next
        last_node.next = new_node # Link the last node to the new
node

    def display(self):
        """Display the contents of the linked list."""
        current_node = self.head # Start from the head
        while current_node: # Traverse through the list
            print(current_node.data, end=' -> ') # Print current
node's data
            current_node = current_node.next # Move to the next
node
        print("None") # Indicate the end of the list
# Example usage
if __name__ == "__main__":
    my_list = LinkedList()
    my_list.insert_at_end(1)
    my_list.insert_at_end(2)
    my_list.insert_at_end(3)

    print("Linked List contents:")
    my_list.display() # Output: Linked List contents: 1 -> 2 -> 3
-> None

```

## OUTPUT:

Linked List contents:  
1 -> 2 -> 3 -> None

## Task Description #4 – Binary Search Tree (BST)

Task: Use AI 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.

## PROMPT:

#Generate a simple Python implementation of a Binary Search Tree with recursive insert and in-order traversal methods, use clear beginner-friendly docstrings, keep the code clean and easy for a college lab record, and include an example inserting values and printing them in sorted order.

## CODE:

```
class TreeNode:  
    """A TreeNode class to represent each node in the binary search tree."""  
  
    def __init__(self, data):  
        """Initialize a tree node with data and pointers to left and right children."""  
        self.data = data # Store the data  
        self.left = None # Initialize left child pointer  
        self.right = None # Initialize right child pointer  
class BinarySearchTree:  
    """A BinarySearchTree class to manage the binary search tree operations."""  
  
    def __init__(self):  
        """Initialize an empty binary search tree."""  
        self.root = None # Start with an empty tree  
  
    def insert(self, data):  
        """Insert a new node with the given data into the binary search tree."""  
        if self.root is None: # If the tree is empty, set the new node as root  
            self.root = TreeNode(data)  
        else:  
            self._insert_recursive(self.root, data) # Insert recursively  
  
    def _insert_recursive(self, node, data):  
        """Helper method to insert a new node recursively."""  
        if data < node.data: # If data is less than current node's data  
            if node.left is None: # If left child is empty, insert here  
                node.left = TreeNode(data)  
            else:  
                self._insert_recursive(node.left, data) # Recur left
```

```

        else: # If data is greater than or equal to current
node's data
            if node.right is None: # If right child is empty,
insert here
                node.right = TreeNode(data)
            else:
                self._insert_recursive(node.right, data) # Recur
right

    def in_order_traversal(self):
        """Perform in-order traversal and return a list of values
in sorted order."""
        return self._in_order_recursive(self.root)

    def _in_order_recursive(self, node):
        """Helper method to perform in-order traversal
recursively."""
        result = []
        if node:
            result.extend(self._in_order_recursive(node.left)) # Traverse left subtree
            result.append(node.data) # Visit current node
            result.extend(self._in_order_recursive(node.right)) # Traverse right subtree
        return result

# Example usage
if __name__ == "__main__":
    bst = BinarySearchTree()
    bst.insert(5)
    bst.insert(3)
    bst.insert(7)
    bst.insert(2)
    bst.insert(4)
    bst.insert(6)
    bst.insert(8)

    print("In-order traversal (sorted order):",
bst.in_order_traversal()) # Output: In-order traversal (sorted
order): [2, 3, 4, 5, 6, 7, 8]

```

## OUTPUT:

In-order traversal (sorted order): [2, 3, 4, 5, 6, 7, 8]

## Task Description #5 – Hash Table

Task: Use AI to implement a hash table with basic insert, search, and delete methods.

Sample Input Code:

class HashTable:

    pass

```

❷ A11.1.py > ...
164     class BinarySearchTree:
165         def __init__(self):
166             self.root = None
167
168         def insert(self, value):
169             if self.root is None:
170                 self.root = Node(value)
171             else:
172                 self._insert(self.root, value)
173
174         def _insert(self, node, value):
175             if value < node.value:
176                 if node.left is None:
177                     node.left = Node(value)
178                 else:
179                     self._insert(node.left, value)
180             else:
181                 if node.right is None:
182                     node.right = Node(value)
183                 else:
184                     self._insert(node.right, value)
185
186         def in_order_traversal(self):
187             result = []
188             self._in_order_recursive(self.root, result)
189             return result
190
191         def _in_order_recursive(self, node, result):
192             if node is not None:
193                 result.append(node.data) # Visit current node
194                 result.extend(self._in_order_recursive(node.right)) # Traverse right subtree
195             return result
196
197     # Example usage
198     if __name__ == "__main__":
199         bst = BinarySearchTree()
200         bst.insert(5)
201         bst.insert(3)
202         bst.insert(7)
203         bst.insert(2)
204         bst.insert(4)
205         bst.insert(6)
206         bst.insert(8)
207
208         print("In-order traversal (sorted order):", bst.in_order_traversal()) # Output: In-order traversal (sorted order): [2, 3, 4, 5, 6, 7, 8]
209
210
211
212
213
214
215

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
/usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
❸ saivenkatesh@Sais-MacBook-Air AI Coding % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
In-order traversal (sorted order): [2, 3, 4, 5, 6, 7, 8]
❹ saivenkatesh@Sais-MacBook-Air AI Coding %

```

Expected Output:

- Collision handling using chaining, with well-commented methods.

## PROMPT:

#Generate a simple Python implementation of a Hash Table using chaining for collision handling with a list of buckets, include insert, search, and delete methods, use a basic hash function, write clear beginner-friendly comments and docstrings, keep the code clean for a college lab record, and provide a small demo example.

## CODE:

```

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

    def __init__(self, size=10):
        """Initialize the hash table with a specified number of
        buckets."""
        self.size = size # Set the size of the hash table
        self.table = [[] for _ in range(size)] # Create a list of
                                                # empty buckets

    def _hash(self, key):

```

```

        """Basic hash function to compute the index for a given
key."""
        return hash(key) % self.size # Compute hash and map it to
the table size

def insert(self, key, value):
    """Insert a key-value pair into the hash table."""
    index = self._hash(key) # Get the index for the key
    bucket = self.table[index] # Get the bucket at that index
    for i, (k, v) in enumerate(bucket): # Check if the key
already exists
        if k == key:
            bucket[i] = (key, value) # Update value if key
exists
            return
    bucket.append((key, value)) # Add new key-value pair if
key does not exist

def search(self, key):
    """Search for a value by its key. Returns None if not
found."""
    index = self._hash(key) # Get the index for the key
    bucket = self.table[index] # Get the bucket at that index
    for k, v in bucket: # Search through the bucket
        if k == key:
            return v # Return value if key is found
    return None # Return None if key is not found

def delete(self, key):
    """Delete a key-value pair from the hash table. Returns
True if deleted, False if not found."""
    index = self._hash(key) # Get the index for the key
    bucket = self.table[index] # Get the bucket at that index
    for i, (k, v) in enumerate(bucket): # Search through the
bucket
        if k == key:
            del bucket[i] # Delete the key-value pair if
found
            return True
    return False # Return False if key is not found

# Example usage
if __name__ == "__main__":
    my_hash_table = HashTable()
    my_hash_table.insert("name", "Alice")
    my_hash_table.insert("age", 30)
    my_hash_table.insert("city", "New York")

    print("Search for 'name':", my_hash_table.search("name")) #
Output: Search for 'name': Alice
    print("Search for 'age':", my_hash_table.search("age")) #
Output: Search for 'age': 30

```

```

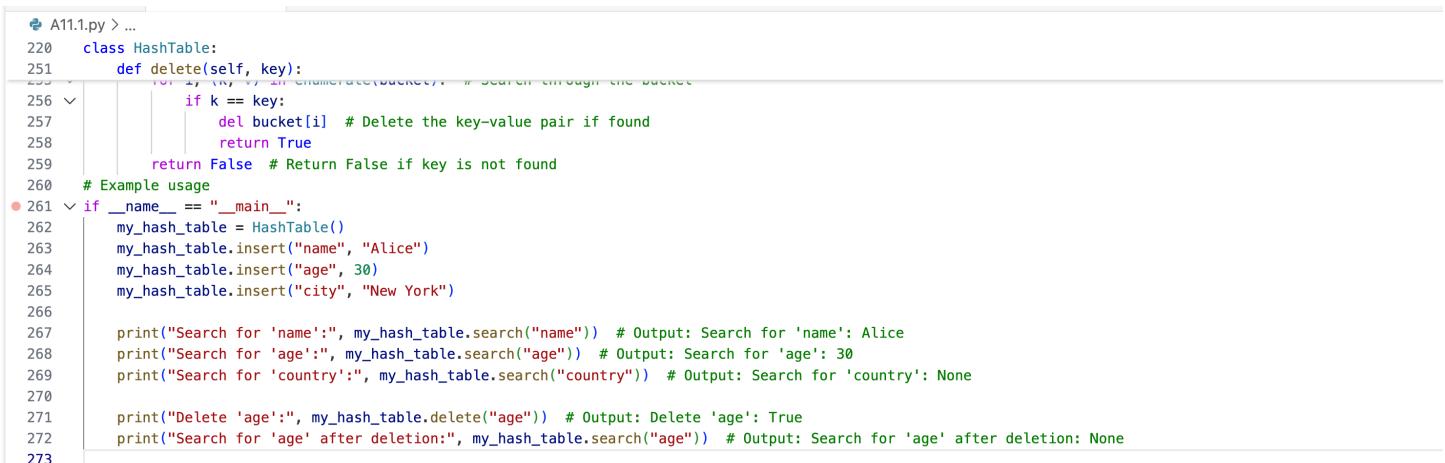
        print("Search for 'country':",
my_hash_table.search("country")) # Output: Search for 'country':
None

        print("Delete 'age':", my_hash_table.delete("age")) # Output:
Delete 'age': True
        print("Search for 'age' after deletion:",
my_hash_table.search("age")) # Output: Search for 'age' after
deletion: None

```

## OUTPUT:

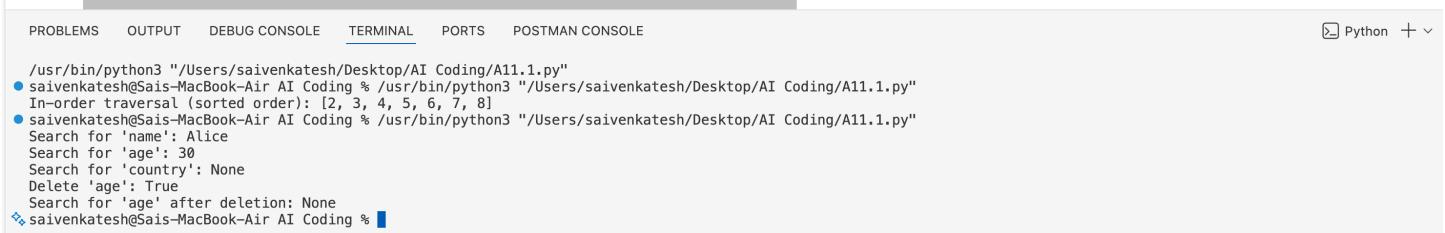
Search for 'name': Alice  
 Search for 'age': 30  
 Search for 'country': None  
 Delete 'age': True  
 Search for 'age' after deletion: None



```

A11.1.py > ...
220  class HashTable:
221      def __init__(self):
222          self.buckets = [None] * 10
223          self.size = 0
224
225      def insert(self, key, value):
226          index = self.get_index(key)
227          if self.buckets[index] is None:
228              self.buckets[index] = [key, value]
229              self.size += 1
230          else:
231              self.buckets[index].append(value)
232
233      def search(self, key):
234          index = self.get_index(key)
235          if self.buckets[index] is None:
236              return None
237          for i in range(len(self.buckets[index])):
238              if self.buckets[index][i] == key:
239                  return self.buckets[index][i+1]
240
241      def delete(self, key):
242          index = self.get_index(key)
243          if self.buckets[index] is None:
244              return False
245          for i in range(len(self.buckets[index])):
246              if self.buckets[index][i] == key:
247                  del self.buckets[index][i]
248                  self.size -= 1
249                  return True
250          return False
251
252      def get_index(self, key):
253          return hash(key) % len(self.buckets)
254
255      def __str__(self):
256          result = "In-order traversal (sorted order): "
257          for i in range(len(self.buckets)):
258              if self.buckets[i] is not None:
259                  result += str(self.buckets[i]) + ", "
260
261  if __name__ == "__main__":
262      my_hash_table = HashTable()
263      my_hash_table.insert("name", "Alice")
264      my_hash_table.insert("age", 30)
265      my_hash_table.insert("city", "New York")
266
267      print("Search for 'name':", my_hash_table.search("name")) # Output: Search for 'name': Alice
268      print("Search for 'age':", my_hash_table.search("age")) # Output: Search for 'age': 30
269      print("Search for 'country':", my_hash_table.search("country")) # Output: Search for 'country': None
270
271      print("Delete 'age':", my_hash_table.delete("age")) # Output: Delete 'age': True
272      print("Search for 'age' after deletion:", my_hash_table.search("age")) # Output: Search for 'age' after deletion: None
273

```



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
Python + ▾

/usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
● saivenkatesh@Sais-MacBook-Air AI Coding % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
In-order traversal (sorted order): [2, 3, 4, 5, 6, 7, 8]
● saivenkatesh@Sais-MacBook-Air AI Coding % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
Search for 'name': Alice
Search for 'age': 30
Search for 'country': None
Delete 'age': True
Search for 'age' after deletion: None
● saivenkatesh@Sais-MacBook-Air AI Coding %

```

## Task Description #6 – Graph Representation

Task: Use AI to 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.

## PROMPT:

```
#Generate a simple and clean Python Graph class using an adjacency list for an undirected graph, include add_vertex, add_edge, and display methods, write clear beginner-friendly docstrings, keep the implementation readable for a college lab record, and include a small example showing vertex connections.
```

## CODE:

```
class Graph:  
    """A simple Graph class using an adjacency list for an undirected graph."""  
  
    def __init__(self):  
        """Initialize an empty graph."""  
        self.graph = {} # Use a dictionary to store the adjacency list  
  
    def add_vertex(self, vertex):  
        """Add a vertex to the graph."""  
        if vertex not in self.graph:  
            self.graph[vertex] = [] # Initialize the vertex with an empty list of neighbors  
  
    def add_edge(self, vertex1, vertex2):  
        """Add an undirected edge between two vertices in the graph."""  
        if vertex1 not in self.graph:  
            self.add_vertex(vertex1) # Add vertex1 if it doesn't exist  
        if vertex2 not in self.graph:  
            self.add_vertex(vertex2) # Add vertex2 if it doesn't exist  
        self.graph[vertex1].append(vertex2) # Add vertex2 to vertex1's neighbors  
        self.graph[vertex2].append(vertex1) # Add vertex1 to vertex2's neighbors  
  
    def display(self):  
        """Display the adjacency list of the graph."""  
        for vertex, neighbors in self.graph.items():  
            print(f"{vertex}: {', '.join(neighbors)}) # Print each vertex and its neighbors  
  
# Example usage  
if __name__ == "__main__":  
    my_graph = Graph()
```

```

my_graph.add_edge("A", "B")
my_graph.add_edge("A", "C")
my_graph.add_edge("B", "D")
my_graph.add_edge("C", "D")
my_graph.add_edge("D", "E")

print("Graph adjacency list:")
my_graph.display() # Output: Graph adjacency list: A: B, C B:
A, D C: A, D D: B, C, E E: D

```

## OUTPUT:

Graph adjacency list:

A: B, C  
 B: A, D  
 C: A, D  
 D: B, C, E  
 E: D

```

A11.1.py > ...
275  class Graph:
295
296      def display(self):
297          """Display the adjacency list of the graph."""
298          for vertex, neighbors in self.graph.items():
299              print(f'{vertex}: {", ".join(neighbors)}') # Print each vertex and its neighbors
300
# Example usage
301 if __name__ == "__main__":
302     my_graph = Graph()
303     my_graph.add_edge("A", "B")
304     my_graph.add_edge("A", "C")
305     my_graph.add_edge("B", "D")
306     my_graph.add_edge("C", "D")]
307     my_graph.add_edge("D", "E")
308
309     print("Graph adjacency list:")
310     my_graph.display() # Output: Graph adjacency list: A: B, C B: A, D C: A, D D: B, C, E E: D
311

```



The screenshot shows a code editor with Python code for a graph class. The code defines a `Graph` class with a `display` method that prints the graph's adjacency list. An example usage of the class is shown, adding edges between vertices A, B, C, D, and E, and then printing the adjacency list. The output terminal shows the execution of the script and the resulting adjacency list: A: B, C B: A, D C: A, D D: B, C, E E: D.

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE Python + v

/usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
● saivenkatesh@Sais-MacBook-Air AI Coding % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
Graph adjacency list:
A: B, C
B: A, D
C: A, D
D: B, C, E
E: D
❖ saivenkatesh@Sais-MacBook-Air AI Coding %

```

## Task Description #7 – Priority Queue

Task: Use AI to 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.

## PROMPT:

```
#Generate a simple Python Priority Queue using the heapq module  
with enqueue(value, priority), dequeue returning the highest  
priority item, and display methods, include short student-style  
docstrings, keep the code clean and readable for a college lab  
record, and provide a small example usage.
```

## CODE:

```
import heapq  
class PriorityQueue:  
    """A simple Priority Queue implementation using the heapq  
    module."""  
  
    def __init__(self):  
        """Initialize an empty priority queue."""  
        self.queue = [] # Use a list to store the priority queue  
  
    def enqueue(self, value, priority):  
        """Add an item to the priority queue with a given  
        priority."""  
        heapq.heappush(self.queue, (priority, value)) # Push a  
        tuple of (priority, value)  
  
    def dequeue(self):  
        """Remove and return the item with the highest priority.  
        Returns None if the queue is empty."""  
        if not self.is_empty():  
            return heapq.heappop(self.queue)[1] # Pop the item  
            with the highest priority  
        else:  
            print("Priority Queue is empty. Cannot dequeue.")  
            return None  
  
    def display(self):  
        """Display the contents of the priority queue."""  
        print("Priority Queue contents:")  
        for priority, value in sorted(self.queue): # Sort by  
        priority for display  
            print(f"Value: {value}, Priority: {priority}")  
  
    def is_empty(self):
```

```

    """Check if the priority queue is empty."""
    return len(self.queue) == 0
# Example usage
if __name__ == "__main__":
    pq = PriorityQueue()
    pq.enqueue("Task 1", priority=2)
    pq.enqueue("Task 2", priority=1)
    pq.enqueue("Task 3", priority=3)

    pq.display() # Output: Priority Queue contents: Value: Task
2, Priority: 1 Value: Task 1, Priority: 2 Value: Task 3, Priority:
3

    print("Dequeued item:", pq.dequeue()) # Output: Dequeued
item: Task 2
    print("Dequeued item:", pq.dequeue()) # Output: Dequeued
item: Task 1
    print("Dequeued item:", pq.dequeue()) # Output: Dequeued
item: Task 3
    print("Dequeued item from empty queue:", pq.dequeue()) # Output:
Priority Queue is empty. Cannot dequeue. Dequeued item
from empty queue: None

```

## OUTPUT:

Priority Queue contents:  
Value: Task 2, Priority: 1  
Value: Task 1, Priority: 2  
Value: Task 3, Priority: 3  
Dequeued item: Task 2  
Dequeued item: Task 1  
Dequeued item: Task 3  
Priority Queue is empty. Cannot dequeue.  
Dequeued item from empty queue: None

## Task Description #8 – Deque

Task: Use AI to implement a double-ended queue using collections.deque.

Sample Input Code:

```
class DequeDS:
```

```
    pass
```

Expected Output:

- Insert and remove from both ends with docstrings.

## PROMPT:

```
#Generate a simple Python Deque implementation using
collections.deque with insert_front, insert_rear, remove_front,
```

```

A11.1.py > ...
314     class PriorityQueue:
315         return None
316
317     def display(self):
318         """Display the contents of the priority queue."""
319         print("Priority Queue contents:")
320         for priority, value in sorted(self.queue): # Sort by priority for display
321             print(f"Value: {value}, Priority: {priority}")
322
323     def is_empty(self):
324         """Check if the priority queue is empty."""
325         return len(self.queue) == 0
326
327     # Example usage
328     if __name__ == "__main__":
329         pq = PriorityQueue()
330         pq.enqueue("Task 1", priority=2)
331         pq.enqueue("Task 2", priority=1)
332         pq.enqueue("Task 3", priority=3)
333
334         pq.display() # Output: Priority Queue contents: Value: Task 2, Priority: 1 Value: Task 1, Priority: 2 Value: Task 3, Priority: 3
335
336         print("Dequeued item:", pq.dequeue()) # Output: Dequeued item: Task 2
337         print("Dequeued item:", pq.dequeue()) # Output: Dequeued item: Task 1
338         print("Dequeued item:", pq.dequeue()) # Output: Dequeued item: Task 3
339         print("Dequeued item from empty queue:", pq.dequeue()) # Output: Priority Queue is empty. Cannot dequeue. Dequeued item from empty queue: None
340
341

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
saivenkatesh@Sais-MacBook-Air AI Coding % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
● saivenkatesh@Sais-MacBook-Air AI Coding % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"
Priority Queue contents:
Value: Task 2, Priority: 1
Value: Task 1, Priority: 2
Value: Task 3, Priority: 3
Dequeued item: Task 2
Dequeued item: Task 1
Dequeued item: Task 3
Priority Queue is empty. Cannot dequeue.
Dequeued item from empty queue: None

```

and `remove_rear` methods, include clear natural docstrings, keep the code clean and readable for a college lab record, and provide a small example usage.

## CODE:

```

from collections import deque
class Deque:
    """A simple Deque (double-ended queue) implementation using
collections.deque."""

    def __init__(self):
        """Initialize an empty deque."""
        self.deque = deque() # Use deque from the collections
module

    def insert_front(self, item):
        """Insert an item at the front of the deque."""
        self.deque.appendleft(item) # Add item to the front

    def insert_rear(self, item):
        """Insert an item at the rear of the deque."""
        self.deque.append(item) # Add item to the rear

    def remove_front(self):
        """Remove and return the item at the front of the deque.
Returns None if the deque is empty."""

```

```

        if not self.is_empty():
            return self.deque.popleft() # Remove and return item
from the front
        else:
            print("Deque is empty. Cannot remove from front.")
            return None

    def remove_rear(self):
        """Remove and return the item at the rear of the deque.
        Returns None if the deque is empty."""
        if not self.is_empty():
            return self.deque.pop() # Remove and return item from
the rear
        else:
            print("Deque is empty. Cannot remove from rear.")
            return None

    def is_empty(self):
        """Check if the deque is empty."""
        return len(self.deque) == 0
# Example usage
if __name__ == "__main__":
    my_deque = Deque()
    my_deque.insert_rear(1)
    my_deque.insert_rear(2)
    my_deque.insert_front(0)

    print("Removed from front:", my_deque.remove_front()) #
Output: Removed from front: 0
    print("Removed from rear:", my_deque.remove_rear()) # Output:
Removed from rear: 2
    print("Removed from rear:", my_deque.remove_rear()) # Output:
Removed from rear: 1
    print("Removed from empty deque:", my_deque.remove_front()) #
Output: Deque is empty. Cannot remove from front. Removed from
empty deque: None
    print("Removed from empty deque:", my_deque.remove_rear()) #
Output: Deque is empty. Cannot remove from rear. Removed from
empty deque: None

```

## OUTPUT:

Removed from front: 0  
 Removed from rear: 2  
 Removed from rear: 1  
 Deque is empty. Cannot remove from front.  
 Removed from empty deque: None  
 Deque is empty. Cannot remove from rear.  
 Removed from empty deque: None

```

A11.1.py > ...
360 class Deque:
375     def remove_front(self):
382         return None
383
384     def remove_rear(self):
385         """Remove and return the item at the rear of the deque.
386         Returns None if the deque is empty."""
387         if not self.is_empty():
388             return self.deque.pop() # Remove and return item from the rear
389         else:
390             print("Deque is empty. Cannot remove from rear.")
391         return None
392
393     def is_empty(self):
394         """Check if the deque is empty."""
395         return len(self.deque) == 0
396
# Example usage
397 if __name__ == "__main__":
398     my_deque = Deque()
399     my_deque.insert_rear(1)
400     my_deque.insert_rear(2)
401     my_deque.insert_front(0)
402
403     print("Removed from front:", my_deque.remove_front()) # Output: Removed from front: 0
404     print("Removed from rear:", my_deque.remove_rear()) # Output: Removed from rear: 2
405     print("Removed from rear:", my_deque.remove_rear()) # Output: Removed from rear: 1
406     print("Removed from empty deque:", my_deque.remove_front()) # Output: Deque is empty. Cannot remove from front. Removed from empty deque: N
407     print("Removed from empty deque:", my_deque.remove_rear()) # Output: Deque is empty. Cannot remove from rear. Removed from empty deque: Non
408

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS    POSTMAN CONSOLE  
 saivenkatesh@Sais-MacBook-Air AI Coding % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"  
 Priority Queue is empty. Cannot dequeue.  
 Dequeued item from empty queue: None  
 ● saivenkatesh@Sais-MacBook-Air AI Coding % /usr/bin/python3 "/Users/saivenkatesh/Desktop/AI Coding/A11.1.py"  
 Removed from front: 0  
 Removed from rear: 2  
 Removed from rear: 1  
 Deque is empty. Cannot remove from front.  
 Removed from empty deque: None  
 Deque is empty. Cannot remove from rear.  
 Removed from empty deque: None

## Task Description #9 Real-Time Application Challenge – Choose the Right Data Structure

### Scenario:

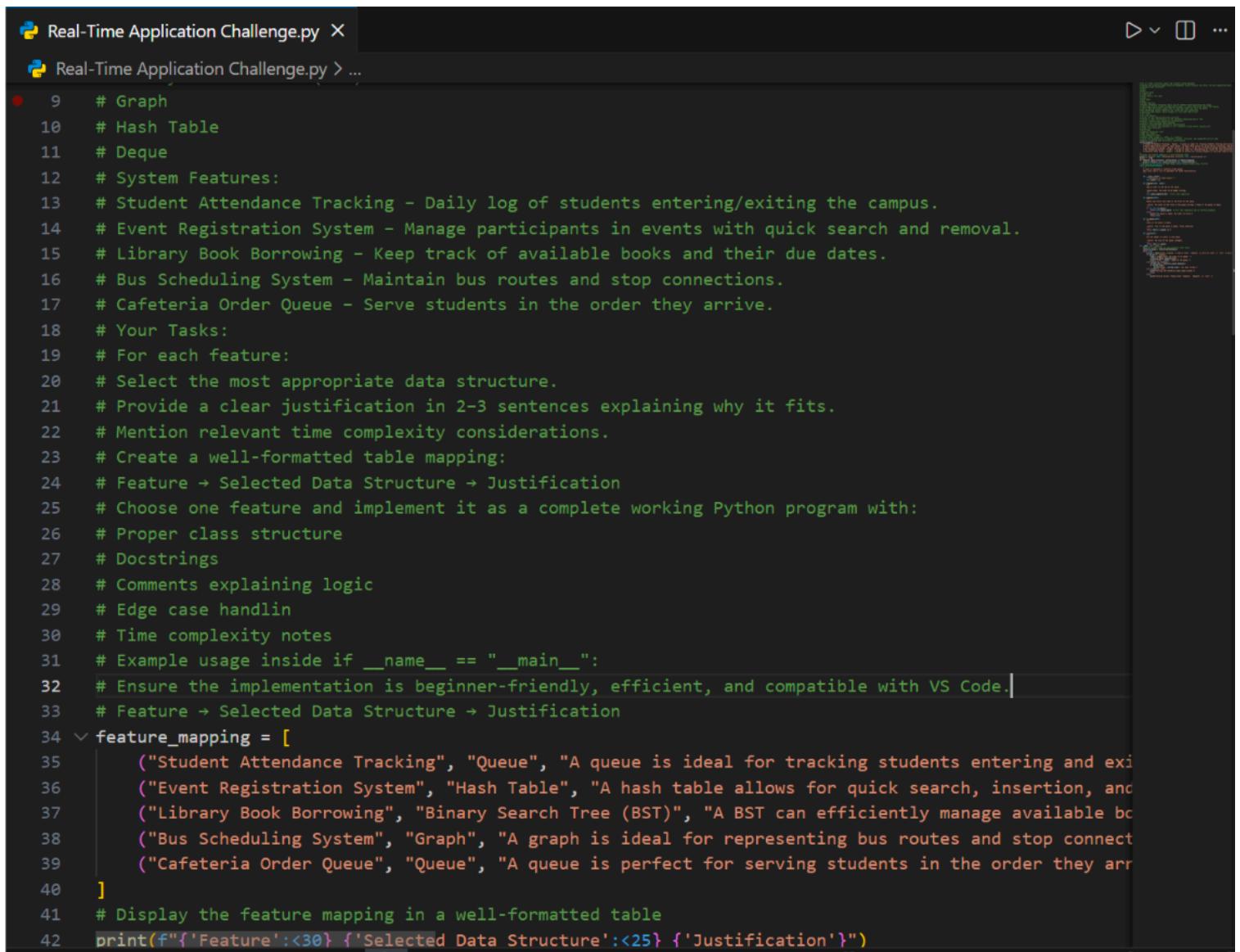
Your college wants to develop a Campus Resource Management System that handles:

1. Student Attendance Tracking – Daily log of students entering/exiting the campus.
2. Event Registration System – Manage participants in events with quick search and removal.
3. Library Book Borrowing – Keep track of available books and their due dates.
4. Bus Scheduling System – Maintain bus routes and stop connections.

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

### Student Task:

- For each feature, select the most appropriate data structure from the list below:
  - Stack
  - Queue
  - Priority Queue
  - Linked List



The screenshot shows a code editor window titled "Real-Time Application Challenge.py". The code is a Python script with the following content:

```
 9 # Graph
10 # Hash Table
11 # Deque
12 # System Features:
13 # Student Attendance Tracking - Daily log of students entering/exiting the campus.
14 # Event Registration System - Manage participants in events with quick search and removal.
15 # Library Book Borrowing - Keep track of available books and their due dates.
16 # Bus Scheduling System - Maintain bus routes and stop connections.
17 # Cafeteria Order Queue - Serve students in the order they arrive.
18 # Your Tasks:
19 # For each feature:
20 # Select the most appropriate data structure.
21 # Provide a clear justification in 2-3 sentences explaining why it fits.
22 # Mention relevant time complexity considerations.
23 # Create a well-formatted table mapping:
24 # Feature → Selected Data Structure → Justification
25 # Choose one feature and implement it as a complete working Python program with:
26 # Proper class structure
27 # Docstrings
28 # Comments explaining logic
29 # Edge case handling
30 # Time complexity notes
31 # Example usage inside if __name__ == "__main__":
32 # Ensure the implementation is beginner-friendly, efficient, and compatible with VS Code.
33 # Feature → Selected Data Structure → Justification
34 feature_mapping = [
35     ("Student Attendance Tracking", "Queue", "A queue is ideal for tracking students entering and exiting the campus."),
36     ("Event Registration System", "Hash Table", "A hash table allows for quick search, insertion, and removal of event participants."),
37     ("Library Book Borrowing", "Binary Search Tree (BST)", "A BST can efficiently manage available books and their due dates."),
38     ("Bus Scheduling System", "Graph", "A graph is ideal for representing bus routes and stop connections."),
39     ("Cafeteria Order Queue", "Queue", "A queue is perfect for serving students in the order they arrive.")
40 ]
41 # Display the feature mapping in a well-formatted table
42 print(f"{'Feature':<30} {'Selected Data Structure':<25} {'Justification'}")
```

- Binary Search Tree (BST)
- Graph
- Hash Table
- Deque
- Justify your choice in 2–3 sentences per feature.
- Implement one selected feature as a working Python program with AI-assisted code generation.

Real-Time Application Challenge.py X

Real-Time Application Challenge.py > ...

```
40     ]
41     # Display the feature mapping in a well-formatted table
42     print(f"{'Feature':<30} {'Selected Data Structure':<25} {'Justification'}")
43     print("-" * 80)
44     for feature, data_structure, justification in feature_mapping:
45         print(f"{feature:<30} {data_structure:<25} {justification}")
46     # Implementing the Cafeteria Order Queue feature using a Queue data structure
47     class CafeteriaOrderQueue:
48         """
49             A class to represent a cafeteria order queue.
50             This class uses a list to implement the queue functionality.
51         """
52
53         def __init__(self):
54             """Initialize an empty queue."""
55             self.queue = []
56
57         def enqueue(self, order):
58             """
59                 Add an order to the end of the queue.
60
61                 :param order: The order to be added (string).
62             """
63             self.queue.append(order) # O(1) time complexity
64
65         def dequeue(self):
66             """
67                 Remove and return the order at the front of the queue.
68
69                 :return: The order at the front of the queue (string) or None if the queue is empty.
70             """
71             if not self.is_empty():
72                 return self.queue.pop(0) # O(n) time complexity due to shifting elements
73             else:
```

Expected Output:

- A table mapping feature → chosen data structure → justification.
- A functional Python program implementing the chosen feature with comments and docstrings.

Real-Time Application Challenge.py X

Real-Time Application Challenge.py > ...

```
47  class CafeteriaOrderQueue:
48      def dequeue(self):
49          """Return the order at the front of the queue (string) or None if the queue is empty.
50          """
51          if not self.is_empty():
52              return self.queue.pop(0) # O(n) time complexity due to shifting elements
53          else:
54              print("The queue is empty. No orders to serve.")
55              return None
56
57      def is_empty(self):
58          """
59          Check if the queue is empty.
60
61          :return: True if the queue is empty, False otherwise.
62          """
63          return len(self.queue) == 0
64
65      def size(self):
66          """
67          Get the number of orders in the queue.
68
69          :return: The size of the queue (integer).
70          """
71          return len(self.queue)
72
73  if __name__ == "__main__":
74      #give me dynamic input for the cafeteria order queue
75      cafeteria_queue = CafeteriaOrderQueue()
76      while True:
77          action = input("Enter 'enqueue' to add an order, 'dequeue' to serve an order, or 'exit' to qu
78          if action == 'enqueue':
79              order = input("Enter the order to be added: ")
80              cafeteria_queue.enqueue(order)
81              print(f"Order '{order}' added to the queue.")
82
83  Ln 32, Col 90  Spaces: 4  UTF-8  CRLF  { } Python  3.14.0  Go Live  Prettier
```

Real-Time Application Challenge.py X

Real-Time Application Challenge.py > ...

```
47  class CafeteriaOrderQueue:
48      def size(self):
49          """
50          Get the number of orders in the queue.
51
52          :return: The size of the queue (integer).
53          """
54          return len(self.queue)
55
56  if __name__ == "__main__":
57      #give me dynamic input for the cafeteria order queue
58      cafeteria_queue = CafeteriaOrderQueue()
59      while True:
60          action = input("Enter 'enqueue' to add an order, 'dequeue' to serve an order, or 'exit' to qu
61          if action == 'enqueue':
62              order = input("Enter the order to be added: ")
63              cafeteria_queue.enqueue(order)
64              print(f"Order '{order}' added to the queue.")
65          elif action == 'dequeue':
66              served_order = cafeteria_queue.dequeue()
67              if served_order:
68                  print(f"Order '{served_order}' has been served.")
69          elif action == 'exit':
70              print("Exiting the cafeteria order queue system.")
71              break
72          else:
73              print("Invalid action. Please enter 'enqueue', 'dequeue', or 'exit'.")
74
75  Ln 32, Col 90  Spaces: 4  UTF-8  CRLF  { } Python  3.14.0  Go Live  Prettier
```

The screenshot shows a terminal window with the following content:

```

PS C:\Users\sonu\Desktop\AI_Coding> & C:/Python314/python.exe "c:/Users/sonu/Desktop/AI_Coding/Real-Time Application Challenge.py"
Feature           Selected Data Structure   Justification
-----
Student Attendance Tracking   Queue           A queue is ideal for tracking students entering and exiting the campus as it follows the First-In-First-Out (FIFO) principle, allowing for efficient logging of attendance. Time complexity for enqueue and dequeue operations is O(1).
Event Registration System    Hash Table       A hash table allows for quick search, insertion, and deletion of participants in events, making it suitable for managing registrations. Time complexity for search, insert, and delete operations is O(1) on average.
Library Book Borrowing      Binary Search Tree (BST) A BST can efficiently manage available books and their due dates by allowing for sorted data storage and quick retrieval. Time complexity for search, insert, and delete operations is O(log n) on average.
Bus Scheduling System        Graph          A graph is ideal for representing bus routes and stop connections as it can model complex relationships between stops and routes. Time complexity for traversal operations is O(V + E), where V is the number of vertices (stops) and E is the number of edges (routes).
Cafeteria Order Queue       Queue          A queue is perfect for serving students in the order they arrive at the cafeteria, ensuring fairness. Time complexity for enqueue and dequeue operations is O(1).
Enter 'enqueue' to add an order, 'dequeue' to serve an order, or 'exit' to quit: enqueue
Enter the order to be added: 5 7 9 10
Order '5 7 9 10' added to the queue.
Enter 'enqueue' to add an order, 'dequeue' to serve an order, or 'exit' to quit: dequeue
Order '5 7 9 10' has been served.
Enter 'enqueue' to add an order, 'dequeue' to serve an order, or 'exit' to quit: exit
Exiting the cafeteria order queue system.
PS C:\Users\sonu\Desktop\AI_Coding>

```

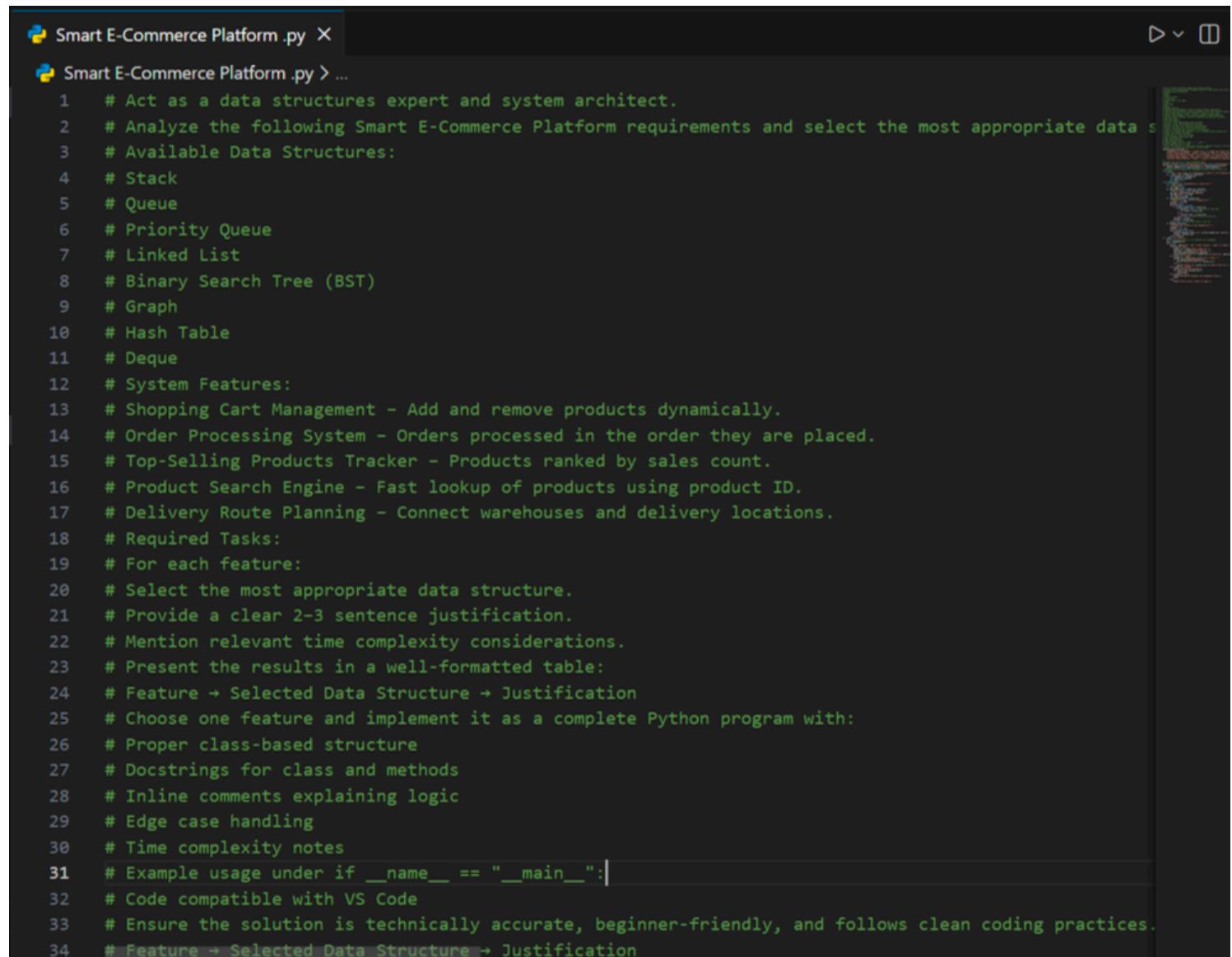
## Task Description #10: Smart E-Commerce Platform – Data Structure Challenge

An e-commerce company wants to build a Smart Online Shopping System with:

1. Shopping Cart Management – Add and remove products dynamically.
2. Order Processing System – Orders processed in the order they are placed.
3. Top-Selling Products Tracker – Products ranked by sales count.
4. Product Search Engine – Fast lookup of products using product ID.
5. Delivery Route Planning – Connect warehouses and delivery locations.

**Student Task:**

- For each feature, select the most appropriate data structure from the list below:
  - Stack
  - Queue
  - Priority Queue
  - Linked List



```

Smart E-Commerce Platform.py X
Smart E-Commerce Platform.py > ...

1 # Act as a data structures expert and system architect.
2 # Analyze the following Smart E-Commerce Platform requirements and select the most appropriate data structures:
3 # Available Data Structures:
4 # Stack
5 # Queue
6 # Priority Queue
7 # Linked List
8 # Binary Search Tree (BST)
9 # Graph
10 # Hash Table
11 # Deque
12 # System Features:
13 # Shopping Cart Management - Add and remove products dynamically.
14 # Order Processing System - Orders processed in the order they are placed.
15 # Top-Selling Products Tracker - Products ranked by sales count.
16 # Product Search Engine - Fast lookup of products using product ID.
17 # Delivery Route Planning - Connect warehouses and delivery locations.
18 # Required Tasks:
19 # For each feature:
20 # Select the most appropriate data structure.
21 # Provide a clear 2-3 sentence justification.
22 # Mention relevant time complexity considerations.
23 # Present the results in a well-formatted table:
24 # Feature → Selected Data Structure → Justification
25 # Choose one feature and implement it as a complete Python program with:
26 # Proper class-based structure
27 # Docstrings for class and methods
28 # Inline comments explaining logic
29 # Edge case handling
30 # Time complexity notes
31 # Example usage under if __name__ == "__main__":
32 # Code compatible with VS Code
33 # Ensure the solution is technically accurate, beginner-friendly, and follows clean coding practices.
34 # Feature → Selected Data Structure → Justification

```

- Binary Search Tree (BST)
- Graph
- Hash Table
- Deque
- Justify your choice in 2–3 sentences per feature.
- Implement one selected feature as a working Python program with AI-assisted code generation.

Expected Output:

- A table mapping feature → chosen data structure → justification.

```
Smart E-Commerce Platform .py X
Smart E-Commerce Platform .py > ...
33     # Ensure the solution is technically accurate, beginner-friendly, and follows clean coding practices.
34     # Feature → Selected Data Structure → Justification
35     features_data_structures = [
36         ("Shopping Cart Management", "Linked List", "A linked list allows for dynamic addition and removal of items from the cart without needing to shift all other items."),
37         ("Order Processing System", "Queue", "A queue is ideal for processing orders in the order they are received, ensuring first-in, first-out (FIFO) behavior."),
38         ("Top-Selling Products Tracker", "Priority Queue", "A priority queue can efficiently keep track of products based on their current popularity or sales volume."),
39         ("Product Search Engine", "Hash Table", "A hash table allows for fast lookup of products using product IDs or names as keys."),
40         ("Delivery Route Planning", "Graph", "A graph is suitable for modeling warehouses and delivery locations, allowing for complex route optimization calculations."),

41     ]
42     # Display the results in a well-formatted table
43     print(f"{'Feature':<30} {'Selected Data Structure':<25} {'Justification'}")
44     for feature, data_structure, justification in features_data_structures:
45         print(f"{feature:<30} {data_structure:<25} {justification}")
46     # Implementing the Shopping Cart Management feature using a Linked List
47     class Node:
48         """A node in the linked list representing a product in the shopping cart."""
49         def __init__(self, product_id, quantity):
50             self.product_id = product_id
51             self.quantity = quantity
52             self.next = None
53     class ShoppingCart:
54         """A shopping cart implemented as a linked list."""
55         def __init__(self):
56             self.head = None
57         def add_product(self, product_id, quantity):
58             """Add a product to the shopping cart."""
59             new_node = Node(product_id, quantity)
60             new_node.next = self.head
61             self.head = new_node
62         def remove_product(self, product_id):
63             """Remove a product from the shopping cart."""
64             current = self.head
65             previous = None
66             while current is not None:
```

- A functional Python program implementing the chosen feature with comments and docstrings.

```
Smart E-Commerce Platform.py X
Smart E-Commerce Platform.py > ...
53 class ShoppingCart:
54     self.head = None
55
56     def remove_product(self, product_id):
57         """Remove a product from the shopping cart."""
58         current = self.head
59         previous = None
60
61         while current is not None:
62             if current.product_id == product_id:
63                 if previous is None: # Removing the head node
64                     self.head = current.next
65                 else:
66                     previous.next = current.next
67
68                 return True # Product removed successfully
69             previous = current
70             current = current.next
71
72         return False # Product not found in the cart
73
74     def display_cart(self):
75         """Display the contents of the shopping cart."""
76         products = []
77
78         current = self.head
79
80         while current is not None:
81             products.append(f"Product ID: {current.product_id}, Quantity: {current.quantity}")
82             current = current.next
83
84         return products
85
86 if __name__ == "__main__":
87     #give me dynamic input for the shopping cart management
88     cart = ShoppingCart()
89     while True:
90         action = input("Enter 'add' to add a product, 'remove' to remove a product, 'display' to show
91         if action == 'add':
92             product_id = input("Enter product ID: ")
93             quantity = int(input("Enter quantity: "))
94             cart.add_product(product_id, quantity)
95             print(f"Added Product ID: {product_id} with Quantity: {quantity}")
96
97         elif action == 'remove':
98             product_id = input("Enter product ID to remove: ")
99             if cart.remove_product(product_id):
100                 print(f"Removed Product ID: {product_id}")
101             else:
102                 print(f"Product ID: {product_id} not found in the cart.")
103
104         elif action == 'display':
105             print("Shopping Cart Contents:")
106             for item in cart.display_cart():
107                 print(item)
108
109         elif action == 'exit':
110             print("Exiting the shopping cart management system.")
111             break
112         else:
113             print("Invalid action. Please try again.")
```

```
Smart E-Commerce Platform.py X
Smart E-Commerce Platform.py > ...
53 class ShoppingCart:
54     def display_cart(self):
55         return products
56
57     if __name__ == "__main__":
58         #give me dynamic input for the shopping cart management
59         cart = ShoppingCart()
60
61         while True:
62             action = input("Enter 'add' to add a product, 'remove' to remove a product, 'display' to show
63             if action == 'add':
64                 product_id = input("Enter product ID: ")
65                 quantity = int(input("Enter quantity: "))
66                 cart.add_product(product_id, quantity)
67                 print(f"Added Product ID: {product_id} with Quantity: {quantity}")
68
69             elif action == 'remove':
70                 product_id = input("Enter product ID to remove: ")
71                 if cart.remove_product(product_id):
72                     print(f"Removed Product ID: {product_id}")
73                 else:
74                     print(f"Product ID: {product_id} not found in the cart.")
75
76             elif action == 'display':
77                 print("Shopping Cart Contents:")
78                 for item in cart.display_cart():
79                     print(item)
80
81             elif action == 'exit':
82                 print("Exiting the shopping cart management system.")
83                 break
84             else:
85                 print("Invalid action. Please try again.")
```

Feature	Selected Data Structure	Justification
Shopping Cart Management	Linked List	A linked list allows for dynamic addition and removal of products without needing to shift elements, making it efficient for managing a shopping cart. Time complexity for adding/removing products is O(1).
Order Processing System	Queue	A queue is ideal for processing orders in the order they are received (FIFO). Time complexity for enqueue and dequeue operations is O(1).
Top-Selling Products Tracker	Priority Queue	A priority queue can efficiently keep track of top-selling products by maintaining a sorted order based on sales count. Time complexity for insertion and retrieval is O(log n).
Product Search Engine	Hash Table	A hash table allows for fast lookup of products using product IDs, with an average time complexity of O(1) for search operations. It also handles collisions effectively.
Delivery Route Planning	Graph	A graph is suitable for modeling warehouses and delivery locations, allowing for efficient route planning using algorithms like Dijkstra's or A*. Time complexity depends on the algorithm used, typically O(V + E) for Dijkstra's algorithm.

```
Enter 'add' to add a product, 'remove' to remove a product, 'display' to show cart contents, or 'exit' to quit: add
Enter product ID: 567903
Enter quantity: 300
Added Product ID: 567903 with Quantity: 300
Enter 'add' to add a product, 'remove' to remove a product, 'display' to show cart contents, or 'exit' to quit: display
Product ID: 567903, Quantity: 300
Enter 'add' to add a product, 'remove' to remove a product, 'display' to show cart contents, or 'exit' to quit: exit
```