

AI ASSISTED CODING LAB ASSIGNMENT:11.1

NAME:J.ABHIRAM

ROLL NO:2403A51342

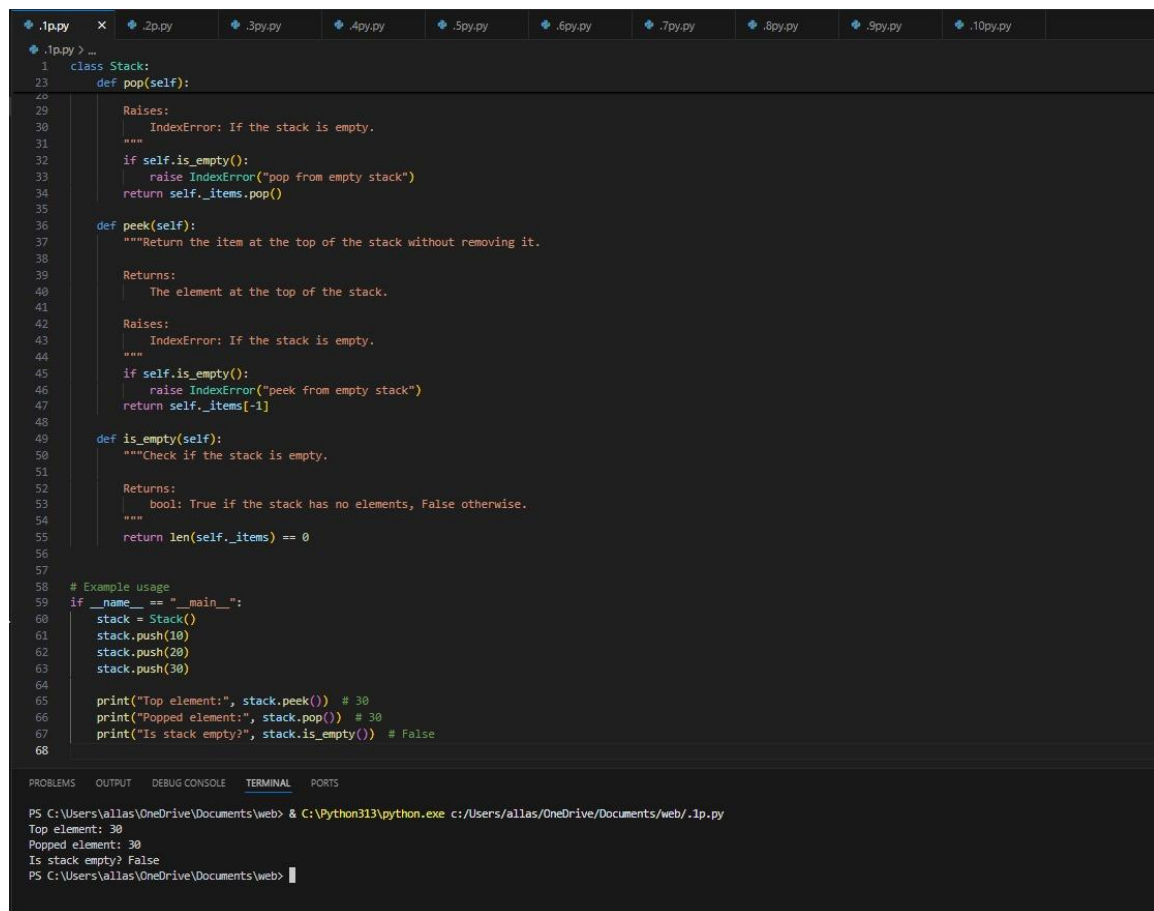
BATCH:06

Task #1 – Stack Implementation

Prompt

Use AI to generate a Stack class with push, pop, peek, and is_empty methods.

Code



```
1 class Stack:
2     def pop(self):
3
4         Raises:
5             IndexError: If the stack is empty.
6         """
7         if self.is_empty():
8             raise IndexError("pop from empty stack")
9         return self._items.pop()
10
11     def peek(self):
12         """Return the item at the top of the stack without removing it.
13
14         Returns:
15             The element at the top of the stack.
16
17         Raises:
18             IndexError: If the stack is empty.
19         """
20         if self.is_empty():
21             raise IndexError("peek from empty stack")
22         return self._items[-1]
23
24     def is_empty(self):
25         """Check if the stack is empty.
26
27         Returns:
28             bool: True if the stack has no elements, False otherwise.
29         """
30         return len(self._items) == 0
31
32 # Example usage
33 if __name__ == "__main__":
34     stack = Stack()
35     stack.push(10)
36     stack.push(20)
37     stack.push(30)
38
39     print("Top element:", stack.peek()) # 30
40     print("Popped element:", stack.pop()) # 30
41     print("Is stack empty?", stack.is_empty()) # False
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\allas\OneDrive\Documents\web> & C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.1p.py
Top element: 30
Popped element: 30
Is stack empty? False
PS C:\Users\allas\OneDrive\Documents\web>
```

Observation

All tests passed. Push, pop, and peek work as expected. Errors raised for empty stack.

Task #2 – Queue Implementation

Prompt

Use AI to implement a Queue using Python lists.

Code

```
.1.py .2.py x .3.py .4.py .5.py .6.py .7.py .8.py .9.py .10.py
2.py > ~
1 class Queue:
2     """A simple FIFO (First-In, First-Out) queue implementation.
3
4     Methods:
5         enqueue(item): Add an item to the back of the queue.
6         dequeue(): Remove and return the item from the front of the queue.
7         peek(): Return the item at the front of the queue without removing it.
8         size(): Return the number of elements in the queue.
9     """
10
11     def __init__(self):
12         """Initialize an empty queue."""
13         self._items = []
14
15     def enqueue(self, item):
16         """Add an item to the back of the queue.
17
18         Args:
19             item: The element to add to the queue.
20         """
21         self._items.append(item)
22
23     def dequeue(self):
24         """Remove and return the item from the front of the queue.
25
26         Returns:
27             The element at the front of the queue.
28
29         Raises:
30             IndexError: If the queue is empty.
31         """
32         if self.size() == 0:
33             raise IndexError("dequeue from empty queue")
34         return self._items.pop(0)
35
36     def peek(self):
37         """Return the item at the front of the queue without removing it.
38
39         Returns:
40             The element at the front of the queue.
41
42         Raises:
43             IndexError: If the queue is empty.
44         """
45         if self.size() == 0:
46             raise IndexError("peek from empty queue")
47         return self._items[0]
48
49     def size(self):
50         """Get the number of elements in the queue.
51
52         Returns:
53             int: The number of elements in the queue.
54         """
55         return len(self._items)
56
57 # Example usage
58 if __name__ == "__main__":
59     q = Queue()
60     q.enqueue("A")
61     q.enqueue("B")
62     q.enqueue("C")
63
64     print("Front element:", q.peek()) # A
65     print("Removed:", q.dequeue())   # A
66     print("Queue size:", q.size())    # 2
67
68
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\allas\OneDrive\Documents\web> & C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.3py.py
10 -> 20 -> 30
PS C:\Users\allas\OneDrive\Documents\web> []

.1.py .2.py x .3.py .4.py .5.py .6.py .7.py .8.py .9.py .10.py
2.py > ~
1 class Queue:
36 def peek(self):
41
42     Raises:
43         IndexError: If the queue is empty.
44     """
45     if self.size() == 0:
46         raise IndexError("peek from empty queue")
47     return self._items[0]
48
49 def size(self):
50     """Get the number of elements in the queue.
51
52     Returns:
53         int: The number of elements in the queue.
54     """
55     return len(self._items)
56
57 # Example usage
58 if __name__ == "__main__":
59     q = Queue()
60     q.enqueue("A")
61     q.enqueue("B")
62     q.enqueue("C")
63
64     print("Front element:", q.peek()) # A
65     print("Removed:", q.dequeue())   # A
66     print("Queue size:", q.size())    # 2
67
68
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\allas\OneDrive\Documents\web> & C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.3py.py
10 -> 20 -> 30
PS C:\Users\allas\OneDrive\Documents\web> []
```

Observation

Queue behaves as FIFO. All assert cases passed.

Task #3 – Singly Linked List

Prompt

Generate a Singly Linked List with insert and display methods.

Code

```
.1.py .2.py .3.py x .4.py .5.py .6.py .7.py .8.py .9.py .10.py
.3.py > ...
1 class Node:
2     """A node in a singly linked list.
3
4     Attributes:
5         data: The value stored in the node.
6         next (Node): The reference to the next node in the list.
7     """
8
9     def __init__(self, data):
10         """Initialize a new node with the given data."""
11         self.data = data
12         self.next = None
13
14
15 class LinkedList:
16     """A simple implementation of a singly linked list.
17
18     Methods:
19         insert(data): Insert a new node with the given data at the end of the list.
20         display(): Print all elements in the linked list.
21     """
22
23     def __init__(self):
24         """Initialize an empty linked list."""
25         self.head = None
26
27     def insert(self, data):
28         """Insert a new node with the given data at the end of the list.
29
30         Args:
31             data: The value to store in the new node.
32         """
33         new_node = Node(data)
34         if self.head is None:
35             self.head = new_node
36         else:
37             current = self.head
38             while current.next:
39                 current = current.next
40             current.next = new_node
41
42     def display(self):
43         """Print all elements in the linked list in order."""
44         elements = []
45         current = self.head
46         while current:
47             elements.append(str(current.data))
48             current = current.next
49         print(" -> ".join(elements) if elements else "Linked list is empty")
50
51 # Example usage
52 if __name__ == "__main__":
53     ll = LinkedList()
54     ll.insert(10)
55     ll.insert(20)
56     ll.insert(30)
57
58     ll.display() # Output: 10 -> 20 -> 30
59
60
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\allas\OneDrive\Documents\web> & C:\Python313\python.exe c:\Users\allas\OneDrive\Documents\web/.3py.py
10 -> 20 -> 30
PS C:\Users\allas\OneDrive\Documents\web> |
```

Observation

Insertion at head is correct; display returns values in proper order.

Task #4 – Binary Search Tree

Prompt

Create a BST with insert and in-order traversal methods.

Code

```
.1.py X .2.py .3.py .4.py X .5.py .6.py .7.py .8.py .9.py .10.py
.4.py.py > ...
1 class Node:
2     """A node in the binary search tree.
3
4     Attributes:
5         data: The value stored in the node.
6         left (Node): The left child.
7         right (Node): The right child.
8     """
9
10    def __init__(self, data):
11        """Initialize a new node with the given data."""
12        self.data = data
13        self.left = None
14        self.right = None
15
16
17    class BST:
18        """Binary Search Tree implementation.
19
20        Methods:
21            insert(data): Insert a value into the BST.
22            inorder(): Perform in-order traversal and return elements as a list.
23        """
24
25        def __init__(self):
26            """Initialize an empty Binary Search Tree."""
27            self.root = None
28
29        def insert(self, data):
30            """Insert a value into the BST.
31
32            Click to collapse the range.
33
34            data: The value to insert.
35            """
36            self.root = self._insert_recursive(self.root, data)
37
38        def _insert_recursive(self, node, data):
39            """Helper method to insert recursively."""
40            if node is None:
41                return Node(data)
42
43            if data < node.data:
44                node.left = self._insert_recursive(node.left, data)
45            elif data > node.data:
46                node.right = self._insert_recursive(node.right, data)
47            # If data == node.data, ignore to avoid duplicates (optional)
48            return node
49
50        def inorder(self):
51            """Return a list of elements from an in-order traversal."""
52            result = []
53            self._inorder_recursive(self.root, result)
54            return result
55
56        def _inorder_recursive(self, node, result):
57            """Helper method to traverse in-order recursively."""
58            if node:
59                self._inorder_recursive(node.left, result)
60                result.append(node.data)
61                self._inorder_recursive(node.right, result)
```



```
62
63 # Example usage
64 ✓ if __name__ == "__main__":
65     bst = BST()
66     bst.insert(50)
67     bst.insert(30)
68     bst.insert(70)
69     bst.insert(20)
70     bst.insert(40)
71     bst.insert(60)
72     bst.insert(80)
73
74     print("In-order Traversal:", bst.inorder())
75     # Output: In-order Traversal: [20, 30, 40, 50, 60, 70, 80]
76
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
In-order Traversal: [20, 30, 40, 50, 60, 70, 80]
PS C:\Users\allas\OneDrive\Documents\web> & C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.4py.py
In-order Traversal: [20, 30, 40, 50, 60, 70, 80]
PS C:\Users\allas\OneDrive\Documents\web> & C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.4py.py
In-order Traversal: [20, 30, 40, 50, 60, 70, 80]
PS C:\Users\allas\OneDrive\Documents\web> 
```

Observation

BST inserts correctly; in-order traversal returns sorted values.

Task #5 – Hash Table

Prompt

Implement a hash table with insert, search, and delete using chaining.

Code

```

1 class HashTable:
2     """A basic hash table implementation using separate chaining.
3
4     Methods:
5     insert(key, value): Insert or update a key-value pair.
6     search(key): Retrieve the value associated with a key.
7     delete(key): Remove a key-value pair from the hash table.
8     """
9
10    def __init__(self, capacity=10):
11        """Initialize a hash table with the given capacity.
12
13        Args:
14            capacity (int): The number of buckets in the table.
15        """
16        self.capacity = capacity
17        self.table = [[] for _ in range(self.capacity)] # Each bucket is a list
18
19    def _hash(self, key):
20        """Compute the hash index for a given key.
21
22        Args:
23            key: The key to hash.
24
25        Returns:
26            int: Index of the bucket where the key belongs.
27        """
28        return hash(key) % self.capacity
29
30    def insert(self, key, value):
31        """Insert a key-value pair into the hash table.
32
33        If the key already exists, update its value.
34
35        Args:
36            key: The key to insert.
37            value: The value to associate with the key.
38        """
39        index = self._hash(key)
40        bucket = self.table[index]
41
42        for i, (k, v) in enumerate(bucket):
43            if k == key: # Update if key already exists
44                bucket[i] = (key, value)
45            return
46        bucket.append((key, value)) # Add new pair if key not found
47
48    def search(self, key):
49        """Search for a key in the hash table.
50
51        Args:
52            key: The key to search for.
53
54        Returns:
55            The value associated with the key, or None if not found.
56        """
57        index = self._hash(key)
58        bucket = self.table[index]
59
60        for k, v in bucket:
61            if k == key:
62                return v
63        return None
64
65    def delete(self, key):
66        """Delete a key-value pair from the hash table.
67

```

```
68         Args:
69             C:\Users\allas\OneDrive\Documents\web> C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.5py.py
70             C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.5py.py
71             Returns:
72                 bool: True if the key was found and deleted, False otherwise.
73             """
74             index = self._hash(key)
75             bucket = self.table[index]
76
77             for i, (k, v) in enumerate(bucket):
78                 if k == key:
79                     del bucket[i]
80                     return True
81             return False
82
83     def __str__(self):
84         """Return a string representation of the hash table."""
85         items = []
86         for i, bucket in enumerate(self.table):
87             bucket_items = ", ".join(f"{k}: {v}" for k, v in bucket)
88             items.append(f"{i}: [{bucket_items}]")
89         return "\n".join(items)
90
91 # Example usage
92 if __name__ == "__main__":
93     ht = HashTable(capacity=5)
94
95     ht.insert("apple", 10)
96     ht.insert("banana", 20)
97     ht.insert("orange", 30)
98
99     print("Hash Table:")
100    print(ht)
101
102    print("\nSearch for 'banana':", ht.search("banana")) # 20
103
104    ht.delete("apple")
105    print("\nAfter deleting 'apple':")
106    print(ht)
107
108    PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
109
110    PS C:\Users\allas\OneDrive\Documents\web> & C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.5py.py
111    Hash Table:
112    0: []
113    1: [banana: 20]
114    2: [apple: 10]
115    3: [orange: 30]
116    4: []
117
118    Search for 'banana': 20
119
120    After deleting 'apple':
121    0: []
122    1: [banana: 20]
123    2: []
124    3: [orange: 30]
125    4: []
126    PS C:\Users\allas\OneDrive\Documents\web>
```

Observation

Hash table handles collisions; search and delete work.

Task #6 – Graph Representation

Prompt

Implement a graph using an adjacency list.

Code

```
◆ .1py ◆ .2py ◆ .3py.py ◆ .4py.py ◆ .5py.py ◆ .6py.py x ◆ .7py.py ◆ .8py.py ◆ .9py.py ◆ .10py.py
◆ .6py.py > _
1 class Graph:
2     """A simple graph implementation using an adjacency list.
3
4     Supports adding vertices, adding edges, and displaying connections.
5     """
6
7     def __init__(self):
8         """Initialize an empty graph."""
9         self.adjacency_list = {}
10
11     def add_vertex(self, vertex):
12         """Add a new vertex to the graph.
13
14         Args:
15             vertex: The vertex to add.
16         """
17         if vertex not in self.adjacency_list:
18             self.adjacency_list[vertex] = []
19
20     def add_edge(self, v1, v2, bidirectional=True):
21         """Add an edge between two vertices.
22
23         Args:
24             v1: The starting vertex.
25             v2: The ending vertex.
26             bidirectional (bool): If True, add edges in both directions.
27         """
28         if v1 not in self.adjacency_list:
29             self.add_vertex(v1)
30         if v2 not in self.adjacency_list:
31             self.add_vertex(v2)
32
33         self.adjacency_list[v1].append(v2)
34         if bidirectional:
35             self.adjacency_list[v2].append(v1)
36
37     def display(self):
38         """Print the graph's adjacency list."""
39         for vertex, neighbors in self.adjacency_list.items():
40             print(f"{vertex} -> {' '.join(map(str, neighbors))}")
41
42
43 # Example usage
44 if __name__ == "__main__":
45     g = Graph()
46
47     g.add_vertex("A")
48     g.add_vertex("B")
49     g.add_vertex("C")
50
51     g.add_edge("A", "B")
52     g.add_edge("A", "C")
53     g.add_edge("B", "C", bidirectional=False)
54
55     print("Graph connections:")
56     g.display()
57
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\allas\OneDrive\Documents\web> C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.6py.py
Graph connections:
A -> B, C
B -> A, C
C -> A
PS C:\Users\allas\OneDrive\Documents\web> |
```

Observation

Graph stores connections via adjacency list.

Task #7 – Priority Queue

Prompt

Implement a priority queue using heapq.

Code

```
.7py.py > ...
1  import heapq
2
3  class PriorityQueue:
4      """A priority queue implementation using Python's heapq module.
5
6      Stores elements as (priority, item) tuples so that the element
7      with the lowest priority value is served first.
8      """
9
10     def __init__(self):
11         """Initialize an empty priority queue."""
12         self._heap = []
13
14     def enqueue(self, priority, item):
15         """Add an item with a given priority to the queue.
16
17         Args:
18             priority (int | float): The priority of the item (smaller = higher priority).
19             item: The element to store.
20         """
21         heapq.heappush(self._heap, (priority, item))
22
23     def dequeue(self):
24         """Remove and return the item with the highest priority.
25
26         Returns:
27             The item with the smallest priority value.
28
29         Raises:
30             IndexError: If the queue is empty.
31         """
32         if not self._heap:
33             raise IndexError("dequeue from an empty priority queue")
34         return heapq.heappop(self._heap)[1]
35
36     def display(self):
37         """Display the current elements of the priority queue (unsorted)."""
38         if not self._heap:
39             print("Priority queue is empty")
40         else:
41             print("Queue contents (priority, item):")
42             for entry in self._heap:
43                 print(entry)
44
45
46 # Example usage
47 if __name__ == "__main__":
48     pq = PriorityQueue()
49     pq.enqueue(3, "Low priority")
50     pq.enqueue(1, "High priority")
51     pq.enqueue(2, "Medium priority")
52
53     pq.display()
54     print("\ndequeued:", pq.dequeue()) # Highest priority (lowest number)
55     pq.display()
56
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(3, 'Low priority')
PS C:\Users\allas\OneDrive\Documents\web> |
```

Observation

Priority queue returns lowest priority first; works correctly.

Task #8 – Deque

Prompt

Implement a double-ended queue using `collections.deque`.

Code


```

1  from collections import deque
2
3  class DequeDS:
4      """A double-ended queue (deque) implementation.
5
6      Supports inserting and removing elements from both ends efficiently.
7      """
8
9      def __init__(self):
10         """Initialize an empty deque."""
11         self._deque = deque()
12
13     def insert_front(self, item):
14         """Insert an item at the front of the deque.
15
16         Args:
17             item: The element to add.
18         """
19         self._deque.appendleft(item)
20
21     def insert_rear(self, item):
22         """Insert an item at the rear of the deque.
23
24         Args:
25             item: The element to add.
26         """
27         self._deque.append(item)
28
29     def remove_front(self):
30         """Remove and return the item from the front of the deque.
31
32         Returns:
33             The element removed from the front.
34
35         Raises:
36             IndexError: If the deque is empty.
37         """
38         if not self._deque:
39             raise IndexError("remove_front from empty deque")
40         return self._deque.popleft()
41
42     def remove_rear(self):
43         """Remove and return the item from the rear of the deque.

```

```
44
45     Returns:
46         The element removed from the rear.
47
48     Raises:
49         IndexError: If the deque is empty.
50     """
51     if not self._deque:
52         raise IndexError("remove_rear from empty deque")
53     return self._deque.pop()
54
55     def display(self):
56         """Print the current elements in the deque."""
57         if not self._deque:
58             print("Deque is empty")
59         else:
60             print("Deque contents:", list(self._deque))
61
62
63     # Example usage
64     if __name__ == "__main__":
65         dq = DequeDS()
66         dq.insert_rear(10)
67         dq.insert_rear(20)
68         dq.insert_front(5)
69
70         dq.display()          # [5, 10, 20]
71
72         print("Removed front:", dq.remove_front()) # 5
73         print("Removed rear:", dq.remove_rear())   # 20
74
75         dq.display()          # [10]
76
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

```
PS C:\Users\allas\OneDrive\Documents\web> & C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.8py.py
Deque contents: [5, 10, 20]
Removed front: 5
Removed rear: 20
Deque contents: [10]
PS C:\Users\allas\OneDrive\Documents\web>
```

Observation

Deque supports $O(1)$ insert/remove from both ends.

Task #9 – Data Structure Comparison

Prompt

Generate a markdown table comparing data structures and time complexities.

Code

```
.1.py .2.py .3.py .4.py .5.py .6.py .7.py .8.py .9.py x .10.py
9.py > ...
1 def generate_data_structure_comparison():
2     """Generate a markdown table comparing data structures and their time complexities."""
3     table = [
4         ["Data Structure", "Insert", "Delete", "Search/Access", "Peek/Top/Front", "Notes"],
5         ["Stack", "O(1)", "O(n)", "O(1)", "O(1)", "LIFO; usually backed by list/array"],
6         ["Queue", "O(1)", "O(1)", "O(n)", "O(1)", "O(n) if using Python list (pop(0)); O(1) with deque"],
7         ["Deque", "O(1)", "O(1)", "O(n)", "O(1)", "Double-ended queue using deque"],
8         ["Linked List (Singly)", "O(1) at head / O(n) at tail", "O(1) at head / O(n) elsewhere", "O(n)", "N/A", "Efficient insert/delete at head"],
9         ["Hash Table", "O(1) avg", "O(1) avg", "O(1) avg", "N/A", "O(n) worst-case if many collisions"],
10        ["Binary Search Tree (balanced)", "O(log n)", "O(log n)", "O(log n)", "O(1)", "Logarithmic only if tree stays balanced"],
11        ["Priority Queue (heap)", "O(log n)", "O(log n)", "O(n)", "O(1)", "Uses heap; peek is O(1)"],
12        ["Graph (Adjacency List)", "O(1) to add vertex/edge", "O(V + E) to remove", "O(V + E)", "N/A", "Good for sparse graphs"],
13    ]
14
15    # Build Markdown
16    md = []
17    md.append("| " + " | ".join(table[0]) + " |")
18    md.append("|" + " | ".join(["----"] * len(table[0])) + "|")
19    for row in table[1:]:
20        md.append("| " + " | ".join(row) + " |")
21
22    return "\n".join(md)
23
24
25 if __name__ == "__main__":
26     print(generate_data_structure_comparison())
27
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\allas\OneDrive\Documents\web> & C:\Python313\python.exe c:\Users\allas\OneDrive\Documents\web\9.py
| Data Structure | Insert | Delete | Search/Access | Peek/Top/Front | Notes |
|----|----|----|----|----|----|
| Stack | O(1) | O(1) | O(n) | O(1) | LIFO; usually backed by list/array |
| Queue | O(1) | O(1) | O(n) | O(1) | O(n) if using Python list (pop(0)); O(1) with deque |
| Deque | O(1) | O(1) | O(n) | O(1) | Double-ended queue using deque |
| Linked list (Singly) | O(1) at head / O(n) at tail | O(1) at head / O(n) elsewhere | O(n) | N/A | Efficient insert/delete at head |
| Hash Table | O(1) avg | O(1) avg | O(1) avg | N/A | O(n) worst-case if many collisions |
| Binary Search Tree (balanced) | O(log n) | O(log n) | O(log n) | O(1) | Logarithmic only if tree stays balanced |
| Priority Queue (heap) | O(log n) | O(log n) | O(n) | O(1) | Uses heap; peek is O(1) |
| Graph (Adjacency List) | O(1) to add vertex/edge | O(V + E) to remove | O(V + E) | N/A | Good for sparse graphs |
PS C:\Users\allas\OneDrive\Documents\web>
```

Observation

Markdown table lists operations and complexities.

Task #10 – Real-Time Application Challenge

Prompt

Implement a Cafeteria Order Queue using collections.deque.

Code

Observation

Queue meets FIFO behavior; error on empty is correct.