AI ASSISTED CODING LAB ASSIGNMENT:11.1

NAME:N.Prudhvi

ROLL NO:2403A510G7

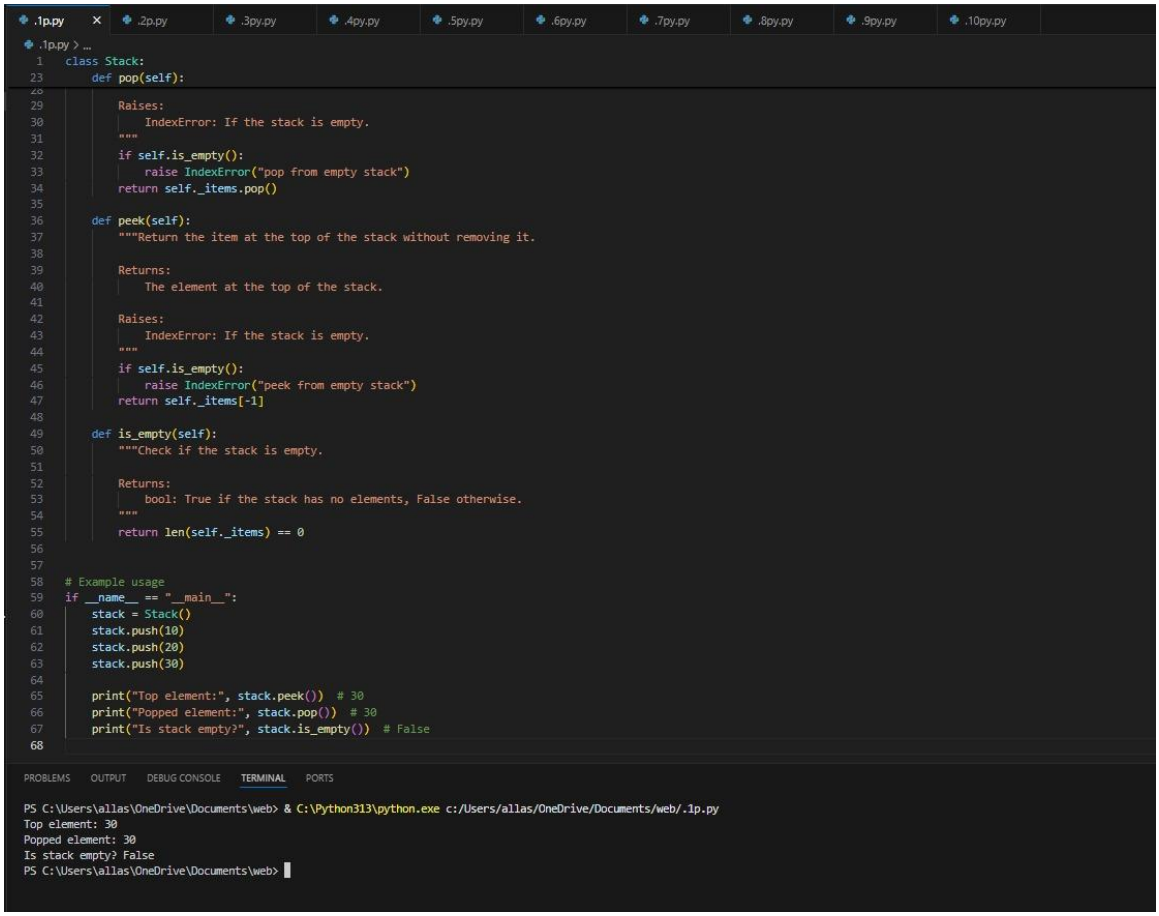
BATCH:06

Task #1 – Stack Implementation

Prompt

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

Code



```
class Stack:
    def pop(self):

        Raises:
            IndexError: If the stack is empty.
        """
        if self.is_empty():
            raise IndexError("pop from empty stack")
        return self._items.pop()

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

        Returns:
            The element at the top of the stack.

        Raises:
            IndexError: If the stack is empty.
        """
        if self.is_empty():
            raise IndexError("peek from empty stack")
        return self._items[-1]

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

        Returns:
            bool: True if the stack has no elements, False otherwise.
        """
        return len(self._items) == 0


# Example usage
if __name__ == "__main__":
    stack = Stack()
    stack.push(10)
    stack.push(20)
    stack.push(30)

    print("Top element:", stack.peek())  # 30
    print("Popped element:", stack.pop())  # 30
    print("Is stack empty?", stack.is_empty())  # False
```

```
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

```python
class Queue:
    """A simple FIFO (First-In, First-Out) queue implementation.

    Methods:
        enqueue(item): Add an item to the back of the queue.
        dequeue(): Remove and return the item from the front of the queue.
        peek(): Return the item at the front of the queue without removing it.
        size(): Return the number of elements in the queue.
    """

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

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

        Args:
            item: The element to add to the queue.
        """
        self._items.append(item)

    def dequeue(self):
        """Remove and return the item from the front of the queue.

        Returns:
            The element at the front of the queue.

        Raises:
            IndexError: If the queue is empty.
        """
        if self.size() == 0:
            raise IndexError("dequeue from empty queue")
        return self._items.pop(0)

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

        Returns:
            The element at the front of the queue.

        Raises:
```

```
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>
```

```python
class Queue:
    def peek(self):

        Raises:
            IndexError: If the queue is empty.
        """
        if self.size() == 0:
            raise IndexError("peek from empty queue")
        return self._items[0]

    def size(self):
        """Get the number of elements in the queue.

        Returns:
            int: The number of elements in the queue.
        """
        return len(self._items)


# Example usage
if __name__ == "__main__":
    q = Queue()
    q.enqueue("A")
    q.enqueue("B")
    q.enqueue("C")

    print("Front element:", q.peek())   # A
    print("Removed:", q.dequeue())      # A
    print("Queue size:", q.size())      # 2

```

```
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

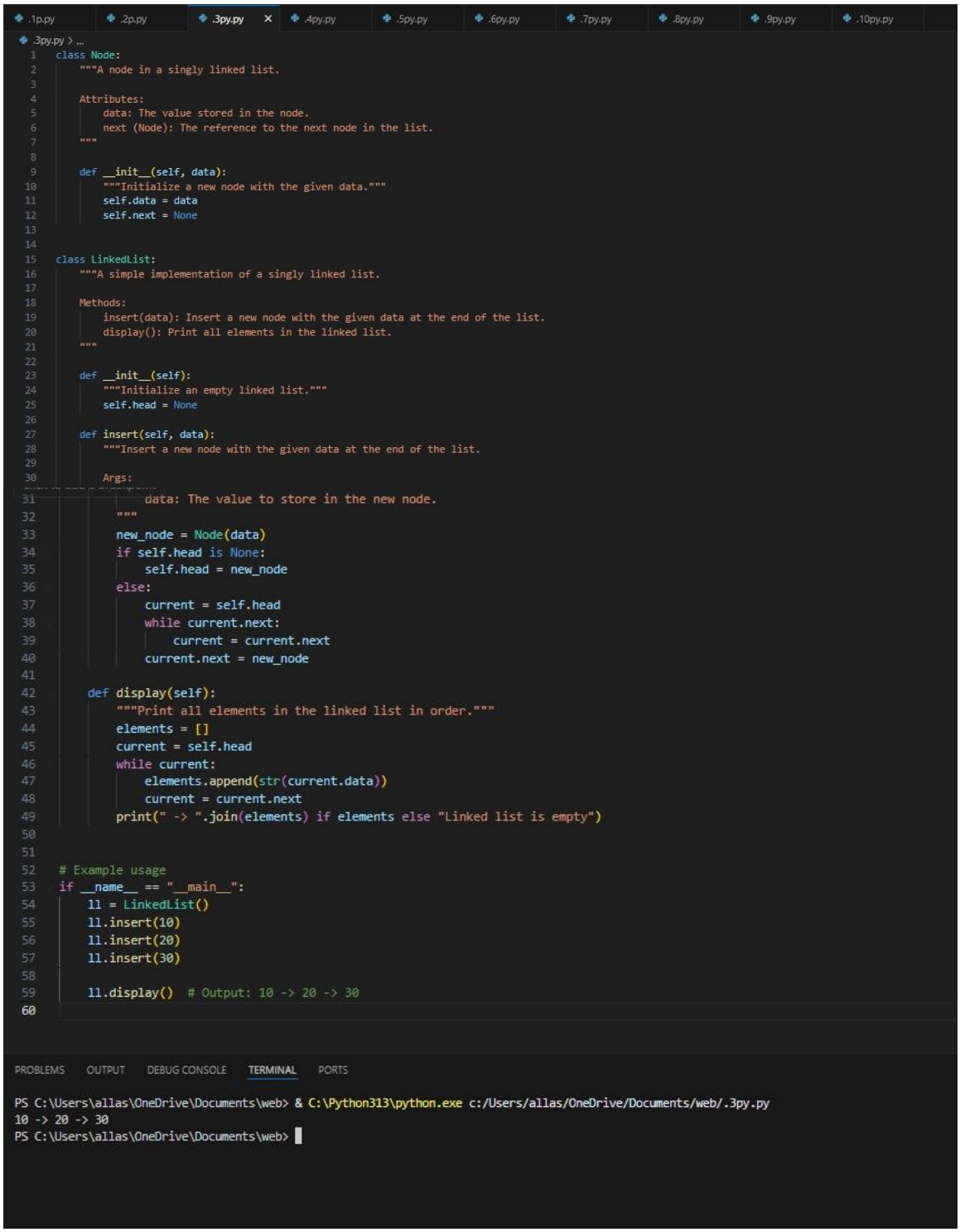
Queue behaves as FIFO. All assert cases passed.

Task #3 – Singly Linked List

Prompt

Generate a Singly Linked List with insert and display methods.

Code



Observation

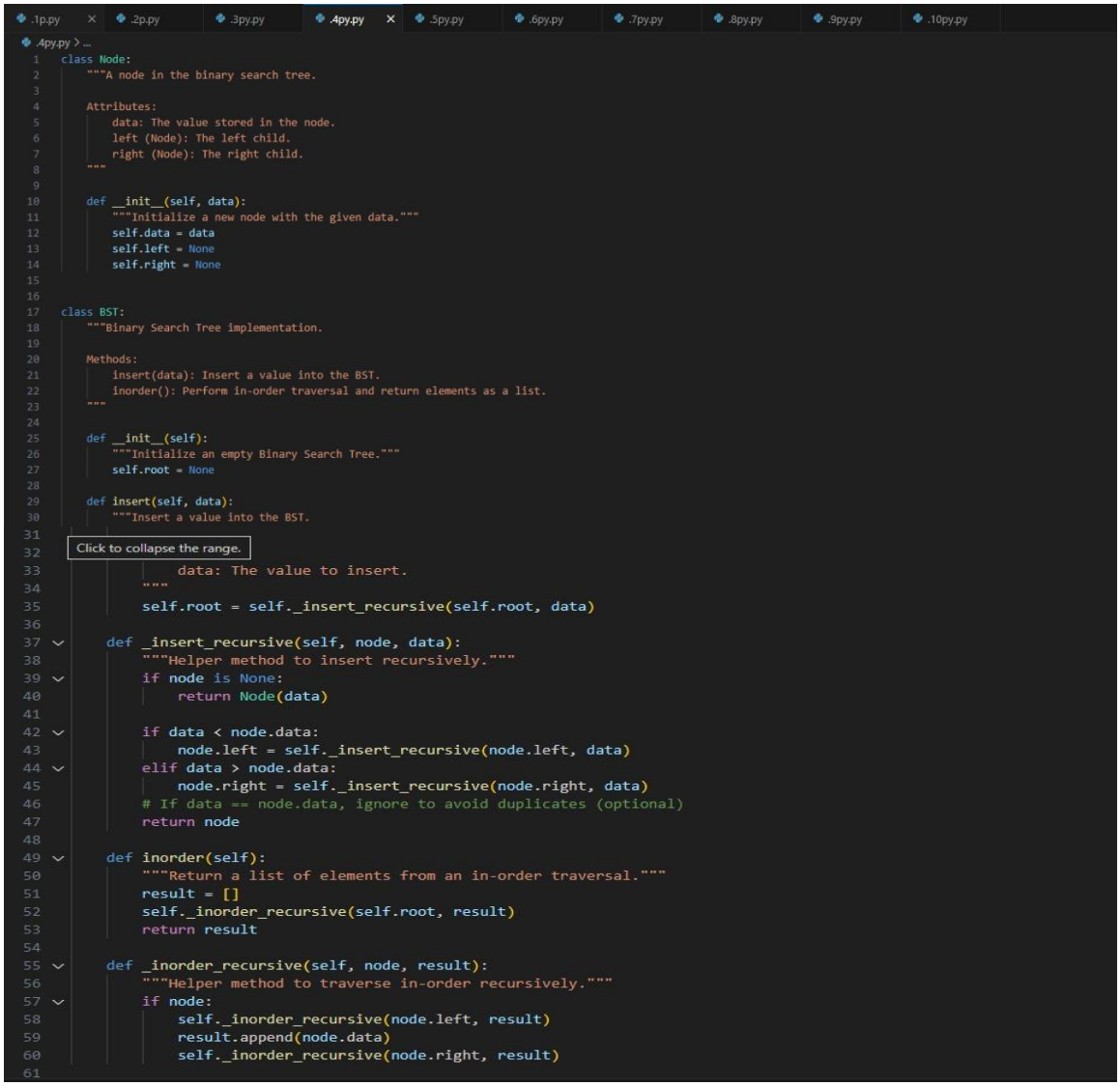
Insertion at head is correct; display returns values in proper order.

Task #4 – Binary Search Tree

Prompt

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

Code




```python
class Node:
    """A node in the binary search tree.

    Attributes:
        data: The value stored in the node.
        left (Node): The left child.
        right (Node): The right child.
    """

    def __init__(self, data):
        """Initialize a new node with the given data."""
        self.data = data
        self.left = None
        self.right = None


class BST:
    """Binary Search Tree implementation.

    Methods:
        insert(data): Insert a value into the BST.
        inorder(): Perform in-order traversal and return elements as a list.
    """

    def __init__(self):
        """Initialize an empty Binary Search Tree."""
        self.root = None

    def insert(self, data):
        """Insert a value into the BST.

            data: The value to insert.
        """
        self.root = self._insert_recursive(self.root, data)

    def _insert_recursive(self, node, data):
        """Helper method to insert recursively."""
        if node is None:
            return Node(data)

        if data < node.data:
            node.left = self._insert_recursive(node.left, data)
        elif data > node.data:
            node.right = self._insert_recursive(node.right, data)
        # If data == node.data, ignore to avoid duplicates (optional)
        return node

    def inorder(self):
        """Return a list of elements from an in-order traversal."""
        result = []
        self._inorder_recursive(self.root, result)
        return result

    def _inorder_recursive(self, node, result):
        """Helper method to traverse in-order recursively."""
        if node:
            self._inorder_recursive(node.left, result)
            result.append(node.data)
            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.

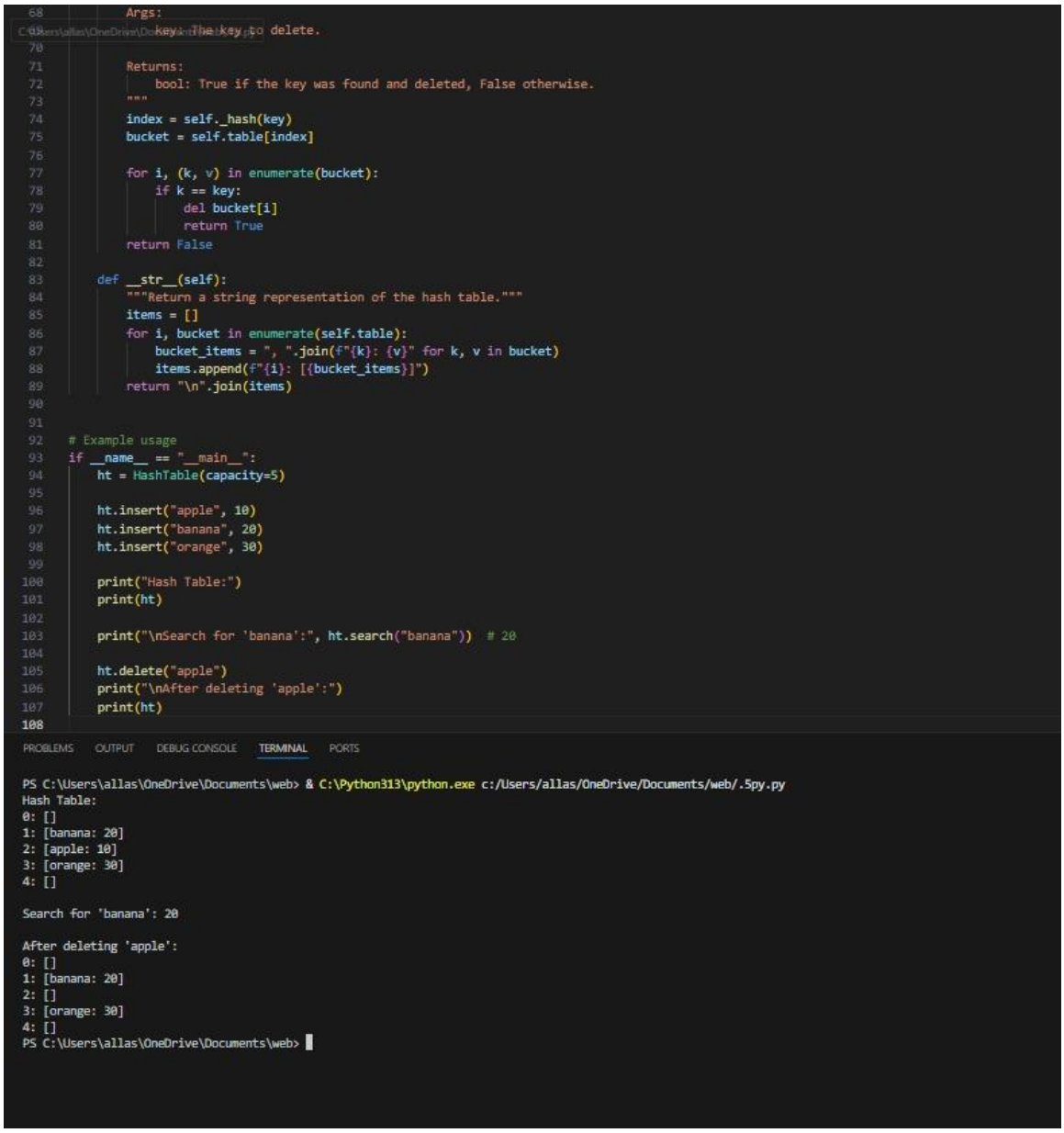
Task #5 – Hash Table

Prompt

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

Code


C:\Users\allan\OneDrive\Documents\web\1.py
class HashTable:
"""A basic hash table implementation using separate chaining.
Methods:
insert(key, value): Insert or update a key-value pair.
search(key): Retrieve the value associated with a key.
delete(key): Remove a key-value pair from the hash table.
"""
def __init__(self, capacity=10):
"""Initialize a hash table with the given capacity.
Args:
capacity (int): The number of buckets in the table.
"""
self.capacity = capacity
self.table = [[] for _ in range(self.capacity)] # Each bucket is a list
def _hash(self, key):
"""Compute the hash index for a given key.
Args:
key: The key to hash.
Returns:
int: Index of the bucket where the key belongs.
"""
return hash(key) % self.capacity
def insert(self, key, value):
"""Insert a key-value pair into the hash table.
If the key already exists, update its value.
Args:
key: The key to insert.
value: The value to associate with the key.
"""
index = self._hash(key)
bucket = self.table[index]
for i, (k, v) in enumerate(bucket):
if k == key: # Update if key already exists
bucket[i] = (key, value)
return
bucket.append((key, value)) # Add new pair if key not found
def search(self, key):
"""Search for a key in the hash table.
Args:
key: The key to search for.
Returns:
The value associated with the key, or None if not found.
"""
index = self._hash(key)
bucket = self.table[index]
for k, v in bucket:
if k == key:
return v
return None
def delete(self, key):
"""Delete a key-value pair from the hash table.



```
        Args:
            key: The key to delete.

        Returns:
            bool: True if the key was found and deleted, False otherwise.
        """
        index = self._hash(key)
        bucket = self.table[index]

        for i, (k, v) in enumerate(bucket):
            if k == key:
                del bucket[i]
                return True
        return False

    def __str__(self):
        """Return a string representation of the hash table."""
        items = []
        for i, bucket in enumerate(self.table):
            bucket_items = ", ".join(f"{k}: {v}" for k, v in bucket)
            items.append(f"{i}: [{bucket_items}]")
        return "\n".join(items)


# Example usage
if __name__ == "__main__":
    ht = HashTable(capacity=5)

    ht.insert("apple", 10)
    ht.insert("banana", 20)
    ht.insert("orange", 30)

    print("Hash Table:")
    print(ht)

    print("\nSearch for 'banana':", ht.search("banana"))  # 20

    ht.delete("apple")
    print("\nAfter deleting 'apple':")
    print(ht)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\allas\OneDrive\Documents\web> & C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.5py.py
Hash Table:
0: []
1: [banana: 20]
2: [apple: 10]
3: [orange: 30]
4: []

Search for 'banana': 20

After deleting 'apple':
0: []
1: [banana: 20]
2: []
3: [orange: 30]
4: []
PS C:\Users\allas\OneDrive\Documents\web>
```

Observation

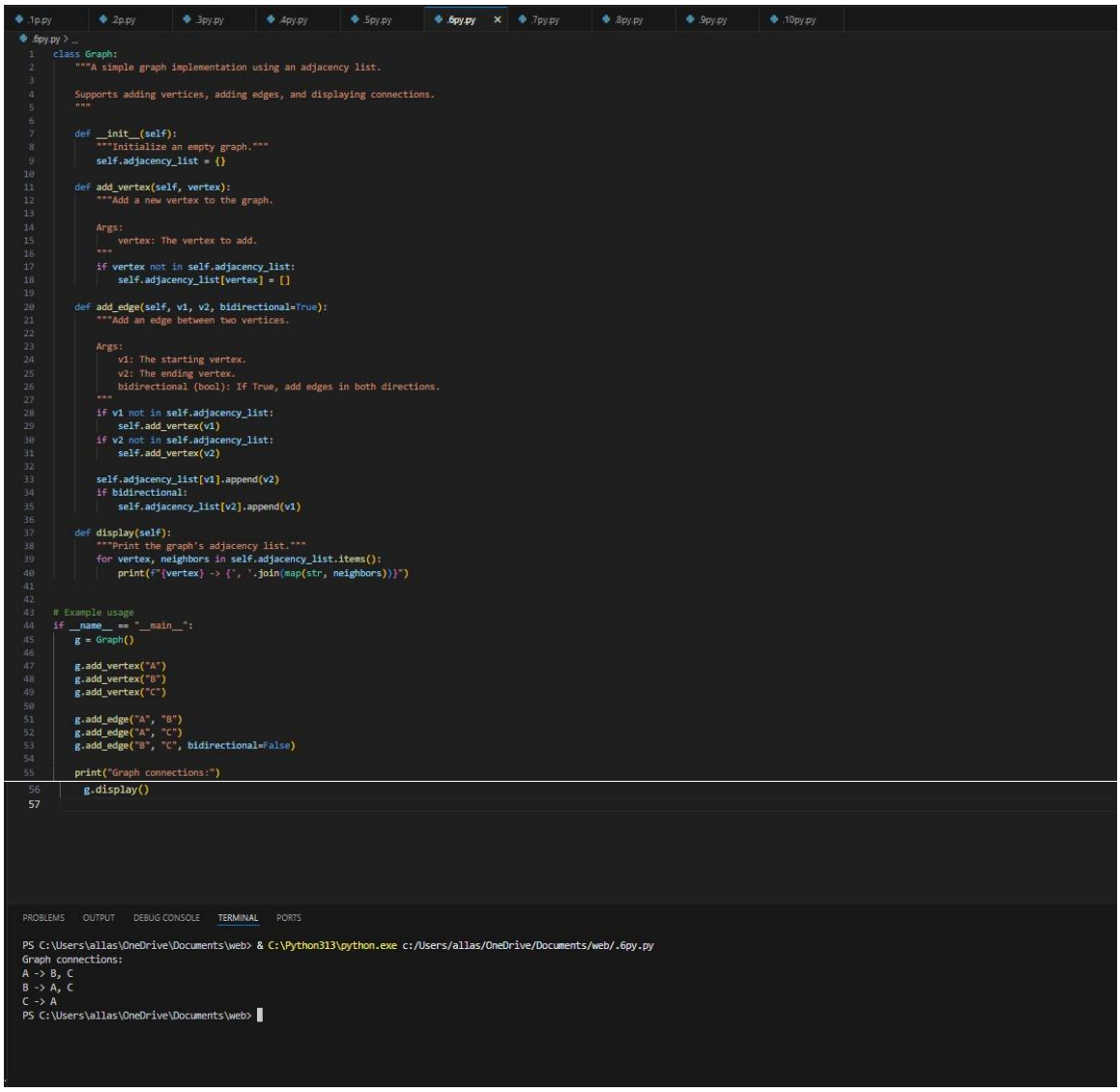
Hash table handles collisions; search and delete work.

Task #6 – Graph Representation

Prompt

Implement a graph using an adjacency list.

Code



Observation

Graph stores connections via adjacency list.

Task #7 – Priority Queue

Prompt

Implement a priority queue using heapq.

Code

```python
import heapq

class PriorityQueue:
    """A priority queue implementation using Python's heapq module.

    Stores elements as (priority, item) tuples so that the element
    with the lowest priority value is served first.
    """

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

    def enqueue(self, priority, item):
        """Add an item with a given priority to the queue.

        Args:
            priority (int | float): The priority of the item (smaller = higher priority).
            item: The element to store.
        """
        heapq.heappush(self._heap, (priority, item))

    def dequeue(self):
        """Remove and return the item with the highest priority.

        Returns:
            The item with the smallest priority value.

        Raises:
            IndexError: If the queue is empty.
        """
        if not self._heap:
            raise IndexError("dequeue from an empty priority queue")
        return heapq.heappop(self._heap)[1]

    def display(self):
        """Display the current elements of the priority queue (unsorted)."""
        if not self._heap:
            print("Priority queue is empty")
        else:
            print("Queue contents (priority, item):")
            for entry in self._heap:
                print(entry)


# Example usage
if __name__ == "__main__":
    pq = PriorityQueue()
    pq.enqueue(3, "Low priority")
    pq.enqueue(1, "High priority")
    pq.enqueue(2, "Medium priority")

    pq.display()
    print("\nDequeued:", pq.dequeue())  # Highest priority (lowest number)
    pq.display()
```

```
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

.8py.py > ...

```python
from collections import deque

class DequeDS:
    """A double-ended queue (deque) implementation.

    Supports inserting and removing elements from both ends efficiently.
    """

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

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

        Args:
            item: The element to add.
        """
        self._deque.appendleft(item)

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

        Args:
            item: The element to add.
        """
        self._deque.append(item)

    def remove_front(self):
        """Remove and return the item from the front of the deque.

        Returns:
            The element removed from the front.

        Raises:
            IndexError: If the deque is empty.
        """
        if not self._deque:
            raise IndexError("remove_front from empty deque")
        return self._deque.popleft()

    def remove_rear(self):
        """Remove and return the item from the rear of the deque.
```

```
        Returns:
            The element removed from the rear.

        Raises:
            IndexError: If the deque is empty.
        """
        if not self._deque:
            raise IndexError("remove_rear from empty deque")
        return self._deque.pop()

    def display(self):
        """Print the current elements in the deque."""
        if not self._deque:
            print("Deque is empty")
        else:
            print("Deque contents:", list(self._deque))


# Example usage
if __name__ == "__main__":
    dq = DequeDS()
    dq.insert_rear(10)
    dq.insert_rear(20)
    dq.insert_front(5)

    dq.display()           # [5, 10, 20]

    print("Removed front:", dq.remove_front())  # 5
    print("Removed rear:", dq.remove_rear())    # 20

    dq.display()           # [10]

```

```
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

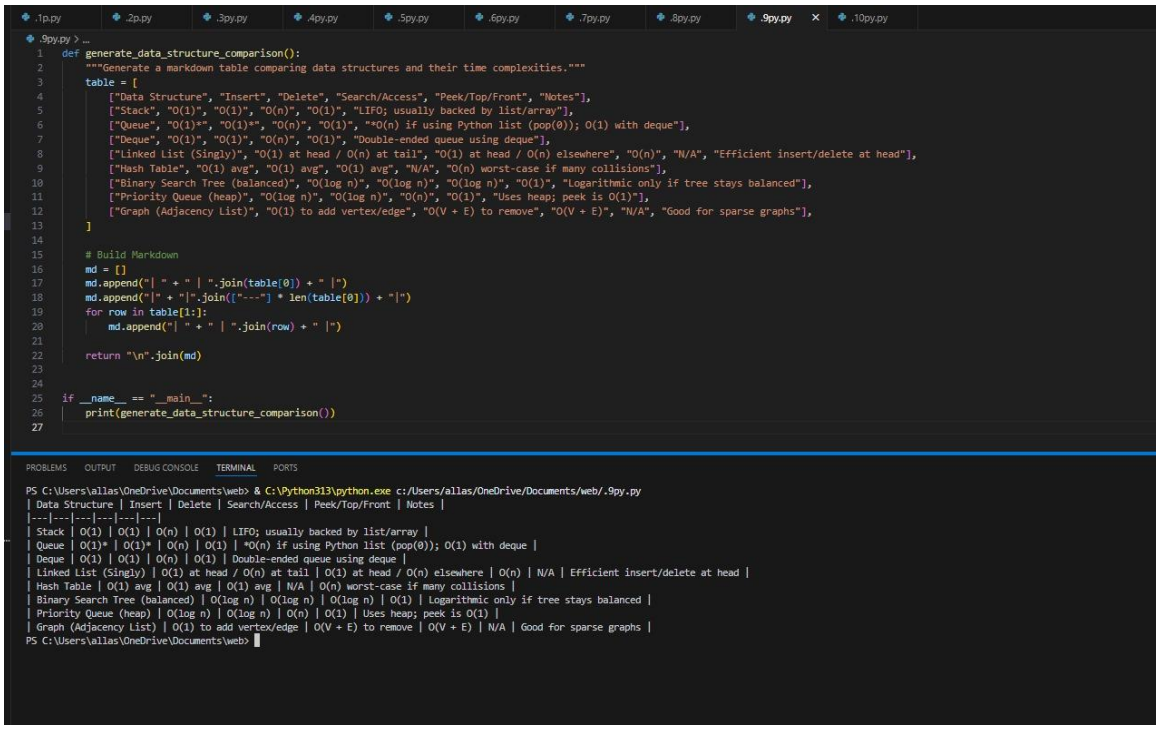
Deque supports O(1) insert/remove from both ends.

Task #9 – Data Structure Comparison

Prompt

Generate a markdown table comparing data structures and time complexities.

Code



```python
def generate_data_structure_comparison():
    """Generate a markdown table comparing data structures and their time complexities."""
    table = [
        ["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"],
    ]

    # Build Markdown
    md = []
    md.append("| " + " | ".join(table[0]) + " |")
    md.append("|" + "|".join(["---"] * len(table[0])) + "|")
    for row in table[1:]:
        md.append("| " + " | ".join(row) + " |")

    return "\n".join(md)


if __name__ == "__main__":
    print(generate_data_structure_comparison())
```

```
PS C:\Users\allas\OneDrive\Documents\web> & C:\Python313\python.exe c:/Users/allas/OneDrive/Documents/web/.9py.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.