AI ASSISTED CODING

LAB ASSIGNMENT: 11.2

NAME: K. SARIKA

H.NO: 2403A52012

B.NO: 04

TASK:

Stack Implementation
Task: Use AI to generate a Stack class with push, pop, peek, and is_empty
methods.
Sample Input Code:
class Stack:
pass.

PROMPT:

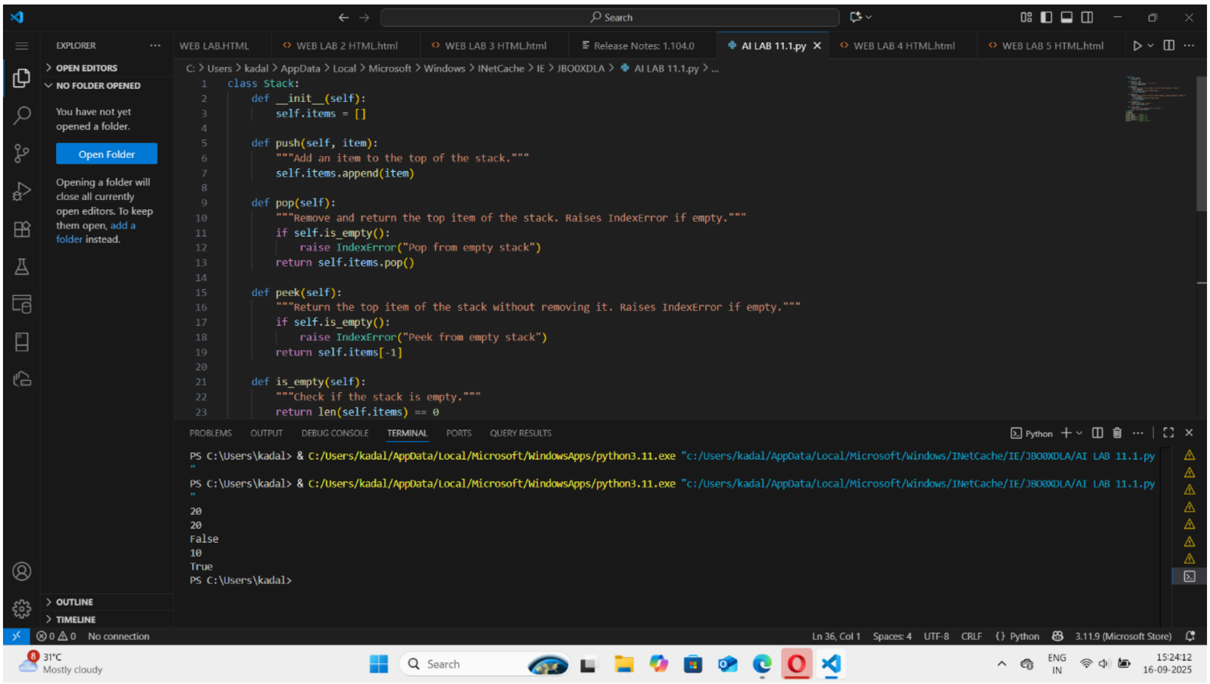
Generate python code and stack
Implementation
Task: Use AI to generate a Stack class with push, pop, peek, and is_empty

methods.

Sample Input Code:

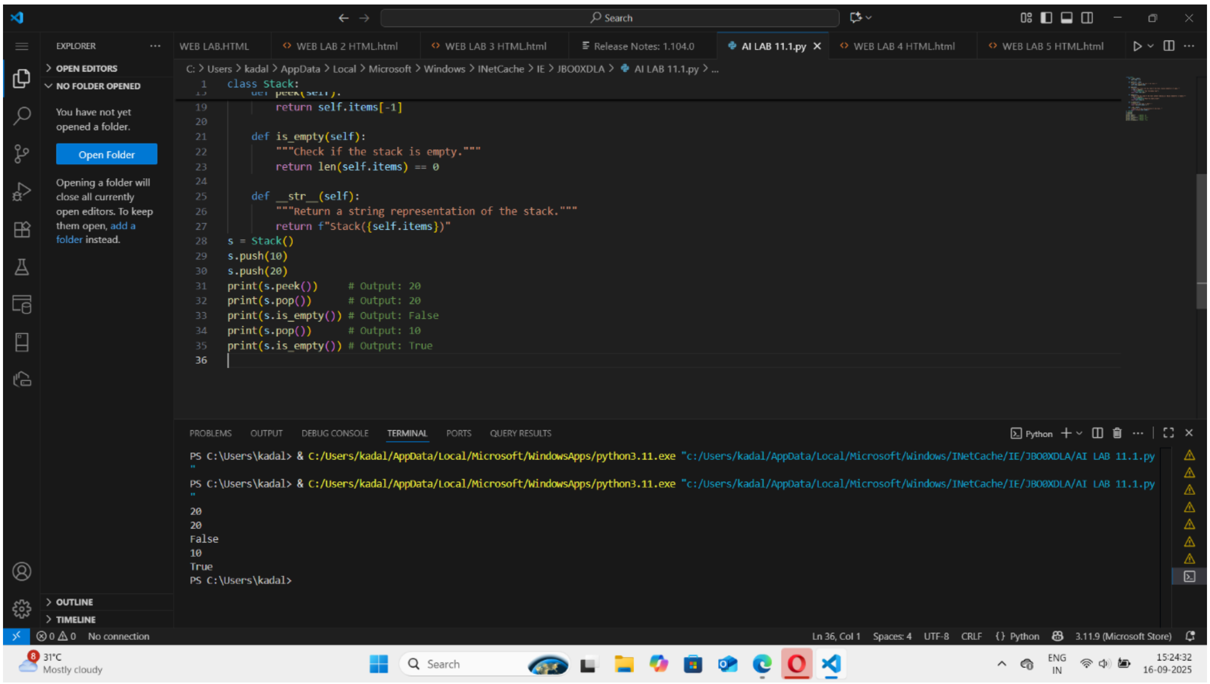
```
class Stack:
pass.
```

CODE & OUTPUT:



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

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

    def pop(self):
        """Remove and return the top item of the stack. Raises IndexError if empty."""
        if self.is_empty():
            raise IndexError("Pop from empty stack")
        return self.items.pop()

    def peek(self):
        """Return the top item of the stack without removing it. Raises IndexError if 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."""
        return len(self.items) == 0
```

```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1.py
"
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1.py
"
20
20
False
10
True
PS C:\Users\kadal>
```



```python
class Stack:
    def peek(self):
        return self.items[-1]

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

    def __str__(self):
        """Return a string representation of the stack."""
        return f"Stack({self.items})"
s = Stack()
s.push(10)
s.push(20)
print(s.peek())     # Output: 20
print(s.pop())      # Output: 20
print(s.is_empty()) # Output: False
print(s.pop())      # Output: 10
print(s.is_empty()) # Output: True
```

```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1.py
"
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1.py
"
20
20
False
10
True
PS C:\Users\kadal>
```

EXPLAINATION:

A **stack** is a linear data structure that follows the **LIFO** principle — **Last In, First Out**. Think of it like a stack of plates:

- You add (push) a plate to the top.
- You remove (pop) the top plate first.
- You can peek at the top plate without removing it.
- You can check if the stack is empty.

TASK 2:

Queue Implementation

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

Sample Input Code:

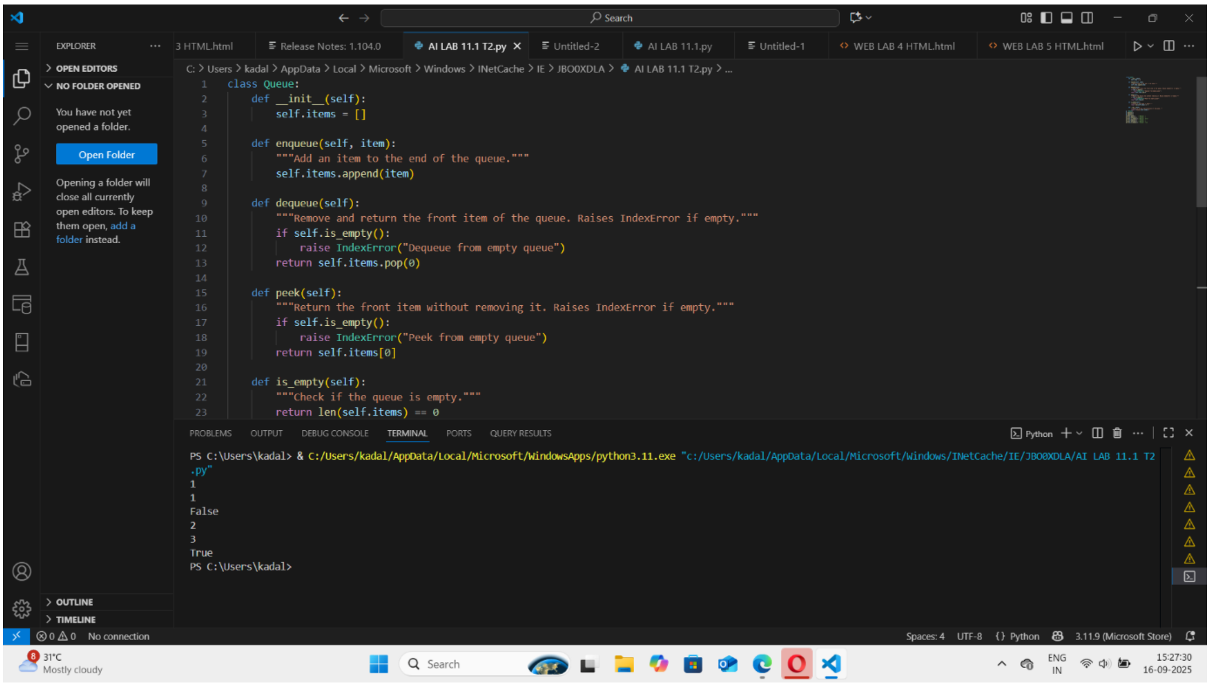
```
class Queue:
pass.
```

PROMPT:

Generate python code and queue
Implementation
Task: Use AI to implement a Queue using Python lists.
Sample Input Code:

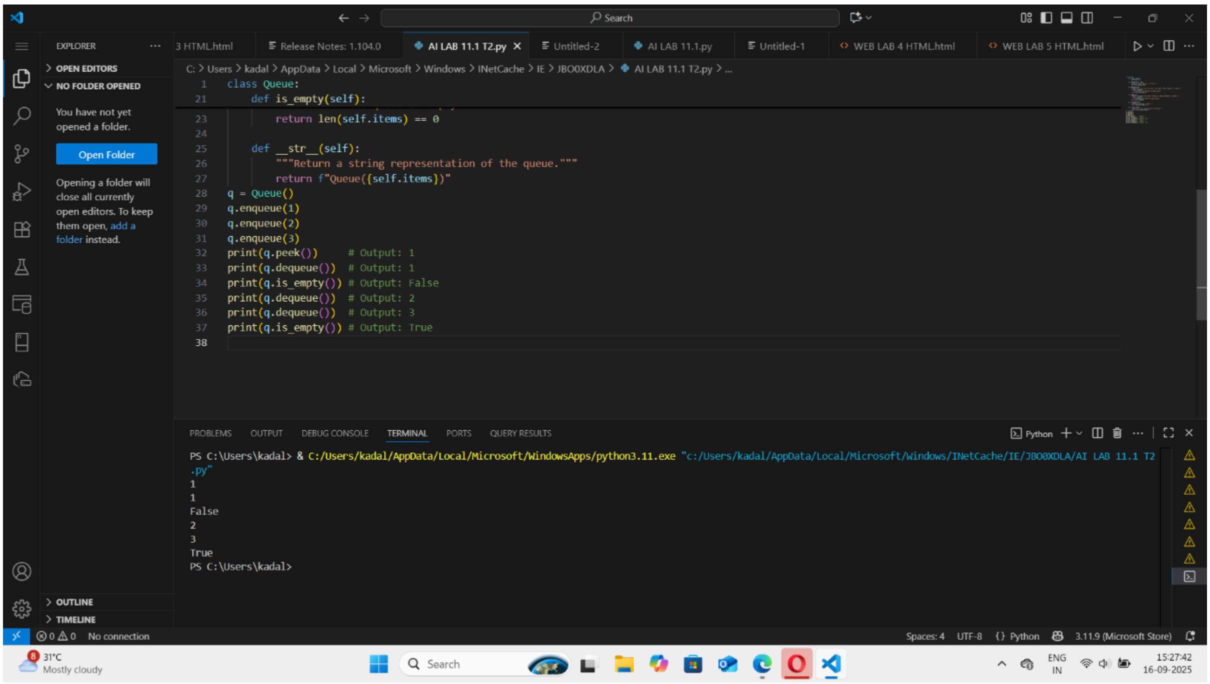
```
class Queue:
pass.
```

CODE & OUTPUT:



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

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

    def dequeue(self):
        """Remove and return the front item of the queue. Raises IndexError if empty."""
        if self.is_empty():
            raise IndexError("Dequeue from empty queue")
        return self.items.pop(0)

    def peek(self):
        """Return the front item without removing it. Raises IndexError if empty."""
        if self.is_empty():
            raise IndexError("Peek from empty queue")
        return self.items[0]

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

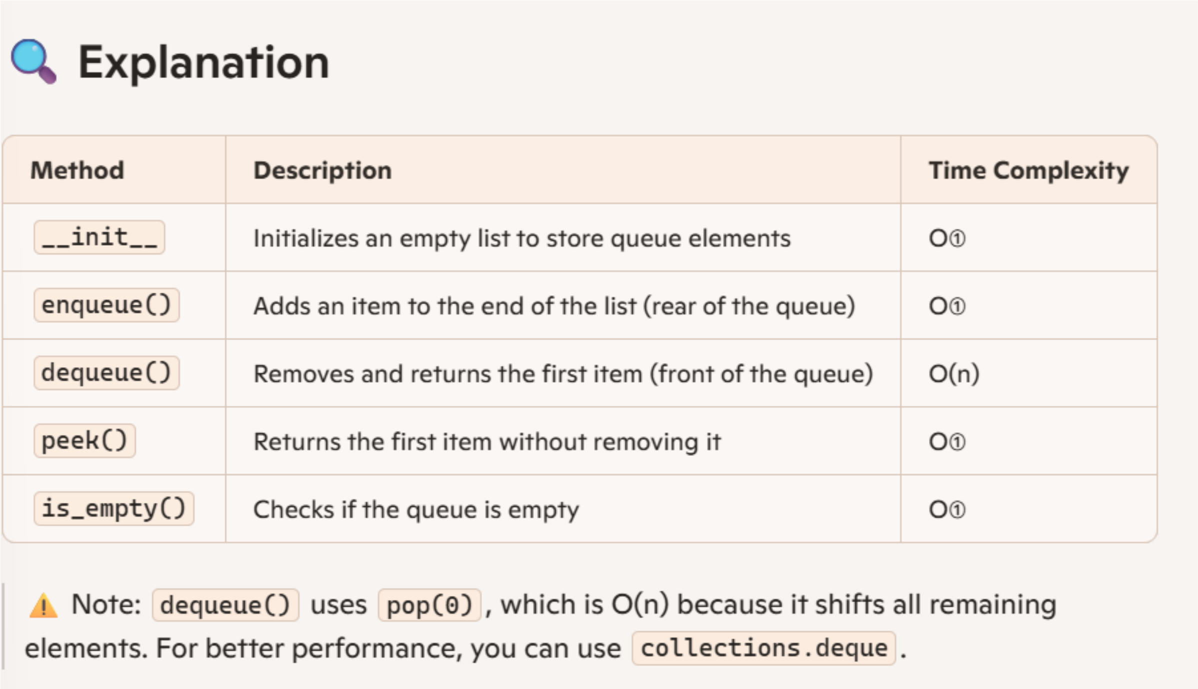
```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T2
.py"
1
1
False
2
3
True
PS C:\Users\kadal>
```



```
class Queue:
    def is_empty(self):
        return len(self.items) == 0

    def __str__(self):
        """Return a string representation of the queue."""
        return f"Queue({self.items})"
q = Queue()
q.enqueue(1)
q.enqueue(2)
q.enqueue(3)
print(q.peek())      # Output: 1
print(q.dequeue())  # Output: 1
print(q.is_empty()) # Output: False
print(q.dequeue())  # Output: 2
print(q.dequeue())  # Output: 3
print(q.is_empty()) # Output: True
```

```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T2
.py"
1
1
False
2
3
True
PS C:\Users\kadal>
```

EXPLAINATION:



🔍 Explanation

Method	Description	Time Complexity
`__init__`	Initializes an empty list to store queue elements	O①
`enqueue()`	Adds an item to the end of the list (rear of the queue)	O①
`dequeue()`	Removes and returns the first item (front of the queue)	O(n)
`peek()`	Returns the first item without removing it	O①
`is_empty()`	Checks if the queue is empty	O①

> ⚠️ Note: `dequeue()` uses `pop(0)`, which is O(n) because it shifts all remaining elements. For better performance, you can use `collections.deque`.

TASK 3:

Linked List
Task: Use AI to generate a Singly Linked List with insert and display methods.
Sample Input Code:

```
class Node:
pass.
```

PROMPT:

Generate python code and linked List
Task: Use AI to generate a Singly Linked List

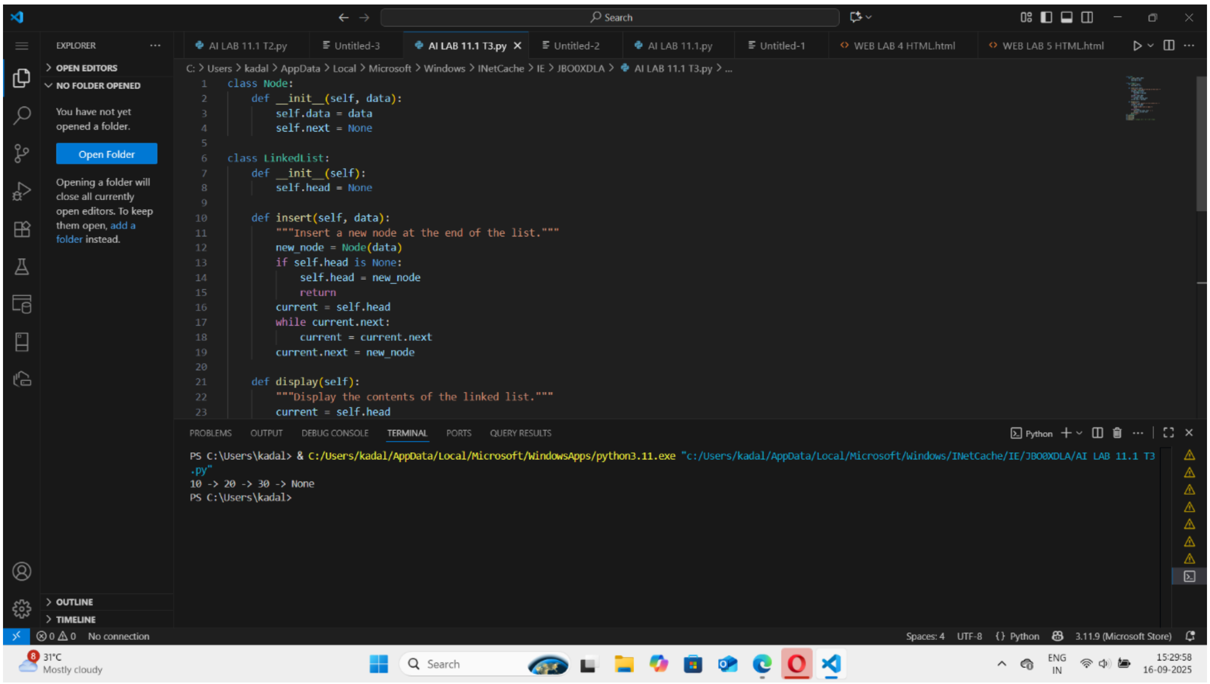
with insert and display methods.

Sample Input Code:

class Node:

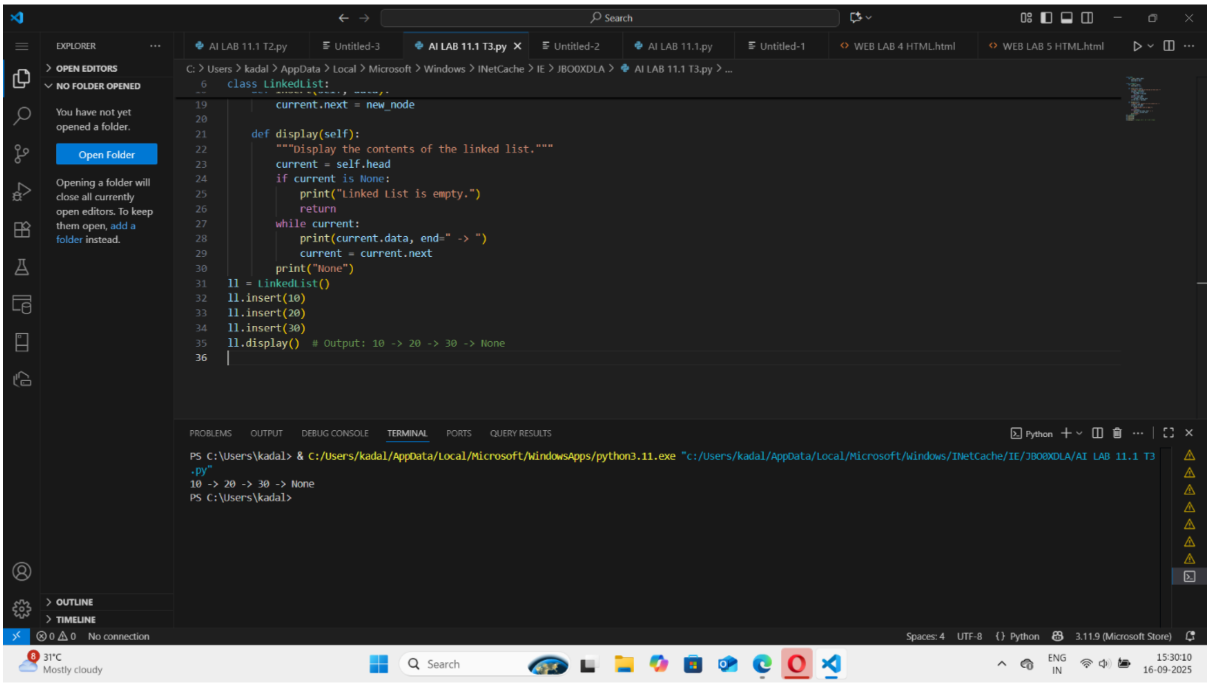
pass.

CODE & OUTPUT:



```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def insert(self, data):
        """Insert a new node at the end of the list."""
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

    def display(self):
        """Display the contents of the linked list."""
        current = self.head
```

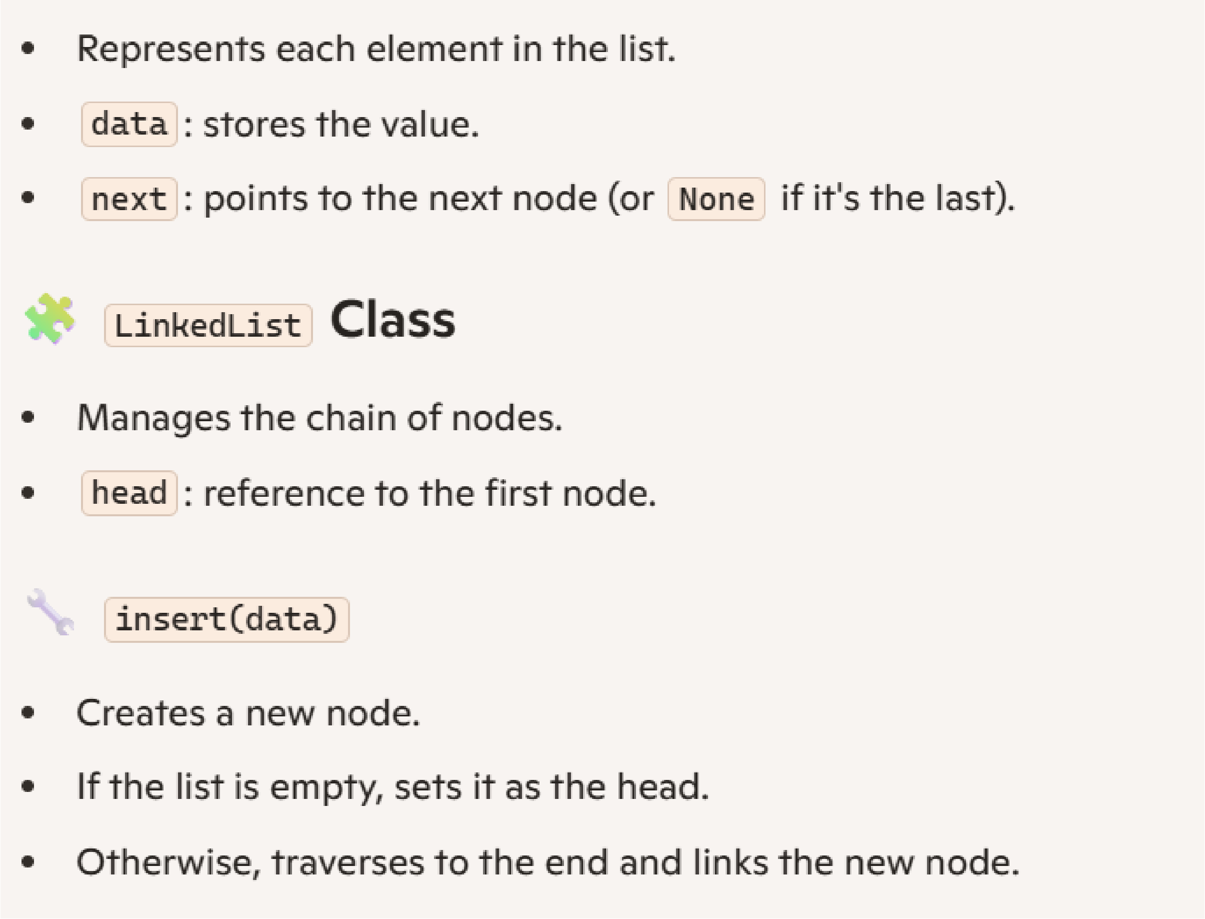
```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T3
.py"
10 -> 20 -> 30 -> None
PS C:\Users\kadal>
```



```
class LinkedList:
        current.next = new_node

    def display(self):
        """Display the contents of the linked list."""
        current = self.head
        if current is None:
            print("Linked List is empty.")
            return
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")
ll = LinkedList()
ll.insert(10)
ll.insert(20)
ll.insert(30)
ll.display()  # Output: 10 -> 20 -> 30 -> None
```

```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T3
.py"
10 -> 20 -> 30 -> None
PS C:\Users\kadal>
```

EXPLAINATION:



- Represents each element in the list.
- `data` : stores the value.
- `next` : points to the next node (or `None` if it's the last).

🧩 `LinkedList` Class

- Manages the chain of nodes.
- `head` : reference to the first node.

🔧 `insert(data)`

- Creates a new node.
- If the list is empty, sets it as the head.
- Otherwise, traverses to the end and links the new node.

TASK 4:

Binary Search Tree (BST)
Task: Use AI to create a BST with insert and in-order traversal methods.
Sample Input Code:

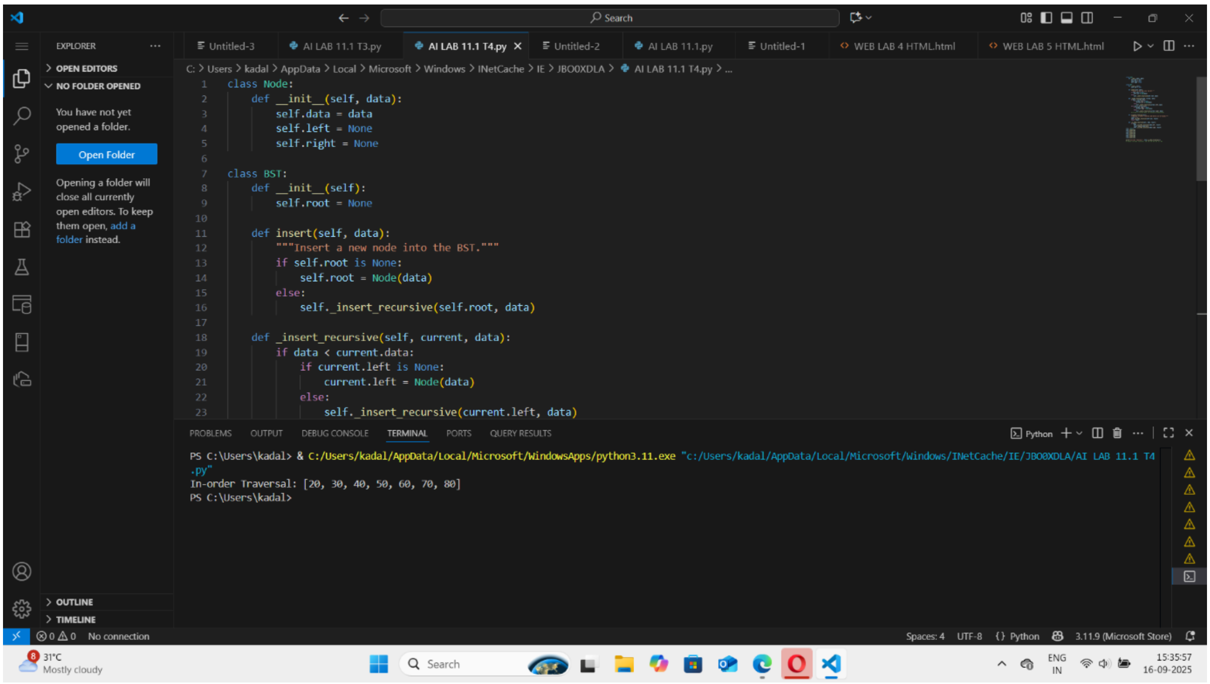
```
class BST:
pass.
```

PROMPT:

Generate python code and binary Search Tree (BST)
Task: Use AI to create a BST with insert and in-order traversal methods.
Sample Input Code:

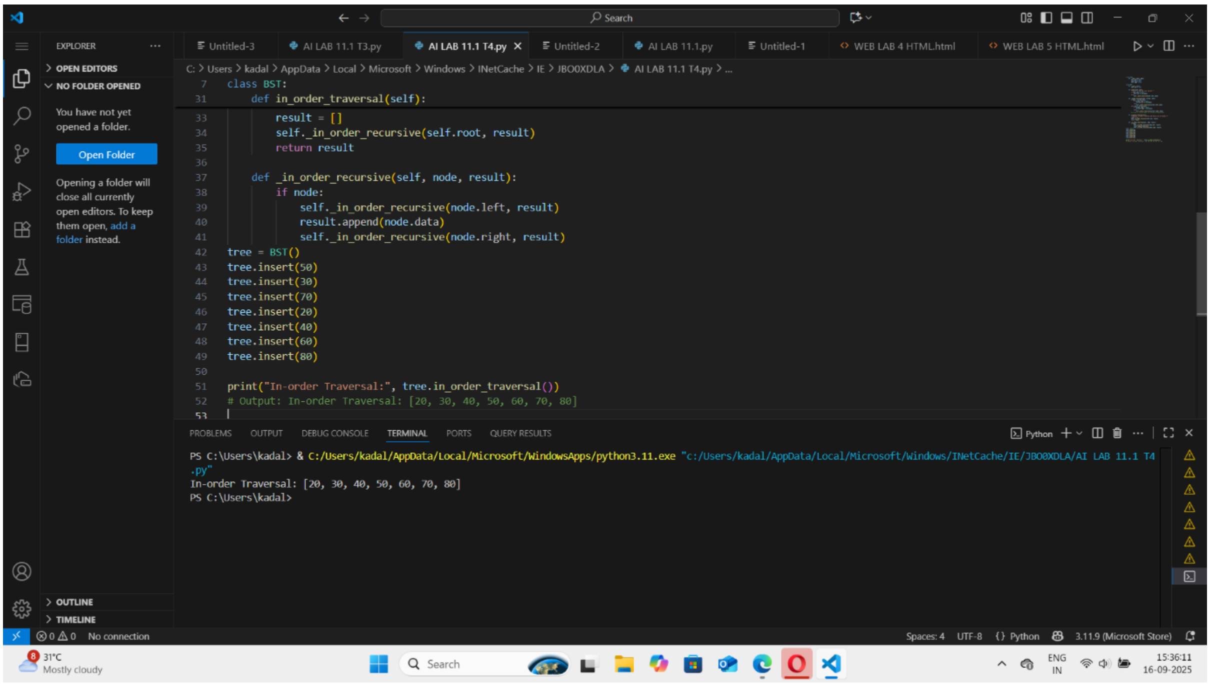
```
class BST:
pass.
```

CODE & OUTPUT:



```python
class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

class BST:
    def __init__(self):
        self.root = None

    def insert(self, data):
        """Insert a new node into the BST."""
        if self.root is None:
            self.root = Node(data)
        else:
            self._insert_recursive(self.root, data)

    def _insert_recursive(self, current, data):
        if data < current.data:
            if current.left is None:
                current.left = Node(data)
            else:
                self._insert_recursive(current.left, data)
```

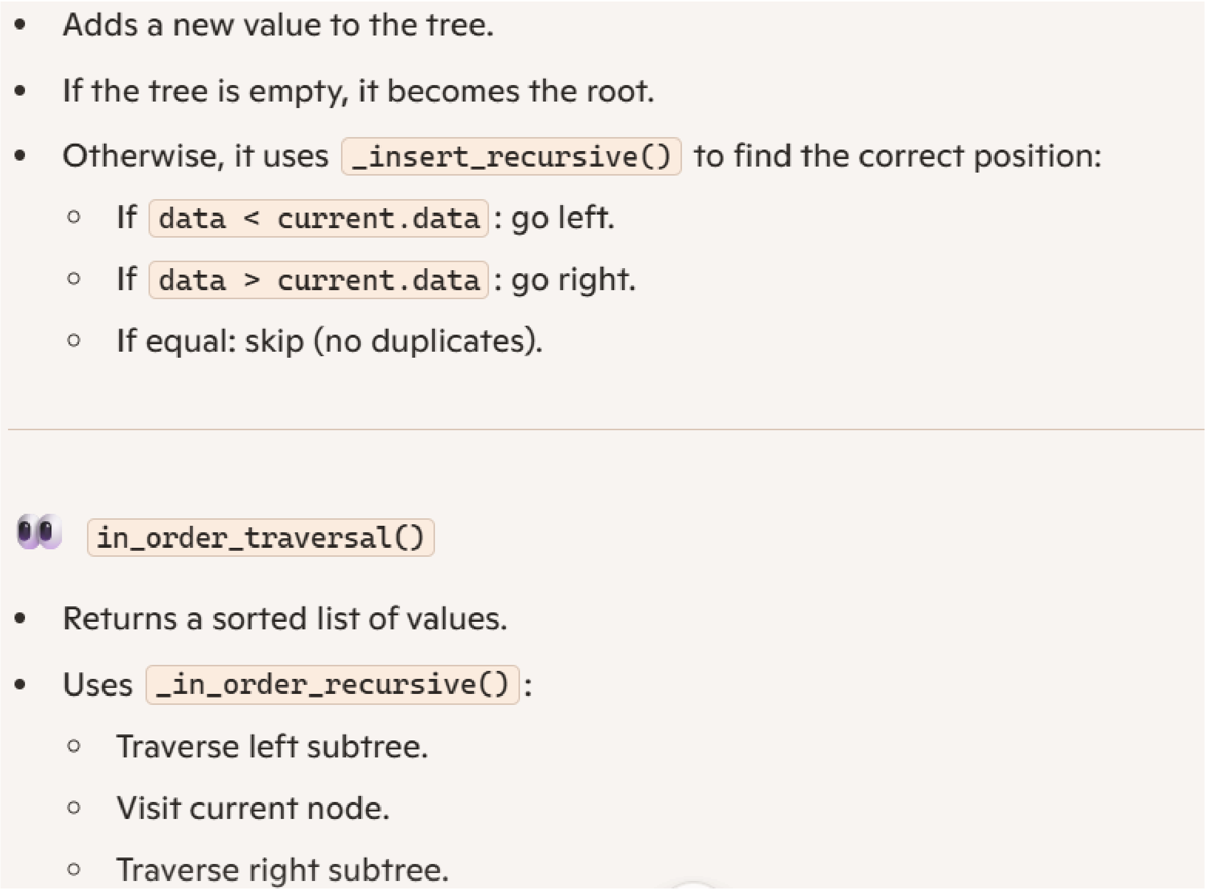
```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T4
.py"
In-order Traversal: [20, 30, 40, 50, 60, 70, 80]
PS C:\Users\kadal>
```



```python
class BST:
    def in_order_traversal(self):
            result = []
            self._in_order_recursive(self.root, result)
            return result

    def _in_order_recursive(self, node, result):
        if node:
            self._in_order_recursive(node.left, result)
            result.append(node.data)
            self._in_order_recursive(node.right, result)
tree = BST()
tree.insert(50)
tree.insert(30)
tree.insert(70)
tree.insert(20)
tree.insert(40)
tree.insert(60)
tree.insert(80)

print("In-order Traversal:", tree.in_order_traversal())
# Output: In-order Traversal: [20, 30, 40, 50, 60, 70, 80]
```

```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T4
.py"
In-order Traversal: [20, 30, 40, 50, 60, 70, 80]
PS C:\Users\kadal>
```

EXPLAINATION:



- Adds a new value to the tree.
- If the tree is empty, it becomes the root.
- Otherwise, it uses `_insert_recursive()` to find the correct position:
  - If `data < current.data`: go left.
  - If `data > current.data`: go right.
  - If equal: skip (no duplicates).

👀 `in_order_traversal()`

- Returns a sorted list of values.
- Uses `_in_order_recursive()`:
  - Traverse left subtree.
  - Visit current node.
  - Traverse right subtree.

TASK 5:

Hash Table

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

Sample Input Code:

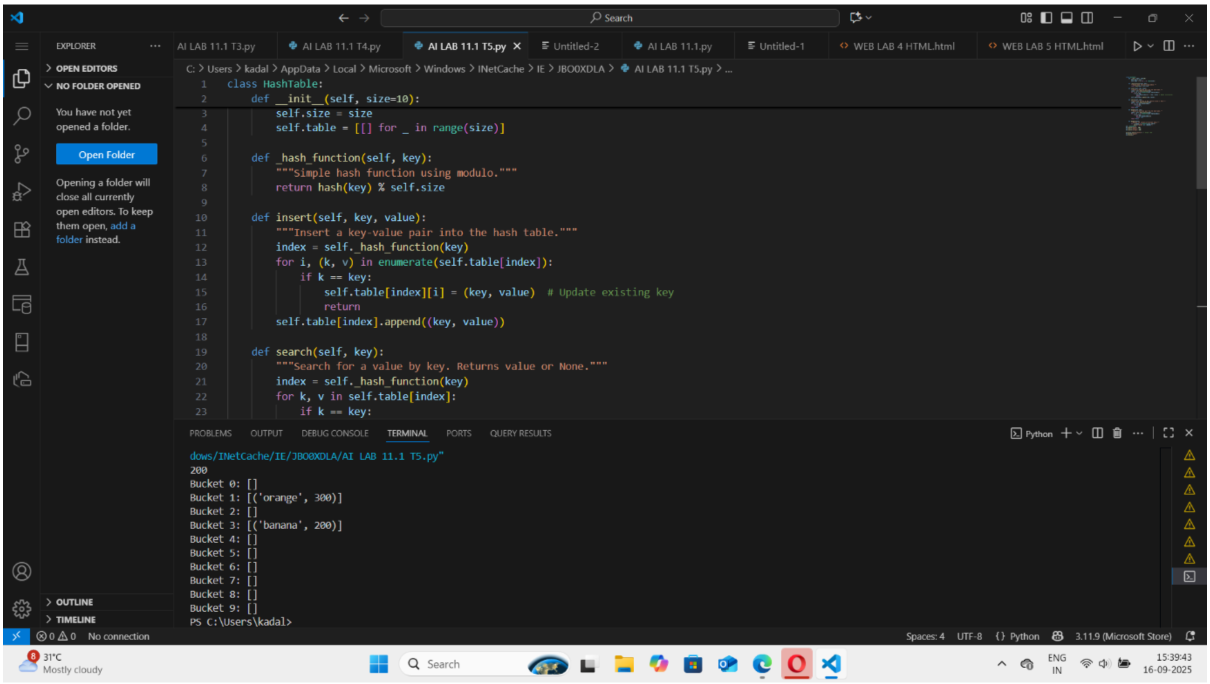
```
class HashTable:
pass.
```

PROMPT:

Generate python code and hash Table
Task: Use AI to implement a hash table with basic insert, search, and delete

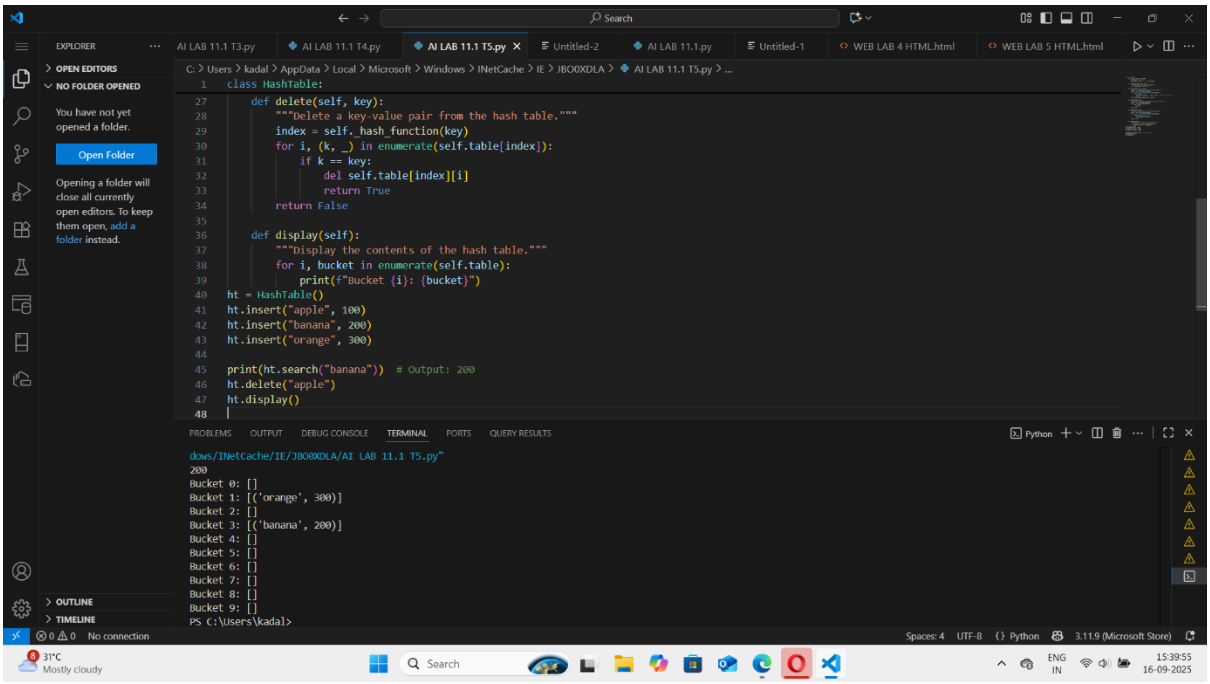
methods.
Sample Input Code:
class HashTable:
pass.

CODE & OUTPUT:



```python
class HashTable:
    def __init__(self, size=10):
        self.size = size
        self.table = [[] for _ in range(size)]

    def _hash_function(self, key):
        """Simple hash function using modulo."""
        return hash(key) % self.size

    def insert(self, key, value):
        """Insert a key-value pair into the hash table."""
        index = self._hash_function(key)
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                self.table[index][i] = (key, value)  # Update existing key
                return
        self.table[index].append((key, value))

    def search(self, key):
        """Search for a value by key. Returns value or None."""
        index = self._hash_function(key)
        for k, v in self.table[index]:
            if k == key:
```

```
dows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T5.py"
200
Bucket 0: []
Bucket 1: [('orange', 300)]
Bucket 2: []
Bucket 3: [('banana', 200)]
Bucket 4: []
Bucket 5: []
Bucket 6: []
Bucket 7: []
Bucket 8: []
Bucket 9: []
PS C:\Users\kadal>
```



```python
class HashTable:
    def delete(self, key):
        """Delete a key-value pair from the hash table."""
        index = self._hash_function(key)
        for i, (k, _) in enumerate(self.table[index]):
            if k == key:
                del self.table[index][i]
                return True
        return False

    def display(self):
        """Display the contents of the hash table."""
        for i, bucket in enumerate(self.table):
            print(f"Bucket {i}: {bucket}")
ht = HashTable()
ht.insert("apple", 100)
ht.insert("banana", 200)
ht.insert("orange", 300)

print(ht.search("banana"))  # Output: 200
ht.delete("apple")
ht.display()

```

```
dows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T5.py"
200
Bucket 0: []
Bucket 1: [('orange', 300)]
Bucket 2: []
Bucket 3: [('banana', 200)]
Bucket 4: []
Bucket 5: []
Bucket 6: []
Bucket 7: []
Bucket 8: []
Bucket 9: []
PS C:\Users\kadal>
```

EXPLAINATION:

🧠 `_hash_function(self, key)`

- Uses Python's built-in `hash()` function.
- Applies modulo to ensure the index fits within the table size.

🔧 `insert(self, key, value)`

- Computes the index using the hash function.
- Checks if the key already exists in the bucket:
    - If yes, updates the value.
    - If no, appends the new `(key, value)` pair.

🔍 `search(self, key)`

- Computes the index and scans the bucket.
- Returns the value if the key is found, otherwise returns `None`.

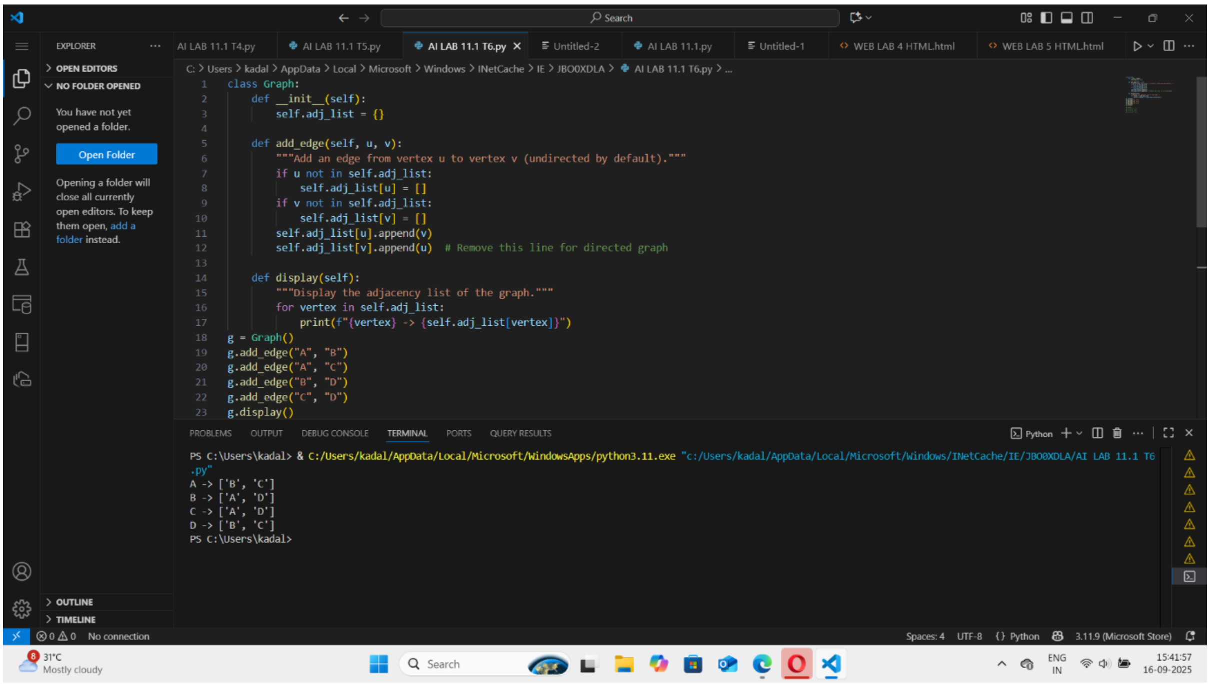
TASK 6:

Graph Representation
Task: Use AI to implement a graph using an adjacency list.
Sample Input Code:
class Graph:
pass.

PROMPT:

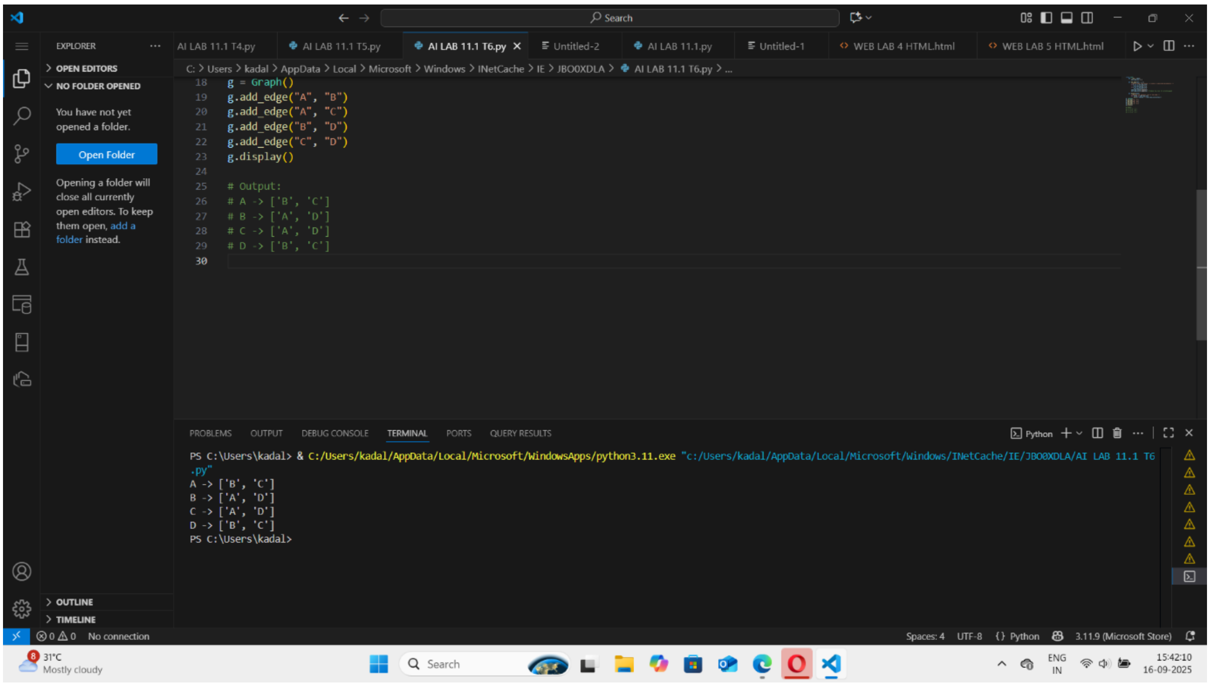
Generate python code and graph Representation
Task: Use AI to implement a graph using an adjacency list.
Sample Input Code:
class Graph:
pass.

CODE & OUTPUT:



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

    def add_edge(self, u, v):
        """Add an edge from vertex u to vertex v (undirected by default)."""
        if u not in self.adj_list:
            self.adj_list[u] = []
        if v not in self.adj_list:
            self.adj_list[v] = []
        self.adj_list[u].append(v)
        self.adj_list[v].append(u)  # Remove this line for directed graph

    def display(self):
        """Display the adjacency list of the graph."""
        for vertex in self.adj_list:
            print(f"{vertex} -> {self.adj_list[vertex]}")
g = Graph()
g.add_edge("A", "B")
g.add_edge("A", "C")
g.add_edge("B", "D")
g.add_edge("C", "D")
g.display()
```

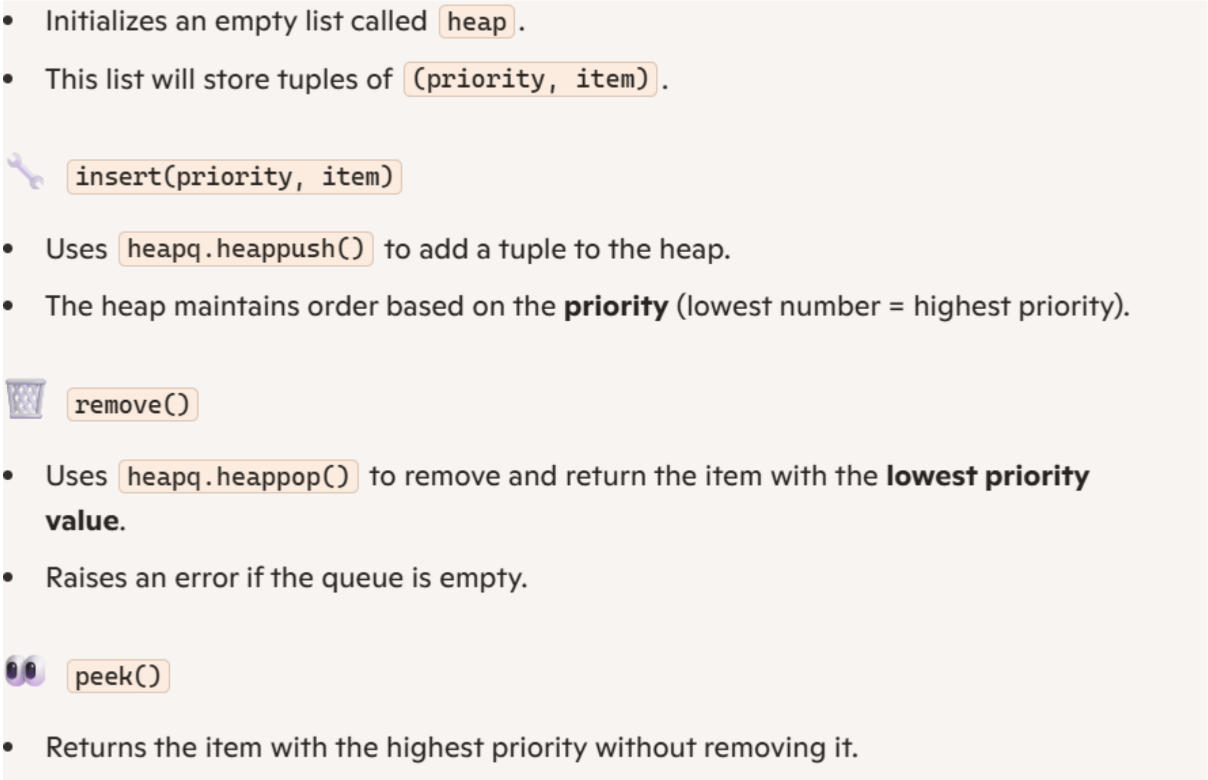
```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T6
.py"
A -> ['B', 'C']
B -> ['A', 'D']
C -> ['A', 'D']
D -> ['B', 'C']
PS C:\Users\kadal>
```



```python
g = Graph()
g.add_edge("A", "B")
g.add_edge("A", "C")
g.add_edge("B", "D")
g.add_edge("C", "D")
g.display()

# Output:
# A -> ['B', 'C']
# B -> ['A', 'D']
# C -> ['A', 'D']
# D -> ['B', 'C']
```

```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T6
.py"
A -> ['B', 'C']
B -> ['A', 'D']
C -> ['A', 'D']
D -> ['B', 'C']
PS C:\Users\kadal>
```

EXPLAINATION:



- Initializes an empty list called `heap`.
- This list will store tuples of `(priority, item)`.

🔧 `insert(priority, item)`

- Uses `heapq.heappush()` to add a tuple to the heap.
- The heap maintains order based on the **priority** (lowest number = highest priority).

🗑️ `remove()`

- Uses `heapq.heappop()` to remove and return the item with the **lowest priority value**.
- Raises an error if the queue is empty.

👀 `peek()`

- Returns the item with the highest priority without removing it.

TASK 7:

Priority Queue
Task: Use AI to implement a priority queue using Python’s heapq module.
Sample Input Code:

```
class PriorityQueue:
pass.
```

PROMPT:

Generate a python code and priority Queue

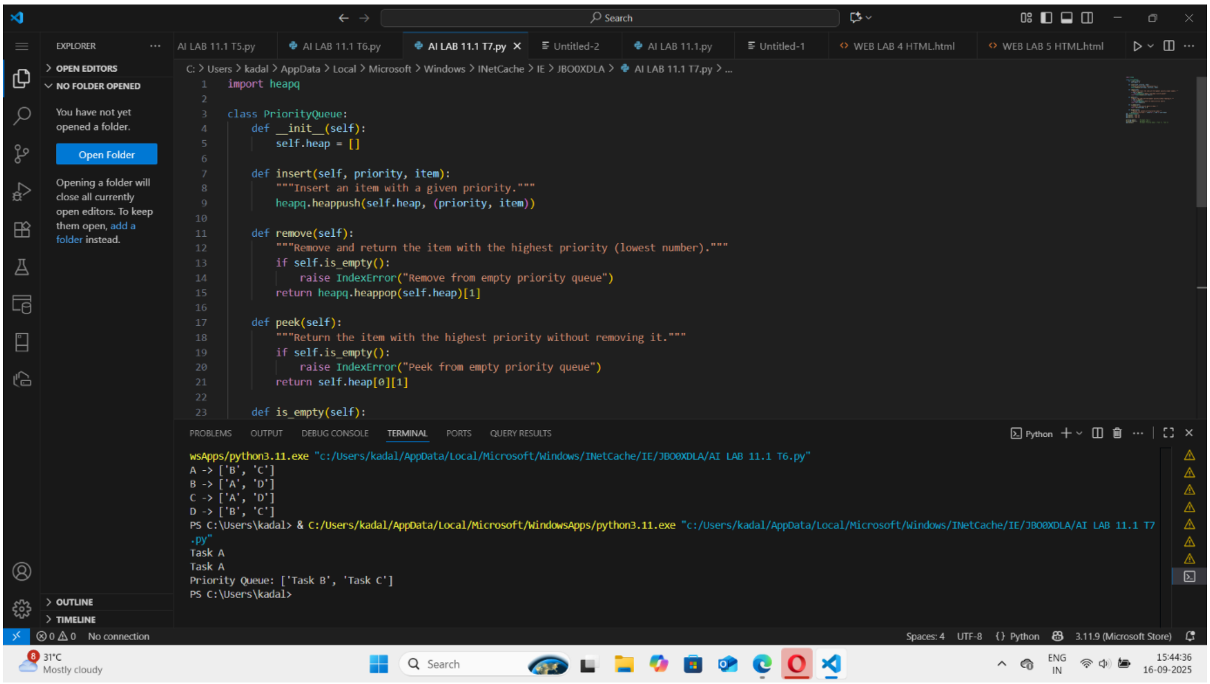
Task: Use AI to implement a priority queue using Python’s heapq module.

Sample Input Code:

class PriorityQueue:

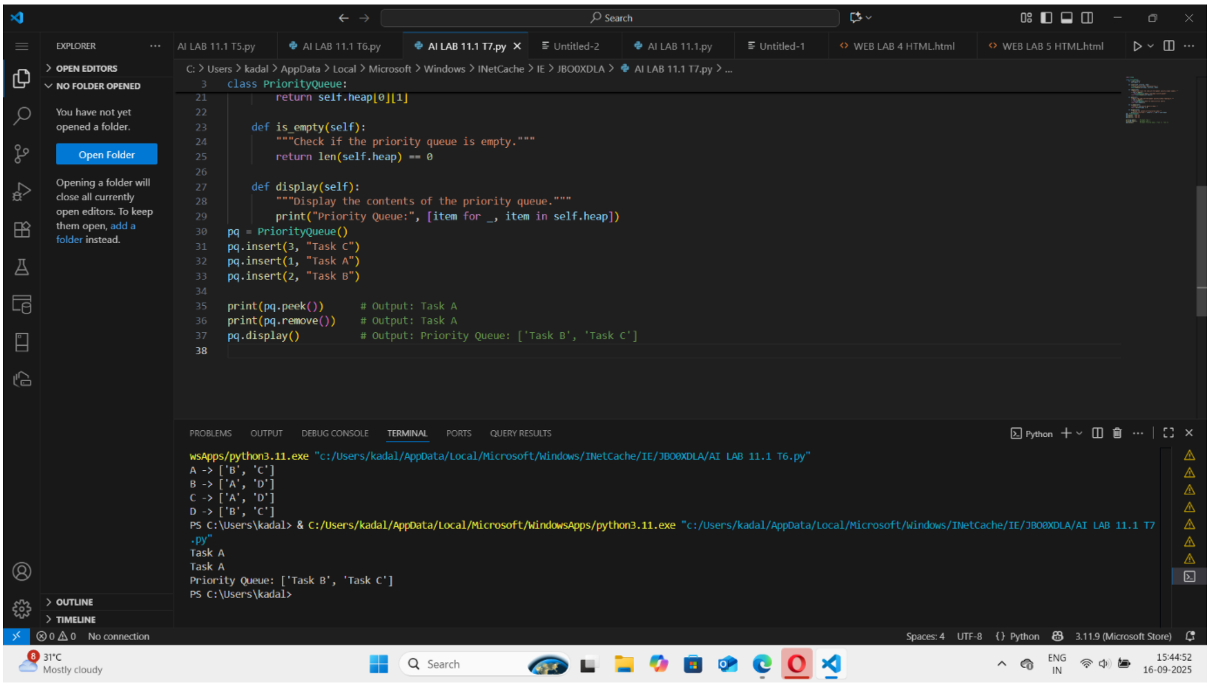
pass.

CODE & OUTPUT:



```python
import heapq

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

    def insert(self, priority, item):
        """Insert an item with a given priority."""
        heapq.heappush(self.heap, (priority, item))

    def remove(self):
        """Remove and return the item with the highest priority (lowest number)."""
        if self.is_empty():
            raise IndexError("Remove from empty priority queue")
        return heapq.heappop(self.heap)[1]

    def peek(self):
        """Return the item with the highest priority without removing it."""
        if self.is_empty():
            raise IndexError("Peek from empty priority queue")
        return self.heap[0][1]

    def is_empty(self):
```

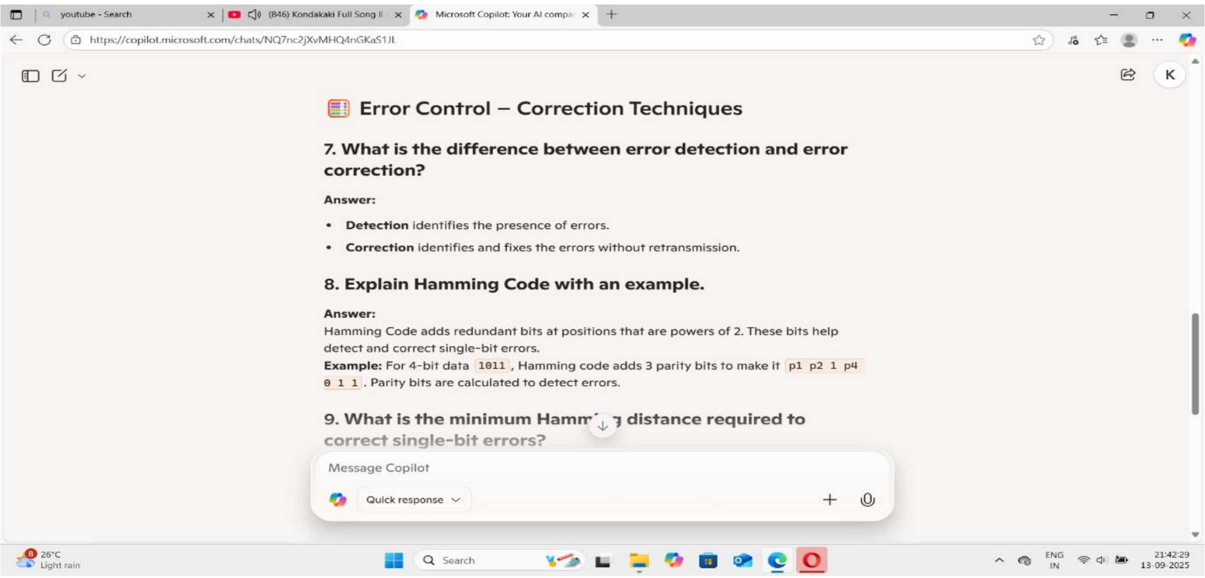
```
wsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T6.py"
A -> ['B', 'C']
B -> ['A', 'D']
C -> ['A', 'D']
D -> ['B', 'C']
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T7
.py"
Task A
Task A
Priority Queue: ['Task B', 'Task C']
PS C:\Users\kadal>
```



```python
class PriorityQueue:
        return self.heap[0][1]

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

    def display(self):
        """Display the contents of the priority queue."""
        print("Priority Queue:", [item for _, item in self.heap])
pq = PriorityQueue()
pq.insert(3, "Task C")
pq.insert(1, "Task A")
pq.insert(2, "Task B")

print(pq.peek())       # Output: Task A
print(pq.remove())     # Output: Task A
pq.display()           # Output: Priority Queue: ['Task B', 'Task C']
```

```
wsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T6.py"
A -> ['B', 'C']
B -> ['A', 'D']
C -> ['A', 'D']
D -> ['B', 'C']
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T7
.py"
Task A
Task A
Priority Queue: ['Task B', 'Task C']
PS C:\Users\kadal>
```

EXPLAINATION:


youtube - Search
(846) Kondakaki Full Song ||
Microsoft Copilot: Your AI compa
https://copilot.microsoft.com/chats/NQ7nc2jXvMHQ4nGKaS1JL
K
Message Copilot
Quick response
26°C
Light rain
Search
ENG
IN
21:42:29
13-09-2025

Error Control – Correction Techniques

7. What is the difference between error detection and error correction?

Answer:

- **Detection** identifies the presence of errors.
- **Correction** identifies and fixes the errors without retransmission.

8. Explain Hamming Code with an example.

Answer:
Hamming Code adds redundant bits at positions that are powers of 2. These bits help detect and correct single-bit errors.
Example: For 4-bit data `1011`, Hamming code adds 3 parity bits to make it `p1 p2 1 p4 0 1 1`. Parity bits are calculated to detect errors.

9. What is the minimum Hamming distance required to correct single-bit errors?

TASK 8:

Deque
Task: Use AI to implement a double-ended queue using collections.deque.
Sample Input Code:

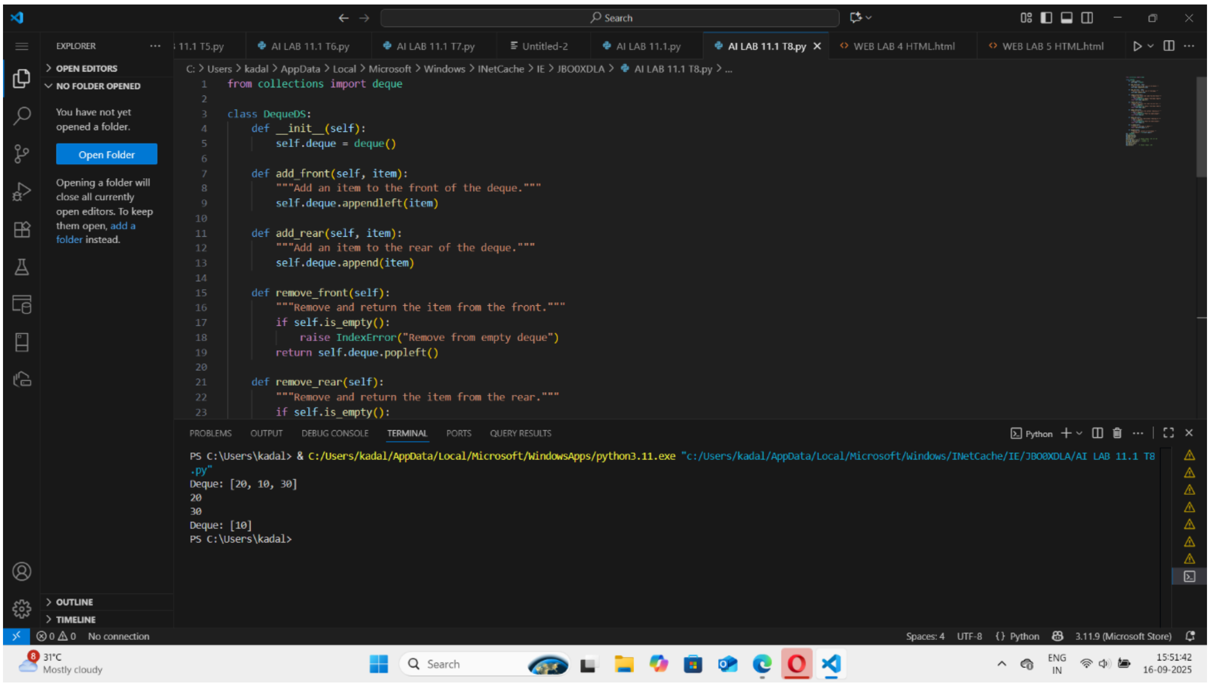
```
class DequeDS:
pass.
```

PROMPT:

Generate python code and deque
Task: Use AI to implement a double-ended queue using collections.deque.
Sample Input Code:

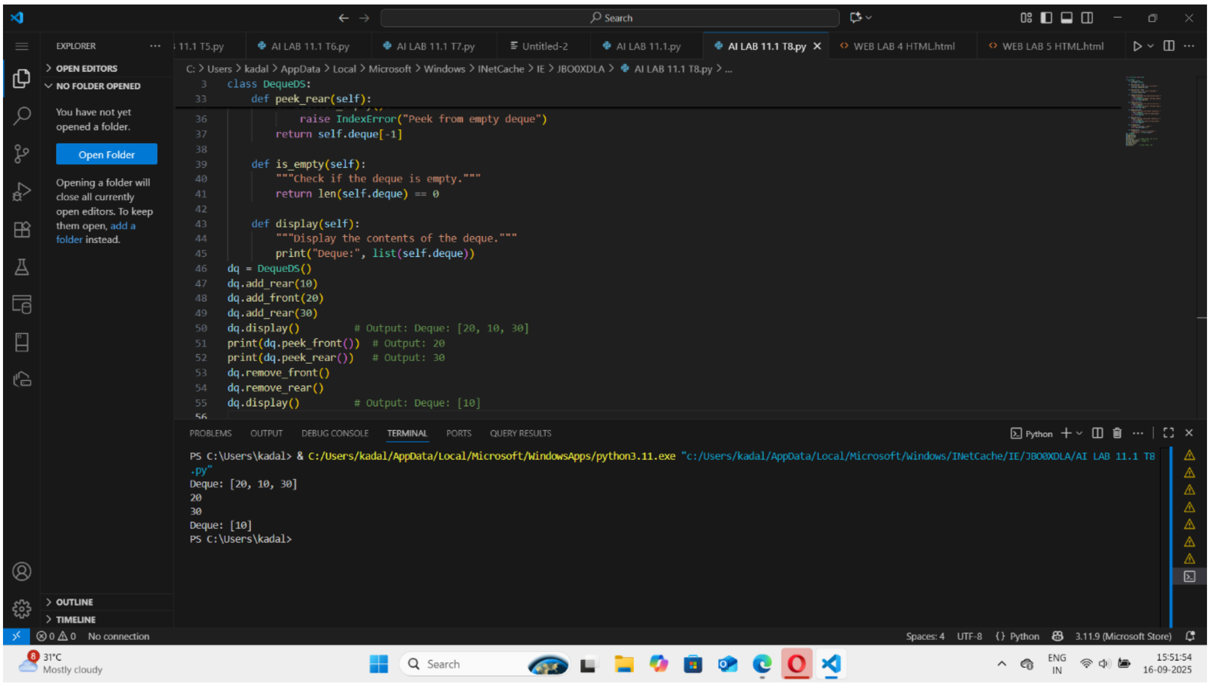
```
class DequeDS:
pass.
```

CODE &OUTPUT:



```
from collections import deque

class DequeDS:
    def __init__(self):
        self.deque = deque()

    def add_front(self, item):
        """Add an item to the front of the deque."""
        self.deque.appendleft(item)

    def add_rear(self, item):
        """Add an item to the rear of the deque."""
        self.deque.append(item)

    def remove_front(self):
        """Remove and return the item from the front."""
        if self.is_empty():
            raise IndexError("Remove from empty deque")
        return self.deque.popleft()

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

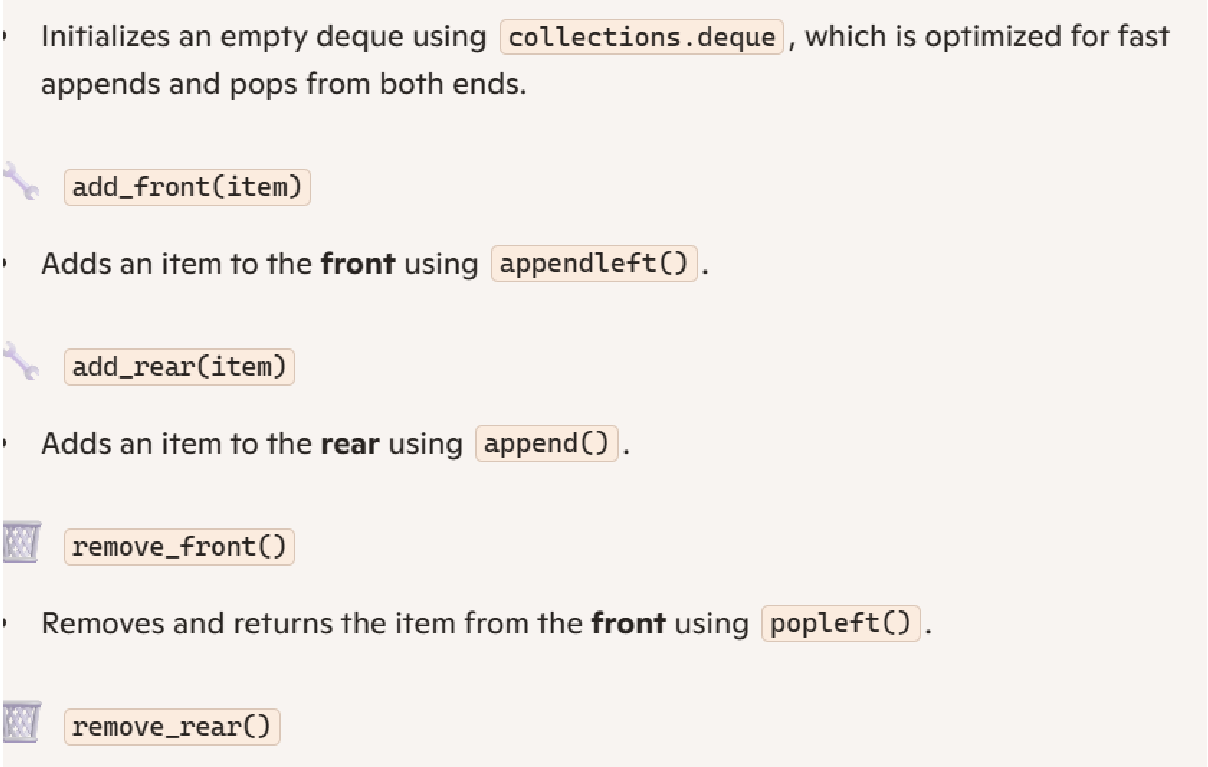
```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T8
.py"
Deque: [20, 10, 30]
20
30
Deque: [10]
PS C:\Users\kadal>
```



```
class DequeDS:
    def peek_rear(self):
            raise IndexError("Peek from empty deque")
        return self.deque[-1]

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

    def display(self):
        """Display the contents of the deque."""
        print("Deque:", list(self.deque))
dq = DequeDS()
dq.add_rear(10)
dq.add_front(20)
dq.add_rear(30)
dq.display()         # Output: Deque: [20, 10, 30]
print(dq.peek_front())  # Output: 20
print(dq.peek_rear())   # Output: 30
dq.remove_front()
dq.remove_rear()
dq.display()         # Output: Deque: [10]
```

```
PS C:\Users\kadal> & C:/Users/kadal/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/kadal/AppData/Local/Microsoft/Windows/INetCache/IE/JBO0XDLA/AI LAB 11.1 T8
.py"
Deque: [20, 10, 30]
20
30
Deque: [10]
PS C:\Users\kadal>
```

EXPLAINATION:



- Initializes an empty deque using `collections.deque`, which is optimized for fast appends and pops from both ends.

🔧 `add_front(item)`

- Adds an item to the **front** using `appendleft()`.

🔧 `add_rear(item)`

- Adds an item to the **rear** using `append()`.

🗑️ `remove_front()`

- Removes and returns the item from the **front** using `popleft()`.

🗑️ `remove_rear()`

TASK 9:

AI-Generated Data Structure Comparisons

Task: Use AI to generate a comparison table of different data structures (stack,
queue, linked list, etc.) including time complexities.

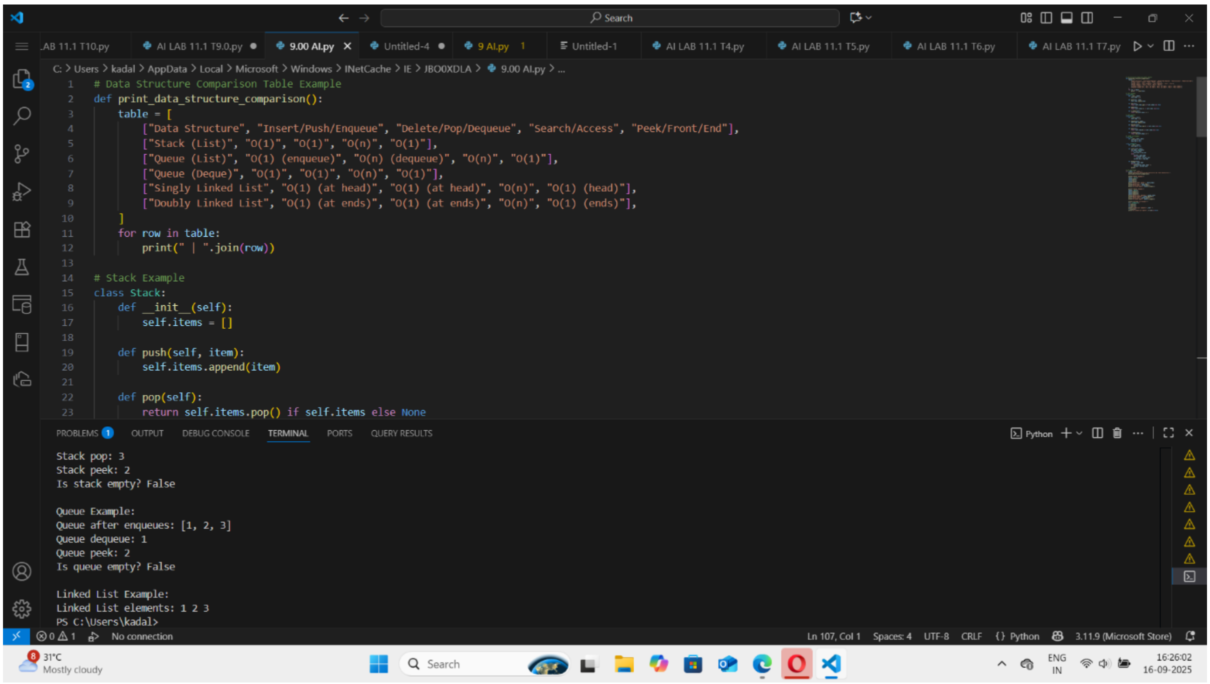
Sample Input Code:

# No code, prompt AI for a data structure comparison table.

PROMPT:

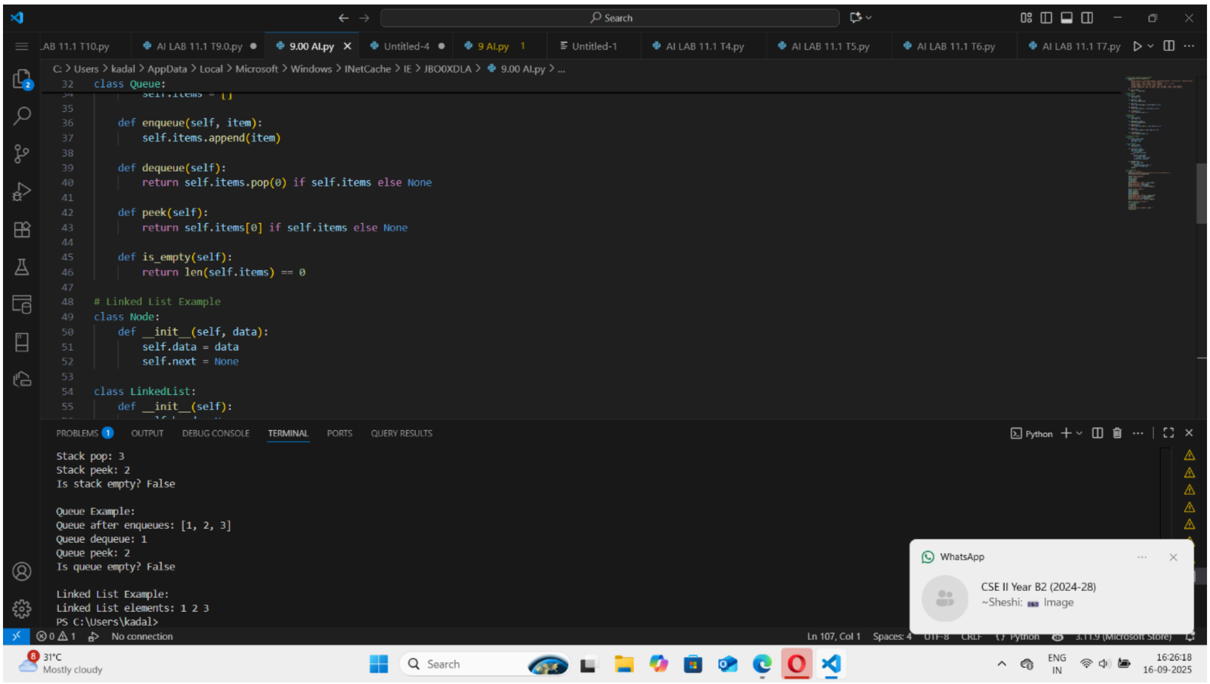
Generate python code and AI-Generated Data Structure Comparisons
Task: Use AI to generate a comparison table of different data structures (stack,
queue, linked list, etc.) including time complexities.
Sample Input Code:
# No code, prompt AI for a data structure comparison table.

CODE & OUTPUT:



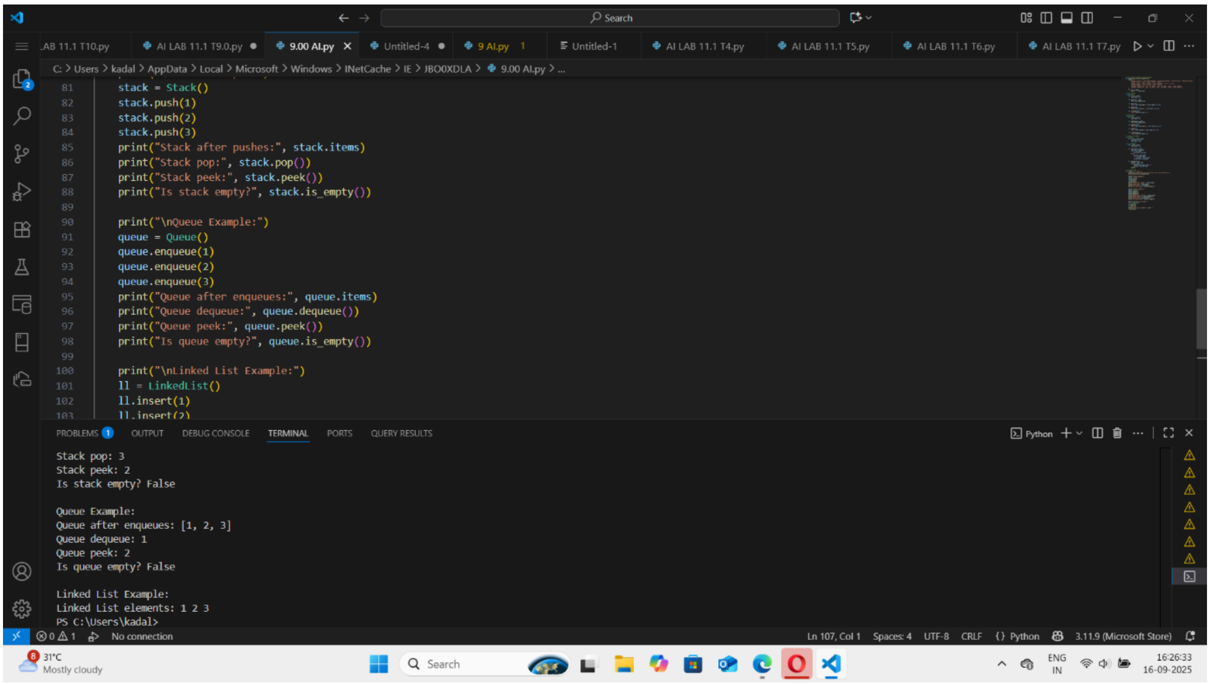
```python
# Data Structure Comparison Table Example
def print_data_structure_comparison():
    table = [
        ["Data Structure", "Insert/Push/Enqueue", "Delete/Pop/Dequeue", "Search/Access", "Peek/Front/End"],
        ["Stack (List)", "O(1)", "O(1)", "O(n)", "O(1)"],
        ["Queue (List)", "O(1) (enqueue)", "O(n) (dequeue)", "O(n)", "O(1)"],
        ["Queue (Deque)", "O(1)", "O(1)", "O(n)", "O(1)"],
        ["Singly Linked List", "O(1) (at head)", "O(1) (at head)", "O(n)", "O(1) (head)"],
        ["Doubly Linked List", "O(1) (at ends)", "O(1) (at ends)", "O(n)", "O(1) (ends)"],
    ]
    for row in table:
        print(" | ".join(row))

# Stack Example
class Stack:
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop() if self.items else None
```

```
Stack pop: 3
Stack peek: 2
Is stack empty? False

Queue Example:
Queue after enqueues: [1, 2, 3]
Queue dequeue: 1
Queue peek: 2
Is queue empty? False

Linked List Example:
Linked List elements: 1 2 3
PS C:\Users\kadal>
```



```python
class Queue:
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        return self.items.pop(0) if self.items else None

    def peek(self):
        return self.items[0] if self.items else None

    def is_empty(self):
        return len(self.items) == 0

# Linked List Example
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
```

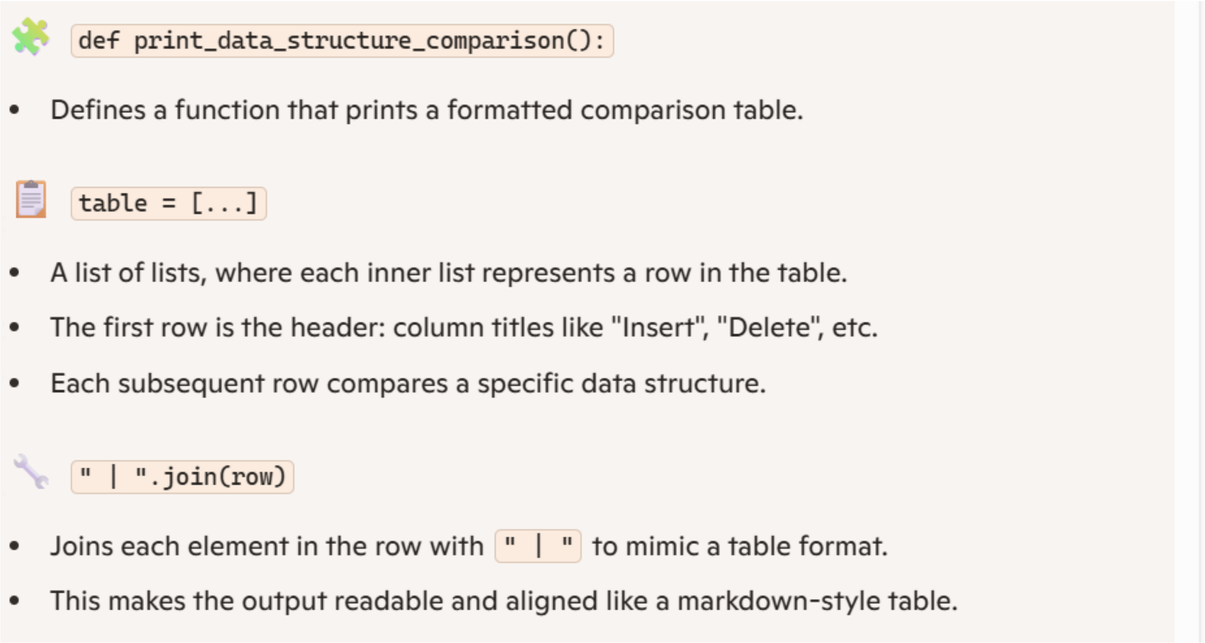
```
Stack pop: 3
Stack peek: 2
Is stack empty? False

Queue Example:
Queue after enqueues: [1, 2, 3]
Queue dequeue: 1
Queue peek: 2
Is queue empty? False

Linked List Example:
Linked List elements: 1 2 3
PS C:\Users\kadal>
```



```python
    stack = Stack()
    stack.push(1)
    stack.push(2)
    stack.push(3)
    print("Stack after pushes:", stack.items)
    print("Stack pop:", stack.pop())
    print("Stack peek:", stack.peek())
    print("Is stack empty?", stack.is_empty())

    print("\nQueue Example:")
    queue = Queue()
    queue.enqueue(1)
    queue.enqueue(2)
    queue.enqueue(3)
    print("Queue after enqueues:", queue.items)
    print("Queue dequeue:", queue.dequeue())
    print("Queue peek:", queue.peek())
    print("Is queue empty?", queue.is_empty())

    print("\nLinked List Example:")
    ll = LinkedList()
    ll.insert(1)
    ll.insert(2)
```

```
Stack pop: 3
Stack peek: 2
Is stack empty? False

Queue Example:
Queue after enqueues: [1, 2, 3]
Queue dequeue: 1
Queue peek: 2
Is queue empty? False

Linked List Example:
Linked List elements: 1 2 3
PS C:\Users\kadal>
```

EXPLAINATION:



🧩 `def print_data_structure_comparison():`

- Defines a function that prints a formatted comparison table.

📋 `table = [...]`

- A list of lists, where each inner list represents a row in the table.
- The first row is the header: column titles like "Insert", "Delete", etc.
- Each subsequent row compares a specific data structure.

🔧 `" | ".join(row)`

- Joins each element in the row with `" | "` to mimic a table format.
- This makes the output readable and aligned like a markdown-style table.

TASK 10:

Task Description #10 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:
  - o Stack
  - o Queue
  - o Priority Queue
  - o Linked List
  - o Binary Search Tree (BST)
  - o Graph
  - o Hash Table
  - o Deque
- Justify your choice in 2–3 sentences per feature.
- Implement one selected feature as a working Python program with AI-assisted code generation.

PROMPT:

Generate python code and task Description #10 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:

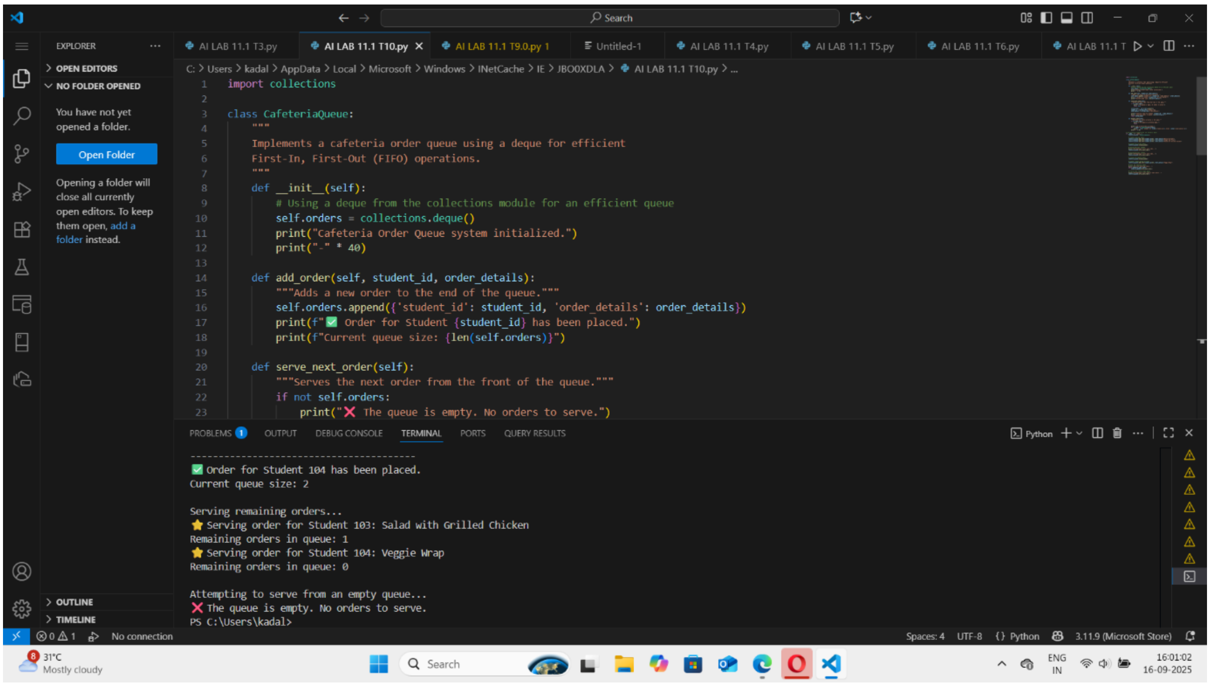
- For each feature, select the most

appropriate data structure from the list below:

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

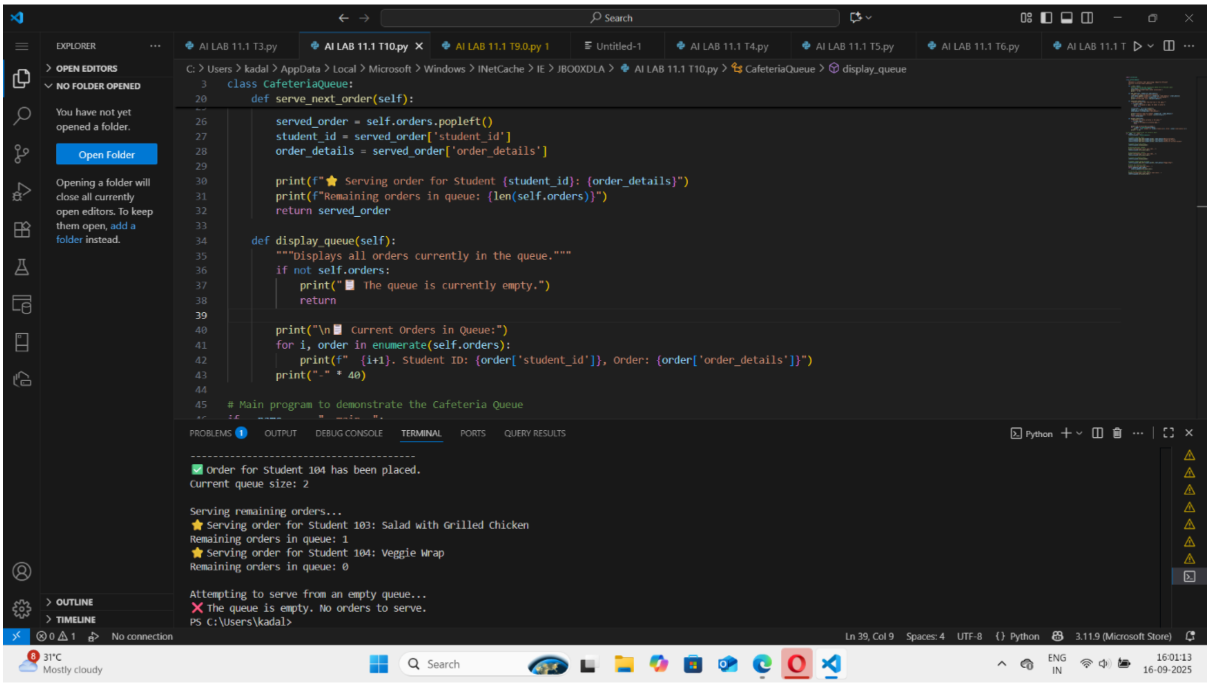
- Justify your choice in 2–3 sentences per feature.
- Implement one selected feature as a working Python program with AI-assisted code generation.

CODE & OUTPUT:



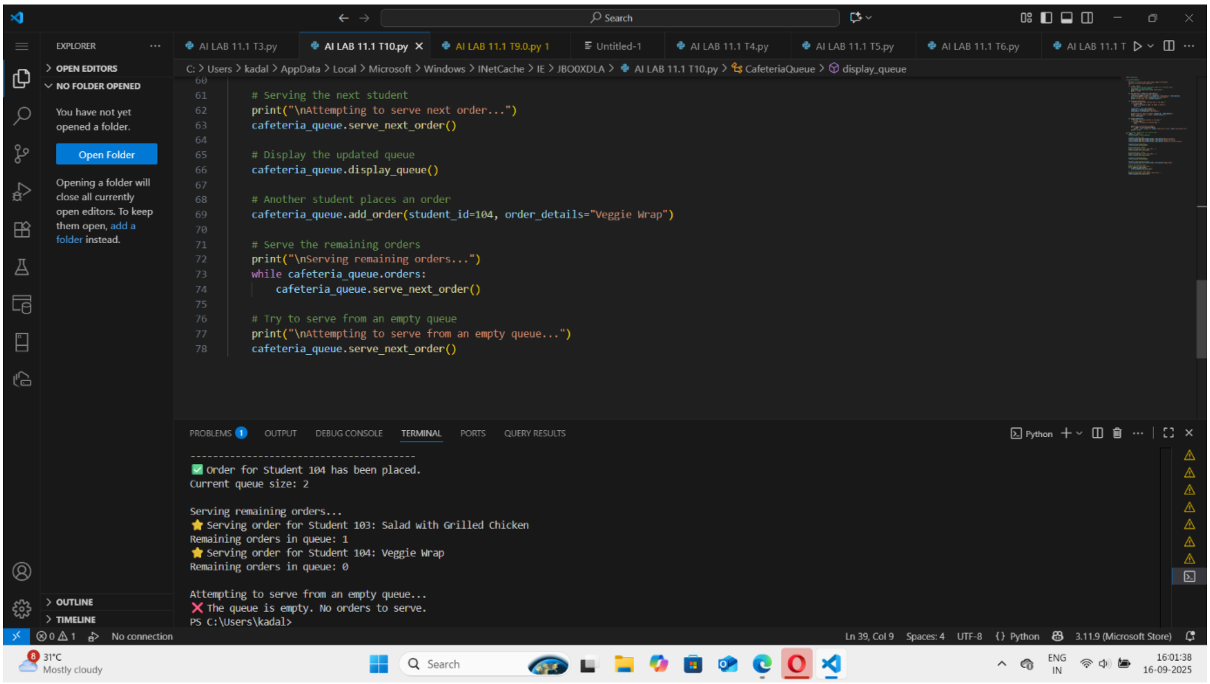
```python
import collections

class CafeteriaQueue:
    """
    Implements a cafeteria order queue using a deque for efficient
    First-In, First-Out (FIFO) operations.
    """
    def __init__(self):
        # Using a deque from the collections module for an efficient queue
        self.orders = collections.deque()
        print("Cafeteria Order Queue system initialized.")
        print("-" * 40)

    def add_order(self, student_id, order_details):
        """Adds a new order to the end of the queue."""
        self.orders.append({'student_id': student_id, 'order_details': order_details})
        print(f"✅ Order for Student {student_id} has been placed.")
        print(f"Current queue size: {len(self.orders)}")

    def serve_next_order(self):
        """Serves the next order from the front of the queue."""
        if not self.orders:
            print("❌ The queue is empty. No orders to serve.")
```

```
----------------------------------------
✅ Order for Student 104 has been placed.
Current queue size: 2

Serving remaining orders...
⭐ Serving order for Student 103: Salad with Grilled Chicken
Remaining orders in queue: 1
⭐ Serving order for Student 104: Veggie Wrap
Remaining orders in queue: 0

Attempting to serve from an empty queue...
❌ The queue is empty. No orders to serve.
PS C:\Users\kadal>
```


AI LAB 11.1 T3.py
AI LAB 11.1 T10.py
AI LAB 11.1 T9.0.py 1
Untitled-1
AI LAB 11.1 T4.py
AI LAB 11.1 T5.py
AI LAB 11.1 T6.py
C: > Users > kadal > AppData > Local > Microsoft > Windows > INetCache > IE > JBO0XDLA > AI LAB 11.1 T10.py > CafeteriaQueue > display_queue
Ln 39, Col 9

```python
class CafeteriaQueue:
    def serve_next_order(self):
            served_order = self.orders.popleft()
            student_id = served_order['student_id']
            order_details = served_order['order_details']

            print(f"⭐ Serving order for Student {student_id}: {order_details}")
            print(f"Remaining orders in queue: {len(self.orders)}")
            return served_order

    def display_queue(self):
        """Displays all orders currently in the queue."""
        if not self.orders:
            print("📋 The queue is currently empty.")
            return

        print("\n📋 Current Orders in Queue:")
        for i, order in enumerate(self.orders):
            print(f"  {i+1}. Student ID: {order['student_id']}, Order: {order['order_details']}")
        print("-" * 40)

# Main program to demonstrate the Cafeteria Queue
```

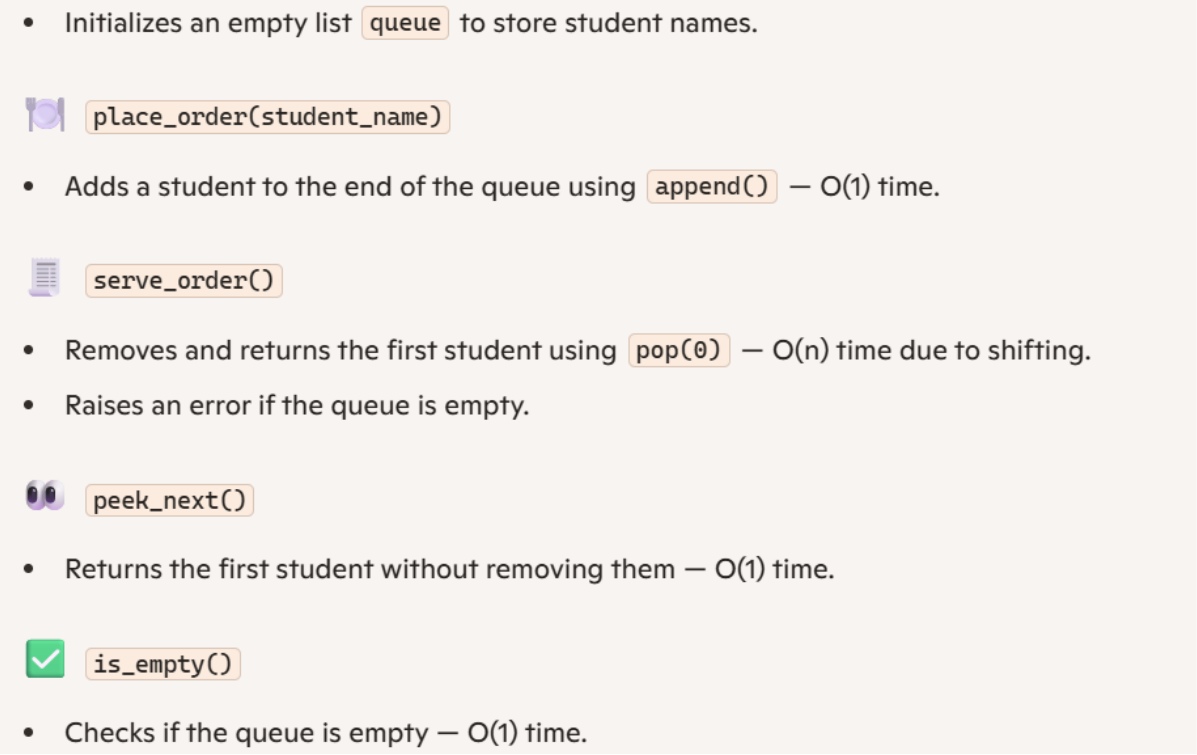
```
----------------------------------------
✅ Order for Student 104 has been placed.
Current queue size: 2

Serving remaining orders...
⭐ Serving order for Student 103: Salad with Grilled Chicken
Remaining orders in queue: 1
⭐ Serving order for Student 104: Veggie Wrap
Remaining orders in queue: 0

Attempting to serve from an empty queue...
❌ The queue is empty. No orders to serve.
PS C:\Users\kadal>
```



```python
    # Serving the next student
    print("\nAttempting to serve next order...")
    cafeteria_queue.serve_next_order()

    # Display the updated queue
    cafeteria_queue.display_queue()

    # Another student places an order
    cafeteria_queue.add_order(student_id=104, order_details="Veggie Wrap")

    # Serve the remaining orders
    print("\nServing remaining orders...")
    while cafeteria_queue.orders:
        cafeteria_queue.serve_next_order()

    # Try to serve from an empty queue
    print("\nAttempting to serve from an empty queue...")
    cafeteria_queue.serve_next_order()
```

```
----------------------------------------
✅ Order for Student 104 has been placed.
Current queue size: 2

Serving remaining orders...
⭐ Serving order for Student 103: Salad with Grilled Chicken
Remaining orders in queue: 1
⭐ Serving order for Student 104: Veggie Wrap
Remaining orders in queue: 0

Attempting to serve from an empty queue...
❌ The queue is empty. No orders to serve.
PS C:\Users\kadal>
```

EXPLAINATION:



- Initializes an empty list `queue` to store student names.

🍽️ `place_order(student_name)`

- Adds a student to the end of the queue using `append()` — O(1) time.

🧾 `serve_order()`

- Removes and returns the first student using `pop(0)` — O(n) time due to shifting.
- Raises an error if the queue is empty.

👀 `peek_next()`

- Returns the first student without removing them — O(1) time.

✅ `is_empty()`

- Checks if the queue is empty — O(1) time.