SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	**DEPARTMENT OF COMPUTER SCIENCE ENGINEERING**

Program Name: B. Tech	**Assignment Type: Lab**	**Academic Year:**2025-2026

Course Coordinator Name	Dr. Rishabh Mittal
Instructor(s) Name	Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju

CourseCode	23CS002PC304	**Course Title**	AI Assisted Coding
Year/Sem	III/II	**Regulation**	R23
Date and Day of Assignment	**Week6 – Monday**	**Time(s)**	23CSBTB01 To 23CSBTB52
Duration	2 Hours	**Applicable to Batches**	All batches

Assignment Number: 11.1(Present assignment number)/**24**(Total number of assignments)

Q.No.	**Question**	***Expected Time to complete***
1	**Lab 11 – Data Structures with AI: Implementing Fundamental Structures** **Lab Objectives** • Use AI to assist in designing and implementing fundamental data structures	Week6 - Monday

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
```

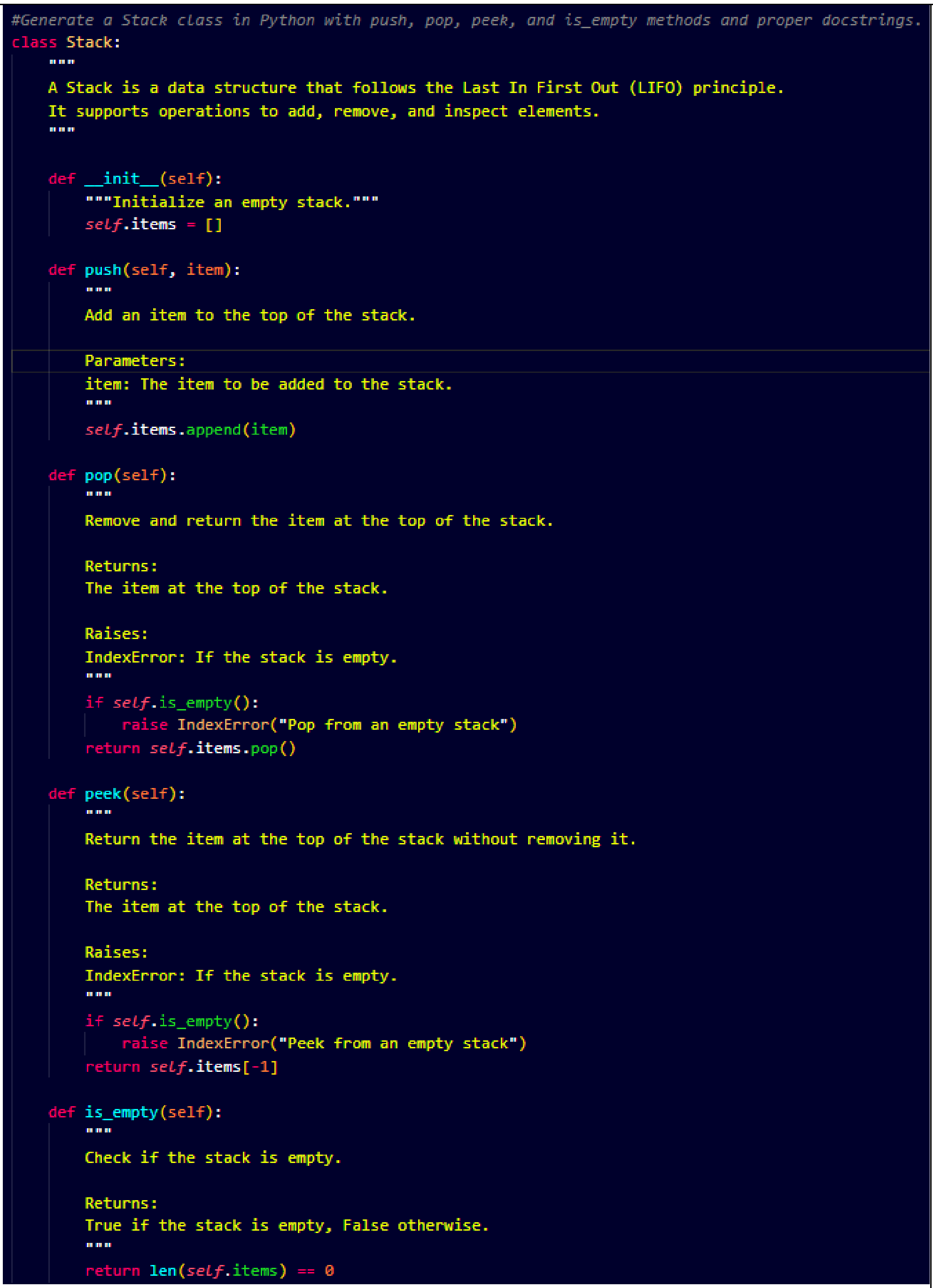
Expected Output:

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

Prompt:

Generate a Stack class in Python with push, pop, peek, and is_empty methods and proper docstrings.

Code:



```python
#Generate a Stack class in Python with push, pop, peek, and is_empty methods and proper docstrings.
class Stack:
    """
    A Stack is a data structure that follows the Last In First Out (LIFO) principle.
    It supports operations to add, remove, and inspect elements.
    """

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

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

        Parameters:
        item: The item to be added to the stack.
        """
        self.items.append(item)

    def pop(self):
        """
        Remove and return the item at the top of the stack.

        Returns:
        The item at the top of the stack.

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

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

        Returns:
        The item at the top of the stack.

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

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

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

Task Description #2 – Queue Implementation

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

Sample Input Code:

```
class Queue:
    pass
```

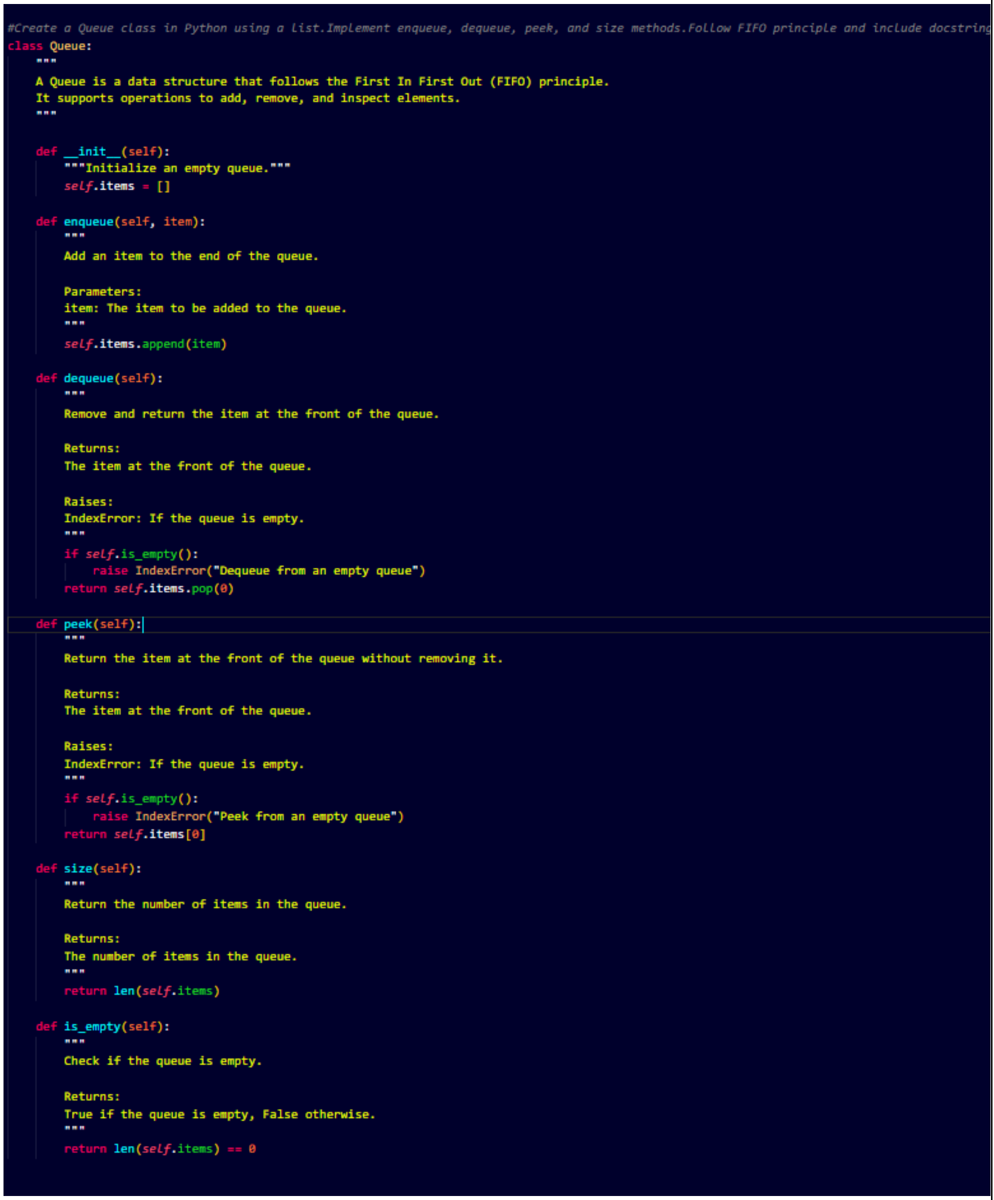
Expected Output:

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

Prompt:

Create a Queue class in Python using a list.Implement enqueue, dequeue, peek, and size methods.Follow FIFO principle and include docstrings.

Output:



```
#Create a Queue class in Python using a list.Implement enqueue, dequeue, peek, and size methods.Follow FIFO principle and include docstring
class Queue:
    """
    A Queue is a data structure that follows the First In First Out (FIFO) principle.
    It supports operations to add, remove, and inspect elements.
    """

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

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

        Parameters:
        item: The item to be added to the queue.
        """
        self.items.append(item)

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

        Returns:
        The item at the front of the queue.

        Raises:
        IndexError: If the queue is empty.
        """
        if self.is_empty():
            raise IndexError("Dequeue from an empty queue")
        return self.items.pop(0)

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

        Returns:
        The item at the front of the queue.

        Raises:
        IndexError: If the queue is empty.
        """
        if self.is_empty():
            raise IndexError("Peek from an empty queue")
        return self.items[0]

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

        Returns:
        The number of items in the queue.
        """
        return len(self.items)

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

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

Task Description #3 – Linked List

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

Sample Input Code:

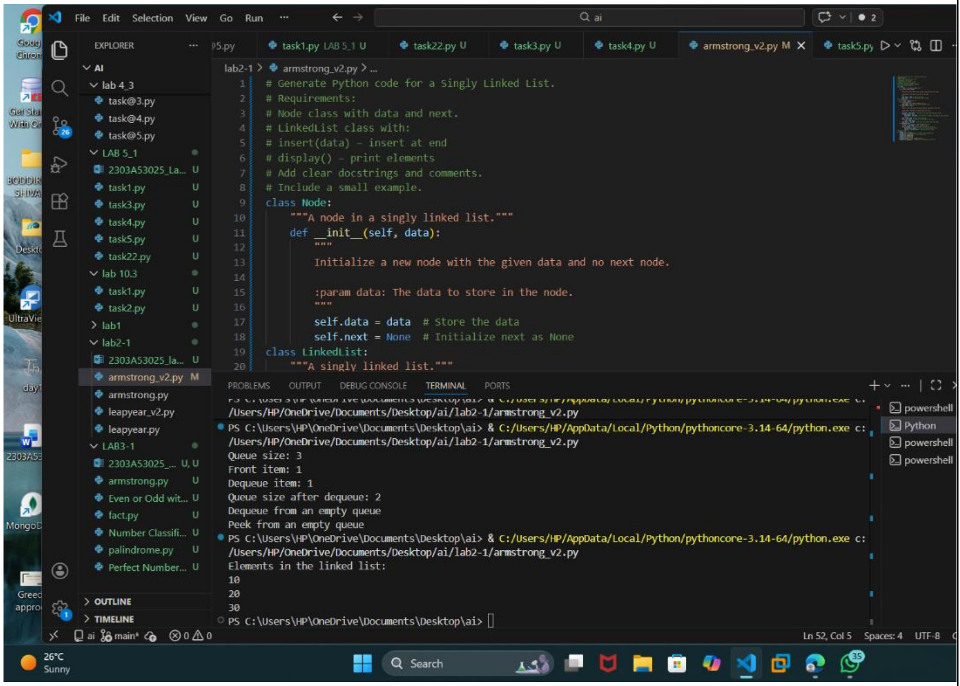
```
class Node:
    pass
```

class LinkedList:

    pass

Expected Output:

- A working linked list implementation with clear method documentation.



```python
# Generate Python code for a Singly Linked List.
# Requirements:
# Node class with data and next.
# LinkedList class with:
# insert(data) - insert at end
# display() - print elements
# Add clear docstrings and comments.
# Include a small example.
class Node:
    """A node in a singly linked list."""
    def __init__(self, data):
        """
        Initialize a new node with the given data and no next node.

        :param data: The data to store in the node.
        """
        self.data = data  # Store the data
        self.next = None  # Initialize next as None
class LinkedList:
    """A singly linked list."""
```

```
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
Queue size: 3
Front item: 1
Dequeue item: 1
Queue size after dequeue: 2
Dequeue from an empty queue
Peek from an empty queue
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
Elements in the linked list:
10
20
30
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> 
```

Task Description #4 – Binary Search Tree (BST)

Task: Use AI to create a BST with insert and in-order traversal methods.

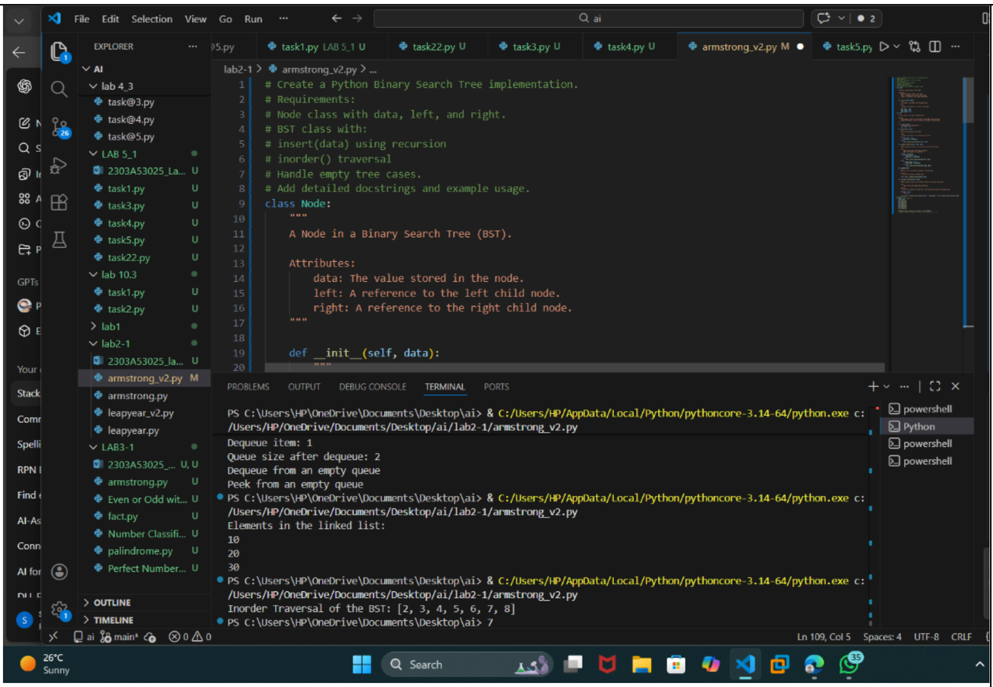
Sample Input Code:

class BST:

    pass

Expected Output:

- BST implementation with recursive insert and traversal methods.



```python
# Create a Python Binary Search Tree implementation.
# Requirements:
# Node class with data, left, and right.
# BST class with:
# insert(data) using recursion
# inorder() traversal
# Handle empty tree cases.
# Add detailed docstrings and example usage.
class Node:
    """
    A Node in a Binary Search Tree (BST).

    Attributes:
        data: The value stored in the node.
        left: A reference to the left child node.
        right: A reference to the right child node.
    """

    def __init__(self, data):
        """
```

```
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
Dequeue item: 1
Queue size after dequeue: 2
Dequeue from an empty queue
Peek from an empty queue
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
Elements in the linked list:
10
20
30
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
Inorder Traversal of the BST: [2, 3, 4, 5, 6, 7, 8]
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> 7
```

Task Description #5 – Hash Table

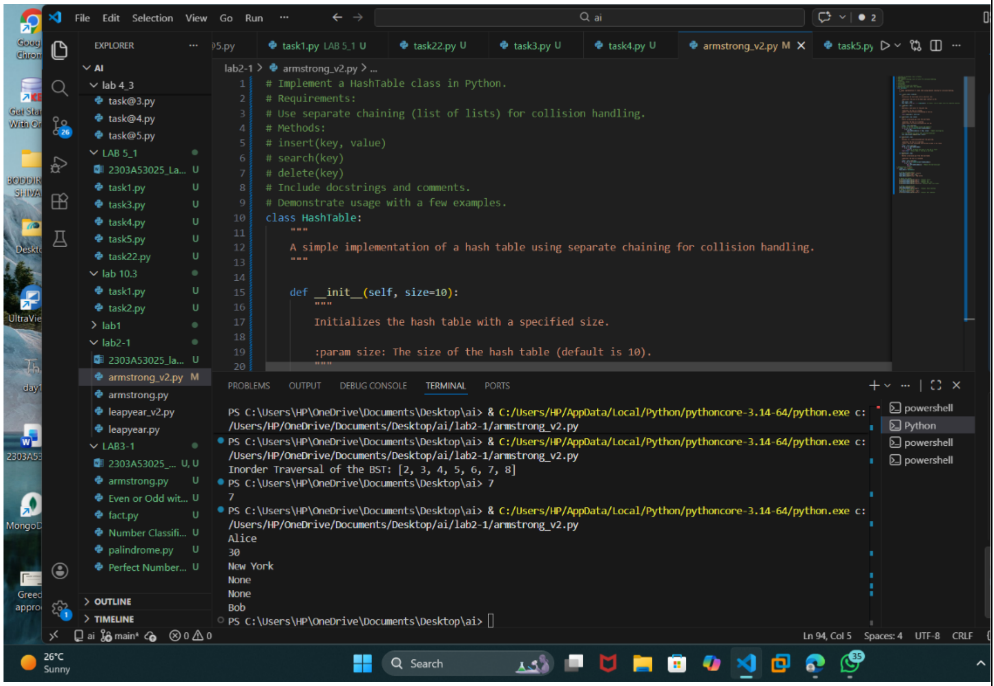
Task: Use AI to implement a hash table with basic insert, search, and delete methods.

Sample Input Code:

```
class HashTable:
    pass
```

Expected Output:

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



```python
# Implement a HashTable class in Python.
# Requirements:
# Use separate chaining (list of lists) for collision handling.
# Methods:
# insert(key, value)
# search(key)
# delete(key)
# Include docstrings and comments.
# Demonstrate usage with a few examples.
class HashTable:
    """
    A simple implementation of a hash table using separate chaining for collision handling.
    """

    def __init__(self, size=10):
        """
        Initializes the hash table with a specified size.

        :param size: The size of the hash table (default is 10).
        """
```

```
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
Inorder Traversal of the BST: [2, 3, 4, 5, 6, 7, 8]
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> 7
7
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
Alice
30
New York
None
None
Bob
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> 
```

Task Description #6 – Graph Representation

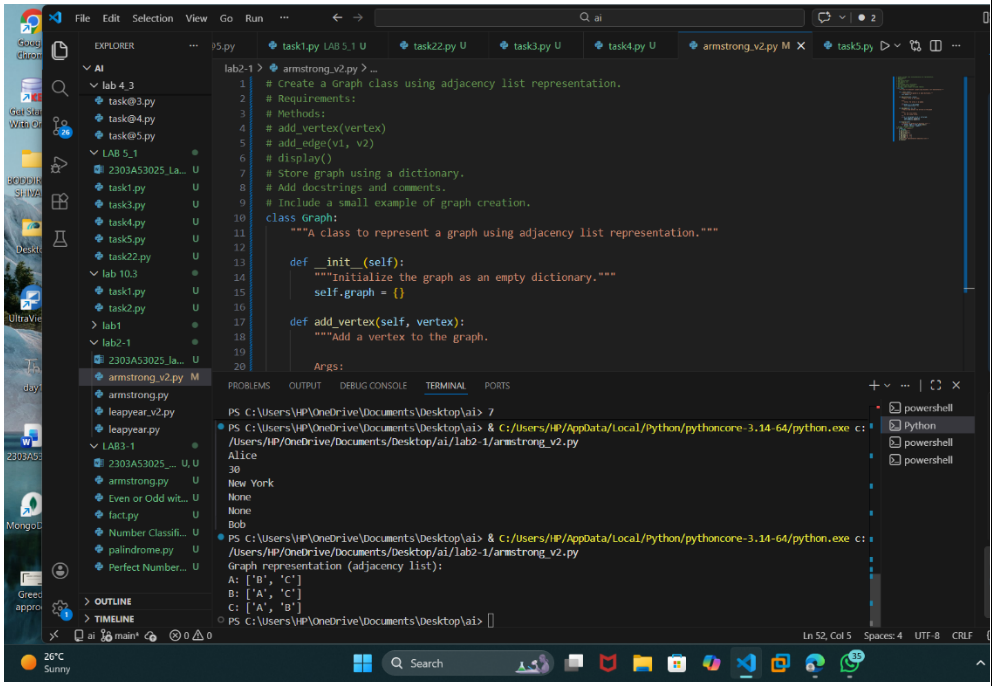
Task: Use AI to implement a graph using an adjacency list.

Sample Input Code:

```
class Graph:
    pass
```

Expected Output:

- Graph with methods to add vertices, add edges, and display connections.



```python
# Create a Graph class using adjacency list representation.
# Requirements:
# Methods:
# add_vertex(vertex)
# add_edge(v1, v2)
# display()
# Store graph using a dictionary.
# Add docstrings and comments.
# Include a small example of graph creation.
class Graph:
    """A class to represent a graph using adjacency list representation."""

    def __init__(self):
        """Initialize the graph as an empty dictionary."""
        self.graph = {}

    def add_vertex(self, vertex):
        """Add a vertex to the graph.

        Args:
```

```
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> 7
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
Alice
30
New York
None
None
Bob
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
Graph representation (adjacency list):
A: ['B', 'C']
B: ['A', 'C']
C: ['A', 'B']
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> 
```

Task Description #7 – Priority Queue

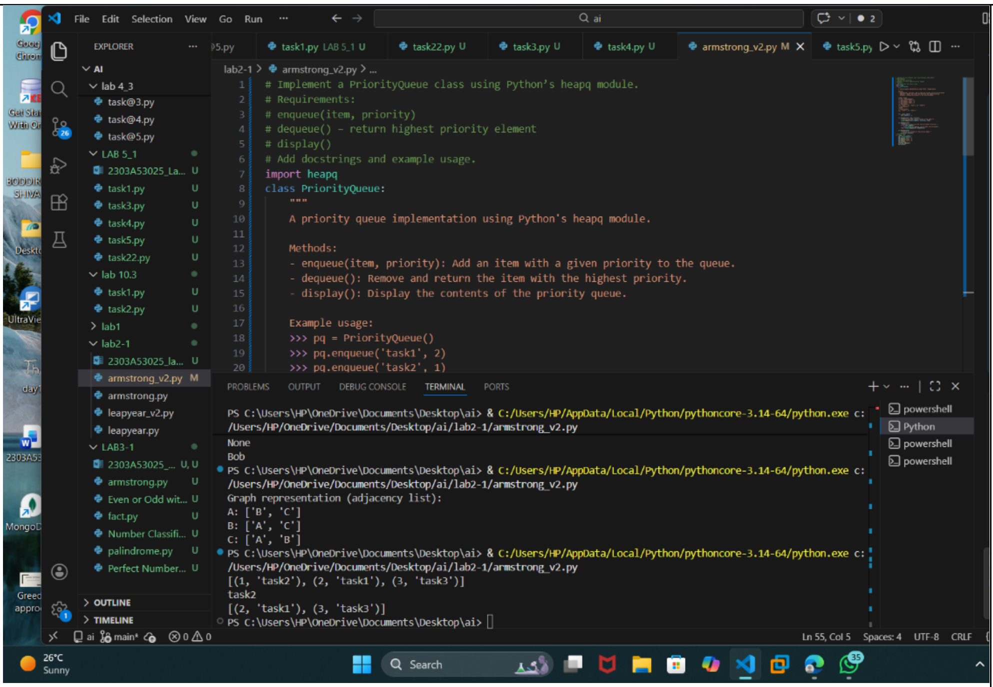
Task: Use AI to implement a priority queue using Python’s heapq module.

Sample Input Code:

```
class PriorityQueue:
    pass
```

Expected Output:

- Implementation with enqueue (priority), dequeue (highest priority), and display methods.


File Edit Selection View Go Run ···
ai
task1.py LAB 5_1 U
task22.py U
task3.py U
task4.py U
armstrong_v2.py M
task5.py
lab2-1 > armstrong_v2.py > ...
EXPLORER
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python
powershell
powershell
Ln 55, Col 5 Spaces: 4 UTF-8 CRLF

```python
# Implement a PriorityQueue class using Python's heapq module.
# Requirements:
# enqueue(item, priority)
# dequeue() - return highest priority element
# display()
# Add docstrings and example usage.
import heapq
class PriorityQueue:
    """
    A priority queue implementation using Python's heapq module.

    Methods:
    - enqueue(item, priority): Add an item with a given priority to the queue.
    - dequeue(): Remove and return the item with the highest priority.
    - display(): Display the contents of the priority queue.

    Example usage:
    >>> pq = PriorityQueue()
    >>> pq.enqueue('task1', 2)
    >>> pq.enqueue('task2', 1)
```

```
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
None
Bob
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
Graph representation (adjacency list):
A: ['B', 'C']
B: ['A', 'C']
C: ['A', 'B']
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
[(1, 'task2'), (2, 'task1'), (3, 'task3')]
task2
[(2, 'task1'), (3, 'task3')]
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> 
```

Task Description #8 – Deque

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

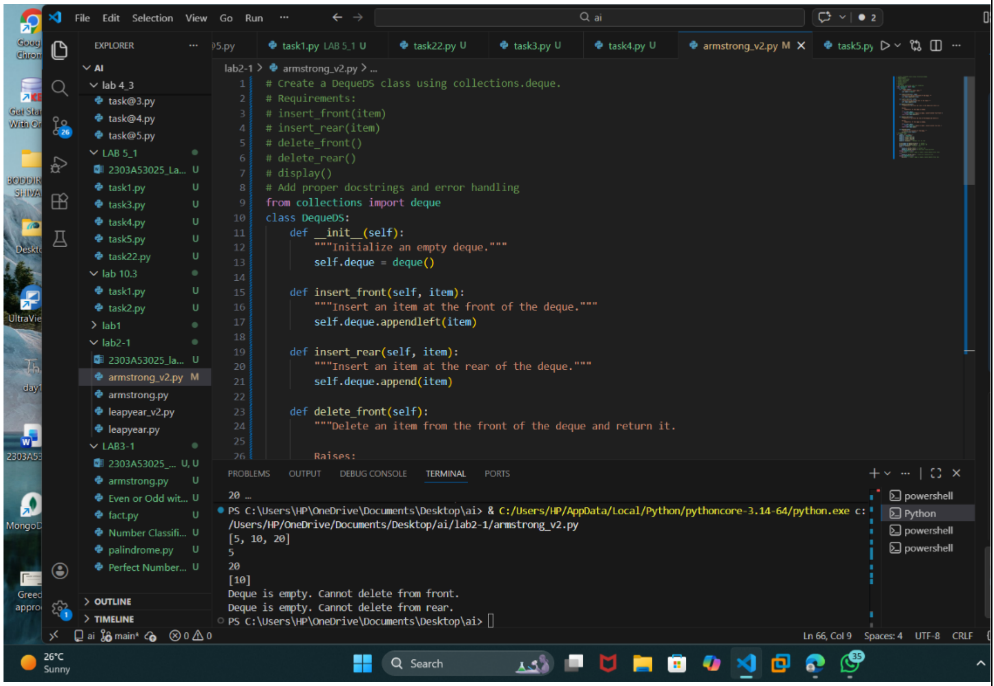
Sample Input Code:

class DequeDS:

    pass

Expected Output:

- Insert and remove from both ends with docstrings.


File Edit Selection View Go Run ···
ai
task1.py LAB 5_1 U
task22.py U
task3.py U
task4.py U
armstrong_v2.py M
task5.py
lab2-1 > armstrong_v2.py > ...
EXPLORER
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
powershell
Python
powershell
powershell
Ln 66, Col 9 Spaces: 4 UTF-8 CRLF

```python
# Create a DequeDS class using collections.deque.
# Requirements:
# insert_front(item)
# insert_rear(item)
# delete_front()
# delete_rear()
# display()
# Add proper docstrings and error handling
from collections import deque
class DequeDS:
    def __init__(self):
        """Initialize an empty deque."""
        self.deque = deque()

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

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

    def delete_front(self):
        """Delete an item from the front of the deque and return it.

        Raises:
```

```
20 …
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe c:
/Users/HP/OneDrive/Documents/Desktop/ai/lab2-1/armstrong_v2.py
[5, 10, 20]
5
20
[10]
Deque is empty. Cannot delete from front.
Deque is empty. Cannot delete from rear.
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> 
```

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:

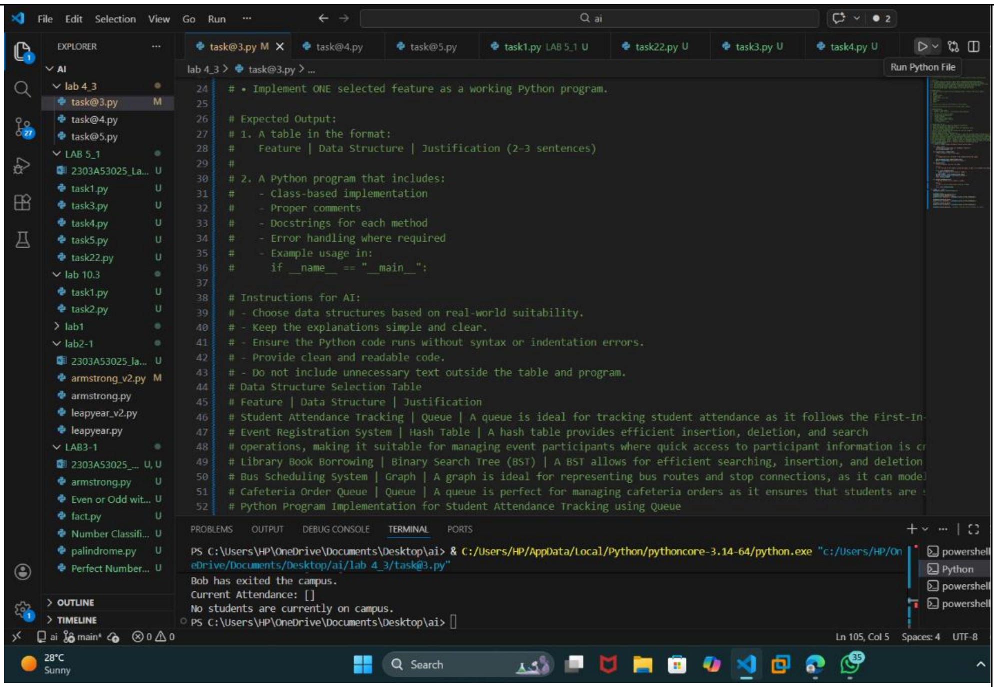
1. Student Attendance Tracking – Daily log of students entering/exiting the campus.
2. Event Registration System – Manage participants in events with quick search and removal.
3. Library Book Borrowing – Keep track of available books and their due dates.
4. Bus Scheduling System – Maintain bus routes 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
    - 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.
- A functional Python program implementing the chosen feature with comments and docstrings.



File Edit Selection View Go Run ···
task@3.py M
task@4.py
task@5.py
task1.py LAB 5_1 U
task22.py U
task3.py U
task4.py U
Run Python File
lab 4_3 > task@3.py > ...
EXPLORER
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Ln 105, Col 5 Spaces: 4 UTF-8

```
# • Implement ONE selected feature as a working Python program.

# Expected Output:
# 1. A table in the format:
#    Feature | Data Structure | Justification (2-3 sentences)
#
# 2. A Python program that includes:
#    - Class-based implementation
#    - Proper comments
#    - Docstrings for each method
#    - Error handling where required
#    - Example usage in:
#      if __name__ == "__main__":

# Instructions for AI:
# - Choose data structures based on real-world suitability.
# - Keep the explanations simple and clear.
# - Ensure the Python code runs without syntax or indentation errors.
# - Provide clean and readable code.
# - Do not include unnecessary text outside the table and program.
# Data Structure Selection Table
# Feature | Data Structure | Justification
# Student Attendance Tracking | Queue | A queue is ideal for tracking student attendance as it follows the First-In-
# Event Registration System | Hash Table | A hash table provides efficient insertion, deletion, and search
# operations, making it suitable for managing event participants where quick access to participant information is cr
# Library Book Borrowing | Binary Search Tree (BST) | A BST allows for efficient searching, insertion, and deletion
# Bus Scheduling System | Graph | A graph is ideal for representing bus routes and stop connections, as it can model
# Cafeteria Order Queue | Queue | A queue is perfect for managing cafeteria orders as it ensures that students are
# Python Program Implementation for Student Attendance Tracking using Queue
```

```
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/HP/On
eDrive/Documents/Desktop/ai/lab 4_3/task@3.py"
Bob has exited the campus.
Current Attendance: []
No students are currently on campus.
PS C:\Users\HP\OneDrive\Documents\Desktop\ai>
```

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:

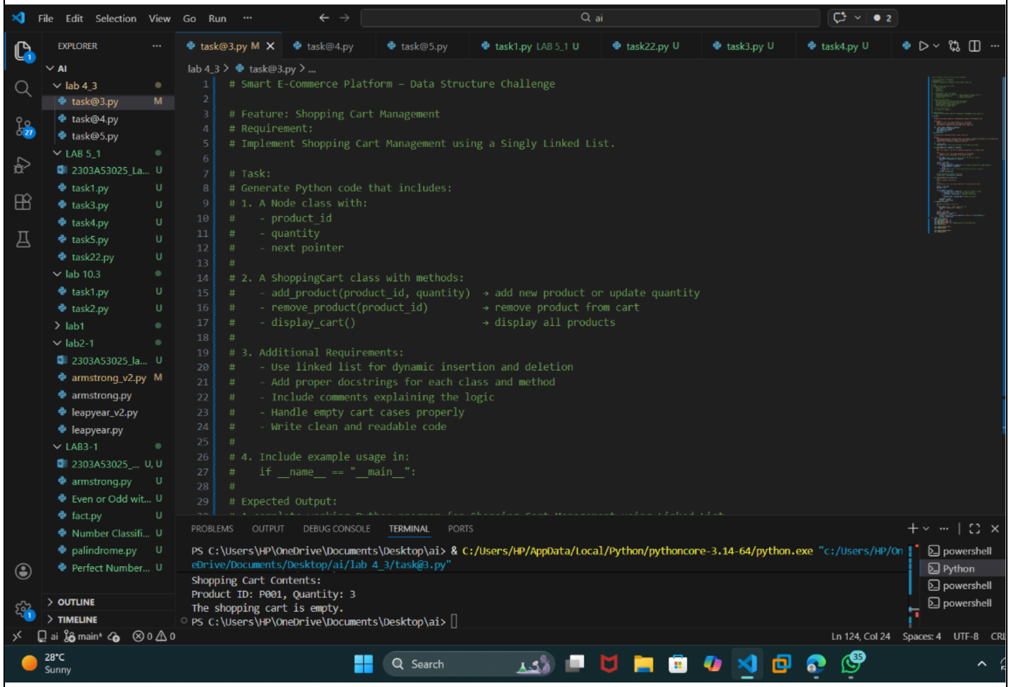
- For each feature, select the most appropriate data structure from the list below:
    - Stack
    - Queue
    - Priority Queue
    - Linked List

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



```python
# Smart E-Commerce Platform – Data Structure Challenge

# Feature: Shopping Cart Management
# Requirement:
# Implement Shopping Cart Management using a Singly Linked List.

# Task:
# Generate Python code that includes:
# 1. A Node class with:
#     - product_id
#     - quantity
#     - next pointer
#
# 2. A ShoppingCart class with methods:
#     - add_product(product_id, quantity)  → add new product or update quantity
#     - remove_product(product_id)         → remove product from cart
#     - display_cart()                     → display all products
#
# 3. Additional Requirements:
#     - Use linked list for dynamic insertion and deletion
#     - Add proper docstrings for each class and method
#     - Include comments explaining the logic
#     - Handle empty cart cases properly
#     - Write clean and readable code
#
# 4. Include example usage in:
#     if __name__ == "__main__":
#
# Expected Output:
```

```
PS C:\Users\HP\OneDrive\Documents\Desktop\ai> & C:/Users/HP/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/HP/OneDrive/Documents/Desktop/ai/lab 4_3/task@3.py"
Shopping Cart Contents:
Product ID: P001, Quantity: 3
The shopping cart is empty.
PS C:\Users\HP\OneDrive\Documents\Desktop\ai>
```