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		Venkataramana Veeramsetty	
Instructor(s) Name		Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)	
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week6 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber: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 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.	Week6 - Monday

- 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
```

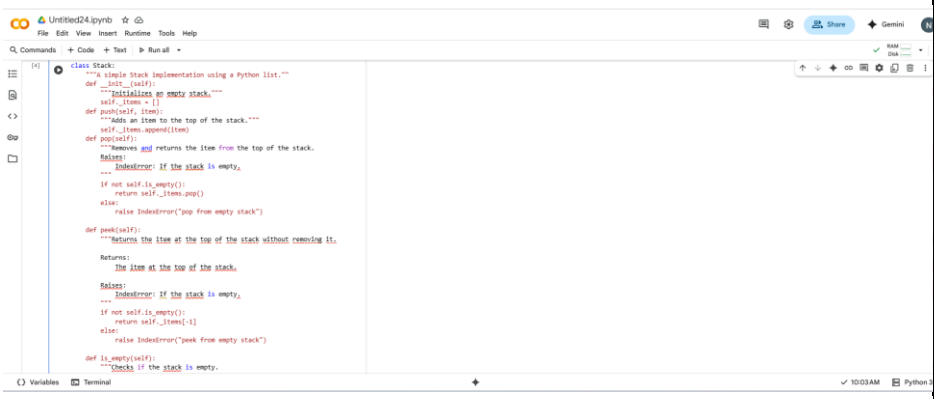
Expected Output:

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

PROMPT1:write a python code of class name Stack .The class should include the push.pop,peek and is_empty.

PROMPT2:Each method should include with a docstring of explaining its functionalities

Code :



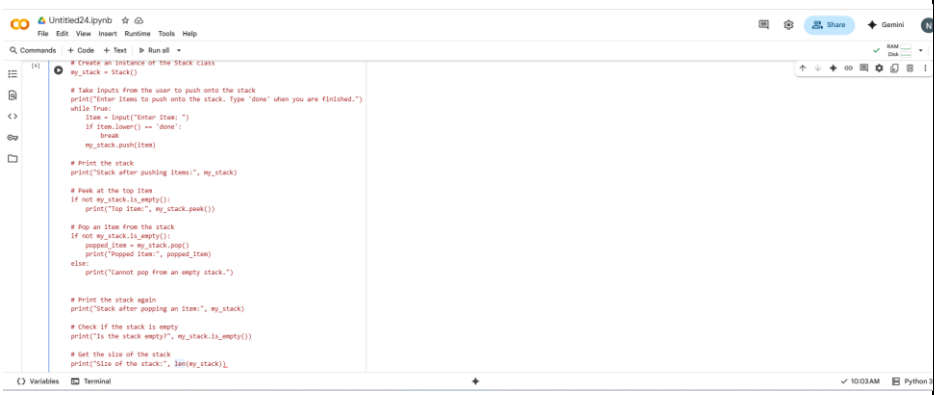
```
class Stack:
    """A simple Stack implementation using a Python list."""
    def __init__(self):
        """Initializes an empty stack."""
        self._items = []
    def push(self, item):
        """Adds an item to the top of the stack."""
        self._items.append(item)
    def pop(self):
        """Removes and returns the item from the top of the stack.
        Raises:
            IndexError: If the stack is empty.
        """
        if not self.is_empty():
            return self._items.pop()
        else:
            raise IndexError("pop from empty stack")

    def peek(self):
        """Returns 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 not self.is_empty():
            return self._items[-1]
        else:
            raise IndexError("peek from empty stack")

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



```
# Create an instance of the Stack class
my_stack = Stack()

# Take inputs from the user to push onto the stack
print("Enter items to push onto the stack. Type 'done' when you are finished.")
while True:
    item = input("Enter item: ")
    if item.lower() == 'done':
        break
    my_stack.push(item)

# Print the stack
print("Stack after pushing items:", my_stack)

# Peek at the top item
if not my_stack.is_empty():
    print("Top item:", my_stack.peek())

# Pop an item from the stack
if not my_stack.is_empty():
    popped_item = my_stack.pop()
    print("Popped item:", popped_item)
else:
    print("Cannot pop from an empty stack.")


# Print the stack again
print("Stack after popping an item:", my_stack)

# Check if the stack is empty
print("Is the stack empty?", my_stack.is_empty())

# Get the size of the stack
print("Size of the stack:", len(my_stack))
```

OUTPUT:

```
Enter items to push onto the stack. Type 'done' when you are finished.
Enter item: 8
Enter item: 12
Enter item: 85
Enter item: 3
Enter item: 1
Enter item: done
Stack after pushing items: ['8', '12', '85', '3', '1']
Top item: 1
Popped item: 1
Stack after popping an item: ['8', '12', '85', '3']
Is the stack empty? False
Size of the stack: 4
```

OBSERVATION:

The above is about Stack Implementation .It is Last In first Out (LIFO). The class should be named as Stack and it includes the method names ast the push() ,pop() ,peek() ,is_empty().it is used for pop means remove the last index value.is_empty checks if an list doesnot contain any numbers then it is true or else it is false.push method means it will add the number and peek method means it will just return the last index value but it wont remove just,it show the value.

Task Description #2 – Queue Implementation

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

Sample Input Code:

```
class Queue:
    pass
```

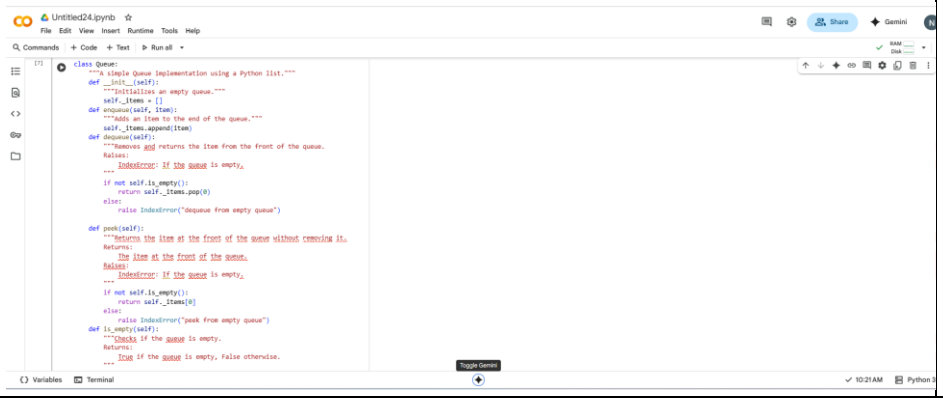
Expected Output:

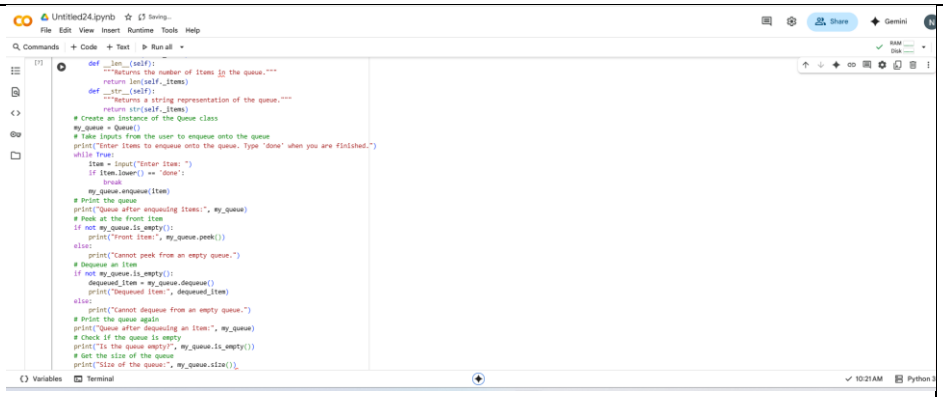
- FIFO-based queue class with enqueue, dequeue, peek, and size methods.

PROMPT1:write a python code of class name as Queue and in that method we should take enqueue,dequeue,peek,size

PROMPT2: Each method should include with a docstring of explaining its functionalities

Code:





```
def __len__(self):
    """Returns the number of items in the queue."""
    return len(self._items)
def __str__(self):
    """Returns a string representation of the queue."""
    return str(self._items)
# Create an instance of the Queue class
my_queue = Queue()
# Take inputs from the user to enqueue onto the queue
print("Enter items to enqueue onto the queue. Type 'done' when you are finished.")
while True:
    item = input("Enter item: ")
    if item.lower() == 'done':
        break
    my_queue.enqueue(item)
# Print the queue
print("Queue after enqueuing items:", my_queue)
# Peek at the front item
if not my_queue.is_empty():
    print("Front item:", my_queue.peek())
else:
    print("Cannot peek from an empty queue.")
# Dequeue an item
if not my_queue.is_empty():
    dequeued_item = my_queue.dequeue()
    print("Dequeued item:", dequeued_item)
else:
    print("Cannot dequeue from an empty queue.")
# Print the queue again
print("Queue after dequeuing an item:", my_queue)
# Check if the queue is empty
print("Is the queue empty?", my_queue.is_empty())
# Get the size of the queue
print("Size of the queue:", my_queue.size())
```

OUTPUT:

```
Enter items to enqueue onto the queue. Type 'done' when you are finished.
Enter item: 12
Enter item: 32
Enter item: 2
Enter item: 42
Enter item: done
Queue after enqueuing items: ['12', '32', '2', '42']
Front item: 12
Dequeued item: 12
Queue after dequeuing an item: ['32', '2', '42']
Is the queue empty? False
Size of the queue: 3
```

OBSERVATION:

In the above python code what is observed is , it is for QUEUE Implementation that means First-In ,First-Out (FIFIO).enqueue method is for adding the elements in the queue.dequeue() method is for remove and also returns the first index of element.peek() method is is will not remove and returns the first index of element.size() this method is for returns the number of elements in the queue.

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:

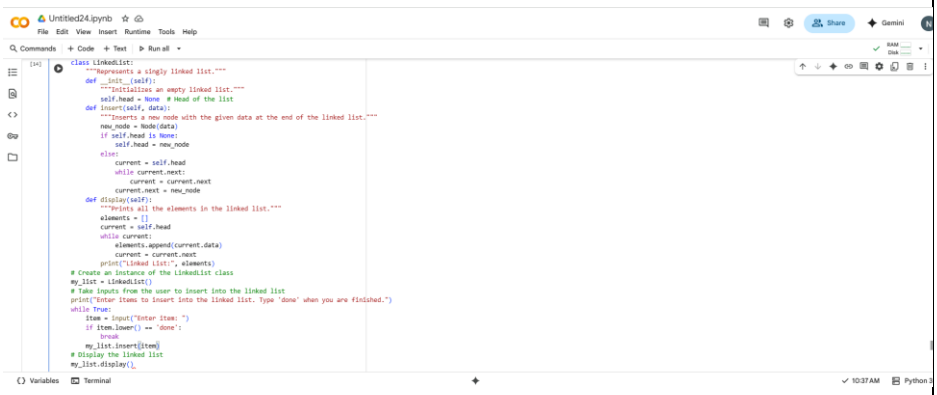
- A working linked list implementation with clear method documentation.

PROMPT1:Write the python code to create two class names Node and LinkedList. In the LinkedList class it should have the two methods insert,display.insert() to add a new node at the end, and display() to print all elements.

PROMPT2: Each method should include with a docstring of explaining its functionalities

PROMPT3:Take input from the user

CODE:



OUTPUT:

```
Enter items to insert into the linked list. Type 'done' when you are finished.
Enter item: 21
Enter item: 3
Enter item: 12
Enter item: 42
Enter item: 2
Enter item: done
Linked List: ['21', '3', '12', '42', '2']
```

OBSERVATION:

From the above code I observed that insert() method is used to add nodes with the given data to the end of the list and display() method is to print (or) also we can say to return the list of elements in order.The above task is about LINKED LIST.And also we creating the 2 classes one if Node and the other is LinkedList

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:

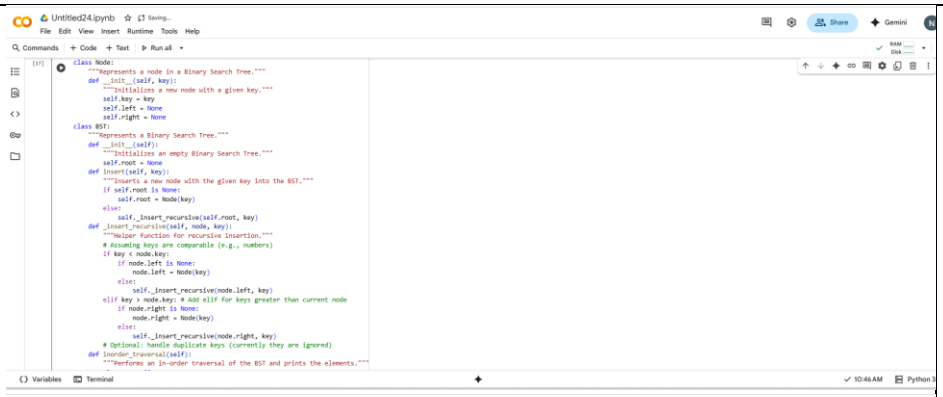
- BST implementation with recursive insert and traversal methods.

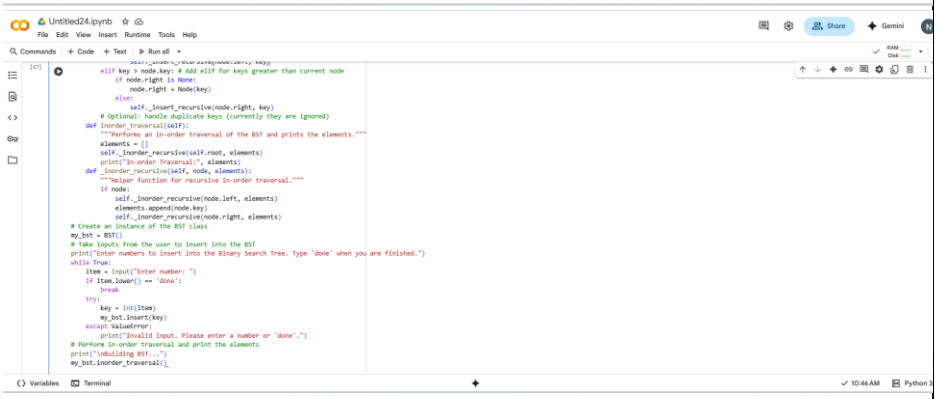
PROMPT1: Write Python code to implement a Binary Search Tree (BST). Create a class BST with methods insert()and inorder_traversal().

PROMPT2: Each method should include with a docstring of explaining its functionalities

PROMPT3:Take input from the user

Code:





OUTPUT:

```
Enter numbers to insert into the Binary Search Tree. Type 'done' when you are finished.
Enter number: 12
Enter number: 3
Enter number: 45
Enter number: 32
Enter number: 45
Enter number: 65
Enter number: done

Building BST...
In-order Traversal: [3, 12, 32, 45, 65]
```

OBSERVATION:

From the above that it is about Binary Search Tree (BST). The class we written as BST .and in the class BST the method we written is insert() and insert_ traversal().insert() method is to add nodes and insert_traversal() method is about the sorting the list of values .

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.

PROMPT1:Write Python code to od class Hash Table class with chaining for collision handling. The class should includes the methods insert(), search(),

	and delete(). Use Python lists for buckets and linked lists (or sub-lists) for handling collisions PROMPT2: Each method should include with a docstring of explaining its functionalities PROMPT3:Take input from the user	
	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.	
	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.	
	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.	
	Task Description #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 Expected Output: • A markdown table with structure names, operations, and complexities.	
	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.

Expected Output:

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

✅ Deliverables (For All Tasks)

1. AI-generated prompts for code and test case generation.
2. At least 3 assert test cases for each task.
3. AI-generated initial code and execution screenshots.
4. Analysis of whether code passes all tests.
5. Improved final version with inline comments and explanation.
6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.