

<b>SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE</b>		<b>DEPARTMENT OF COMPUTER SCIENCE ENGINEERING</b>	
<b>Program Name:</b> B. Tech		<b>Assignment Type:</b> Lab	
<b>Course Coordinator Name</b>		Venkataramana Veeramsetty	
<b>Instructor(s) Name</b>		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)	
<b>Course Code</b>	24CS002PC215	<b>Course Title</b>	AI Assisted Coding
<b>Year/Sem</b>	II/I	<b>Regulation</b>	R24
<b>Date and Day of Assignment</b>	Week6 - Monday	<b>Time(s)</b>	
<b>Duration</b>	2 Hours	<b>Applicable to Batches</b>	
<b>AssignmentNumber:</b> 11.1(Present assignment number)/ <b>24</b> (Total number of assignments)			

<b>Q.No.</b>	<b>Question</b>	<b>Expected Time to complete</b>
1	<b>Lab 11 – Data Structures with AI: Implementing Fundamental Structures</b> <b>Lab Objectives</b> <ul style="list-style-type: none"> <li>• Use AI to assist in designing and implementing fundamental data structures in Python.</li> </ul>	Week6 - Monday

	<ul style="list-style-type: none"><li>• Learn how to prompt AI for structure creation, optimization, and documentation.</li><li>• Improve understanding of Lists, Stacks, Queues, Linked Lists, Trees, Graphs, and Hash Tables.</li><li>• Enhance code quality with AI-generated comments and performance suggestions.</li></ul>	
	<p><b>Task Description #1 – Stack Implementation</b></p> <p>Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.</p> <p>Sample Input Code:</p> <pre>class Stack:     pass</pre> <p>Expected Output:</p> <ul style="list-style-type: none"><li>• A functional stack implementation with all required methods and docstrings.</li></ul> <p><b>Prompt:</b></p> <p>Generate a Python program that implements a Stack class with push, pop, peek, and is_empty methods and docstrings.</p> <p><b>Code:</b></p>	

The screenshot shows a Jupyter Notebook interface with a single code cell containing Python code for a `Stack` class. The notebook title is `Untitled22.ipynb`. The code defines a `Stack` class with methods `push`, `pop`, and `peek`, each with detailed docstrings.

```
class Stack:
    """A simple Stack class implementation."""

    def __init__(self):
        """Initializes an empty stack."""
        self._items = []

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

        Args:
            item: The item to be added.
        """
        self._items.append(item)

    def pop(self):
        """
        Removes and returns the item from the top of the stack.

        Returns:
            The item removed from the stack.
        """
        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.
        """
        if not self.is_empty():
            return self._items[-1]
        else:
            raise IndexError("peek from empty stack")
```

The screenshot shows a Jupyter Notebook interface with a single code cell. The cell contains Python code for a stack class. The code includes docstrings for the class and its methods, and handles user input for operations like Push, Pop, Peek, and Check if empty.

```
Untitled22.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

[7] ✓ lm

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.

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

my_stack = Stack()

while True:
    print("\nChoose an operation:")
    print("1. Push")
    print("2. Pop")
    print("3. Peek")
    print("4. Check if empty")
    print("5. Exit")

    choice = input("Enter your choice (1-5): ")

    if choice == '1':
        item = input("Enter the item to push: ")
        my_stack.push(item)
        print(f"{item} pushed onto the stack.")
    elif choice == '2':
        try:
            item = my_stack.pop()
            print(f"Popped item: {item}")
        except IndexError as e:
            print(e)
    elif choice == '3':
        try:
            item = my_stack.peek()
            print(f"Top item: {item}")
        except IndexError as e:
            print(e)
```

```

Untitled22.ipynb  ⌂ ⌂
File Edit View Insert Runtime Tools Help
Commands | + Code + Text | ▾ Run all ▾
[7] ✓ In
    elif choice == '3':
        try:
            item = my_stack.peek()
            print(f"Top item: {item}")
        except IndexError as e:
            print(e)
    elif choice == '4':
        if my_stack.is_empty():
            print("The stack is empty.")
        else:
            print("The stack is not empty.")
    elif choice == '5':
        print("Exiting.")
        break
    else:
        print("Invalid choice. Please enter a number between 1 and 5.")

Choose an operation:
1. Push
2. Pop
3. Peek
4. Check if empty
5. Exit
Enter your choice (1-5): 1
Enter the items to push: 12
12 pushed onto the stack.

Choose an operation:
1. Push
2. Pop
3. Peek
4. Check if empty
5. Exit
Enter your choice (1-5): 2
Popped item: 12

Choose an operation:
1. Push
2. Pop
3. Peek
4. Check if empty
5. Exit
Enter your choice (1-5): 3
peek from empty stack

Choose an operation:
1. Push
2. Pop
3. Peek
4. Check if empty
5. Exit
Enter your choice (1-5): 4
The stack is empty.

Choose an operation:
1. Push
2. Pop
3. Peek
4. Check if empty
5. Exit
Enter your choice (1-5): 5
Exiting.

```

### Code Explanation:

**Class Definition** – A Stack class is created to represent the stack data structure.

**Push Method** – Adds an element to the top of the stack using append().

**Pop Method** – Removes and returns the top element using pop(). If empty, it shows a warning.

**Peek Method** – Returns the top element without removing it. If empty, it shows a warning.

**is\_empty Method** – Checks whether the stack has no elements and returns True or False.

---

### **Task Description #2 – Queue Implementation**

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

Sample Input Code:

```
class Queue:
```

```
    pass
```

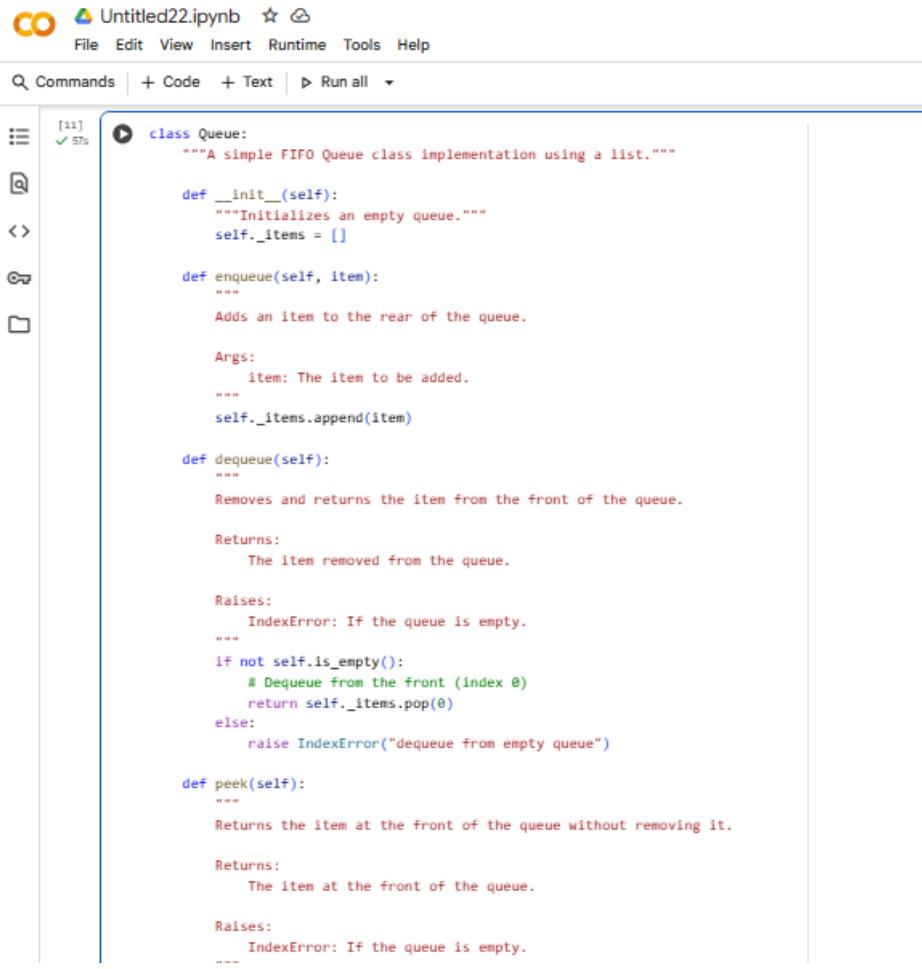
Expected Output:

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

**Prompt:**

Generate a Python class to implement a Queue using lists with enqueue, dequeue, peek, size, and is\_empty methods.

**Code:**



The screenshot shows a Jupyter Notebook cell with the following code:

```
[11]: class Queue:
    """A simple FIFO Queue class implementation using a list."""

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

    def enqueue(self, item):
        """
        Adds an item to the rear of the queue.

        Args:
            item: The item to be added.
        """
        self._items.append(item)

    def dequeue(self):
        """
        Removes and returns the item from the front of the queue.

        Returns:
            The item removed from the queue.
        """
        if not self.is_empty():
            # Dequeue from the front (index 0)
            return self._items.pop(0)
        else:
            raise IndexError("dequeue from empty queue")

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

        Returns:
            The item at the front of the queue.
        """
        if not self.is_empty():
            return self._items[0]
        else:
            raise IndexError("peek from empty queue")

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

Untitled22.ipynb

```
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

[11] ✓ 57s
if not self.is_empty():
    return self._items[@]
else:
    raise IndexError("peek from empty queue")

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

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

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

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

# Create a Queue instance
my_queue = Queue()

while True:
    print("\nChoose an operation:")
    print("1. Enqueue")
    print("2. Dequeue")
    print("3. Peek")
    print("4. Check if empty")
    print("5. Get size")
    print("6. Exit")

    choice = input("Enter your choice (1-6): ")

    if choice == '1':
        item = input("Enter the item to enqueue: ")
        my_queue.enqueue(item)
        print(f"{item} enqueued onto the queue.")
    elif choice == '2':
        try:
            item = my_queue.dequeue()
            print(f"Dequeued item: {item}")
        except IndexError as e:
            print(e)
    elif choice == '3':
        try:
            item = my_queue.peek()
            print(f"Front item: {item}")
        except IndexError as e:
            print(e)
    elif choice == '4':
        if my_queue.is_empty():
            print("The queue is empty.")
        else:
            print("The queue is not empty.")
    elif choice == '5':
        print(f"Queue size: {my_queue.size()}")
    elif choice == '6':
        print("Exiting.")
        break
    else:
        print("Invalid choice. Please enter a number between 1 and 6.")
```

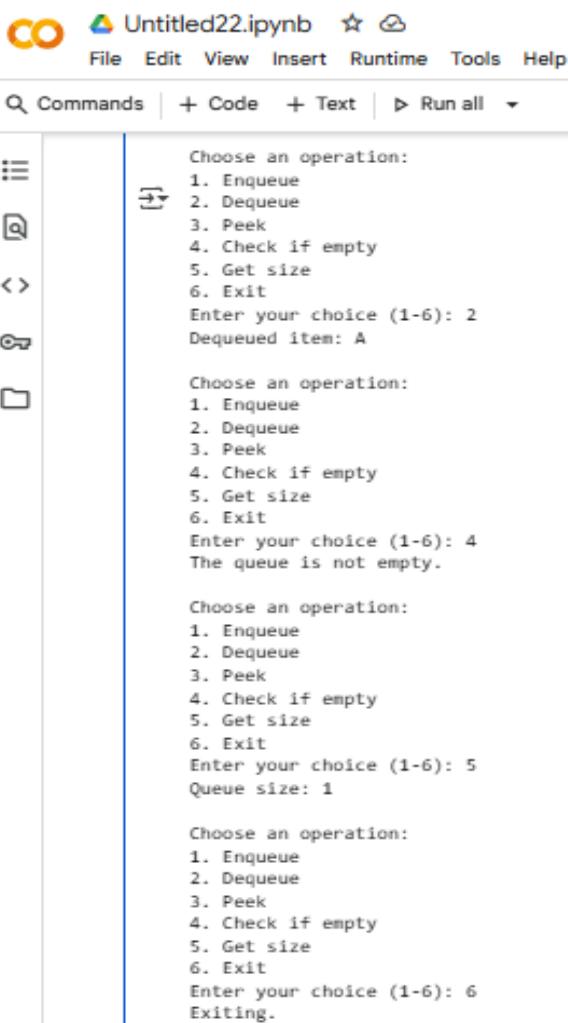
Untitled22.ipynb

```
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

[11] ✓ 57s
    item = my_queue.dequeue()
    print(f"Dequeued item: {item}")
except IndexError as e:
    print(e)
elif choice == '3':
    try:
        item = my_queue.peek()
        print(f"Front item: {item}")
    except IndexError as e:
        print(e)
elif choice == '4':
    if my_queue.is_empty():
        print("The queue is empty.")
    else:
        print("The queue is not empty.")
elif choice == '5':
    print(f"Queue size: {my_queue.size()}")
elif choice == '6':
    print("Exiting.")
    break
else:
    print("Invalid choice. Please enter a number between 1 and 6.")

2. Dequeue
3. Peek
4. Check if empty
5. Get size
6. Exit
Enter your choice (1-6): 1
Enter the item to enqueue: B
B enqueued onto the queue.

Choose an operation:
1. Enqueue
2. Dequeue
3. Peek
4. Check if empty
5. Get size
6. Exit
Enter your choice (1-6): 3
Front item: A
```



```

CO Untitled22.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

Choose an operation:
1. Enqueue
2. Dequeue
3. Peek
4. Check if empty
5. Get size
6. Exit
Enter your choice (1-6): 2
Dequeued item: A

Choose an operation:
1. Enqueue
2. Dequeue
3. Peek
4. Check if empty
5. Get size
6. Exit
Enter your choice (1-6): 4
The queue is not empty.

Choose an operation:
1. Enqueue
2. Dequeue
3. Peek
4. Check if empty
5. Get size
6. Exit
Enter your choice (1-6): 5
Queue size: 1

Choose an operation:
1. Enqueue
2. Dequeue
3. Peek
4. Check if empty
5. Get size
6. Exit
Enter your choice (1-6): 6
Exiting.

```

### Code Explanation:

**Class Definition** – A Queue class is created to represent the FIFO queue..

**Enqueue ()** – Adds an element to the end of the queue using append().

**Dequeue ()** – Removes and returns the first element using pop(0). If empty, it shows a warning.

**Peek ()** – Returns the first element without removing it. If empty, it shows a warning.

**Size ()** – Returns the total number of elements currently in the queue.

**is\_empty()** – Checks whether the queue has no elements and returns True or False.

### **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
```

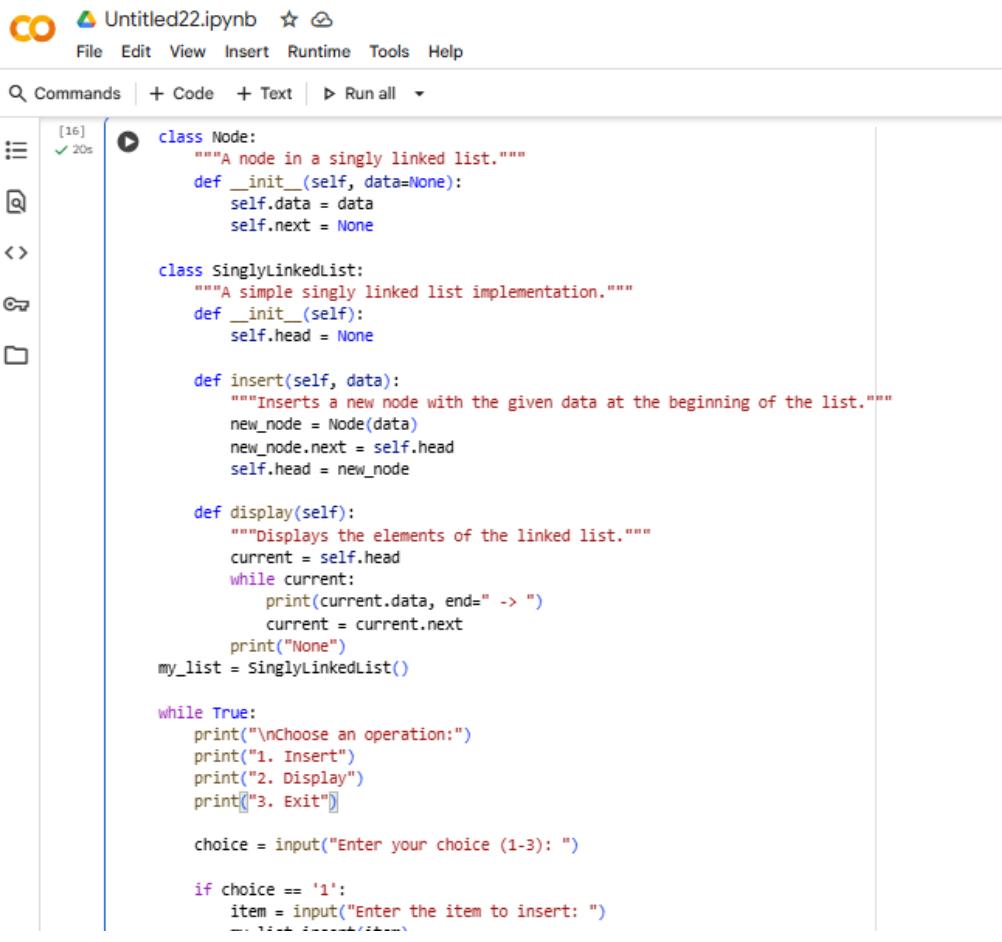
Expected Output:

- A working linked list implementation with clear method documentation.

**Prompt:**

Generate a Python program to implement a Singly Linked List with insert and display methods, including docstrings.

**Code:**



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Untitled22.ipynb
- Toolbar:** File, Edit, View, Insert, Runtime, Tools, Help
- Search Bar:** Commands | + Code | + Text | ▶ Run all ▾
- Code Cell:** Contains Python code for a singly linked list. The code defines a `Node` class and a `SinglyLinkedList` class. It includes methods for inserting nodes at the beginning and displaying the list. A user interaction loop is also present.

```
[16] ✓ 20s
class Node:
    """A node in a singly linked list."""
    def __init__(self, data=None):
        self.data = data
        self.next = None

class SinglyLinkedList:
    """A simple singly linked list implementation."""
    def __init__(self):
        self.head = None

    def insert(self, data):
        """Inserts a new node with the given data at the beginning of the list."""
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def display(self):
        """Displays the elements of the linked list."""
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")
my_list = SinglyLinkedList()

while True:
    print("\nChoose an operation:")
    print("1. Insert")
    print("2. Display")
    print("3. Exit")

    choice = input("Enter your choice (1-3): ")

    if choice == '1':
        item = input("Enter the item to insert: ")
```

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code for a linked list. The code defines a `Node` class with `data` and `next` attributes, and a `LinkedList` class with a `head` pointer. It includes methods for inserting nodes at the end of the list and displaying the list's values. The code is run three times, each time prompting the user to choose an operation (1. Insert, 2. Display, 3. Exit) and providing feedback on the list state.

```
my_list.insert(item)
print(f"{item} inserted into the list.")

elif choice == '2':
    print("Linked List:")
    my_list.display()

elif choice == '3':
    print("Exiting.")
    break

else:
    print("Invalid choice. Please enter a number between 1 and 3.")

Choose an operation:
1. Insert
2. Display
3. Exit
Enter your choice (1-3): 1
Enter the item to insert: 20
20 inserted into the list.

Choose an operation:
1. Insert
2. Display
3. Exit
Enter your choice (1-3): 1
Enter the item to insert: 30
30 inserted into the list.

Choose an operation:
1. Insert
2. Display
3. Exit
Enter your choice (1-3): 2
Linked List:
30 -> 20 -> None
```

### Code Explanation:

**Node class** – Defines each element of the linked list with `data` (value) and `next` (pointer to next node).

**LinkedList class** – Manages the list using the `head` pointer that refers to the first node.

**insert(value)** – Creates a new node and appends it at the end of the list; if the list is empty, the new node becomes the head.

**Display ()** – Traverses from head to tail, collects node values in a list, and prints them in a readable format

#### 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
```

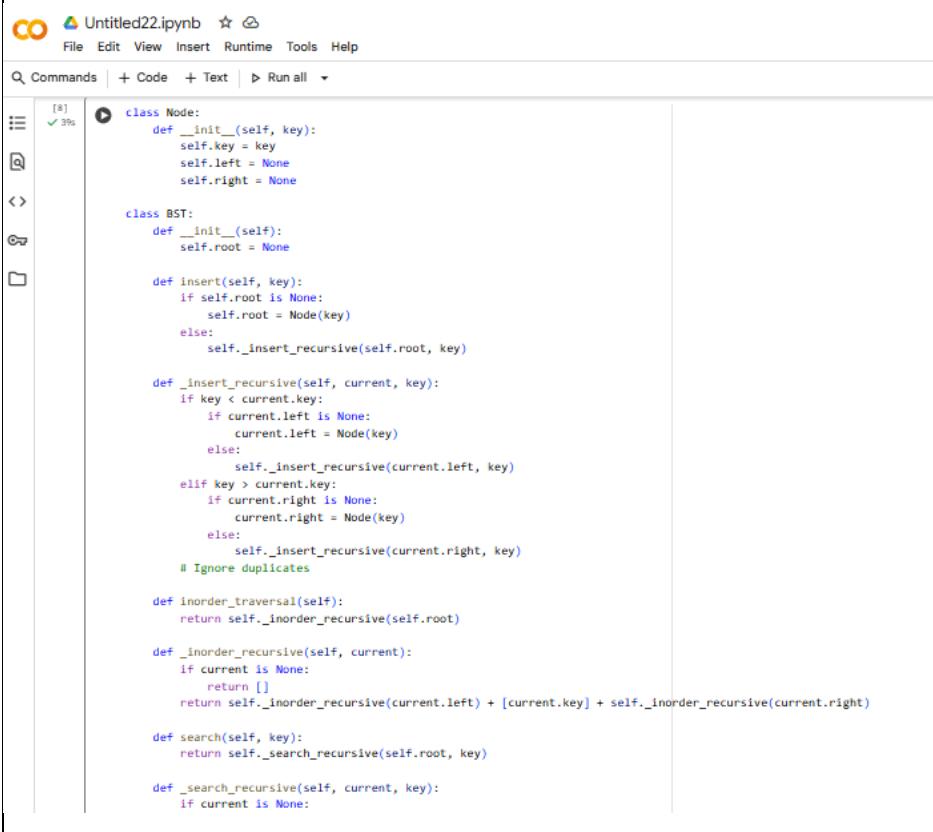
Expected Output:

- BST implementation with recursive insert and traversal methods.

**Prompt:**

Generate a Python program to implement a Binary Search Tree (BST) with recursive insert and in-order traversal methods.

**Code:**



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

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

    def insert(self, key):
        if self.root is None:
            self.root = Node(key)
        else:
            self._insert_recursive(self.root, key)

    def _insert_recursive(self, current, key):
        if key < current.key:
            if current.left is None:
                current.left = Node(key)
            else:
                self._insert_recursive(current.left, key)
        elif key > current.key:
            if current.right is None:
                current.right = Node(key)
            else:
                self._insert_recursive(current.right, key)
        # Ignore duplicates

    def inorder_traversal(self):
        return self._inorder_recursive(self.root)

    def _inorder_recursive(self, current):
        if current is None:
            return []
        return self._inorder_recursive(current.left) + [current.key] + self._inorder_recursive(current.right)

    def search(self, key):
        return self._search_recursive(self.root, key)

    def _search_recursive(self, current, key):
        if current is None:
            return None
        if current.key == key:
            return current
        if current.key < key:
            return self._search_recursive(current.right, key)
        else:
            return self._search_recursive(current.left, key)
```

```

Untitled22.ipynb  ⭐ ⌂
File Edit View Insert Runtime Tools Help
Commands | + Code | + Text | ▶ Run all ▾
[8] ❸
    elif key < current.key:
        return self._search_recursive(current.left, key)
    else:
        return self._search_recursive(current.right, key)
# ----- Menu Driven Program -----
bst = BST()
while True:
    print("\n---- Binary Search Tree Menu ----")
    print("1. Insert")
    print("2. In-order Traversal")
    print("3. Search")
    print("4. Exit")
    choice = input("Enter your choice: ")
    if choice == '1':
        while True:
            val_str = input("Enter value to insert: ")
            try:
                val = int(val_str)
                bst.insert(val)
                print(f"{val} inserted into BST.")
                break
            except ValueError:
                print("Invalid input. Please enter an integer.")
    elif choice == '2':
        print("In-order Traversal:", bst.inorder_traversal())
    elif choice == '3':
        while True:
            val_str = input("Enter value to search: ")
            try:
                val = int(val_str)
                if bst.search(val):
                    print(f"{val} found in BST.")
                else:
                    print(f"{val} not found in BST.")
                break
            except ValueError:
                print("Invalid input. Please enter an integer.")
    elif choice == '4':
        print("Exiting program...")
        break
    else:
        print("Invalid choice! Please try again.")

```

---

```

Untitled22.ipynb  ⭐ ⌂
File Edit View Insert Runtime Tools Help
Commands | + Code | + Text | ▶ Run all ▾
[8] ❸
    print("Invalid choice! Please try again.")

---- Binary Search Tree Menu ----
1. Insert
2. In-order Traversal
3. Search
4. Exit
Enter your choice: 1
Enter value to insert: 30
30 inserted into BST.

---- Binary Search Tree Menu ----
1. Insert
2. In-order Traversal
3. Search
4. Exit
Enter your choice: 1
Enter value to insert: 10
10 inserted into BST.

---- Binary Search Tree Menu ----
1. Insert
2. In-order Traversal
3. Search
4. Exit
Enter your choice: 2
In-order Traversal: [10, 30]

---- Binary Search Tree Menu ----
1. Insert
2. In-order Traversal
3. Search
4. Exit
Enter your choice: 3
Enter value to search: 30
30 found in BST.

---- Binary Search Tree Menu ----
1. Insert
2. In-order Traversal
3. Search
4. Exit
Enter your choice: 4
Exiting program...

```

### Code Explanation:

**Node class** – Defines a tree node with `key`, `left`, and `right` attributes.

**BST class** – Manages the Binary Search Tree with a `root` pointer.

**insert(key)** – Public method that calls a recursive helper to place the new key in the correct position.

**insert\_recursive(node, key)** – Recursively finds the correct spot: smaller values go left, larger values go right.

**inorder\_traversal()** – Returns the elements of the BST in ascending order using recursion.

### **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`

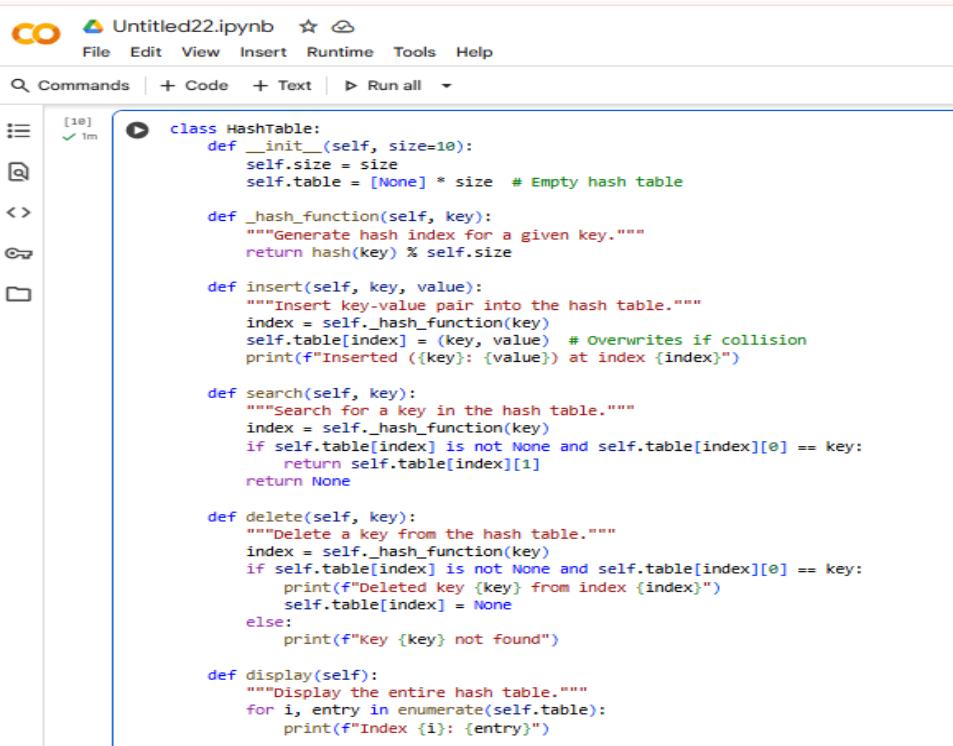
Expected Output:

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

### Prompt:

Generate a Python program to implement a Hash Table with insert, search, and delete methods using chaining for collision handling, including docstrings.

### Code:



The screenshot shows a Jupyter Notebook interface with the following details:

- File:** Untitled22.ipynb
- Cells:** [1e] (1m) - One cell is currently active, indicated by a blue border.
- Code Content:**

```
class HashTable:  
    def __init__(self, size=10):  
        self.size = size  
        self.table = [None] * size # Empty hash table  
  
    def _hash_function(self, key):  
        """Generate hash index for a given key."""  
        return hash(key) % self.size  
  
    def insert(self, key, value):  
        """Insert key-value pair into the hash table."""  
        index = self._hash_function(key)  
        self.table[index] = (key, value) # Overwrites if collision  
        print(f"Inserted ({key}: {value}) at index {index}")  
  
    def search(self, key):  
        """Search for a key in the hash table."""  
        index = self._hash_function(key)  
        if self.table[index] is not None and self.table[index][0] == key:  
            return self.table[index][1]  
        return None  
  
    def delete(self, key):  
        """Delete a key from the hash table."""  
        index = self._hash_function(key)  
        if self.table[index] is not None and self.table[index][0] == key:  
            print(f"Deleted key {key} from index {index}")  
            self.table[index] = None  
        else:  
            print(f"Key {key} not found")  
  
    def display(self):  
        """Display the entire hash table."""  
        for i, entry in enumerate(self.table):  
            print(f"Index {i}: {entry}")
```

```

Untitled22.ipynb  ⭐  ⚙
File Edit View Insert Runtime Tools Help
Commands | + Code + Text | ▶ Run all ▾

[18]  ✓ 1m
# ----- Menu Driven Program -----
size = int(input("Enter size of hash table: "))
ht = HashTable(size)

while True:
    print("\n---- Hash Table Menu ----")
    print("1. Insert")
    print("2. Search")
    print("3. Delete")
    print("4. Display")
    print("5. Exit")

    choice = int(input("Enter your choice: "))

    if choice == 1:
        key = input("Enter key: ")
        value = input("Enter value: ")
        ht.insert(key, value)
    elif choice == 2:
        key = input("Enter key to search: ")
        result = ht.search(key)
        if result is not None:
            print(f"Found: {key} -> {result}")
        else:
            print("Key not found.")
    elif choice == 3:
        key = input("Enter key to delete: ")
        ht.delete(key)
    elif choice == 4:
        ht.display()
    elif choice == 5:
        print("Exiting program...")
        break
    else:
        print("Invalid choice! Try again.")


Untitled22.ipynb  ⭐  ⚙
File Edit View Insert Runtime Tools Help
Commands | + Code + Text | ▶ Run all ▾

1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 1
Enter key: pen
Enter value: 10
Inserted (pen: 10) at index 1

---- Hash Table Menu ----
1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 2
Enter key to search: pen
Found: pen -> 10

---- Hash Table Menu ----
1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 4
Index 0: None
Index 1: ('pen', '10')

---- Hash Table Menu ----
1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 3
Enter key to delete: pen
Deleted key pen from index 1

---- Hash Table Menu ----
1. Insert
2. Search
3. Delete
4. Display
5. Exit
Enter your choice: 5

```

### **Code Explanation:**

**Initialization** – Creates a hash table with a fixed number of buckets, each represented as a list for chaining collisions.

**Hash Function** – Uses Python's built-in hash() and modulo operator to map a key to an index within the table size.

**Insert** – Finds the correct bucket using the hash function; if the key already exists, updates its value, otherwise appends a new (key, value) pair.

**Search** – Looks in the bucket for the given key and returns its value if found; returns None if the key is absent.

**Delete** – Finds the bucket, searches for the key, and removes the key-value pair if it exists; otherwise shows a warning message.

**Display** – Prints the current state of the hash table, showing each index and its chained list of key-value pairs.

### **Task Description #6 – Graph Representation**

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

Sample Input Code:

```
class Graph:
```

```
    pass
```

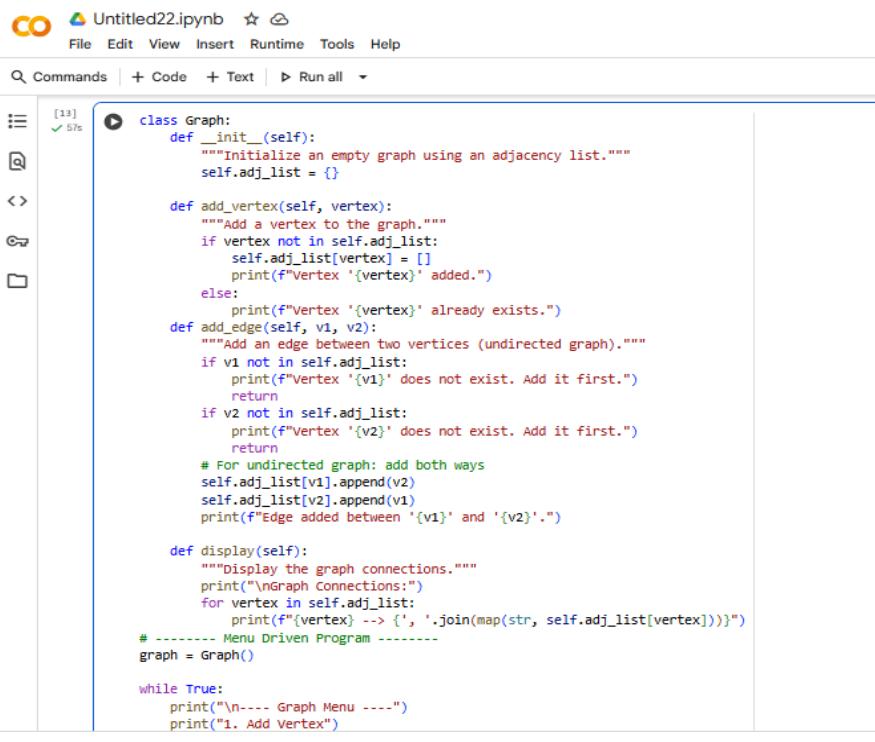
Expected Output:

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

### **Prompt:**

Generate a code to implement a graph using an adjacency list. The graph should have methods to add vertices, add edges, check connections, and display the graph.

### **Code:**



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** CO Untitled22.ipynb
- Toolbar:** File Edit View Insert Runtime Tools Help
- Command Bar:** Q Commands + Code + Text ▶ Run all
- Code Cell:** Contains the Python code for the Graph class implementation.

```
[13] ✓ 57s
class Graph:
    def __init__(self):
        """Initialize an empty graph using an adjacency list."""
        self.adj_list = {}

    def add_vertex(self, vertex):
        """Add a vertex to the graph."""
        if vertex not in self.adj_list:
            self.adj_list[vertex] = []
            print(f"Vertex '{vertex}' added.")
        else:
            print(f"Vertex '{vertex}' already exists.")

    def add_edge(self, v1, v2):
        """Add an edge between two vertices (undirected graph)."""
        if v1 not in self.adj_list:
            print(f"Vertex '{v1}' does not exist. Add it first.")
            return
        if v2 not in self.adj_list:
            print(f"Vertex '{v2}' does not exist. Add it first.")
            return
        # For undirected graph: add both ways
        self.adj_list[v1].append(v2)
        self.adj_list[v2].append(v1)
        print(f"Edge added between '{v1}' and '{v2}'.")

    def display(self):
        """Display the graph connections."""
        print("\nGraph Connections:")
        for vertex in self.adj_list:
            print(f"({vertex}) --> {', '.join(map(str, self.adj_list[vertex]))}")

# ----- Menu Driven Program -----
graph = Graph()

while True:
    print("\n---- Graph Menu ----")
    print("1. Add Vertex")
```

Untitled22.ipynb

```
File Edit View Insert Runtime Tools Help  
Commands + Code + Text ▶ Run all ▾  
[13] ✓ 57s  
print("2. Add Edge")  
print("3. Display Graph")  
print("4. Exit")  
choice = int(input("Enter your choice: "))  
if choice == 1:  
    v = input("Enter vertex name: ")  
    graph.add_vertex(v)  
elif choice == 2:  
    v1 = input("Enter first vertex: ")  
    v2 = input("Enter second vertex: ")  
    graph.add_edge(v1, v2)  
elif choice == 3:  
    graph.display()  
elif choice == 4:  
    print("Exiting program...")  
    break  
else:  
    print("Invalid choice! Try again.")
```

Untitled22.ipynb

```
File Edit View Insert Runtime Tools Help  
Commands + Code + Text ▶ Run all ▾  
[13] ✓ 57s  
---- Graph Menu ----  
1. Add Vertex  
2. Add Edge  
3. Display Graph  
4. Exit  
Enter your choice: 1  
Enter vertex name: A  
Vertex 'A' added.  
  
---- Graph Menu ----  
1. Add Vertex  
2. Add Edge  
3. Display Graph  
4. Exit  
Enter your choice: 1  
Enter vertex name: B  
Vertex 'B' added.  
  
---- Graph Menu ----  
1. Add Vertex  
2. Add Edge  
3. Display Graph  
4. Exit  
Enter your choice: 2  
Enter first vertex: A  
Enter second vertex: B  
Edge added between 'A' and 'B'.  
  
---- Graph Menu ----  
1. Add Vertex  
2. Add Edge  
3. Display Graph  
4. Exit  
Enter your choice: 3  
  
Graph Connections:  
A --> B  
B --> A  
  
---- Graph Menu ----  
1. Add Vertex  
2. Add Edge  
3. Display Graph  
4. Exit  
Enter your choice: 4  
Exiting program...
```

( ) Variables Terminal

### **Code Explanation:**

1. Graph is stored as a dictionary.
- 2.**.add\_vertex(v)** → Adds a new vertex if it doesn't exist.
- 3.**.add\_edge(u, v)** → Adds an edge between two vertices.
4. **.display()** → Prints the graph in adjacency list form.

### **Task Description #7 – Priority Queue**

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

Sample Input Code:

```
class PriorityQueue:
```

```
    pass
```

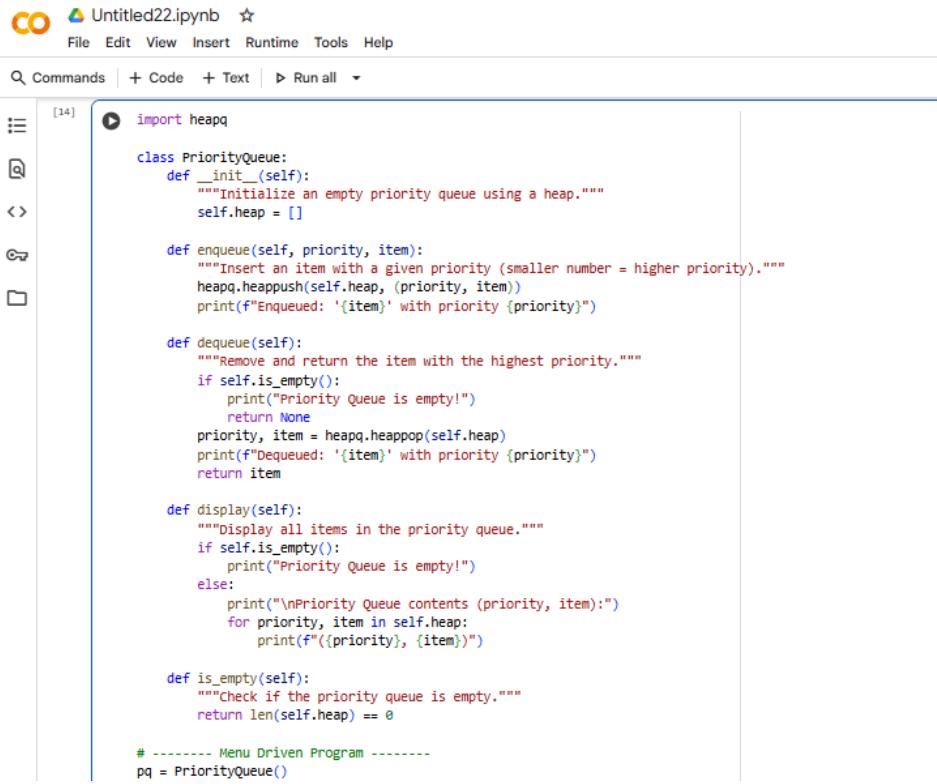
Expected Output:

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

### **Prompt:**

Use AI to implement a priority queue using Python's heapq module. The class should support methods like enqueue , dequeue, check if the queue is empty, and display contents.

### **Code:**



```
Untitled22.ipynb  ☆
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all ▾
[14] import heapq

class PriorityQueue:
    def __init__(self):
        """Initialize an empty priority queue using a heap."""
        self.heap = []

    def enqueue(self, priority, item):
        """Insert an item with a given priority (smaller number = higher priority)."""
        heapq.heappush(self.heap, (priority, item))
        print(f"Enqueued: '{item}' with priority {priority}")

    def dequeue(self):
        """Remove and return the item with the highest priority."""
        if self.is_empty():
            print("Priority Queue is empty!")
            return None
        priority, item = heapq.heappop(self.heap)
        print(f"Dequeued: '{item}' with priority {priority}")
        return item

    def display(self):
        """Display all items in the priority queue."""
        if self.is_empty():
            print("Priority Queue is empty!")
        else:
            print("\nPriority Queue contents (priority, item):")
            for priority, item in self.heap:
                print(f"({priority}, {item})")

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

# ----- Menu Driven Program -----
pq = PriorityQueue()
```

```

CO Untitled22.ipynb ☆ ⚙
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text ▶ Run all
[14] ⏪ pq = PriorityQueue()

while True:
    print("\n---- Priority Queue Menu ----")
    print("1. Enqueue")
    print("2. Dequeue")
    print("3. Display")
    print("4. Exit")

    choice = int(input("Enter your choice: "))

    if choice == 1:
        item = input("Enter item: ")
        priority = int(input("Enter priority (smaller number = higher priority): "))
        pq.enqueue(priority, item)
    elif choice == 2:
        pq.dequeue()
    elif choice == 3:
        pq.display()
    elif choice == 4:
        print("Exiting program...")
        break
    else:
        print("Invalid choice! Try again.")

---- Priority Queue Menu ----
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter item: Task A
Enter priority (smaller number = higher priority): 1
Enqueued: 'Task A' with priority 1

```

```

CO Untitled22.ipynb ☆ ⚙
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text ▶ Run all
[14] ⏪ Enter priority (smaller number = higher priority): 3
Enqueued: 'Task B' with priority 3

---- Priority Queue Menu ----
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter item: Task C
Enter priority (smaller number = higher priority): 2
Enqueued: 'Task C' with priority 2

---- Priority Queue Menu ----
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3

Priority Queue contents (priority, item):
(1, Task A)
(3, Task B)
(2, Task C)

---- Priority Queue Menu ----
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued: 'Task A' with priority 1

---- Priority Queue Menu ----
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
Exiting program...

```

Variables
Terminal

**Code Explanation:**

1. **Uses heapq**- Python's built-in module to create a priority queue.
2. **\_\_init\_\_** → Initializes an empty list to act as the heap.
3. **Enqueue (item, priority)** → Adds an element with its priority into the heap.
4. **Dequeue ()** → Removes and returns the element with the highest priority.
5. **Display ()** → Prints all items with their priorities.
6. **is\_empty()** → Checks if the queue is empty.

**Task Description #8 – Deque**

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

Sample Input Code:

class DequeDS:

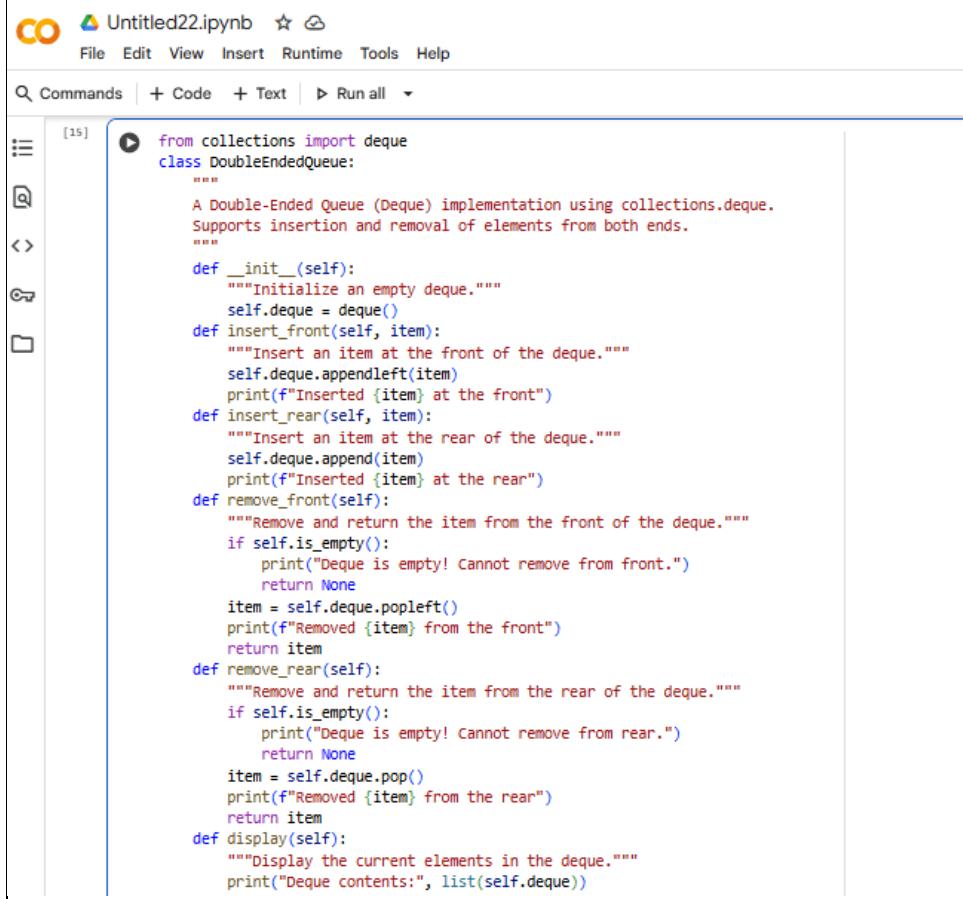
```
    pass
```

Expected Output:

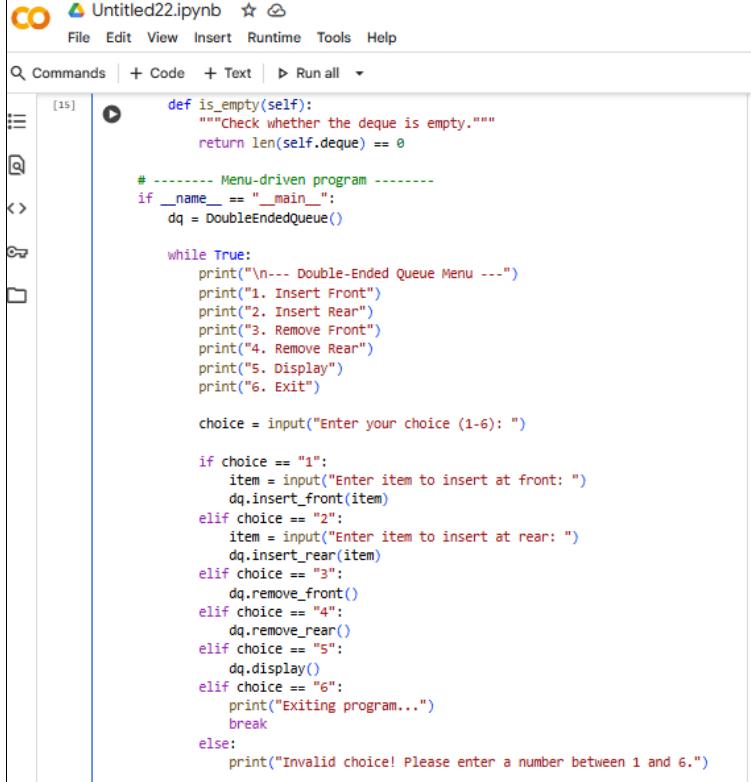
- Insert and remove from both ends with docstrings.

**Prompt:**

Generate code to implement a double-ended queue using collections.deque. The class should support methods to insert and remove elements from both the front and rear, check if the deque is empty, and display its contents. Include docstrings for each method.

**Code:**


```
[15] In [ ] from collections import deque
class DoubleEndedQueue:
    """
    A Double-Ended Queue (Deque) implementation using collections.deque.
    Supports insertion and removal of elements from both ends.
    """
    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)
        print(f"Inserted {item} at the front")
    def insert_rear(self, item):
        """
        Insert an item at the rear of the deque.
        """
        self.deque.append(item)
        print(f"Inserted {item} at the rear")
    def remove_front(self):
        """
        Remove and return the item from the front of the deque.
        """
        if self.is_empty():
            print("Deque is empty! Cannot remove from front.")
            return None
        item = self.deque.popleft()
        print(f"Removed {item} from the front")
        return item
    def remove_rear(self):
        """
        Remove and return the item from the rear of the deque.
        """
        if self.is_empty():
            print("Deque is empty! Cannot remove from rear.")
            return None
        item = self.deque.pop()
        print(f"Removed {item} from the rear")
        return item
    def display(self):
        """
        Display the current elements in the deque.
        """
        print("Deque contents:", list(self.deque))
```



```

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

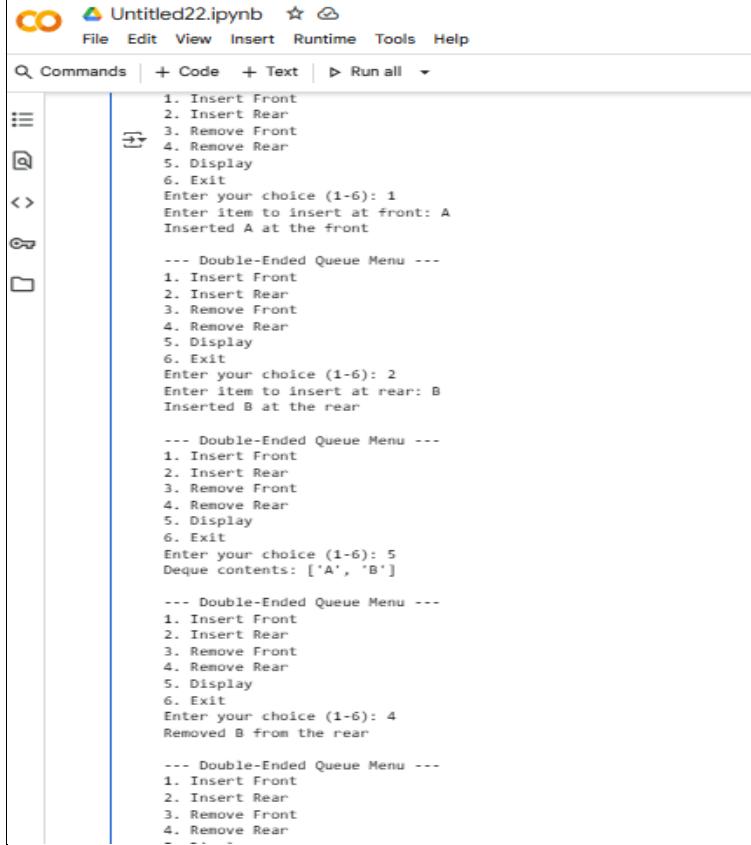
# ----- Menu-driven program -----
if __name__ == "__main__":
    dq = DoubleEndedQueue()

    while True:
        print("\n--- Double-Ended Queue Menu ---")
        print("1. Insert Front")
        print("2. Insert Rear")
        print("3. Remove Front")
        print("4. Remove Rear")
        print("5. Display")
        print("6. Exit")

        choice = input("Enter your choice (1-6): ")

        if choice == "1":
            item = input("Enter item to insert at front: ")
            dq.insert_front(item)
        elif choice == "2":
            item = input("Enter item to insert at rear: ")
            dq.insert_rear(item)
        elif choice == "3":
            dq.remove_front()
        elif choice == "4":
            dq.remove_rear()
        elif choice == "5":
            dq.display()
        elif choice == "6":
            print("Exiting program...")
            break
        else:
            print("Invalid choice! Please enter a number between 1 and 6.")

```

```

1. Insert Front
2. Insert Rear
3. Remove Front
4. Remove Rear
5. Display
6. Exit
Enter your choice (1-6): 1
Enter item to insert at front: A
Inserted A at the front

--- Double-Ended Queue Menu ---
1. Insert Front
2. Insert Rear
3. Remove Front
4. Remove Rear
5. Display
6. Exit
Enter your choice (1-6): 2
Enter item to insert at rear: B
Inserted B at the rear

--- Double-Ended Queue Menu ---
1. Insert Front
2. Insert Rear
3. Remove Front
4. Remove Rear
5. Display
6. Exit
Enter your choice (1-6): 5
Deque contents: ['A', 'B']

--- Double-Ended Queue Menu ---
1. Insert Front
2. Insert Rear
3. Remove Front
4. Remove Rear
5. Display
6. Exit
Enter your choice (1-6): 4
Removed B from the rear

--- Double-Ended Queue Menu ---
1. Insert Front
2. Insert Rear
3. Remove Front
4. Remove Rear
- - -

```

**Code Expanation:**

- 1.**`.collections.deque`** - A built-in Python class that allows fast insert and delete from both ends.
2. **`_init_`** - Creates an empty deque.
3. **`.insert_front(item)`** - Adds an element at the front.
4. **`.insert_rear(item)`** - Adds an element at the rear (end).
5. **`.remove_front()`** - Removes and returns the front element.
6. **`.remove_rear()`** - Removes and returns the rear element.
7. **`.display()`** - Prints the current contents of the deque.
8. **`.is_empty()`** - Checks if the deque has no elements.

**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.

**Comparison Table:**

Data Structure	Operation	Average Time Complexity	Worst-Case Time Complexity	Notes
Array	Access	O(1)	O(1)	
	Search	O(n)	O(n)	
	Insertion (end)	O(1)	O(n)	Unsorted array
	Insertion (middle)	O(n)	O(n)	Amortized for dynamic arrays, O(n) if resize
	Deletion (end)	O(1)	O(1)	
	Deletion (middle)	O(n)	O(n)	
Linked List	Access	O(n)	O(n)	
	Search	O(n)	O(n)	
	Insertion (front)	O(1)	O(1)	
	Insertion (end)	O(n)	O(n)	O(1) if tail pointer is maintained
	Deletion (front)	O(1)	O(1)	
	Deletion (end)	O(n)	O(n)	O(1) with doubly linked list and tail pointer
Doubly Linked List	Insertion (front)	O(1)	O(1)	
	Insertion (end)	O(1)	O(1)	
	Deletion (front)	O(1)	O(1)	
	Deletion (end)	O(1)	O(1)	
Stack	Push	O(1)	O(1)	Using array or linked list
	Pop	O(1)	O(1)	Using array or linked list
	Peek	O(1)	O(1)	Using array or linked list
Queue	Enqueue	O(1)	O(1)	Using linked list or deque
	Dequeue	O(1)	O(1)	Using linked list or deque
	Peek	O(1)	O(1)	Using linked list or deque
Hash Table	Insert	O(1)	O(n)	Average case assumes good hash function and low collision
	Search	O(1)	O(n)	Average case assumes good hash function and low collision
	Delete	O(1)	O(n)	Average case assumes good hash function and low collision
Binary Search Tree	Insert	O(log n)	O(n)	O(n) in case of skewed tree
	Search	O(log n)	O(n)	O(n) in case of skewed tree
	Deletion	O(log n)	O(n)	O(n) in case of skewed tree

Variables Terminal

**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.

**Prompt:**

Generate a Python program to simulate a Cafeteria Order Queue system using a queue data structure. The program should allow students to arrive, be added to the queue, and be served in the order they arrived.

**Code:**

Feature	Chosen Data Structure
Student Attendance Tracking	Hash Table
Event Registration System	Hash Table
Library Book Borrowing	Hash Table
Bus Scheduling System	Graph
Cafeteria Order Queue	Queue

```

from queue import Queue

class CafeteriaOrderQueue:
    """
    A cafeteria order management system using a Queue.
    Implements FIFO (First-In, First-out).
    """

    def __init__(self):
        self.queue = Queue()

    def place_order(self, student_name, order_item):
        """Add a new order to the queue."""
        self.queue.put((student_name, order_item))
        print(f"✓ Order placed: {student_name} ordered {order_item}")

    def serve_order(self):
        """Serve the next order in the queue."""
        if not self.queue.empty():
            student_name, order_item = self.queue.get()
            print(f"☛ Order served: {student_name}'s {order_item}")
        else:
            print("⚠ No pending orders!")

    def display_orders(self):
        """Display all pending orders (snapshot)."""
        if self.queue.empty():
            print("⚠ No orders in the queue.")
        else:
            print("\n▣ Pending Orders:")
            # Convert queue to list for snapshot
            temp_list = list(self.queue.queue)
            for student, item in temp_list:
                print(f"- {student} ordered {item}")

    # ----- User Input Menu -----

if __name__ == "__main__":
    cafeteria = CafeteriaOrderQueue()

    while True:
        print("\n===== Cafeteria Order Queue =====")
        print("1. Place Order")
        print("2. Serve Order")
        print("3. Display Orders")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == "1":
            name = input("Enter student name: ")
            item = input("Enter order item: ")
            cafeteria.place_order(name, item)

        elif choice == "2":
            cafeteria.serve_order()

        elif choice == "3":
            cafeteria.display_orders()

        elif choice == "4":
            print("👋 Exiting... Thank you!")
            break

        else:
            print("⚠ Invalid choice. Please enter 1-4.")


```

===== Cafeteria Order Queue =====  
1. Place Order  
2. Serve Order  
3. Display Orders  
4. Exit

es Terminal

```

Enter your choice (1-4): 1
Enter student name: charan
Enter order item: pizza
✓ Order placed: charan ordered pizza

===== Cafeteria Order Queue =====
1. Place Order
2. Serve Order
3. Display Orders
4. Exit
Enter your choice (1-4): 1
Enter student name: vishnu
Enter order item: Burger
✓ Order placed: vishnu ordered Burger

===== Cafeteria Order Queue =====
1. Place Order
2. Serve Order
3. Display Orders
4. Exit
Enter your choice (1-4): 2
Order served: charan's pizza

===== Cafeteria Order Queue =====
1. Place Order
2. Serve Order
3. Display Orders
4. Exit
Enter your choice (1-4): 3

Pending Orders:
- vishnu ordered Burger

===== Cafeteria Order Queue =====
1. Place Order
2. Serve Order
3. Display Orders
4. Exit
Enter your choice (1-4): 4
✖ Exiting... Thank you!

```

### **Code Explanation:**

1. **Class: CafeteriaOrderQueue** - Manages cafeteria orders using a queue.
2. **init\_()**:Creates an empty queue for storing orders.
3. **place\_order(student\_name, order\_item)**:Adds a new order to the queue.Shows a message confirming the order.
4. **serve\_order()**:Serves (removes) the first order in the queue.
5. **display\_orders()**:Shows all current orders in the queue.

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.