

ASSIGNMENT 11.1

Data Structures with AI: Implementing Fundamental Structures

A.NagaKoushik

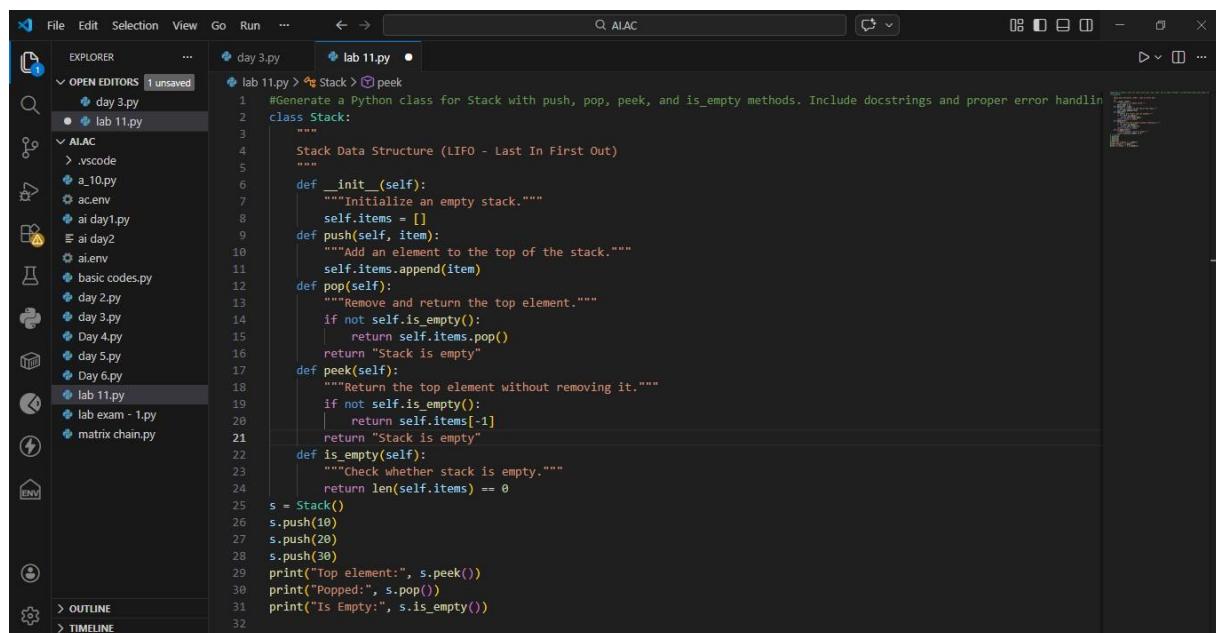
2403A51L22

Batch-51

Task 1: Stack Implementation

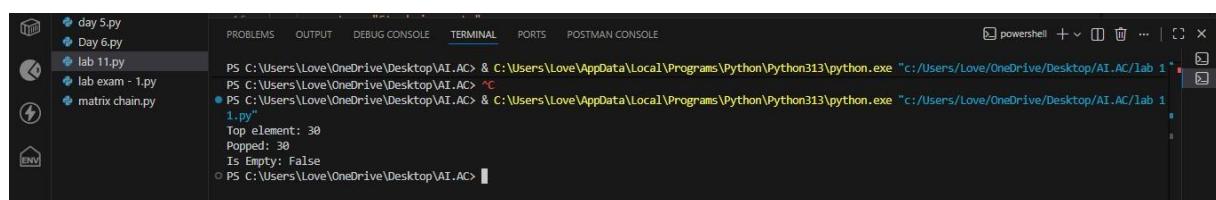
Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.

Prompt: Generate a Python class for Stack with push, pop, peek, and is_empty methods. Include docstrings and proper error handling.



```
#Generate a Python class for Stack with push, pop, peek, and is_empty methods. Include docstrings and proper error handling
class Stack:
    """
    Stack Data Structure (LIFO - Last In First Out)
    """
    def __init__(self):
        """Initialize an empty stack."""
        self.items = []
    def push(self, item):
        """Add an element to the top of the stack."""
        self.items.append(item)
    def pop(self):
        """Remove and return the top element."""
        if not self.is_empty():
            return self.items.pop()
        return "Stack is empty"
    def peek(self):
        """Return the top element without removing it."""
        if not self.is_empty():
            return self.items[-1]
        return "Stack is empty"
    def is_empty(self):
        """Check whether stack is empty."""
        return len(self.items) == 0
s = Stack()
s.push(10)
s.push(20)
s.push(30)
print("Top element:", s.peek())
print("Popped:", s.pop())
print("Is Empty:", s.is_empty())
```

OUTPUT:



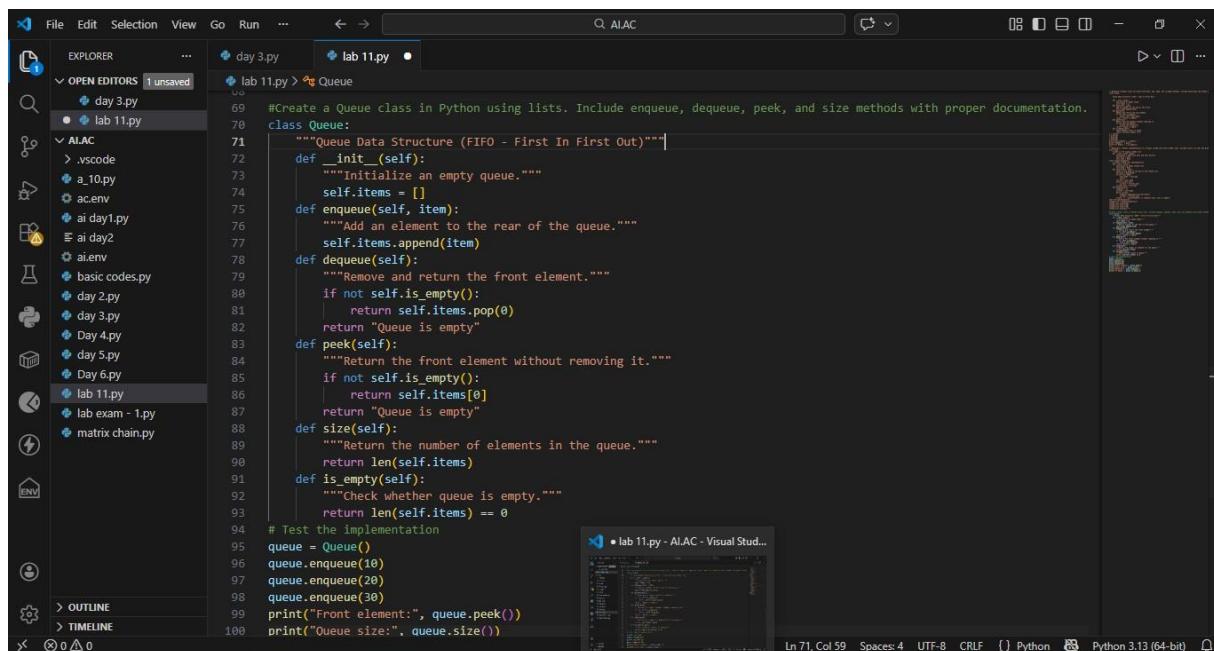
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1.py"
PS C:\Users\Love\OneDrive\Desktop\AI.AC & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1.py"
Top element: 30
Popped: 30
Is Empty: False
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

Explanation: A Stack is a linear data structure that follows the LIFO (Last In First Out) principle, where the last element inserted is the first one removed. Operations such as push, pop, and peek are performed at one end called the top. It is commonly used in function calls, undo operations, and expression evaluation.

Task Description #2: Queue Implementation

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

Prompt: Create a Queue class in Python using lists. Include enqueue, dequeue, peek, and size methods with proper documentation.



The screenshot shows the Visual Studio Code interface with the following details:

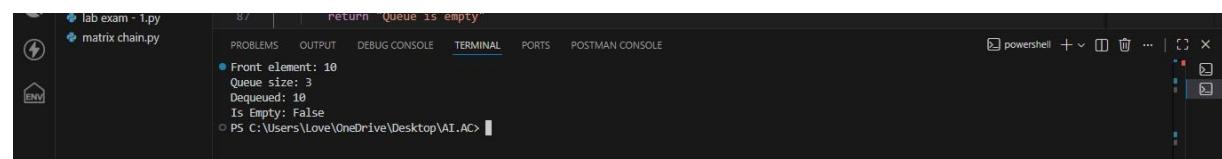
- File Explorer:** Shows files like day 3.py, lab 11.py, and lab 11.py (the active file).
- Code Editor:** Displays the Python code for the Queue class. The code includes methods for enqueue, dequeue, peek, and size, along with their docstrings.
- Terminal:** Shows the command "ln 71, Col 59" and other status information.
- Output:** Shows the execution results of the code, including the output of the print statements.

```

class Queue:
    """Create a Queue class in Python using lists. Include enqueue, dequeue, peek, and size methods with proper documentation."""
    def __init__(self):
        """Initialize an empty queue."""
        self.items = []
    def enqueue(self, item):
        """Add an element to the rear of the queue."""
        self.items.append(item)
    def dequeue(self):
        """Remove and return the front element."""
        if not self.is_empty():
            return self.items.pop(0)
        return "Queue is empty"
    def peek(self):
        """Return the front element without removing it."""
        if not self.is_empty():
            return self.items[0]
        return "Queue is empty"
    def size(self):
        """Return the number of elements in the queue."""
        return len(self.items)
    def is_empty(self):
        """Check whether queue is empty."""
        return len(self.items) == 0
# Test the implementation
queue = Queue()
queue.enqueue(10)
queue.enqueue(20)
queue.enqueue(30)
print("Front element:", queue.peek())
print("Queue size:", queue.size())

```

OUTPUT:



The terminal window shows the following output:

```

Front element: 10
Queue size: 3
Dequeued: 10
Is Empty: False
PS C:\Users\Love\OneDrive\Desktop\AI.AC>

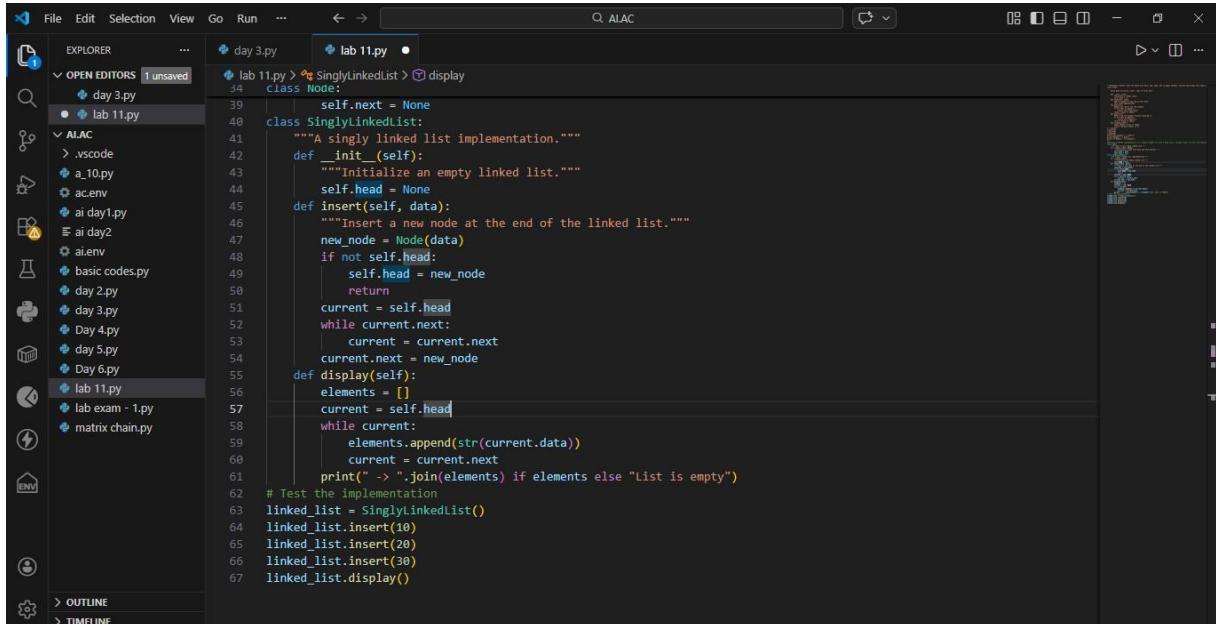
```

Explanation: A Queue is a linear data structure that follows the FIFO (First In First Out) principle. This means the first element inserted is the first one removed.

Task Description #3: Linked List

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

Prompt : Generate a Python implementation of a Singly Linked List with a Node class. Include insert (at end) and display methods with docstrings.

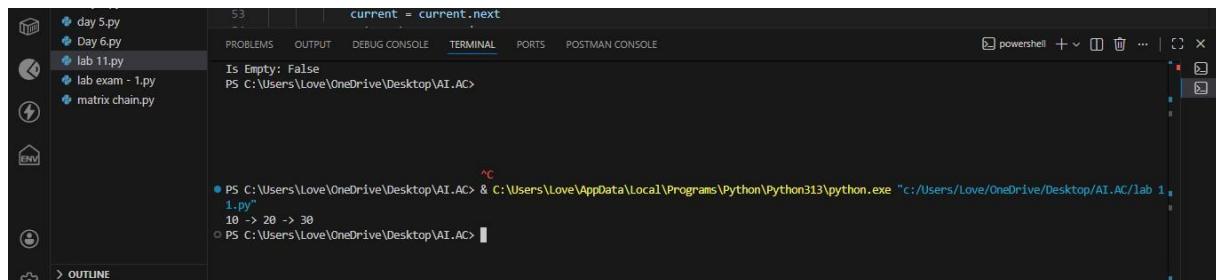


The screenshot shows the VS Code interface with the following details:

- EXPLORER** view: Shows files like day 3.py, lab 11.py, and lab 11.py (the active editor).
- OPEN EDITORS**: One file is open: lab 11.py.
- CODE**: The content of lab 11.py is displayed:

```
class Node:  
    self.next = None  
class SinglyLinkedList:  
    """A singly linked list implementation."""  
    def __init__(self):  
        """Initialize an empty linked list."""  
        self.head = None  
    def insert(self, data):  
        """Insert a new node at the end of the linked list."""  
        new_node = Node(data)  
        if not self.head:  
            self.head = new_node  
            return  
        current = self.head  
        while current.next:  
            current = current.next  
        current.next = new_node  
    def display(self):  
        elements = []  
        current = self.head  
        while current:  
            elements.append(str(current.data))  
            current = current.next  
        print(" -> ".join(elements) if elements else "List is empty")  
# Test the implementation  
linked_list = SinglyLinkedList()  
linked_list.insert(10)  
linked_list.insert(20)  
linked_list.insert(30)  
linked_list.display()
```

OUTPUT:



The screenshot shows the VS Code interface with the following details:

- TERMINAL** tab: Displays the following command-line session:

```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 11.py"  
IsEmpty: False  
10 -> 20 -> 30
```

Explanation: A Singly Linked List is a dynamic data structure where elements (nodes) are connected using pointers. Linked Lists are useful when frequent insertions and deletions are required, as they do not require shifting elements like arrays.

Task Description #4: Binary Search Tree (BST)

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

Prompt: Create a Binary Search Tree in Python with recursive insert and inorder traversal methods. Include proper class structure and documentation.

```

150 ## TASK-4: Create a Binary Search Tree in Python with a nested Node class. Implement recursive insert and in-order traversal
151 # methods following BST properties. Add proper docstrings.
152
153 class Node:
154     def __init__(self, data):
155         self.data = data
156         self.left = None
157         self.right = None
158
159 class BinarySearchTree:
160     def __init__(self):
161         self.root = None
162
163     def insert(self, data):
164         if self.root is None:
165             self.root = Node(data)
166             print(f"{data} inserted as root of the BST.")
167         else:
168             self._insert_recursive(self.root, data)
169
170     def _insert_recursive(self, node, data):
171         if data < node.data:
172             if node.left is None:
173                 node.left = Node(data)
174                 print(f"{data} inserted to the left of {node.data}.")
175             else:
176                 self._insert_recursive(node.left, data)
177
178             if node.right is None:
179                 node.right = Node(data)
180                 print(f"{data} inserted to the right of {node.data}.")
181             else:
182                 self._insert_recursive(node.right, data)
183
184     def in_order_traversal(self):
185         elements = []
186         self._in_order_recursive(self.root, elements)
187         print("In-order Traversal: " + ", ".join(map(str, elements)))
188
189     def _in_order_recursive(self, node, elements):
190         if node:
191             self._in_order_recursive(node.left, elements)
192             elements.append(node.data)
193             self._in_order_recursive(node.right, elements)
194
195 bst = BinarySearchTree()
196 while True:
197     print("1. Insert")
198     print("2. In-order Traversal")
199     print("3. Exit")
200     choice = input("Enter your choice: ")
201     if choice == "1":
202         value = input("Enter value to insert: ")
203         bst.insert(value)
204     elif choice == "2":
205         bst.in_order_traversal()
206     elif choice == "3":
207         print("Exiting program...")
208         break
209     else:
210         print("Invalid choice! Try again.")

```

```

150 ## TASK-4: Create a Binary Search Tree in Python with a nested Node class. Implement recursive insert and in-order traversal
151 # methods following BST properties. Add proper docstrings.
152
153 class Node:
154     def __init__(self, data):
155         self.data = data
156         self.left = None
157         self.right = None
158
159 class BinarySearchTree:
160     def __init__(self):
161         self.root = None
162
163     def insert(self, data):
164         if self.root is None:
165             self.root = Node(data)
166             print(f"{data} inserted as root of the BST.")
167         else:
168             self._insert_recursive(self.root, data)
169
170     def _insert_recursive(self, node, data):
171         if data < node.data:
172             if node.left is None:
173                 node.left = Node(data)
174                 print(f"{data} inserted to the left of {node.data}.")
175             else:
176                 self._insert_recursive(node.left, data)
177
178             if node.right is None:
179                 node.right = Node(data)
180                 print(f"{data} inserted to the right of {node.data}.")
181             else:
182                 self._insert_recursive(node.right, data)
183
184     def in_order_traversal(self):
185         elements = []
186         self._in_order_recursive(self.root, elements)
187         print("In-order Traversal: " + ", ".join(map(str, elements)))
188
189     def _in_order_recursive(self, node, elements):
190         if node:
191             self._in_order_recursive(node.left, elements)
192             elements.append(node.data)
193             self._in_order_recursive(node.right, elements)
194
195 bst = BinarySearchTree()
196 while True:
197     print("1. Insert")
198     print("2. In-order Traversal")
199     print("3. Exit")
200     choice = input("Enter your choice: ")
201     if choice == "1":
202         value = input("Enter value to insert: ")
203         bst.insert(value)
204     elif choice == "2":
205         bst.in_order_traversal()
206     elif choice == "3":
207         print("Exiting program...")
208         break
209     else:
210         print("Invalid choice! Try again.")

```

OUTPUT:

```

PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> & c:/Users/sarik/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/sarik/OneD...
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 1
Enter value to insert: 11
11 inserted as root of the BST.

1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 2
Enter value to insert: 14
14 inserted to the right of 11.

1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 3

```

Explanation: A Binary Search Tree is a hierarchical data structure where the left child contains smaller values and the right child contains larger values than the root. This property makes searching, insertion, and deletion efficient.

Task Description #5: Hash Table

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

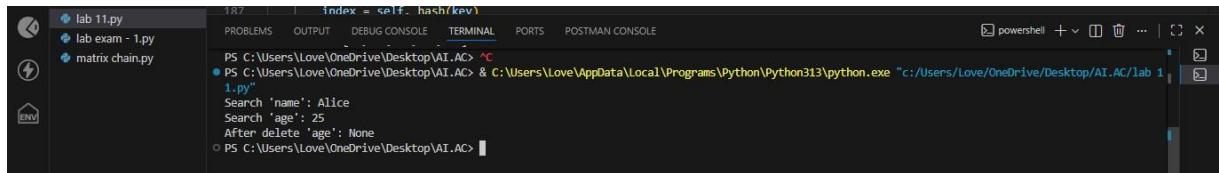
Prompt: Implement a Hash Table in Python using chaining for collision handling. Include insert, search, and delete methods with comments.

```
File Edit Selection View Go Run ... 🔍 AIAC

EXPLORER ... day 3.py lab 11.py ●
OPEN EDITORS 1 unsaved
lab 11.py
AIAC
.vscode
a_10.py
a.env
ai day1.py
ai day2
ai.env
basic codes.py
day 2.py
day 3.py
Day 4.py
day 5.py
Day 6.py
lab 11.py
lab exam - 1.py
matrix chain.py

158     #Implement a Hash Table in Python using chaining for collision handling. Include insert, search, and delete methods with c
159     class HashTable:
160         """Hash Table implementation using chaining for collision handling."""
161         def __init__(self, size=10):
162             """Initialize hash table with given size."""
163             self.size = size
164             self.table = [[] for _ in range(size)]
165         def __hash__(self, key):
166             """Compute hash index for given key."""
167             return hash(key) % self.size
168         def insert(self, key, value):
169             """Insert key-value pair into hash table."""
170             index = self.__hash__(key)
171             # Check if key already exists and update
172             for i, (k, v) in enumerate(self.table[index]):
173                 if k == key:
174                     self.table[index][i] = (key, value)
175                     return
176             # Add new key-value pair
177             self.table[index].append((key, value))
178         def search(self, key):
179             """Search for value by key. Return value or None."""
180             index = self.__hash__(key)
181             for k, v in self.table[index]:
182                 if k == key:
183                     return v
184
185     class HashTable:
186         def insert(self, key, value):
187             self.table[index][i] = (key, value)
188             return
189         # Add new key-value pair
190         self.table[index].append((key, value))
191         def search(self, key):
192             """Search for value by key. Return value or None."""
193             index = self.__hash__(key)
194             for k, v in self.table[index]:
195                 if k == key:
196                     return v
197             return None
198         def delete(self, key):
199             """Delete key-value pair from hash table."""
200             index = self.__hash__(key)
201             for i, (k, v) in enumerate(self.table[index]):
202                 if k == key:
203                     self.table[index].pop(i)
204                     return True
205             return False
206
207     # Test the implementation
208     hash_table = HashTable()
209     hash_table.insert("name", "Alice")
210     hash_table.insert("age", 25)
211     hash_table.insert("city", "NVC")
212     print("Search 'name':", hash_table.search("name"))
213     print("Search 'age':", hash_table.search("age"))
214     hash_table.delete("age")
215     print("After delete 'age':", hash_table.search("age"))
```

OUTPUT:



The screenshot shows the VS Code interface with the terminal tab active. The terminal window displays the following Python script execution:

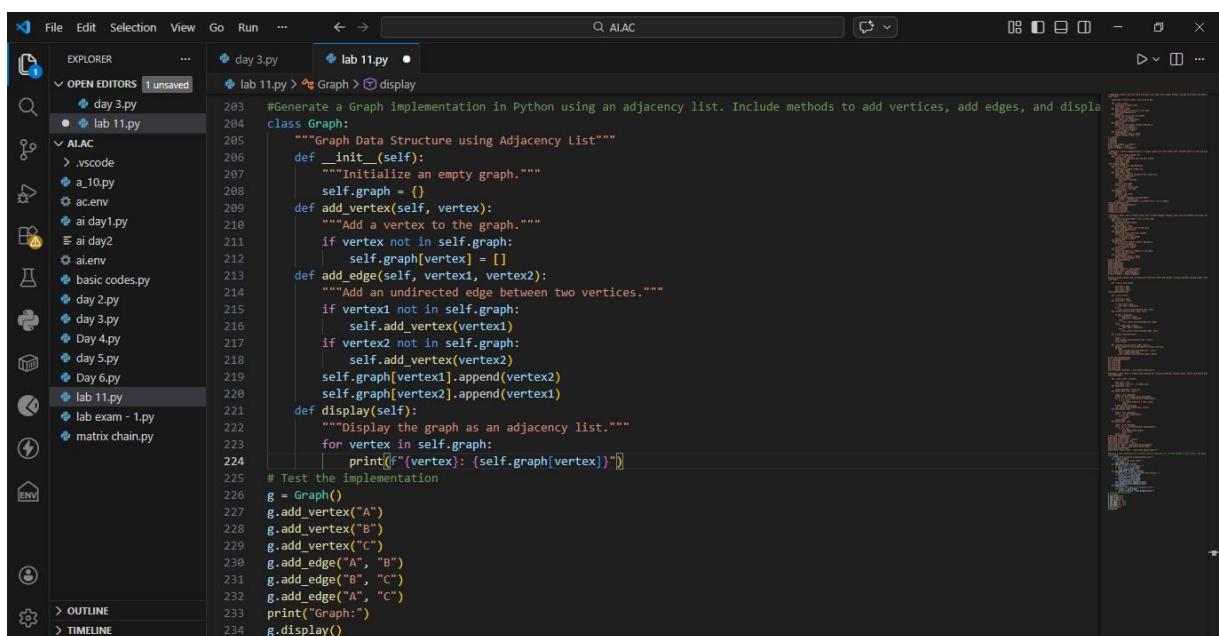
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> python lab 11.py
Search 'name': Alice
Search 'age': 25
After delete 'age': None
```

Explanation: A Hash Table stores data in key-value pairs using a hash function to compute an index. It provides fast average-case time complexity for search, insertion, and deletion operations.

Task Description #6: Graph Representation

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

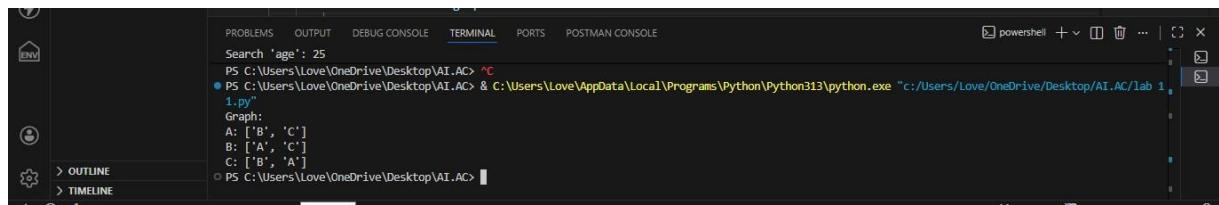
Prompt: Generate a Graph implementation in Python using an adjacency list. Include methods to add vertices, add edges, and display the graph.



The screenshot shows the VS Code interface with the code editor tab active. The code editor displays a Python file named `lab 11.py` containing the following implementation of a Graph class using an adjacency list:

```
#Generate a Graph implementation in Python using an adjacency list. Include methods to add vertices, add edges, and display
class Graph:
    """Graph Data Structure using Adjacency List"""
    def __init__(self):
        """Initialize an empty graph."""
        self.graph = {}
    def add_vertex(self, vertex):
        """Add a vertex to the graph."""
        if vertex not in self.graph:
            self.graph[vertex] = []
    def add_edge(self, vertex1, vertex2):
        """Add an undirected edge between two vertices."""
        if vertex1 not in self.graph:
            self.add_vertex(vertex1)
        if vertex2 not in self.graph:
            self.add_vertex(vertex2)
        self.graph[vertex1].append(vertex2)
        self.graph[vertex2].append(vertex1)
    def display(self):
        """Display the graph as an adjacency list."""
        for vertex in self.graph:
            print(f'{vertex}: {self.graph[vertex]}')
# Test the implementation
g = Graph()
g.add_vertex("A")
g.add_vertex("B")
g.add_vertex("C")
g.add_edge("A", "B")
g.add_edge("B", "C")
g.add_edge("A", "C")
print("Graph:")
g.display()
```

Output:



The screenshot shows the VS Code interface with the terminal tab selected. The terminal window displays the following Python code and its output:

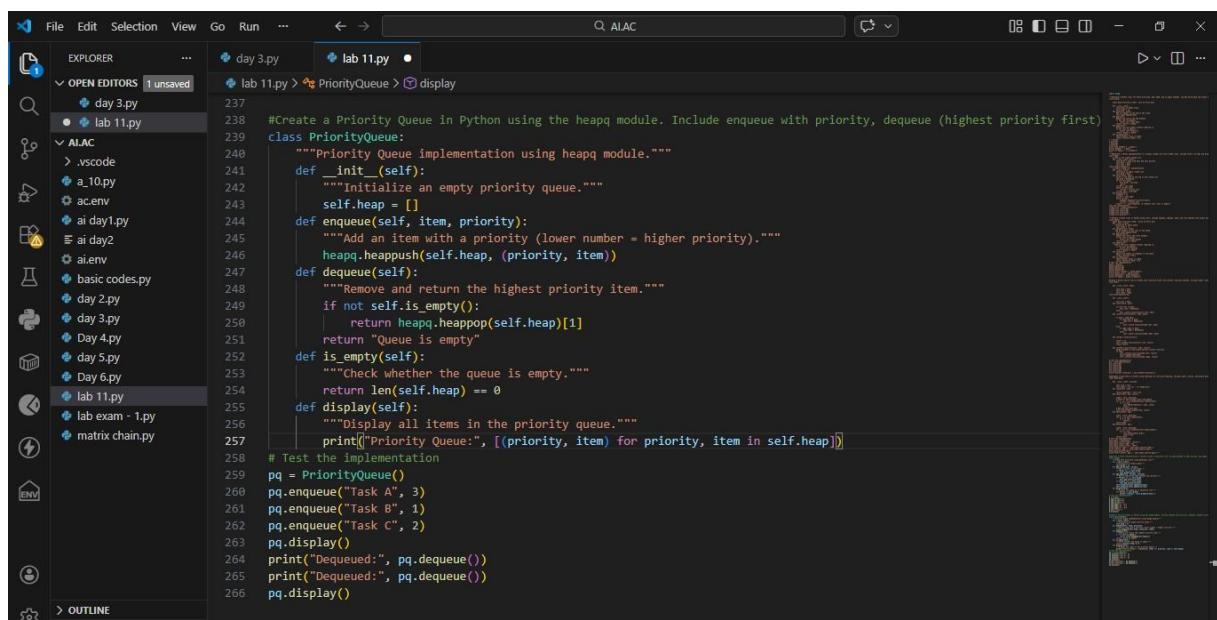
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
Search 'age': 25
PS C:\Users\Love\OneDrive\Desktop\AI.AC & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1.py"
● PS C:\Users\Love\OneDrive\Desktop\AI.AC & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1.py"
Graph:
A: ['B', 'C']
B: ['A', 'C']
C: ['B', 'A']
```

Explanation: A Graph is a non-linear data structure used to represent relationships between entities. It consists of vertices (nodes) and edges (connections).

Task Description #7: Priority Queue

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

Prompt: Create a Priority Queue in Python using the heapq module. Include enqueue with priority, dequeue (highest priority first), and display methods.



The screenshot shows the VS Code interface with the Explorer tab selected. The left sidebar shows a file tree with several Python files. The current file being edited is `lab 11.py`. The code implements a Priority Queue using the `heapq` module:

```
File Edit Selection View Go Run ... ← → Q AI.AC
EXPLORER OPEN EDITORS 1 unsaved
day 3.py lab 11.py
lab 11.py > Priority Queue > display
238 #Create a Priority Queue in Python using the heapq module. Include enqueue with priority, dequeue (highest priority first)
239 class PriorityQueue:
240     """Priority Queue implementation using heapq module."""
241     def __init__(self):
242         """Initialize an empty priority queue."""
243         self.heap = []
244     def enqueue(self, item, priority):
245         """Add an item with a priority (lower number = higher priority)."""
246         heapq.heappush(self.heap, (priority, item))
247     def dequeue(self):
248         """Remove and return the highest priority item."""
249         if not self.is_empty():
250             return heapq.heappop(self.heap)[1]
251         return "Queue is empty"
252     def is_empty(self):
253         """Check whether the queue is empty."""
254         return len(self.heap) == 0
255     def display(self):
256         """Display all items in the priority queue."""
257         print("Priority Queue:", [(priority, item) for priority, item in self.heap])
258
# Test the implementation
259 pq = PriorityQueue()
260 pq.enqueue("Task A", 3)
261 pq.enqueue("Task B", 1)
262 pq.enqueue("Task C", 2)
263 pq.display()
264 print("Dequeued:", pq.dequeue())
265 print("Dequeued:", pq.dequeue())
266 pq.display()
```

Output:

```
matrix_chain.py
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
powershell + - x
PS C:\Users\Love\OneDrive\Desktop\AI.AC> ^C
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1/matrix_chain.py"
Graph:
A: ['B', 'C']
B: ['A', 'C']
C: ['B', 'A']
Priority Queue: [(1, 'Task B'), (3, 'Task A'), (2, 'Task C')]
Dequeued: Task B
Dequeued: Task C
Priority Queue: [(3, 'Task A')]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

Explanation: A Priority Queue is a special type of queue where elements are removed based on priority rather than order of insertion. Higher priority elements are processed first. It is typically implemented using a heap for efficiency.

Task Description #8 – Deque

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

Prompt: Implement a double-ended queue (Deque) in Python using collections, deque. Include methods to insert and remove from both ends with documentation.



The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure with files like `day 3.py`, `lab 11.py`, `vscode`, `.env`, and `matrix chain.py`.
- Editor Area (Center):** Displays the content of `lab 11.py`, which is a Python script for implementing a double-ended queue (Dequeue) using `collections.deque`. The code includes methods for adding items to the front and rear, removing items from both ends, and peeking at the front item.
- Search Bar (Top):** Contains the text "Q A1AC".
- Activity Bar (Bottom):** Shows various icons for file operations like Open, Save, Find, and Run.

The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar: Shows files like .vscode, day 3.py, lab 11.py, AIAC, a_10.py, ac.env, ai day1.py, ai day2.py, ai.env, basic codes.py, day 2.py, day 3.py, Day 4.py, day 5.py, Day 6.py, lab 11.py, lab exam - 1.py, and matrix chain.py.
- CODE EDITOR**: The active file is lab 11.py, containing Python code for a Deque class. The code includes methods for adding items at front and rear, removing items from front and rear, peeking at front and rear, checking if the deque is empty, and displaying all items.
- OUTPUT** panel: Shows the execution of the code, outputting the priority queue and the results of various operations.

Output:

The terminal output shows the execution of the deque code:

```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> python lab 11.py
Priority Queue: [(3, 'Task A')]
Deque: [5, 10, 20, 30]
Front: 5
Rear: 30
Removed from front: 5
Removed from rear: 30
Deque: [10, 20]
```

Explanation: A Deque (Double Ended Queue) allows insertion and deletion of elements from both the front and rear ends.