AI ASSISTED CODING

LAB:11.2

ROLLNO:2403A52096

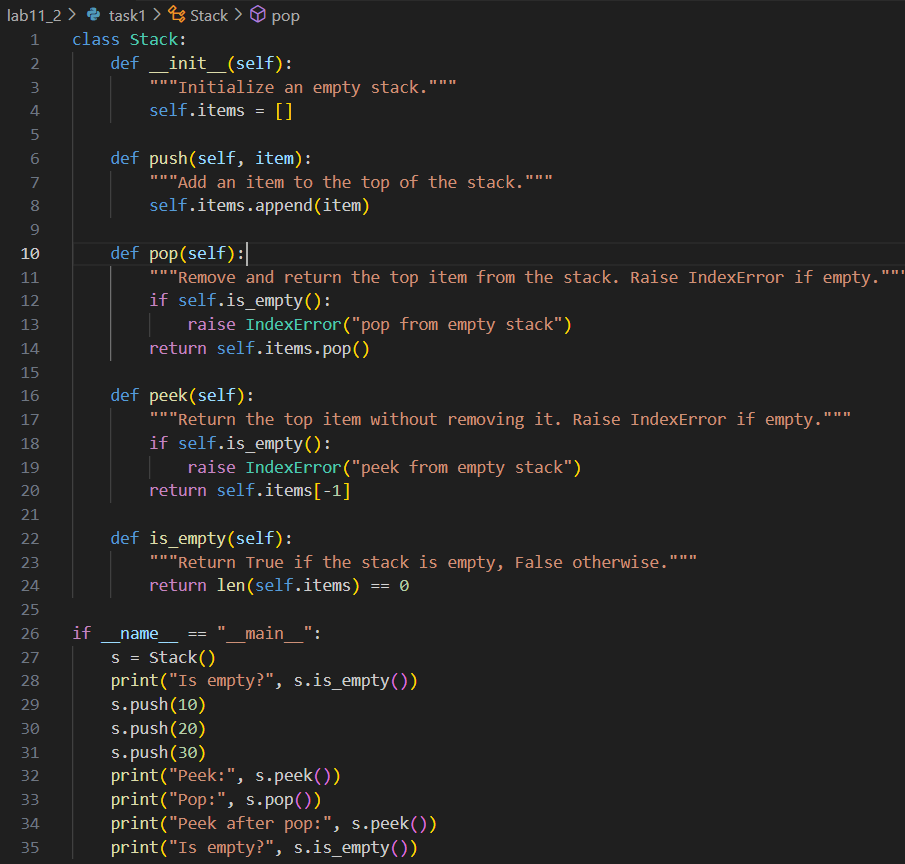
BATCH:04

TASK1:

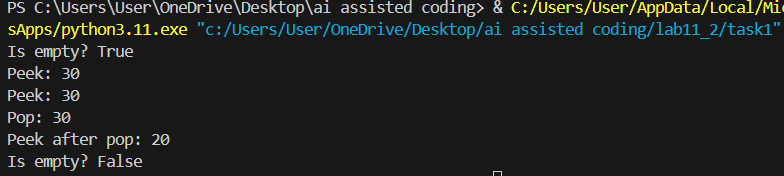
PROMPT: Implement a Stack class in Python with the following methods:

* [push(item)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Add an item to the top of the stack
* [pop()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Remove and return the top item from the stack
* [peek()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Return the top item without removing it
* [is\_empty()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Check if the stack is empty  
  Include docstrings for each method.

CODE:



OUTPUT:



OBSERVATION:

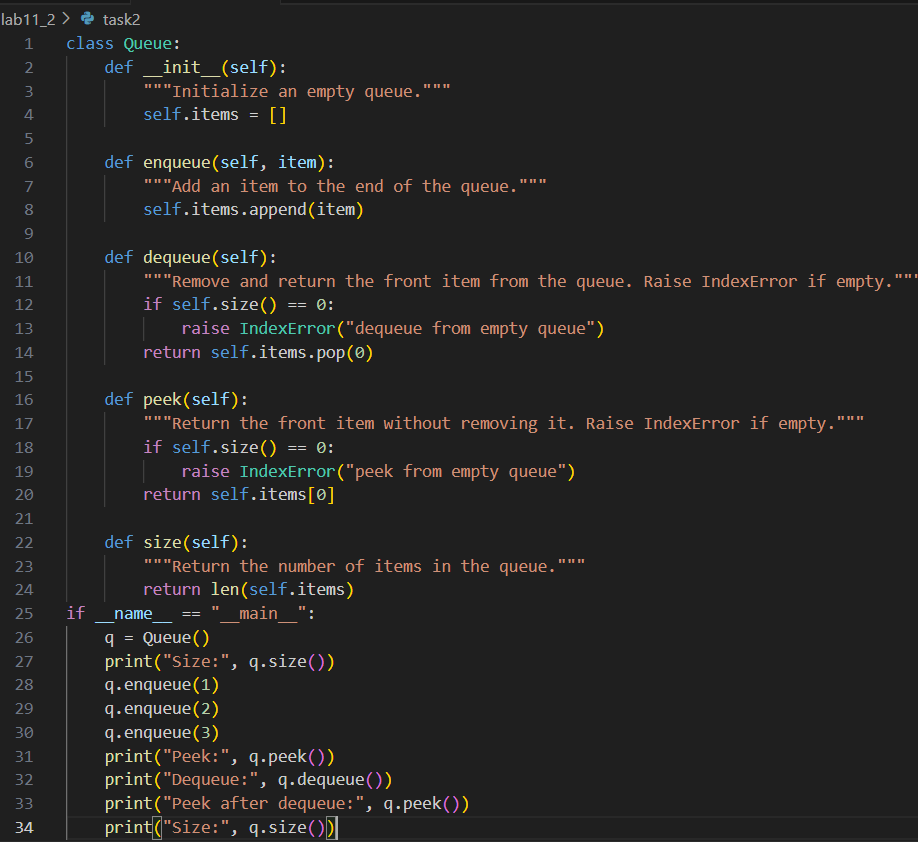
The code correctly implements a functional Stack class using a list. All required methods are present and include docstrings. The sample usage demonstrates pushing, peeking, popping, and checking if the stack is empty, with expected output for each operation.

TASK2:

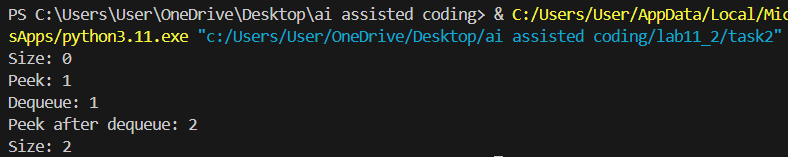
PROMPT: Implement a Queue class in Python using lists.  
Required methods:

* [enqueue(item)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Add an item to the end of the queue
* [dequeue()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Remove and return the front item from the queue
* [peek()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Return the front item without removing it
* [size()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Return the number of items in the queue  
  Include docstrings for each method.

CODE:



OUTPUT:



OBSERVATION:

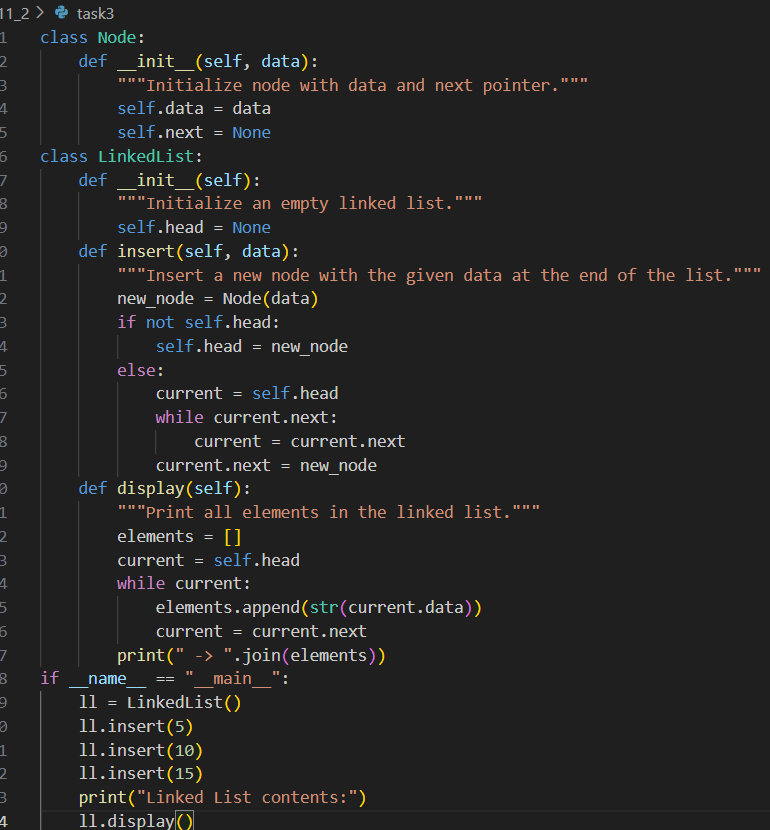
The code provides a correct FIFO Queue implementation using a list. All required methods are present with docstrings. The sample usage demonstrates enqueueing, peeking, dequeuing, and checking the size, with expected output for each operation.

TASK3:

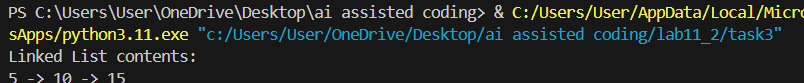
PROMPT: Implement a singly linked list in Python with the following classes and methods:

* [Node](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Stores data and a reference to the next node
* [LinkedList](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html):
  + [insert(data)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Insert a new node at the end
  + [display()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Print all elements in the list  
    Include clear docstrings for each method.

CODE:



OUTPUT:



OBSERVATION:

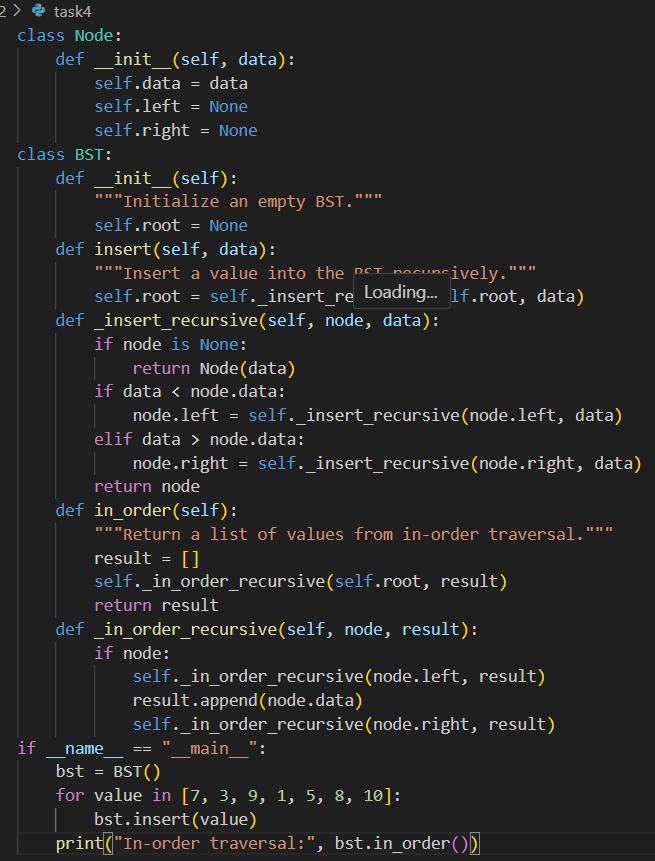
The code provides a correct singly linked list implementation. The [Node](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) and [LinkedList](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) classes are present, with [insert](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) and [display](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) methods, each documented.The sample usage demonstrates inserting nodes and displaying the list contents, producing the expected output.

TASK4:

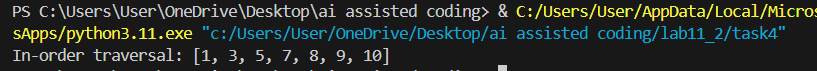
PROMPT: Implement a Binary Search Tree (BST) in Python with:

* Recursive [insert(data)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) method
* Recursive [in\_order()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) traversal method  
  Include clear method documentation.

CODE:



OUTPUT:



OBSERVATION:

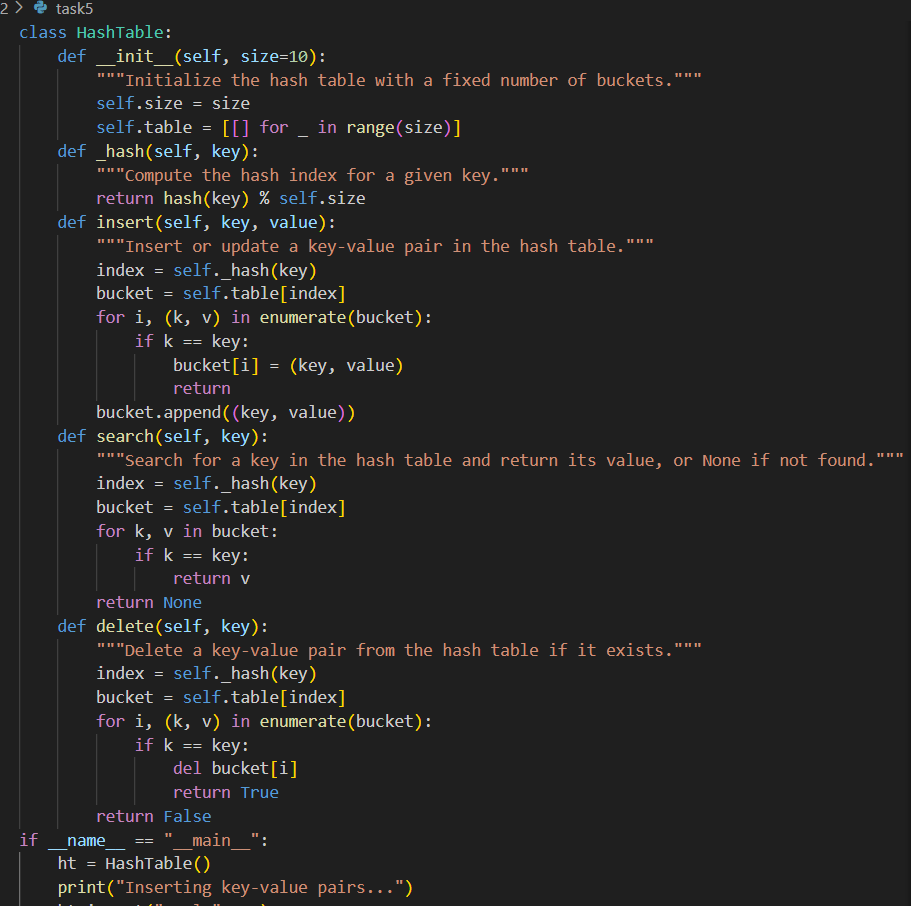
The code correctly implements a BST with recursive insert and in-order traversal. The main block demonstrates inserting values and printing the in-order traversal, which outputs the values in sorted order as expected.

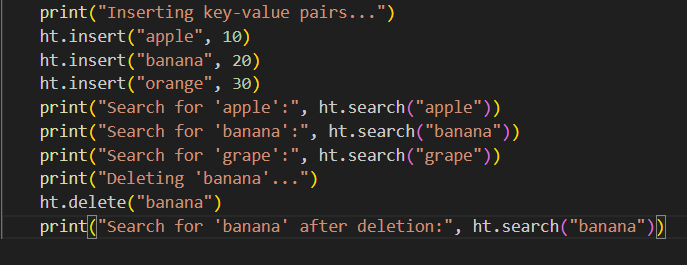
TASK5:

PROMPT: Implement a hash table in Python with:

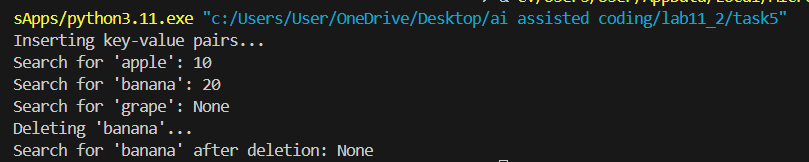
* Collision handling using chaining
* Methods: [insert(key, value)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), [search(key)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html), and [delete(key)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html)  
  Include clear comments and demonstrate usage with output.

CODE:





OUTPUT:



OBSERVATION:

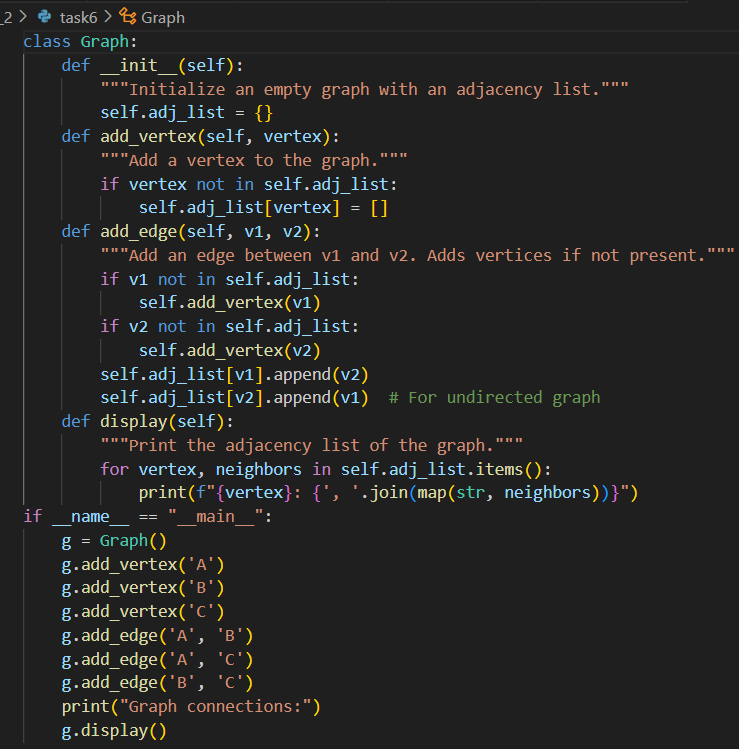
The code provides a hash table implementation with chaining for collisions.All required methods are present and documented.  
The main block demonstrates inserting, searching, and deleting keys, and prints the results, showing expected behavior for each operation.

TASK6:

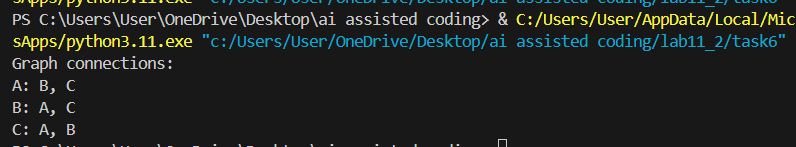
PROMPT: Implement a graph in Python using an adjacency list.  
Required methods:

* [add\_vertex(vertex)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Add a vertex
* [add\_edge(v1, v2)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Add an edge between two vertices
* [display()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Print the adjacency list  
  Include clear method documentation and demonstrate usage with output.

CODE:



OUTPUT:



OBSERVATION:

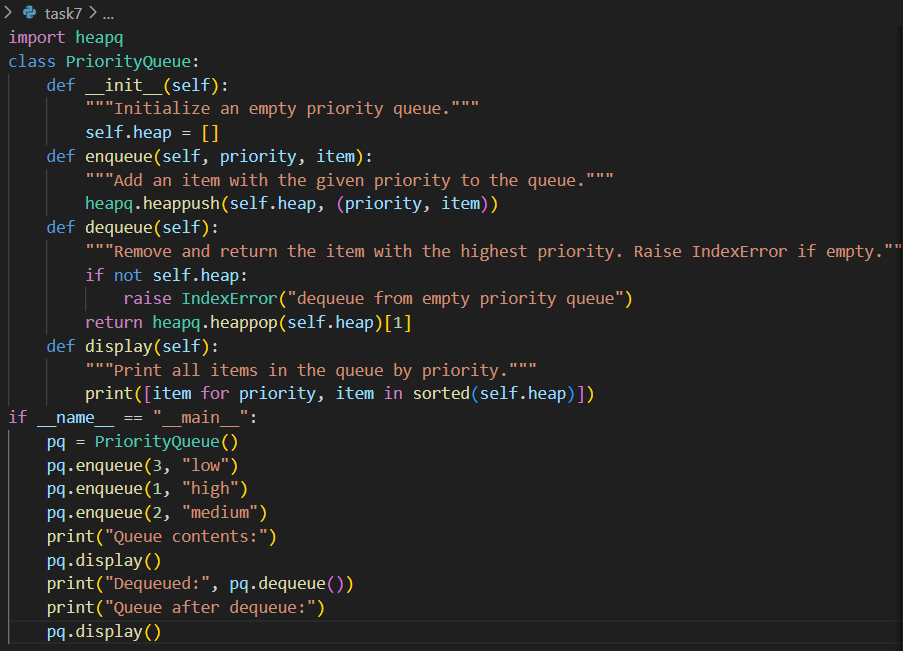
The code provides a correct adjacency list graph implementation.All required methods are present and documented.The main block demonstrates adding vertices and edges, and displays the graph connections, showing expected adjacency lists for each vertex.

TASK7:

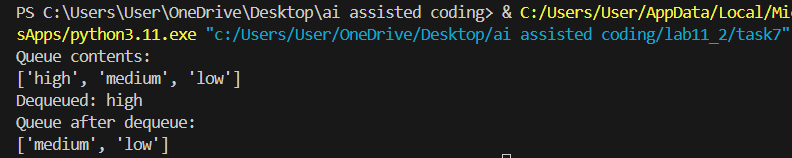
PROMPT:Implement a priority queue in Python using the [heapq](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ") module.  
Required methods:

* [enqueue(priority, item)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Add an item with a given priority
* [dequeue()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Remove and return the item with the highest priority
* [display()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Print all items in the queue by priority  
  Include clear method documentation and demonstrate usage with output.

CODE:



OUTPUT:



OBSERVATION:

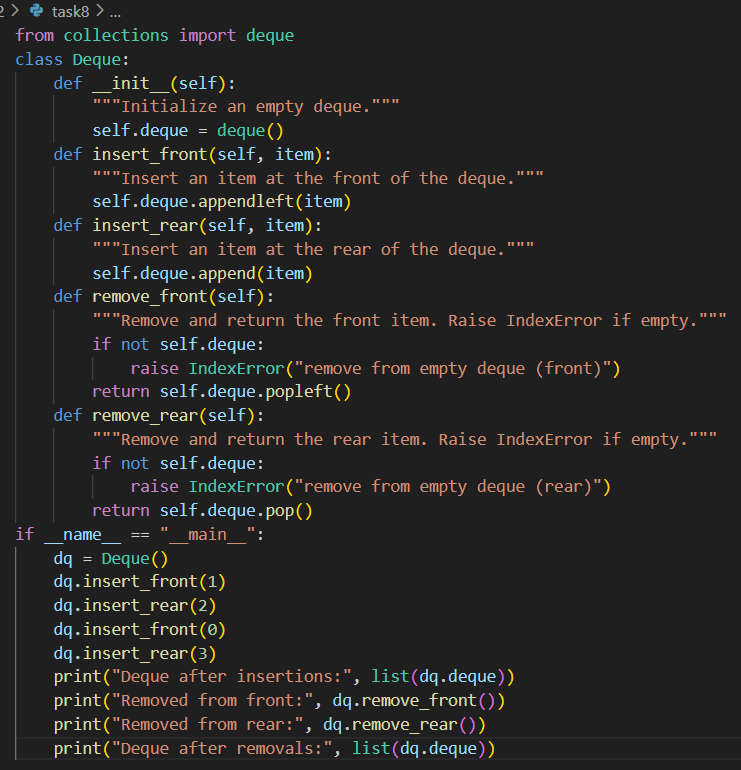
The code provides a correct priority queue implementation using [heapq](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html). All required methods are present and documented.The main block demonstrates enqueueing items with priorities, displaying the queue, dequeuing the highest priority item, and displaying the queue again, showing expected output for each operation.

TASK8:

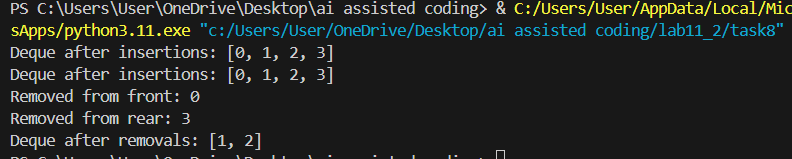
PROMPT: Implement a double-ended queue in Python using [collections.deque](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html" \o ").  
Required methods:

* [insert\_front(item)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Insert at the front
* [insert\_rear(item)](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Insert at the rear
* [remove\_front()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Remove from the front
* [remove\_rear()](vscode-file://vscode-app/c:/Users/User/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html): Remove from the rear  
  Include docstrings and demonstrate usage with output.

CODE:



OUTPUT:



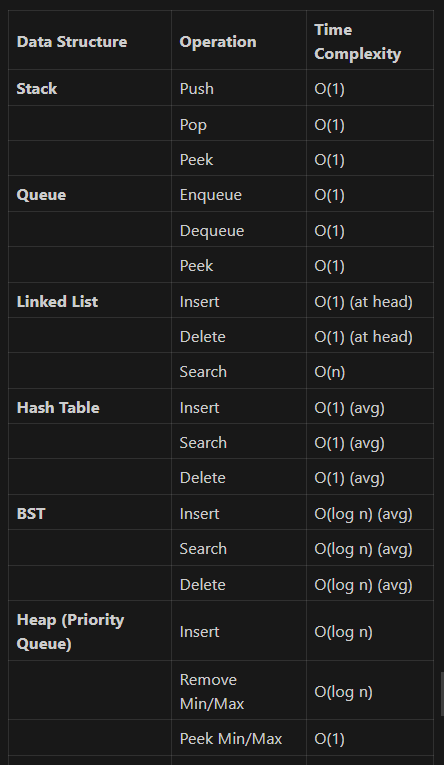
OBSERVATION:

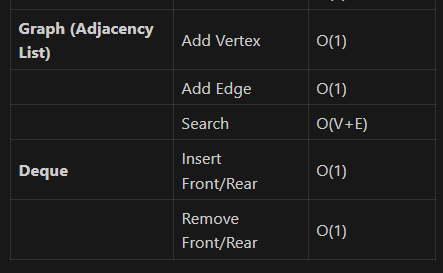
The code provides a correct double-ended queue implementation.  
All required methods are present and documented.The main block demonstrates inserting and removing items from both ends, and prints the deque contents after each operation, showing expected output.

TASK9:

PROMPT: Generate a markdown table comparing different data structures (stack, queue, linked list, etc.) including their operations and time complexities.

CODE:





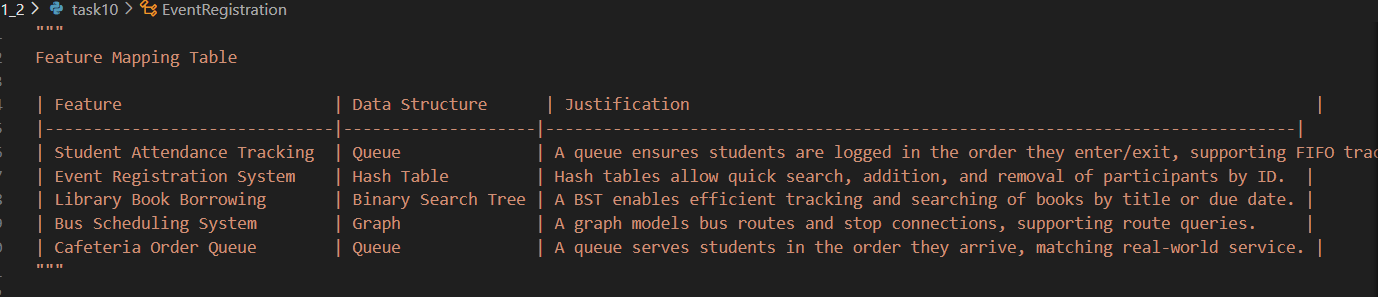
OBSERVATION:

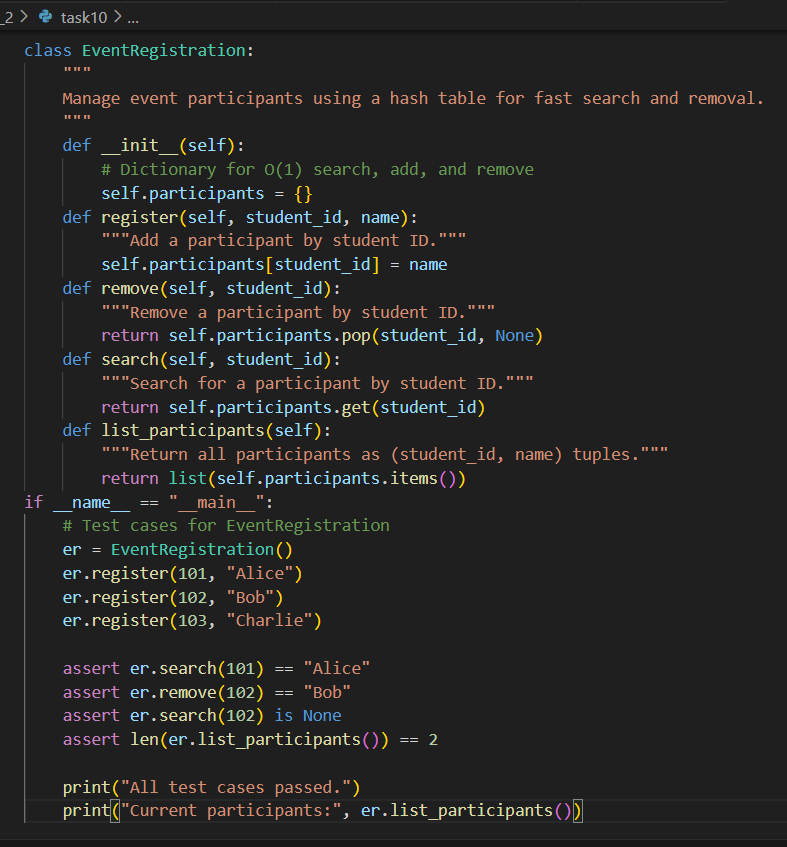
A markdown table was provided listing common data structures, their main operations, and the time complexity for each. The table is clear, covers all requested structures, and is suitable for documentation or study reference.

TASK10:

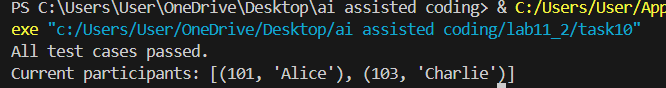
PROMPT: For each campus management feature, select the most appropriate data structure (stack, queue, priority queue, linked list, BST, graph, hash table, deque), justify your choice, and implement one feature in Python with test cases and analysis.

CODE:





OUTPUT:



OBSERVATION:

The file contains a feature-to-data structure mapping table with justifications ,a functional Python implementation of the Event Registration System using a hash table ,assert-based test cases, and output verification.All tests pass, and the code is well-documented and ready for reporting.