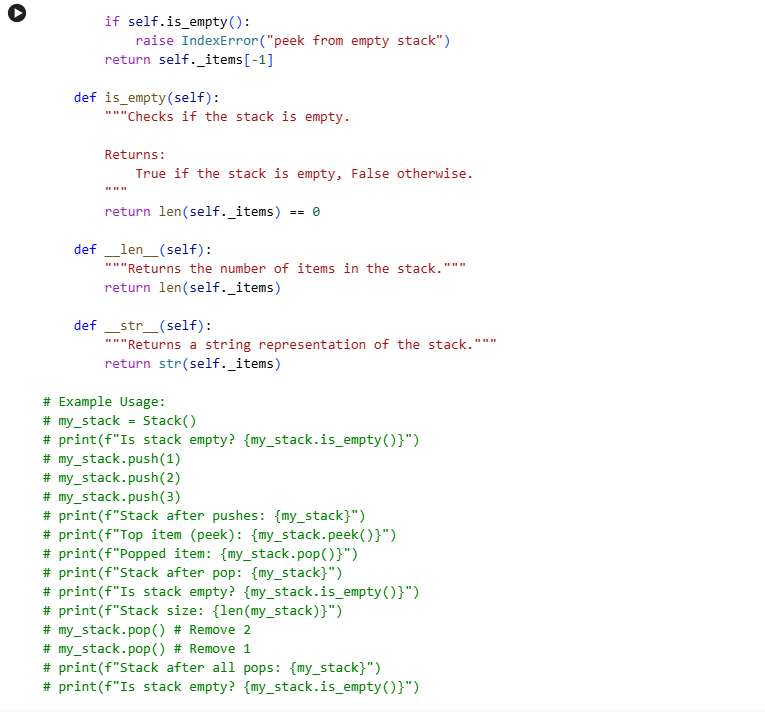
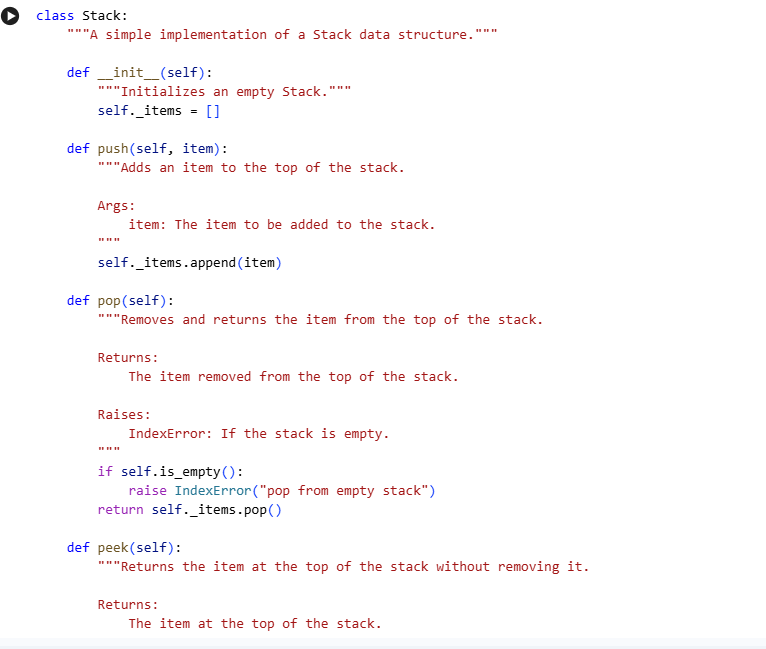
H.T.NO:2303A51656

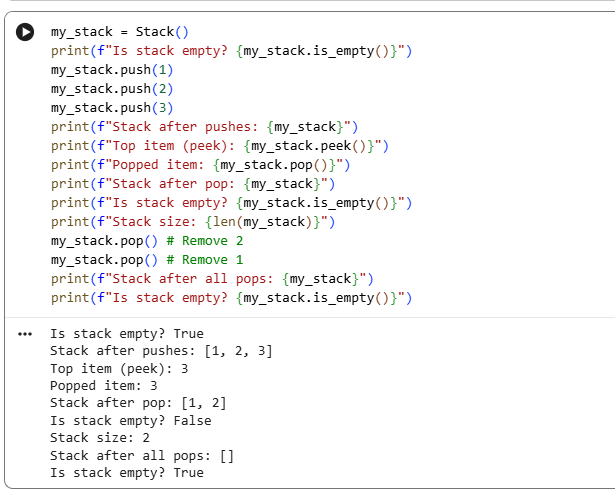
BATCH 28

TASK 1:

Prompt:

Use AI to generate a Stack class with push, pop, peek, and is\_empty  
methods.

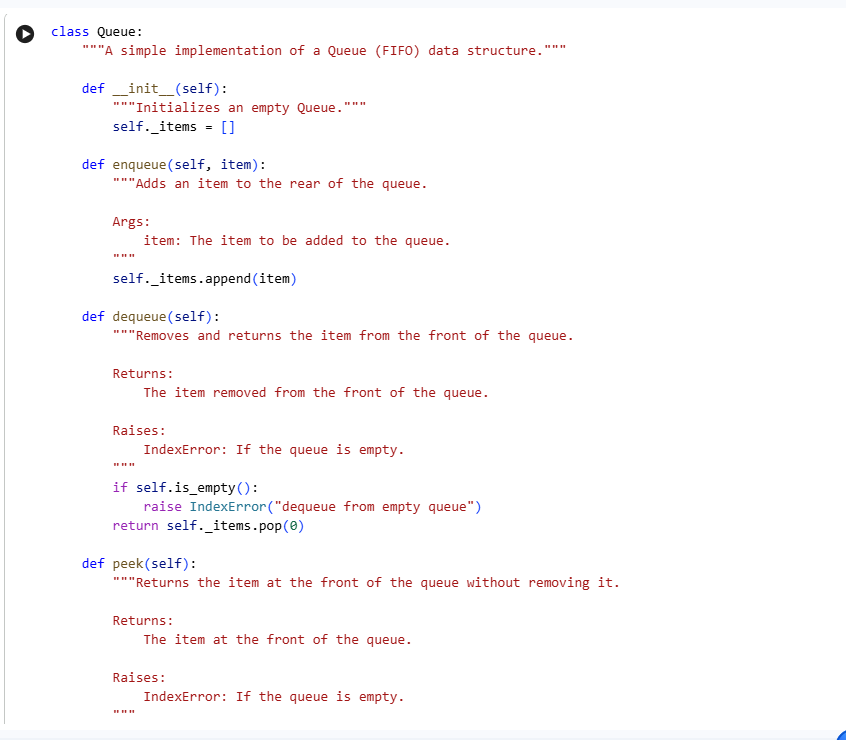


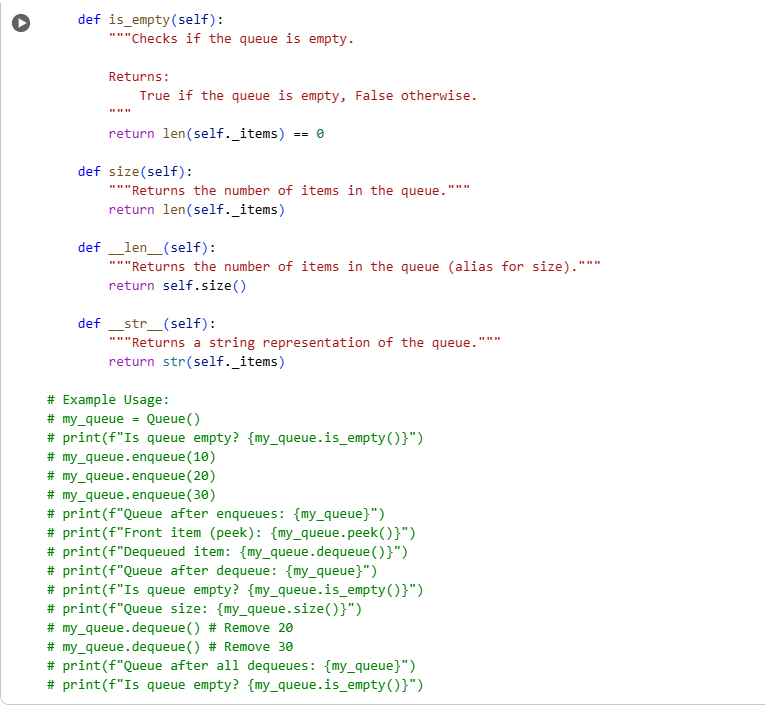


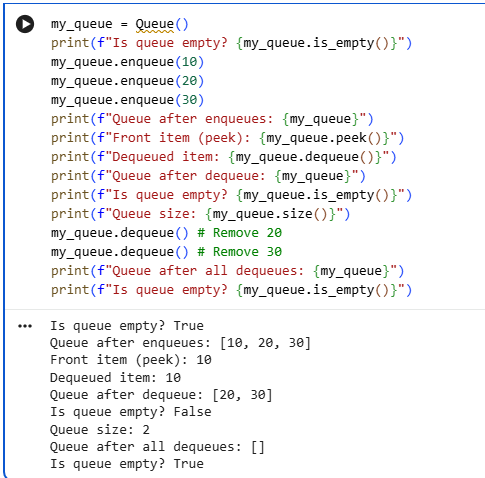
TASK 2:

Prompt:

Use AI to implement a Queue using Python lists.



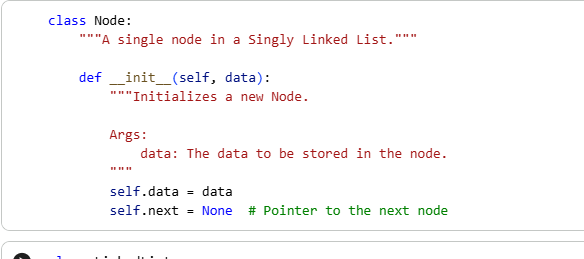


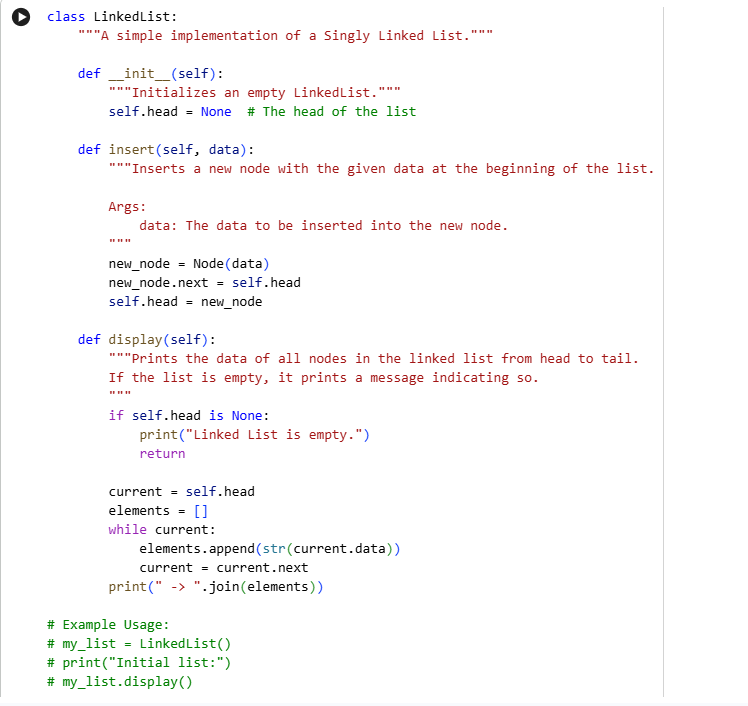


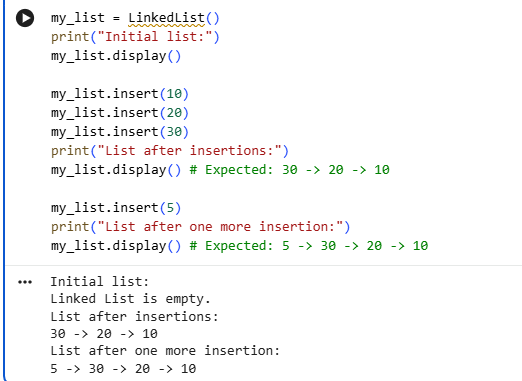
TASK 3:

Prompt:

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



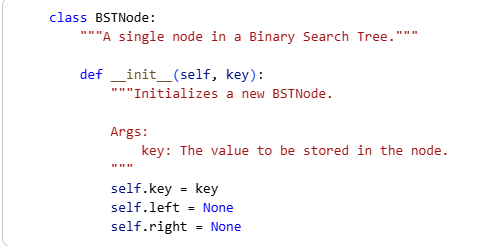


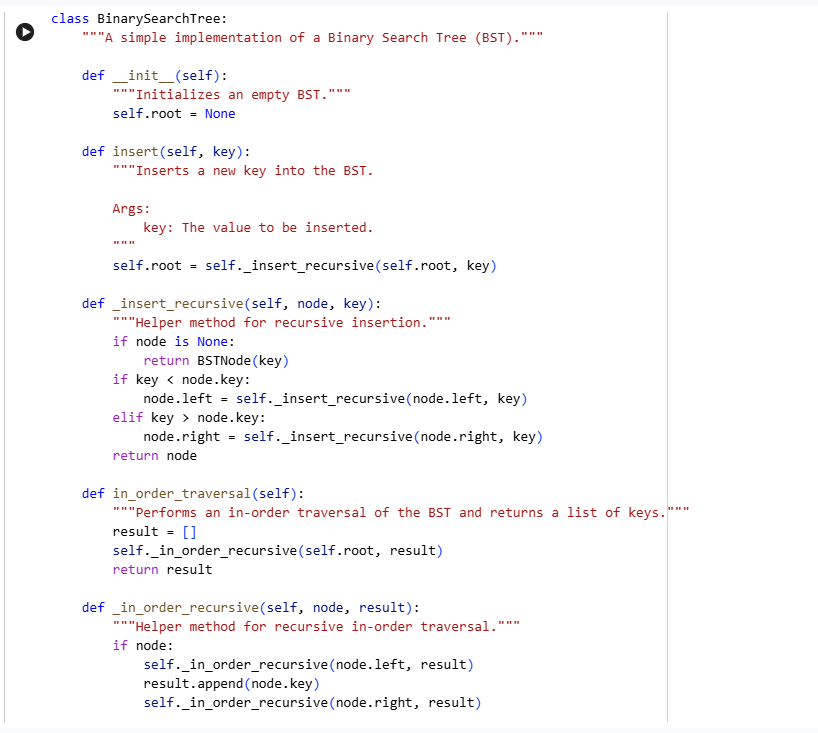


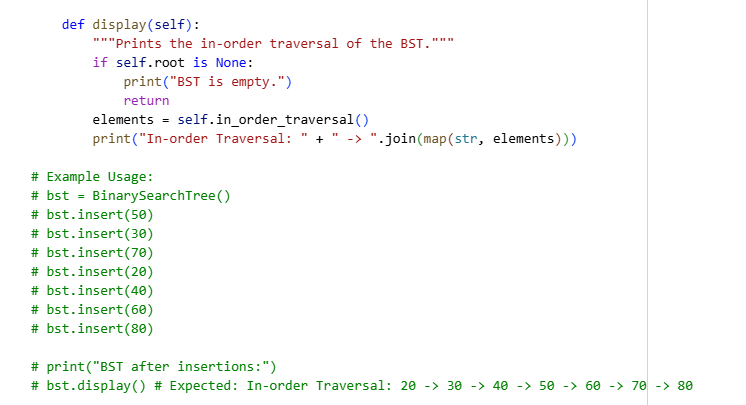
TASK 4:

Prompt:

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



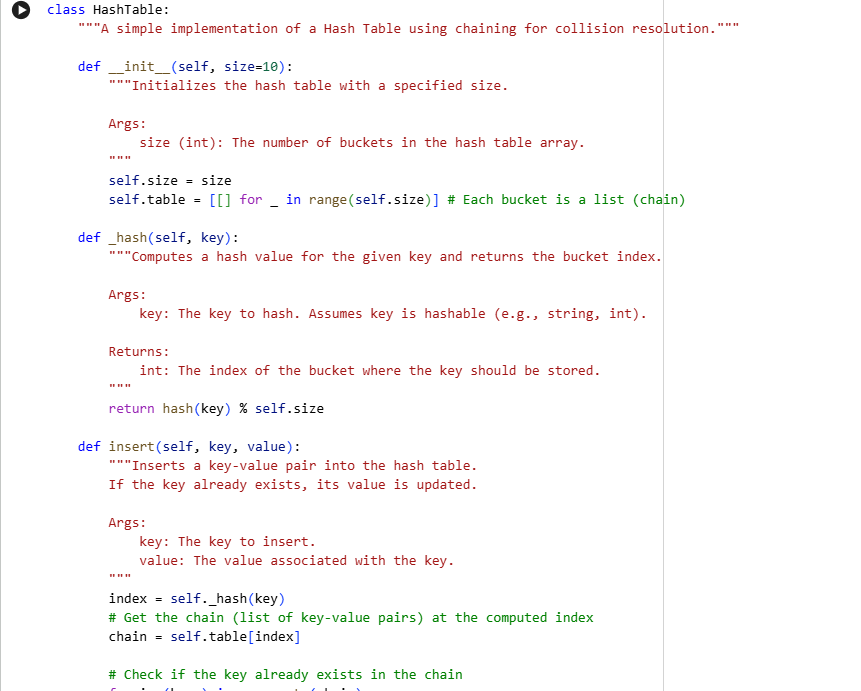


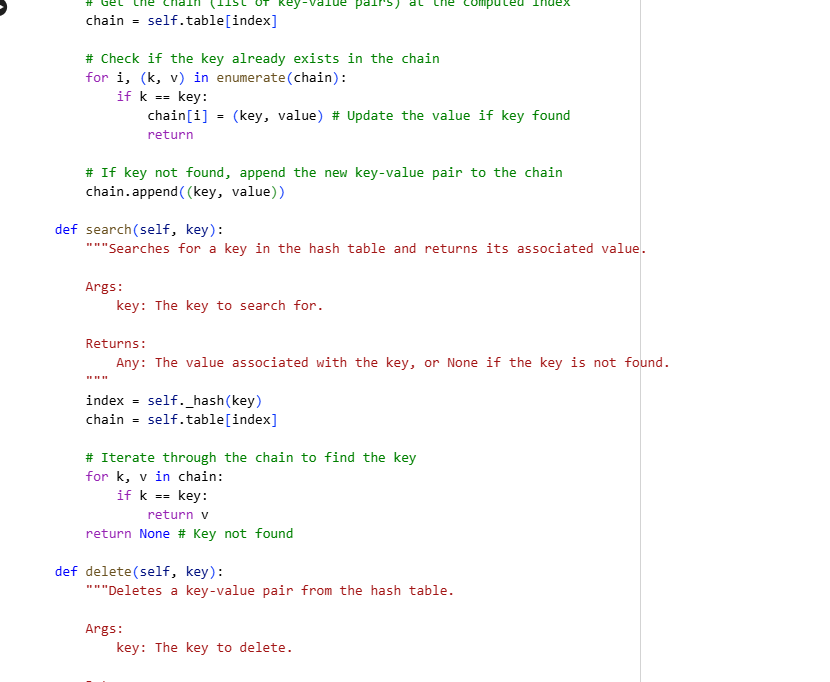




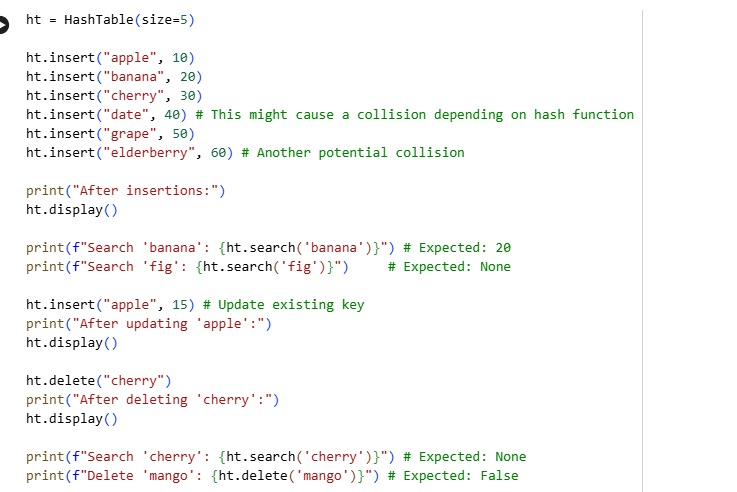
TASK 5:

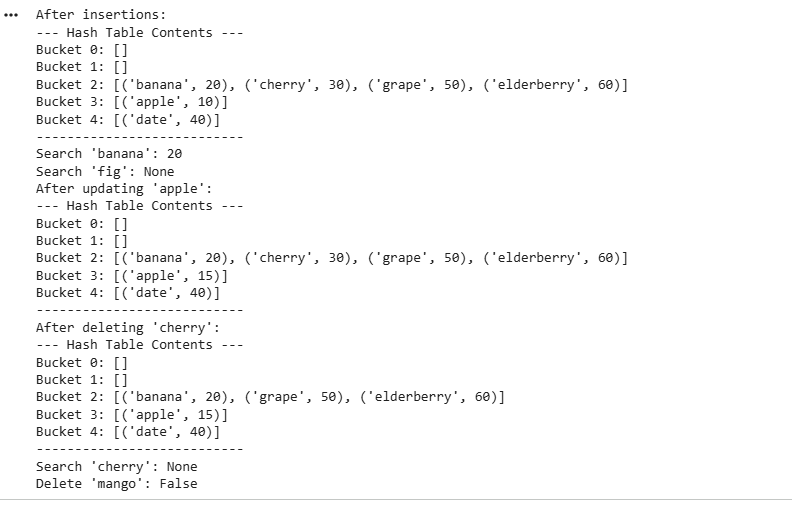
Use AI to implement a hash table with basic insert, search, and delete methods.







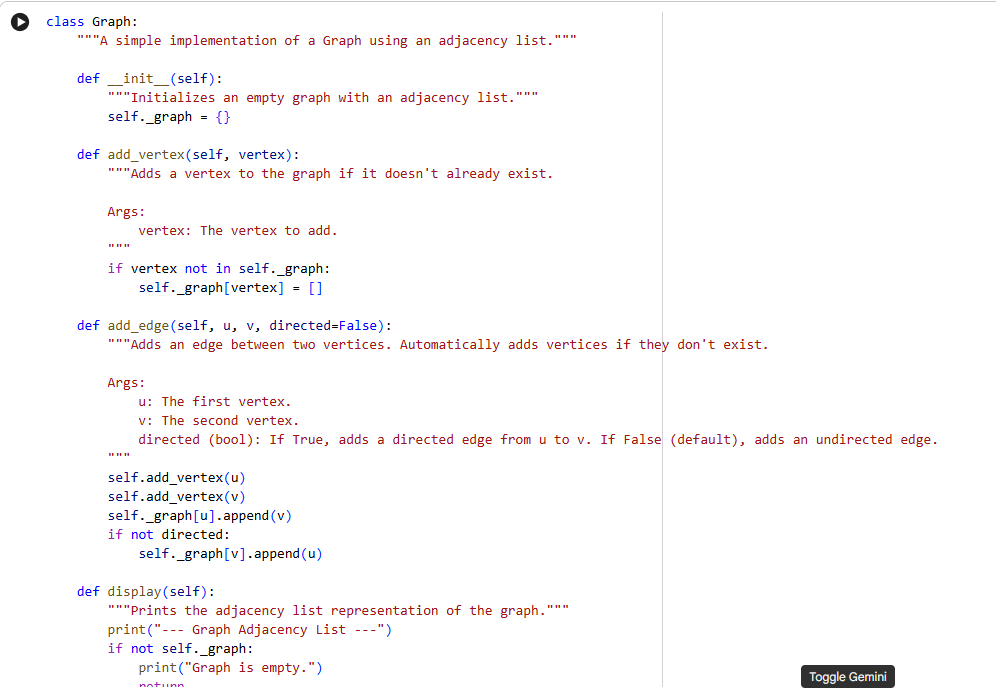


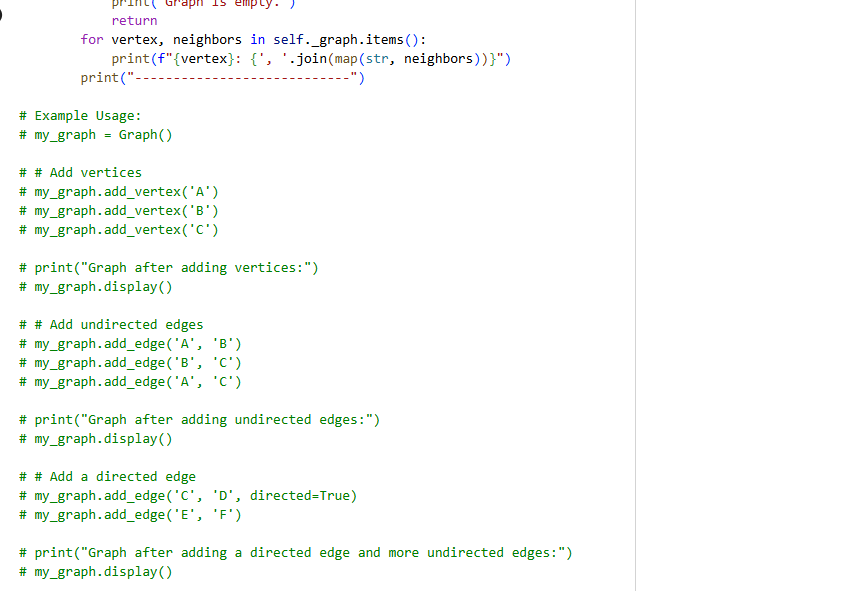
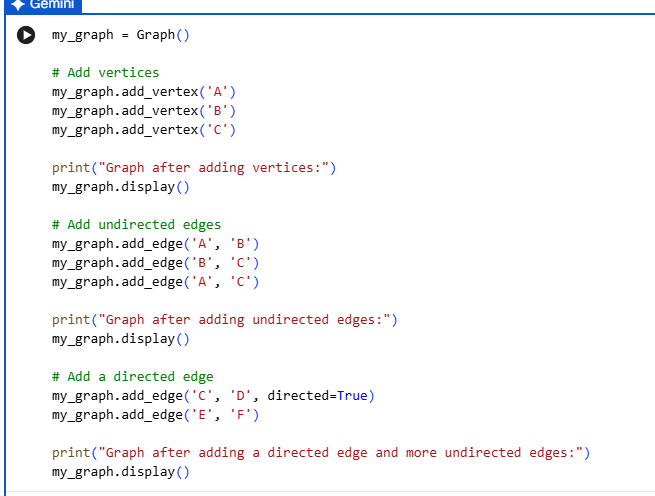


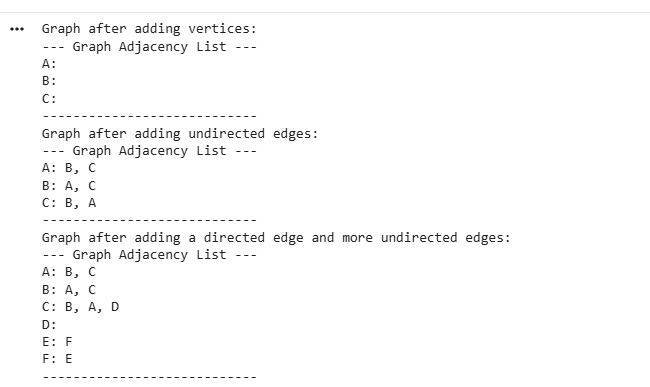
TASK 6:

Prompt:

Use AI to implement a graph using an adjacency list.



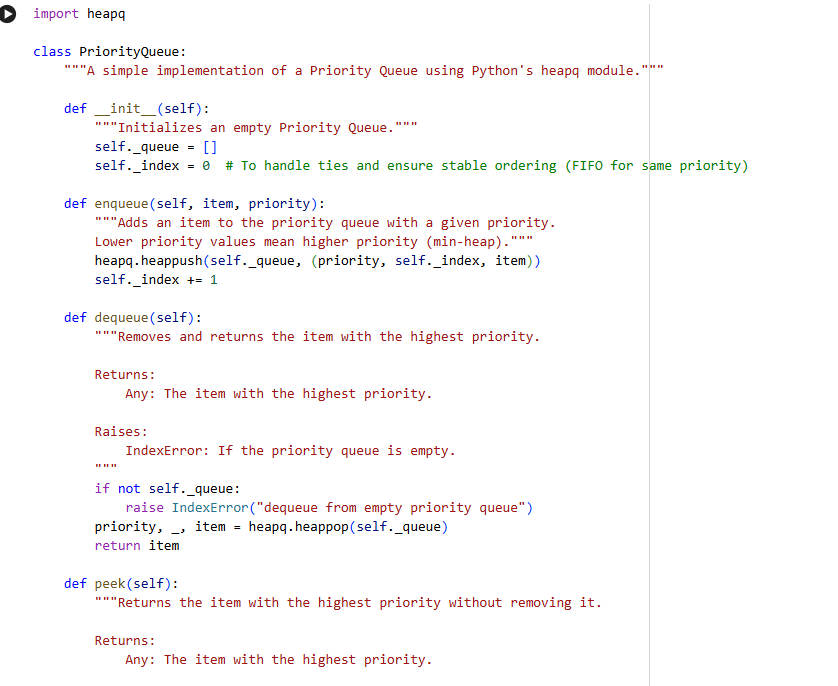


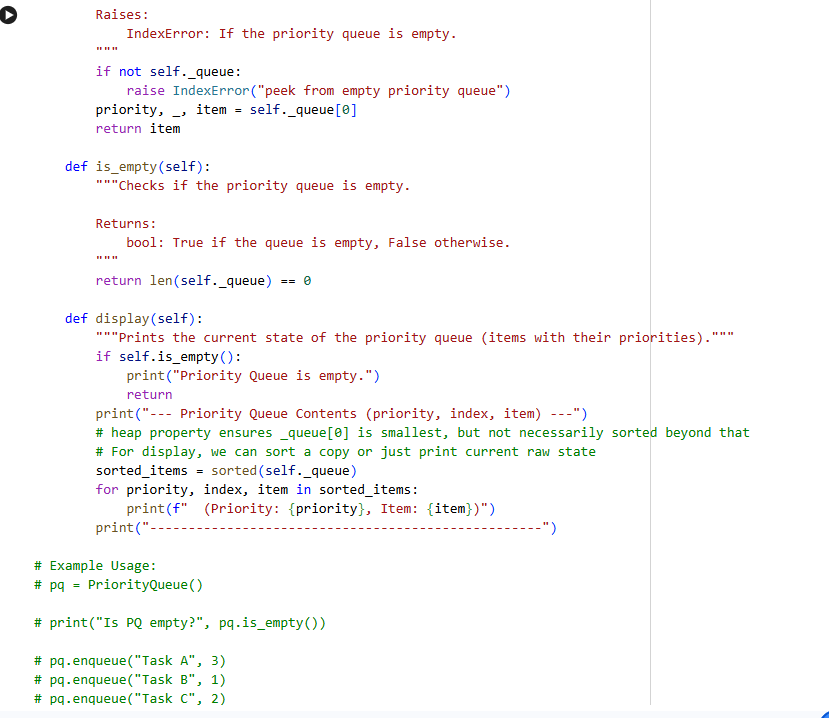


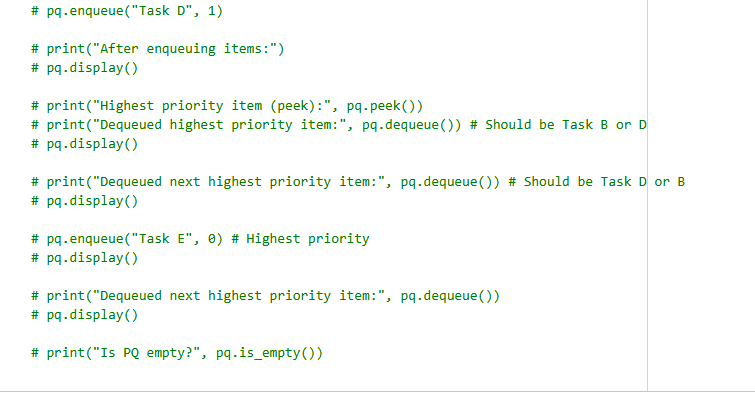
TASK 7:

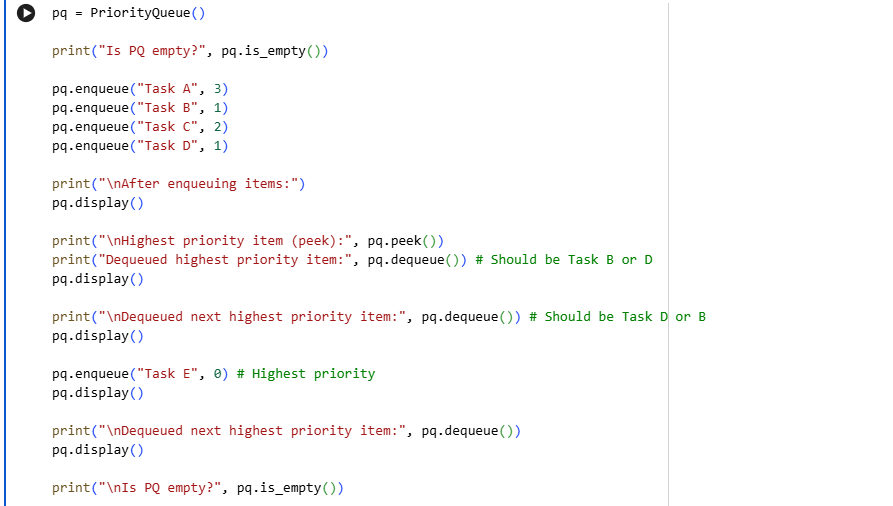
Prompt:

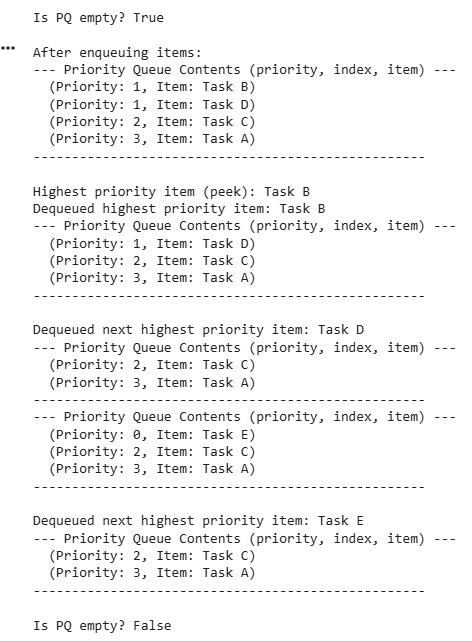
Use AI to implement a priority queue using Python’s heapq module.









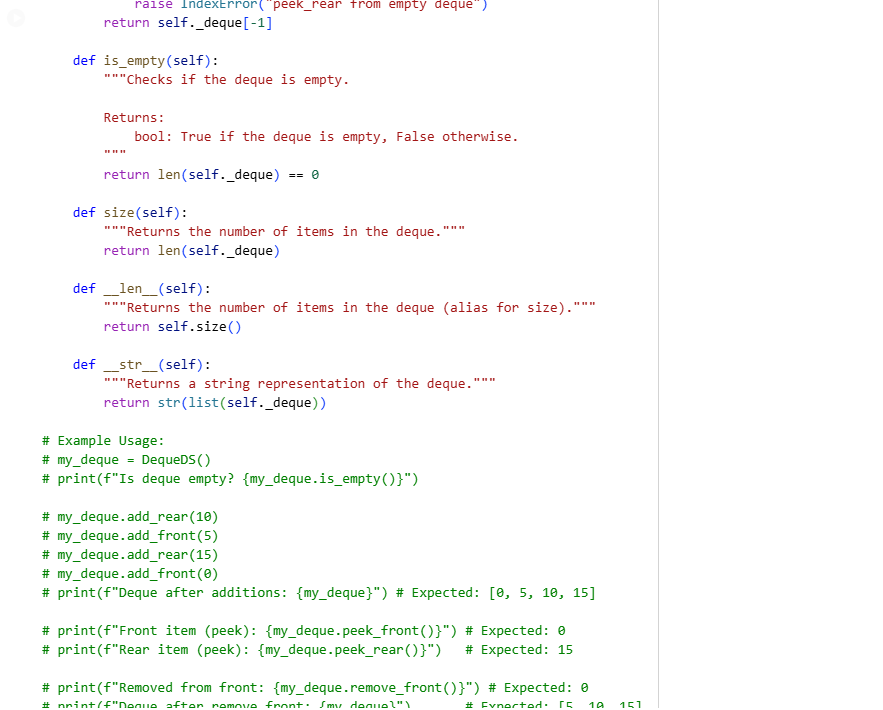


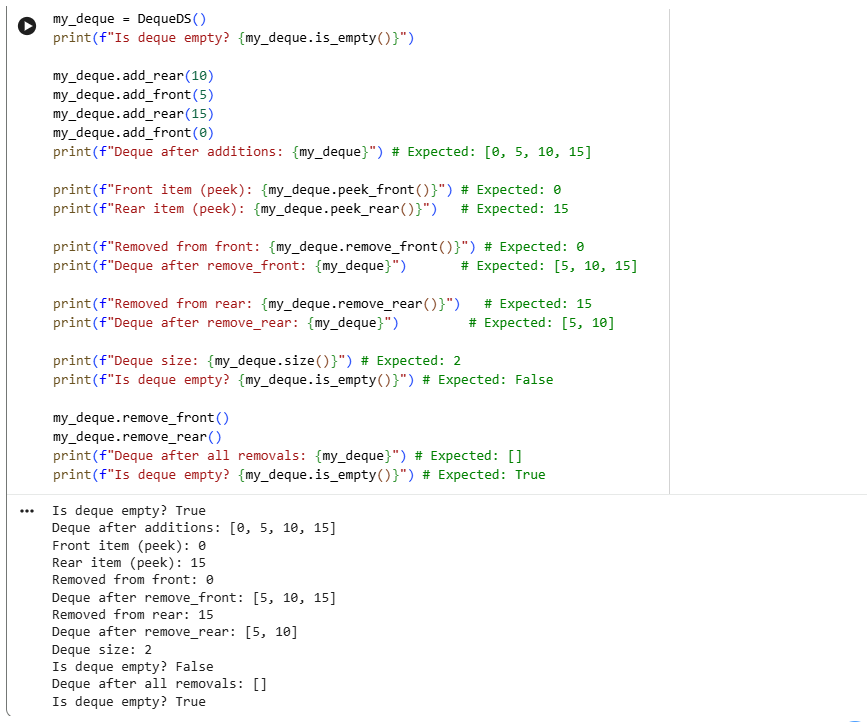
TASK 8:

Prompt:

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



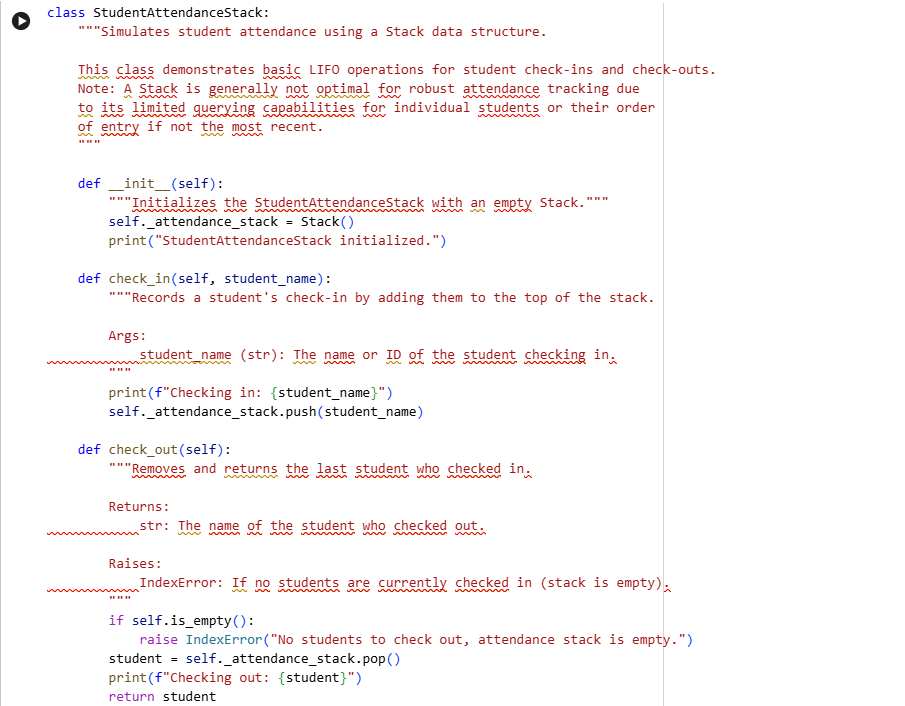


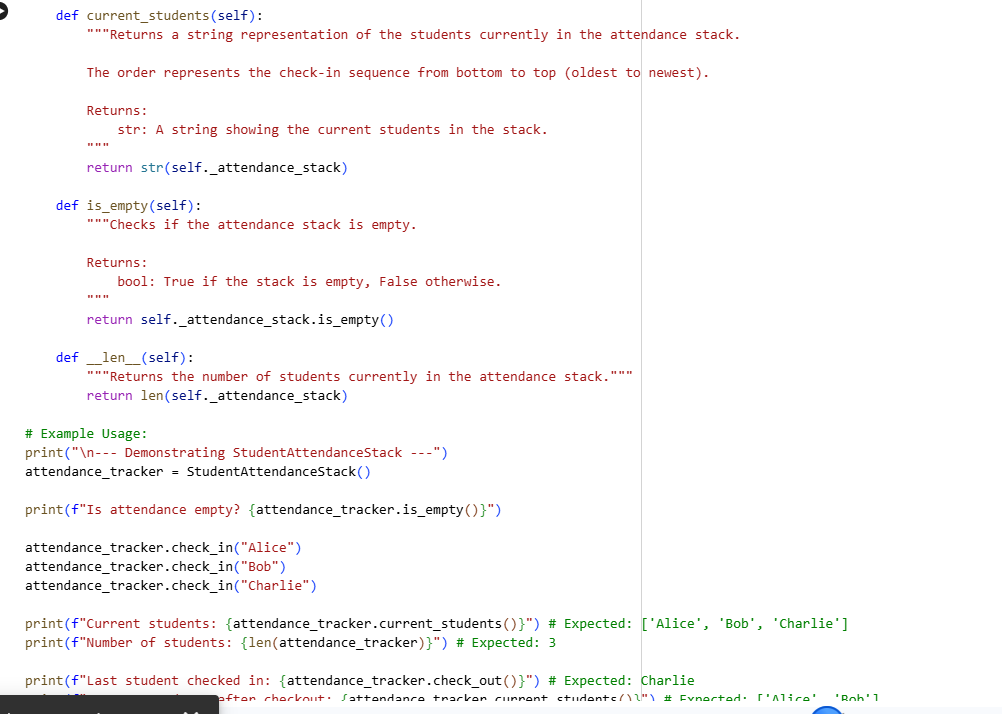


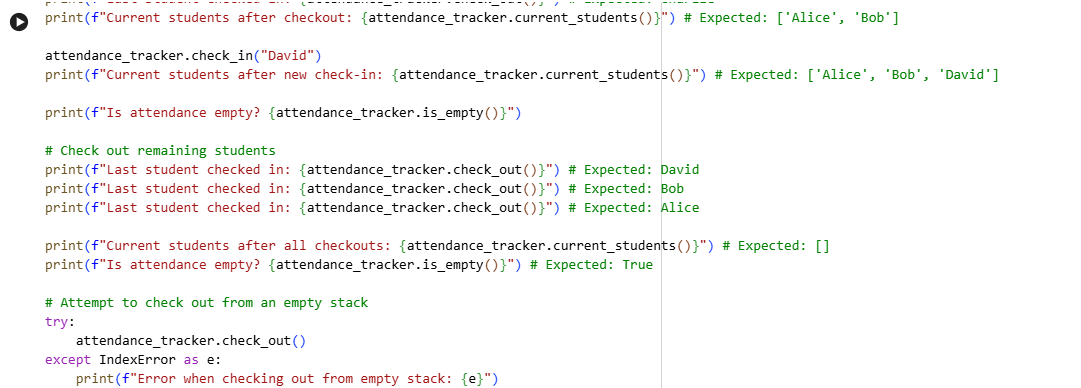
TASK 9:

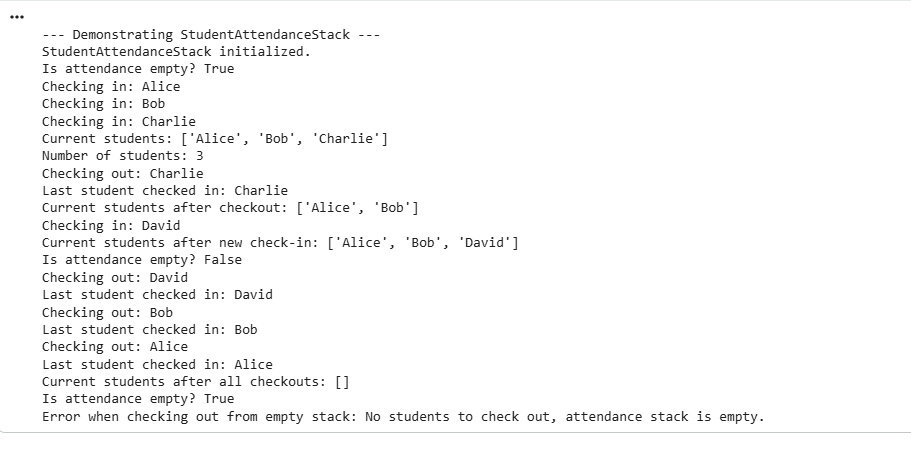
Prompt:

Try to generate a python program for student attendance tracking by using stack, Event registration system using queues, Library book borrowing by using priority queue, Bus scheduling system using stack,Cafeteria order queue using queues.









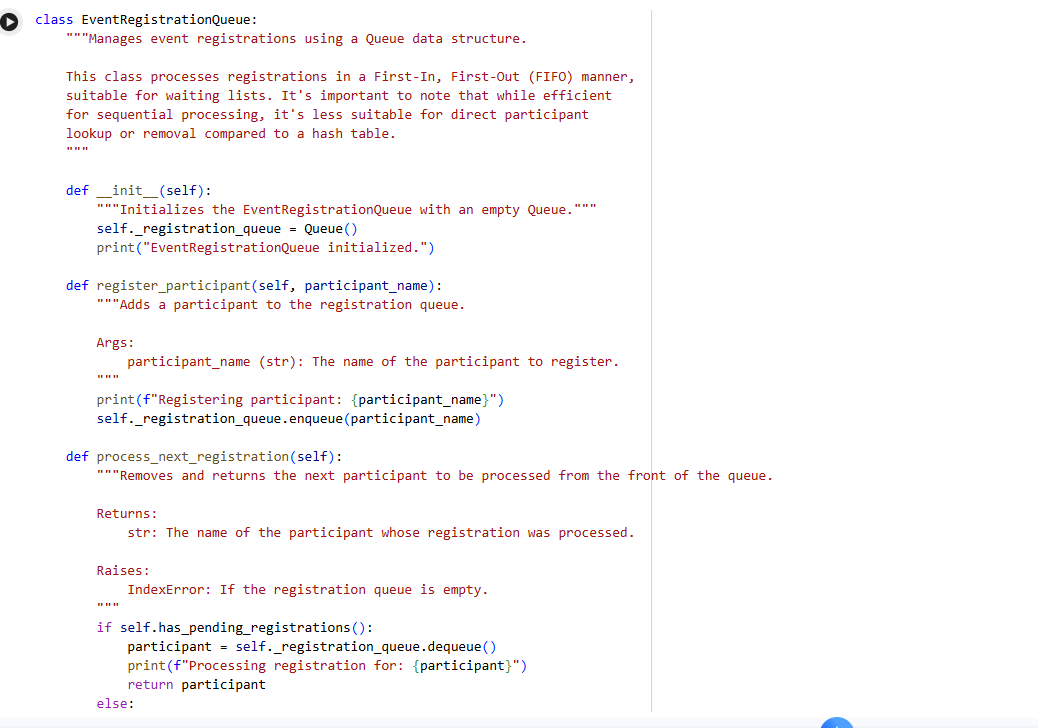
Event Registration System (Queue)

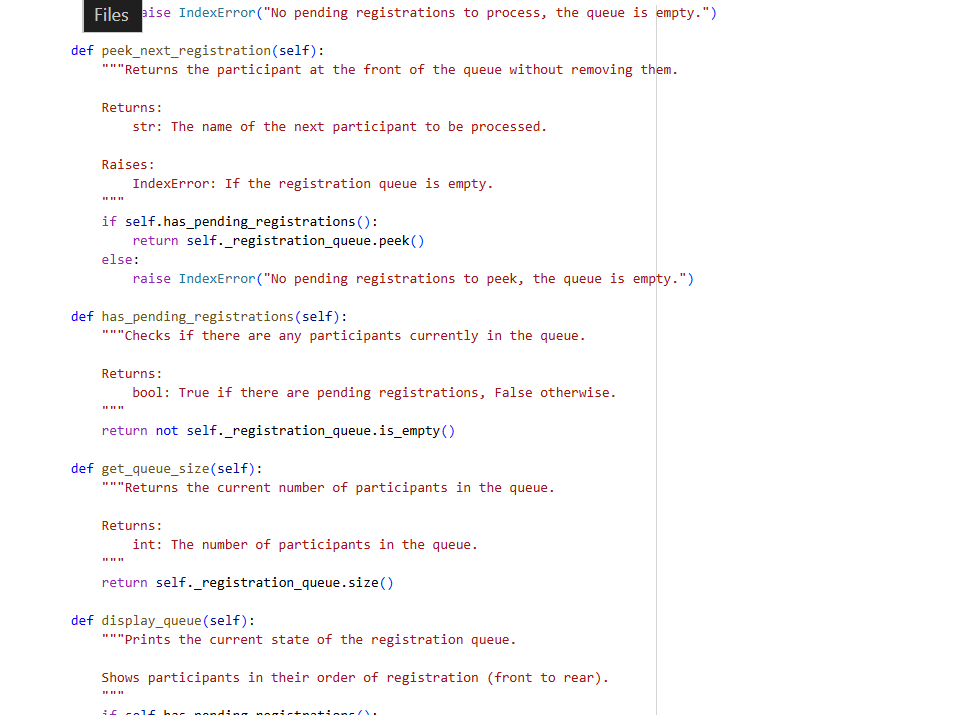
Subtask:

Implement an EventRegistrationQueue class using a Queue to manage participants. This will process registrations in a FIFO manner, suitable for waiting lists, but less efficient for direct participant lookup or removal compared to a hash table.

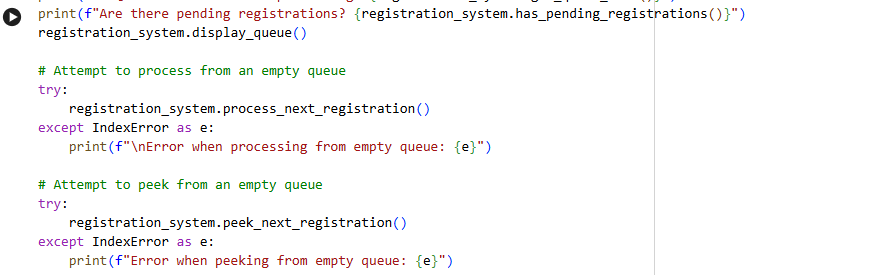
**Reasoning**: I will define the EventRegistrationQueue class, leveraging the Queue class previously established in the notebook. This implementation will include methods for adding participants (register\_participant), processing them in FIFO order (process\_next\_registration), peeking at the next participant (peek\_next\_registration), checking for pending registrations (has\_pending\_registrations), querying the queue size (get\_queue\_size), and displaying the queue's current state (display\_queue). All methods will feature docstrings, comments, and comprehensive example usage, including error handling for an empty queue.

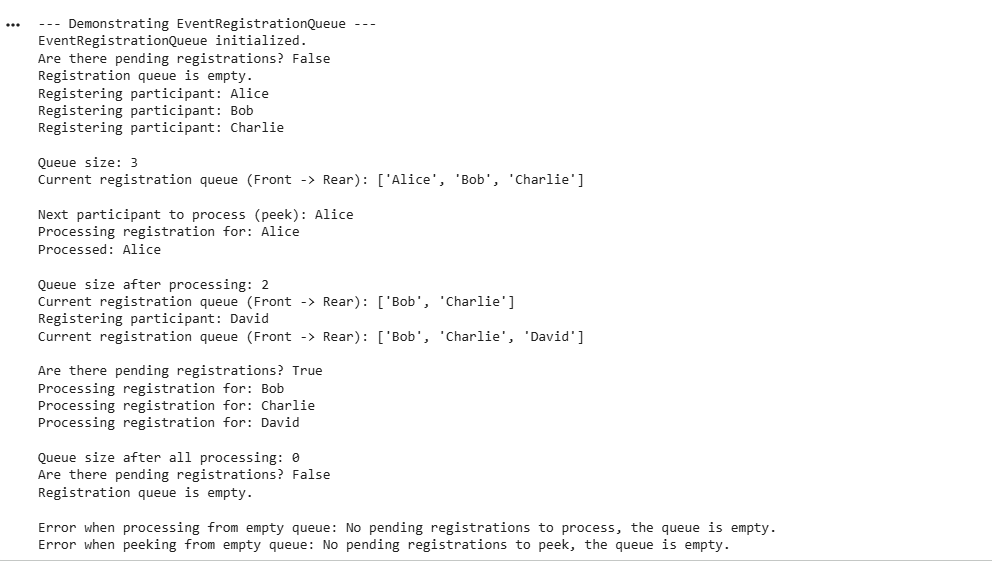
**Reasoning**: I need to implement the EventRegistrationQueue class as instructed, utilizing the Queue class for its core functionality and ensuring all specified methods, docstrings, comments, and example usage are included.









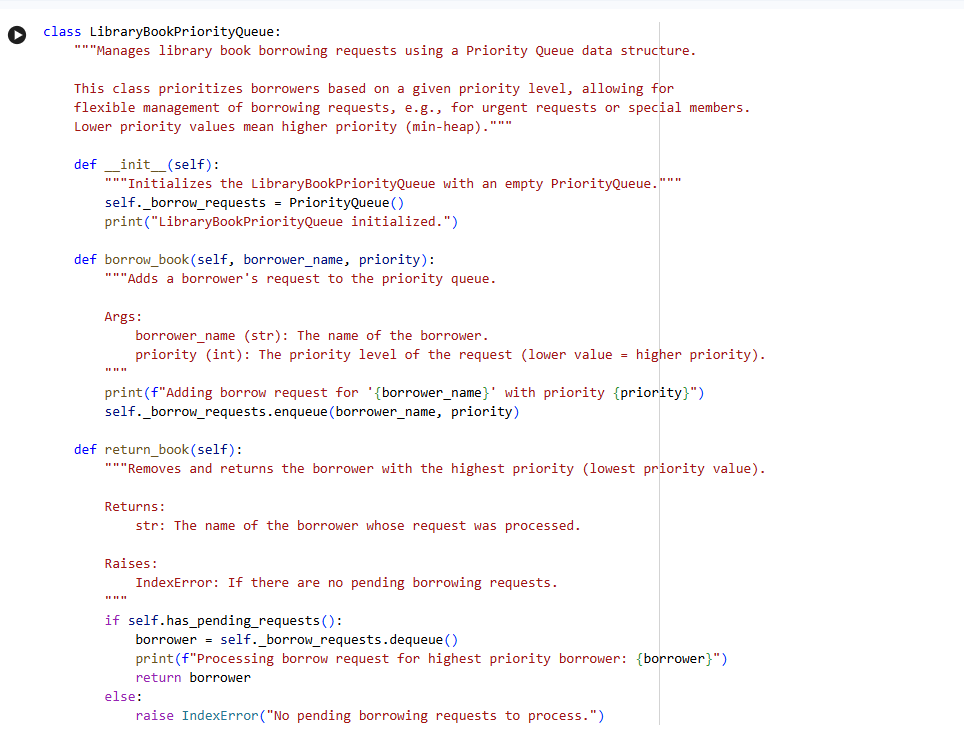


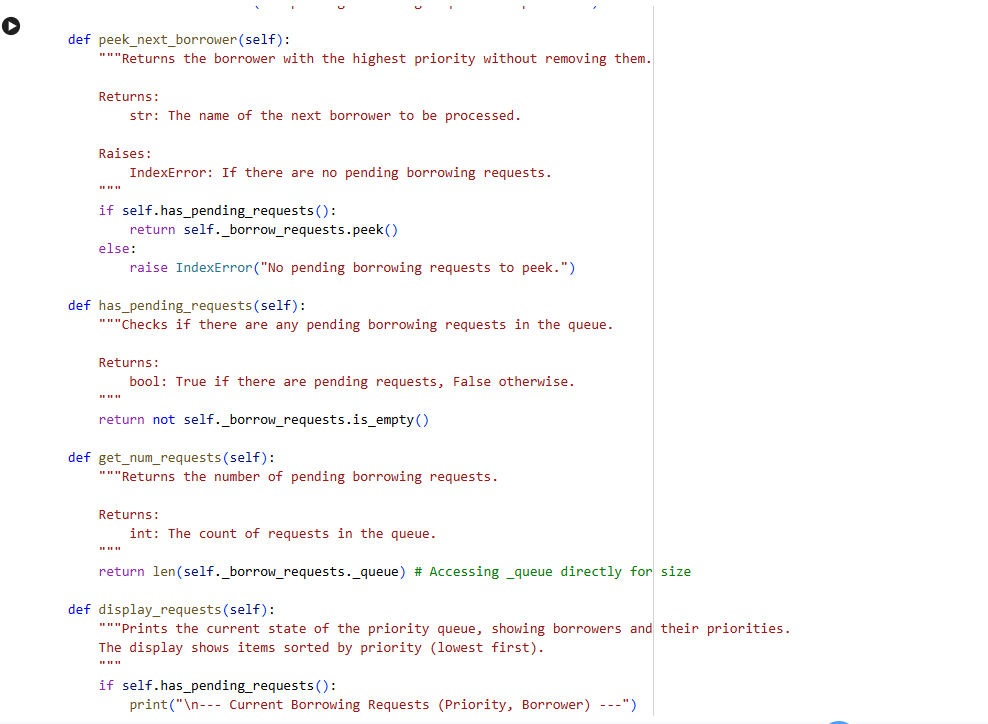
Library Book Borrowing (Priority Queue)

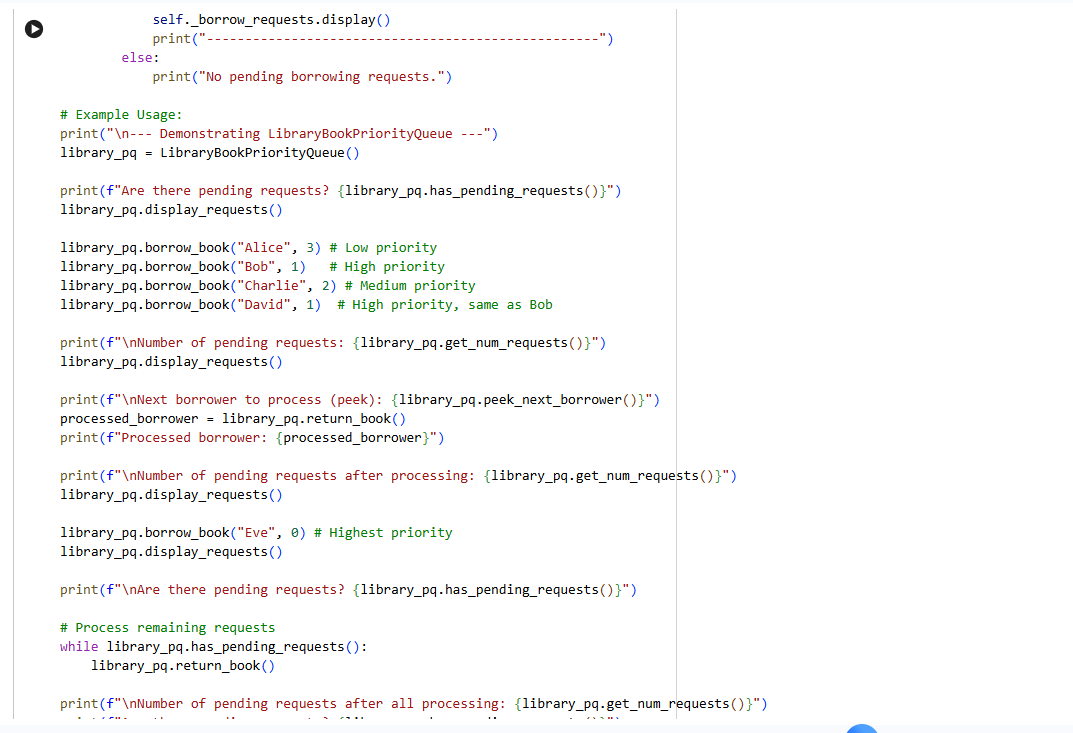
Subtask:

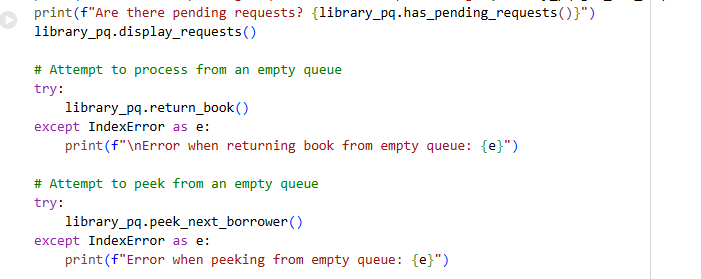
Implement a LibraryBookPriorityQueue class that uses a Priority Queue to manage book borrowing. This will allow for prioritization of borrowers, for instance, based on their status or time of request.

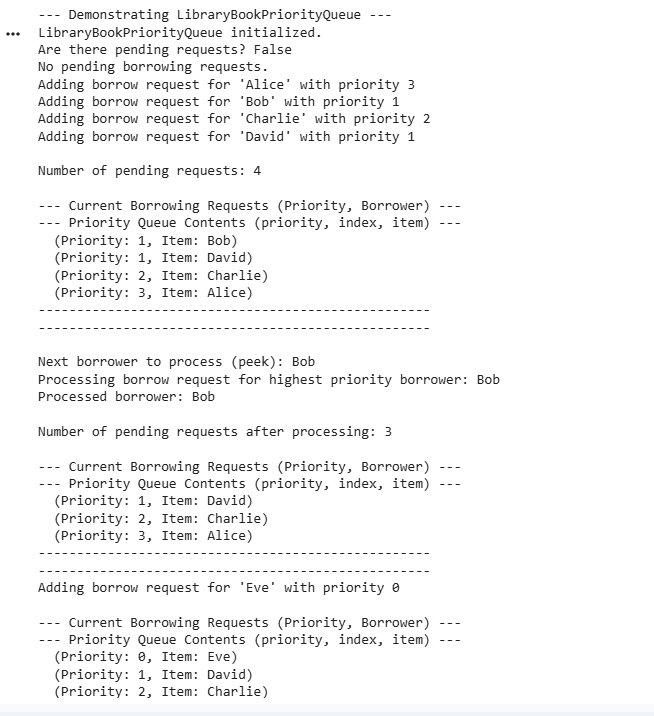
**\*\*Reasoning\*\***:  
I will define the `LibraryBookPriorityQueue` class, leveraging the `PriorityQueue` class previously established in the notebook.This implementation will include methods for adding borrowers (`borrow\_book`), processing them based on priority (`return\_book`), peeking at the next borrower (`peek\_next\_borrower`), checking for pending requests (`has\_pending\_requests`), querying the queue size (`get\_num\_requests`), and displaying the queue's current state (`display\_requests`). All methods will feature docstrings, comments, and comprehensive example usage, including error handling for an empty queue.

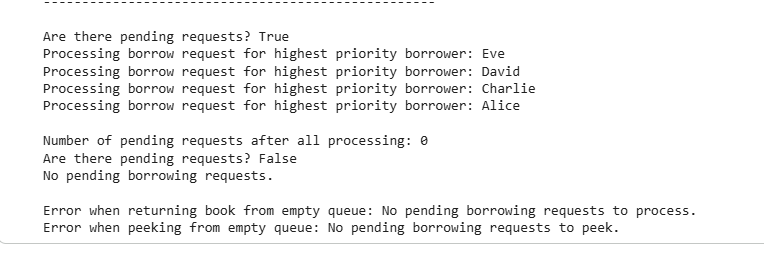










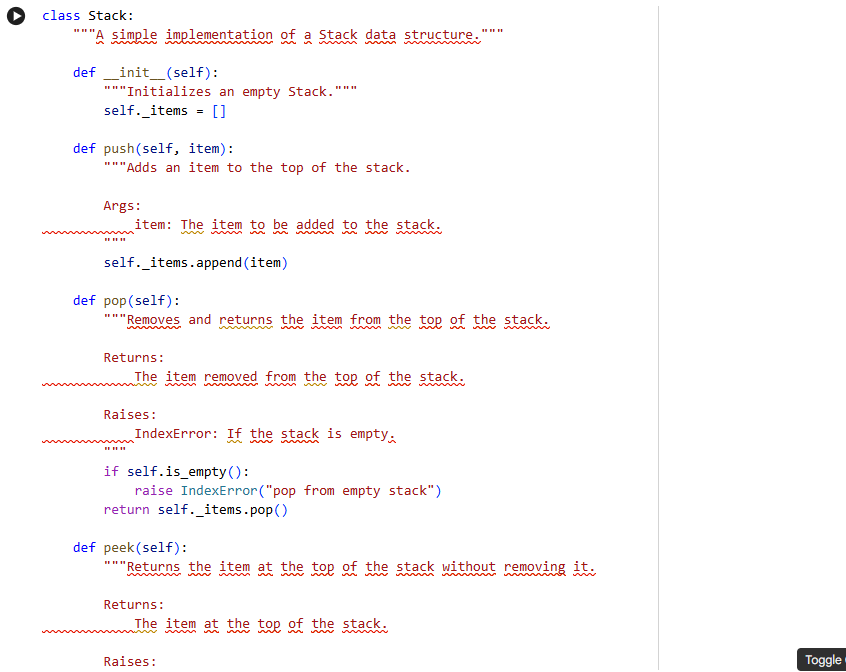


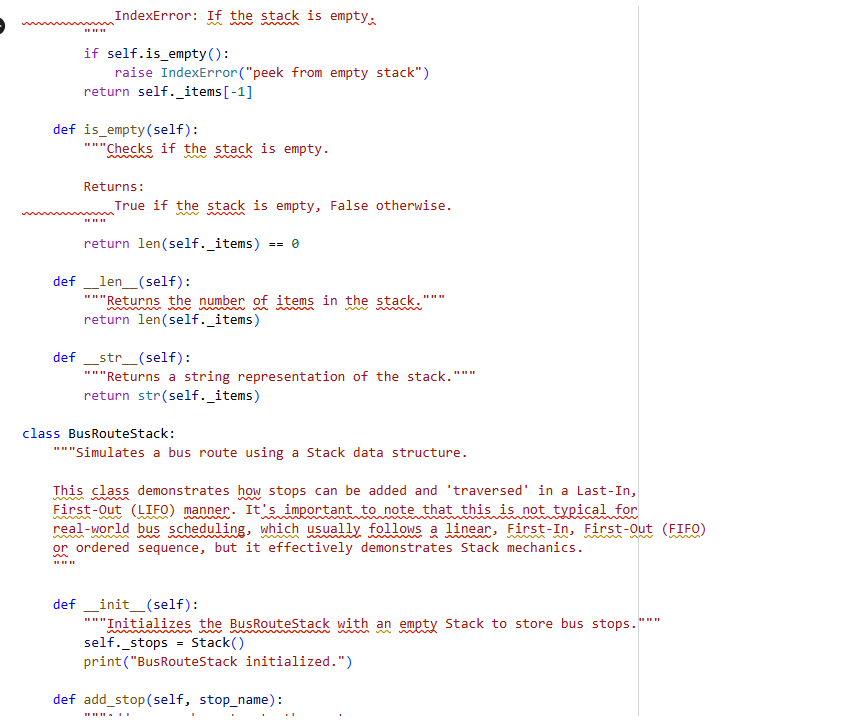
Bus Scheduling System (Stack)

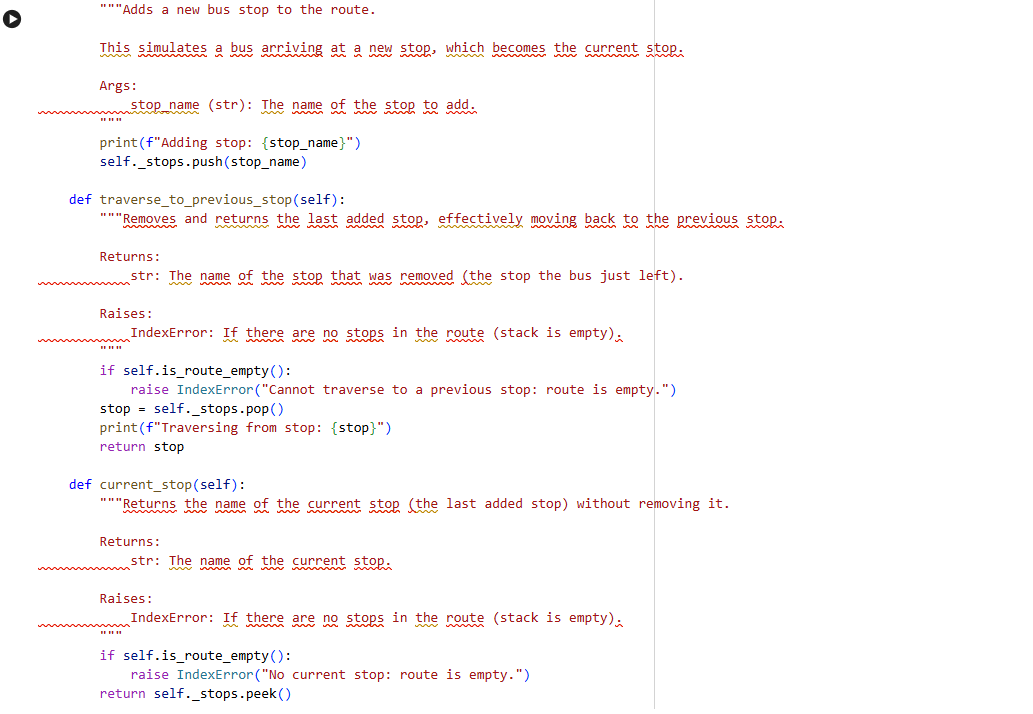
Subtask:

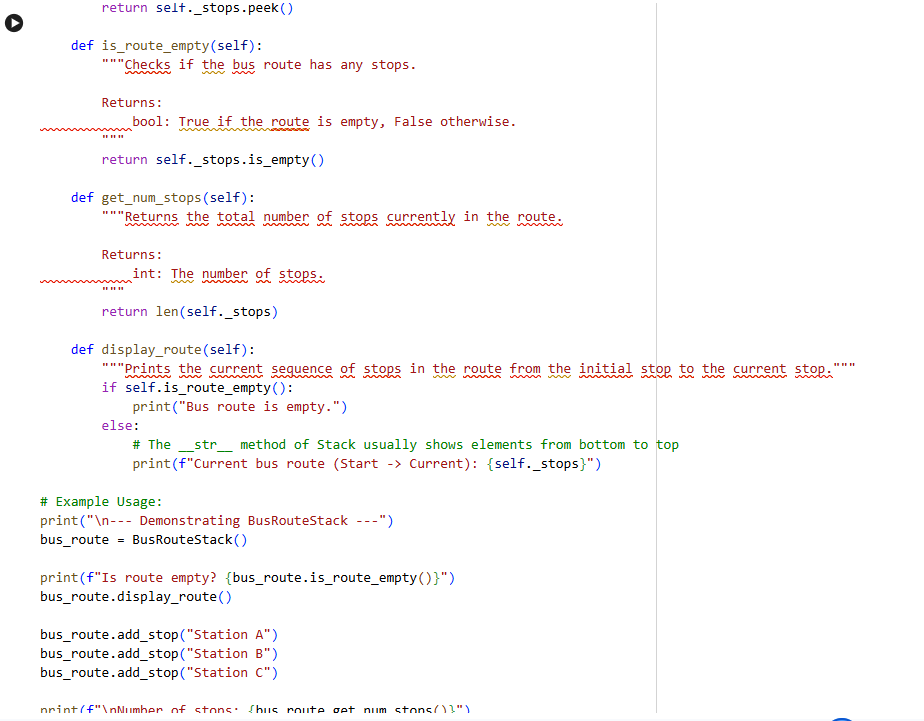
Implement a BusRouteStack class using a Stack to represent a bus route. This will show how stops can be added and 'traversed' in a LIFO manner, which is not typical for real-world bus scheduling but demonstrates Stack mechanics.

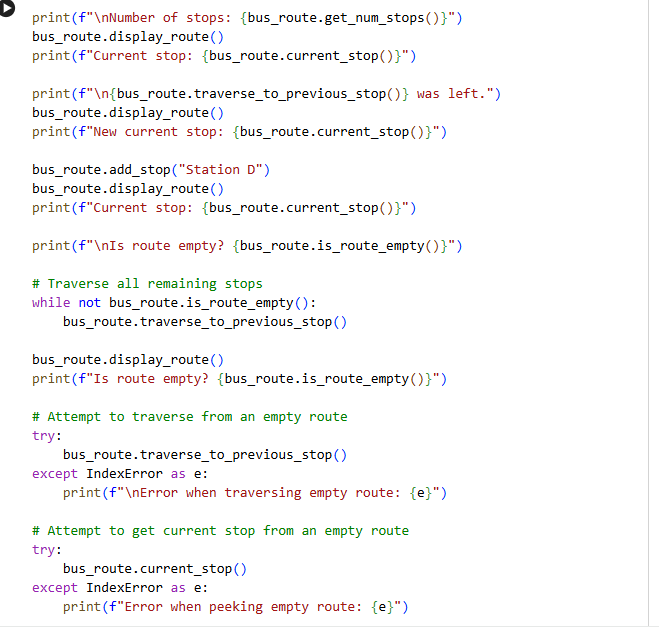
**Reasoning**: I need to define the BusRouteStack class, leveraging the Stack class previously established in the notebook. This implementation will include all the specified methods, docstrings, comments, and comprehensive example usage, including error handling for an empty route.









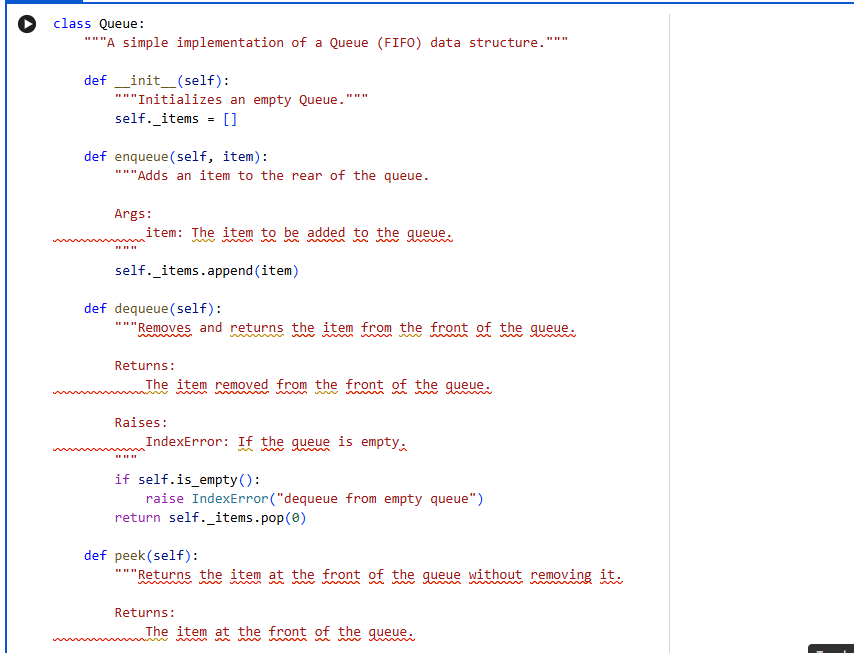


Cafeteria Order Queue (Queue)

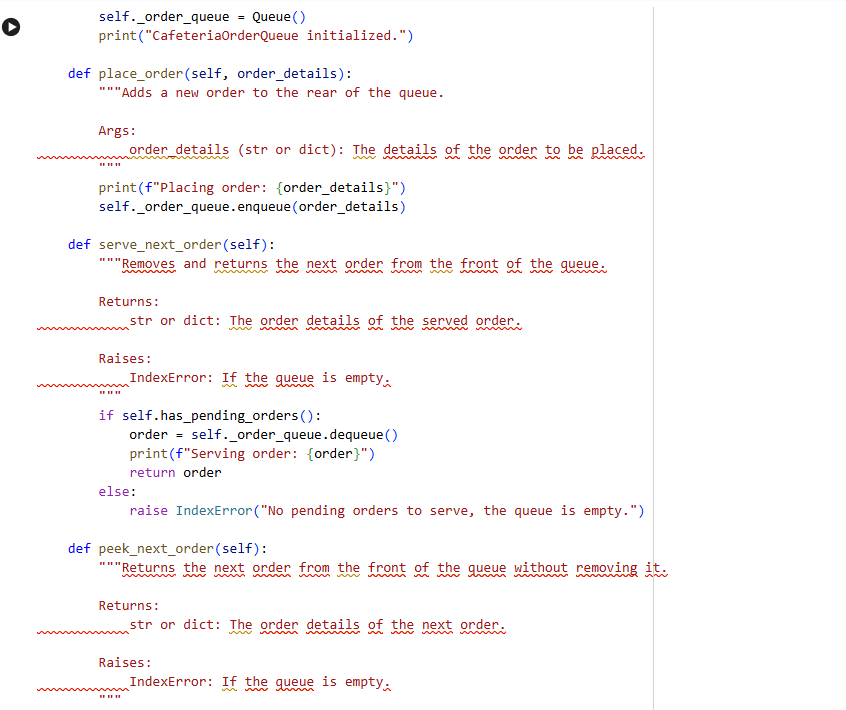
Subtask:

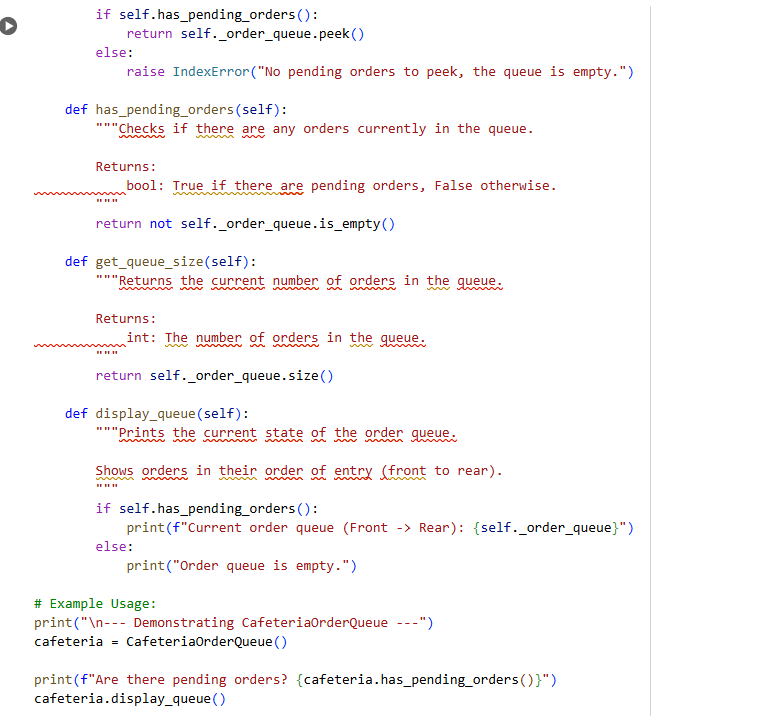
Implement a CafeteriaOrderQueue class using a Queue to handle orders. This is a highly appropriate use case, ensuring orders are served in a First-In, First-Out (FIFO) sequence.

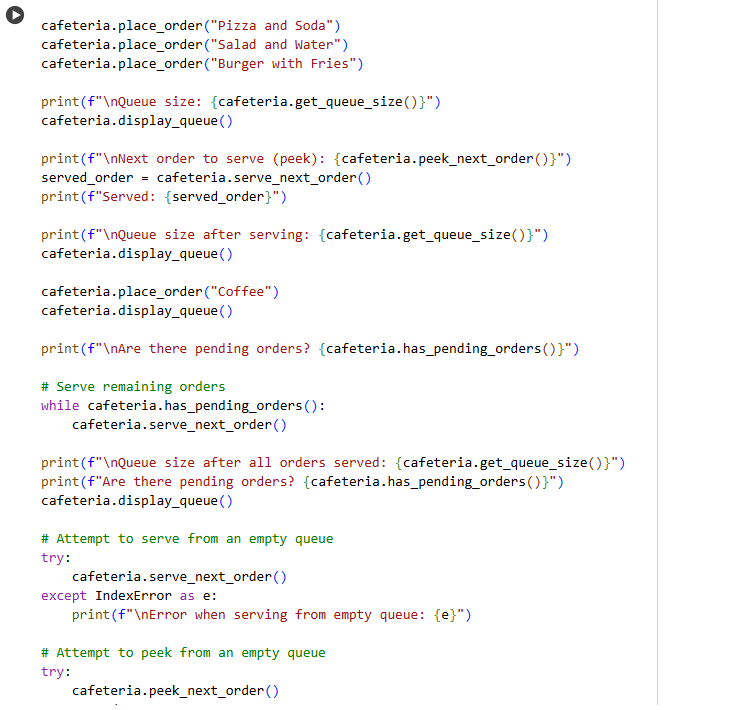
**Reasoning**: I need to define the CafeteriaOrderQueue class as instructed, utilizing the Queue class for its core functionality and ensuring all specified methods, docstrings, comments, and example usage are included. This will demonstrate the FIFO principle for managing cafeteria orders.



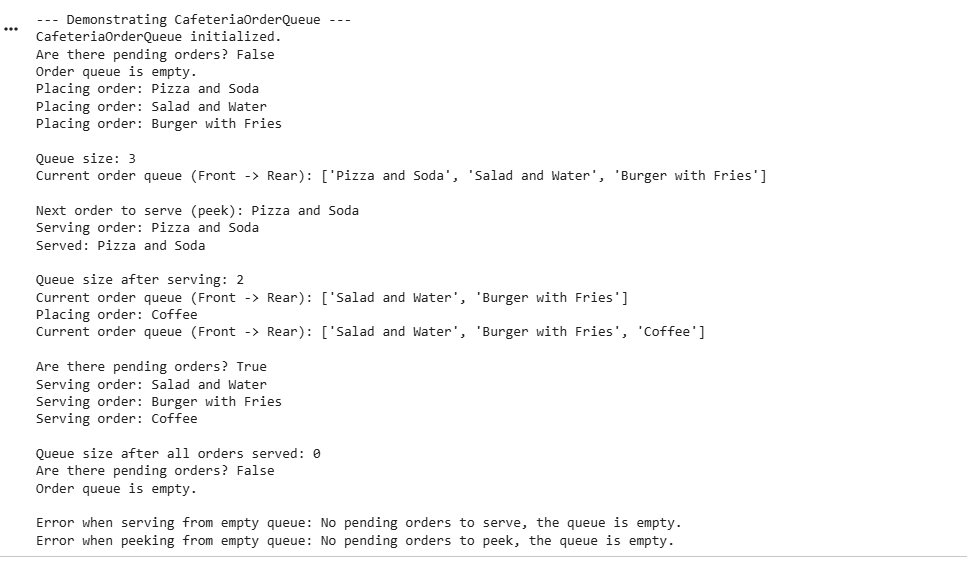


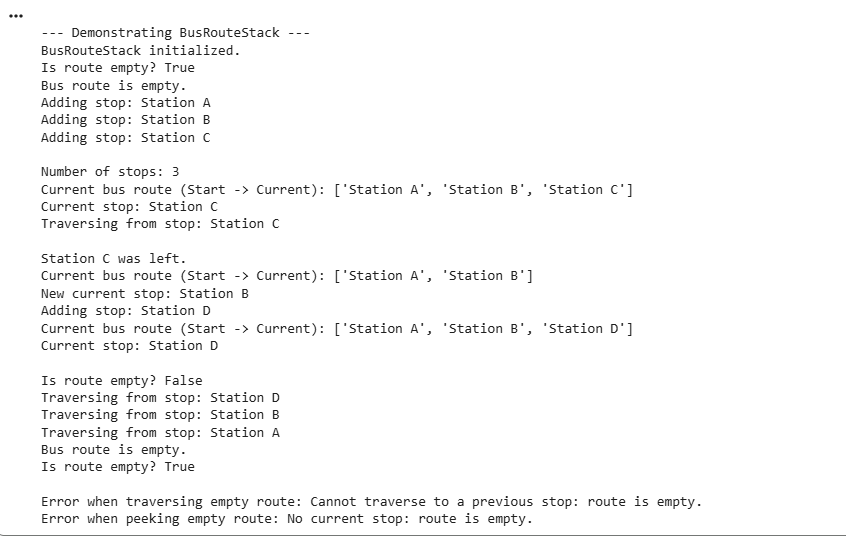








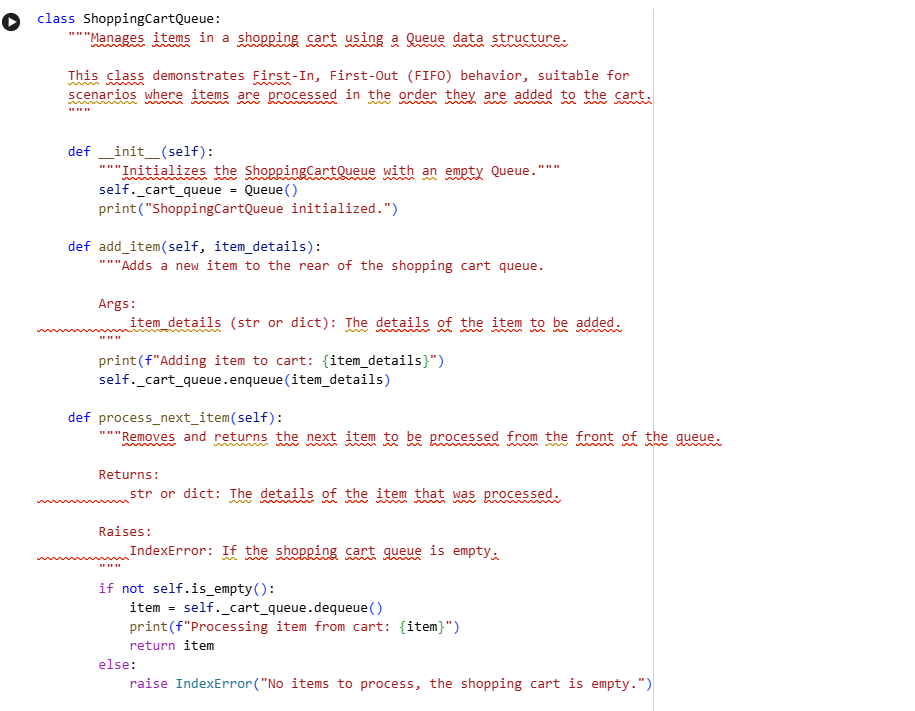


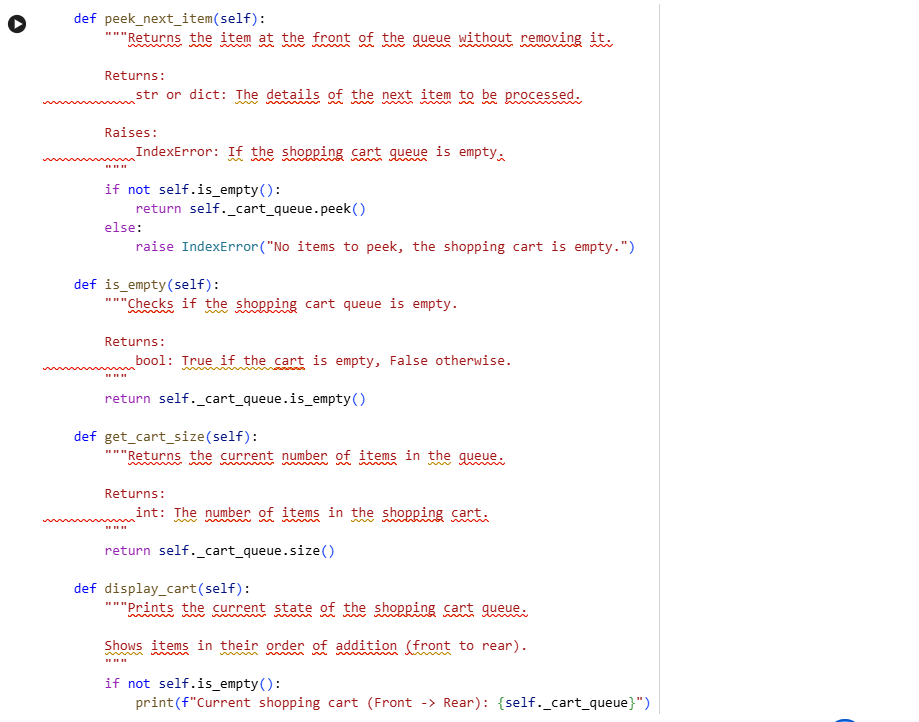


TASK 10:

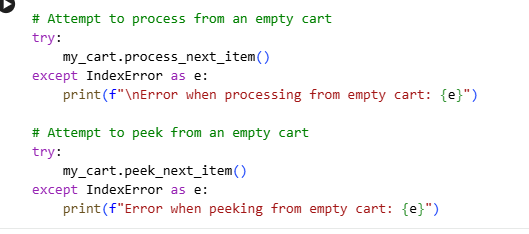
Prompt:

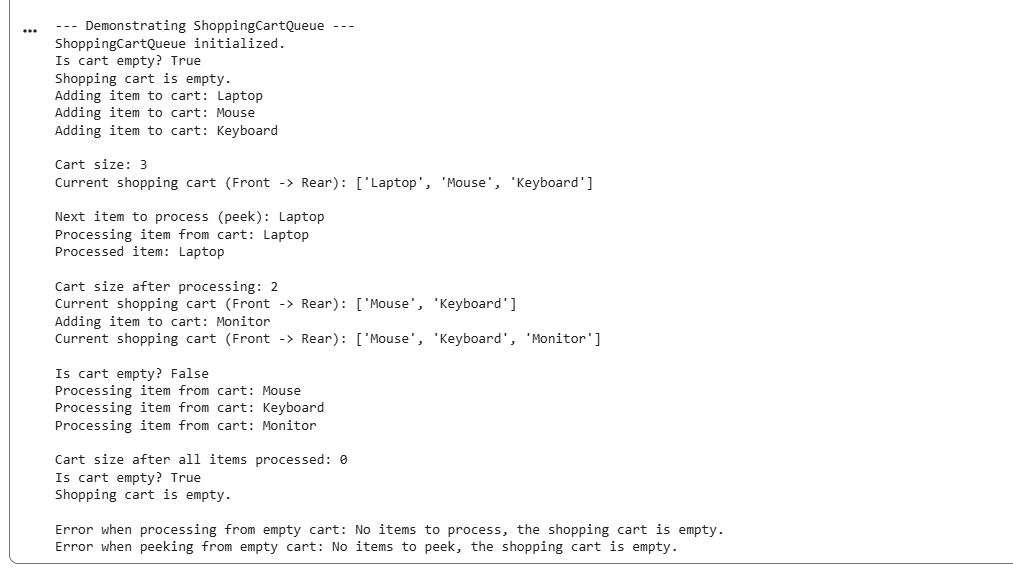
Try to generate a python program for shopping cart management using queues, Order Processing System using Stacks, Top-selling products tracker using stacks, Product search engine using stacks, Delivery route planning using Linked Lists



****

****

****

****

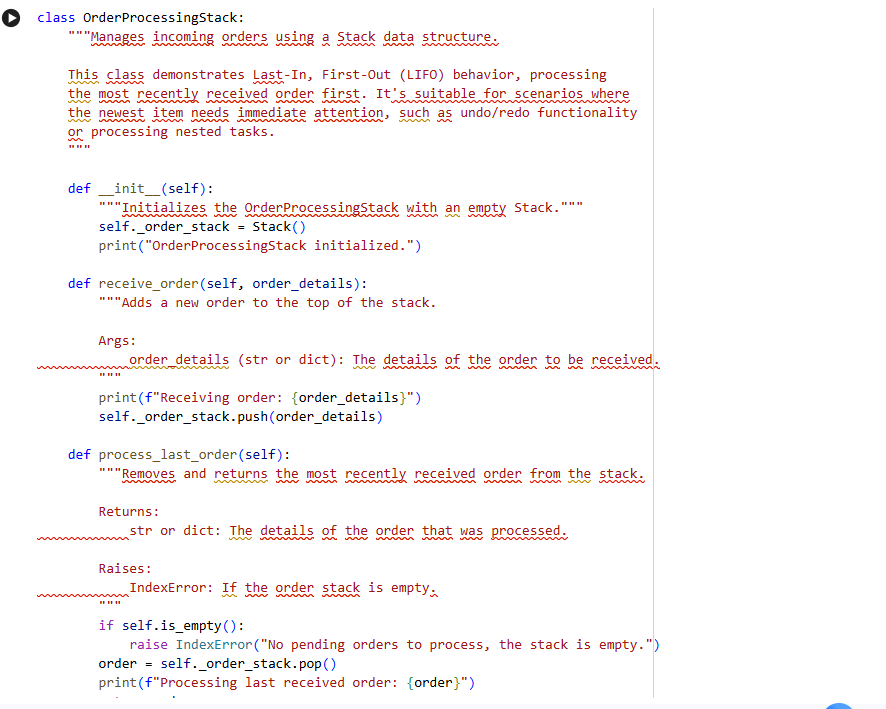
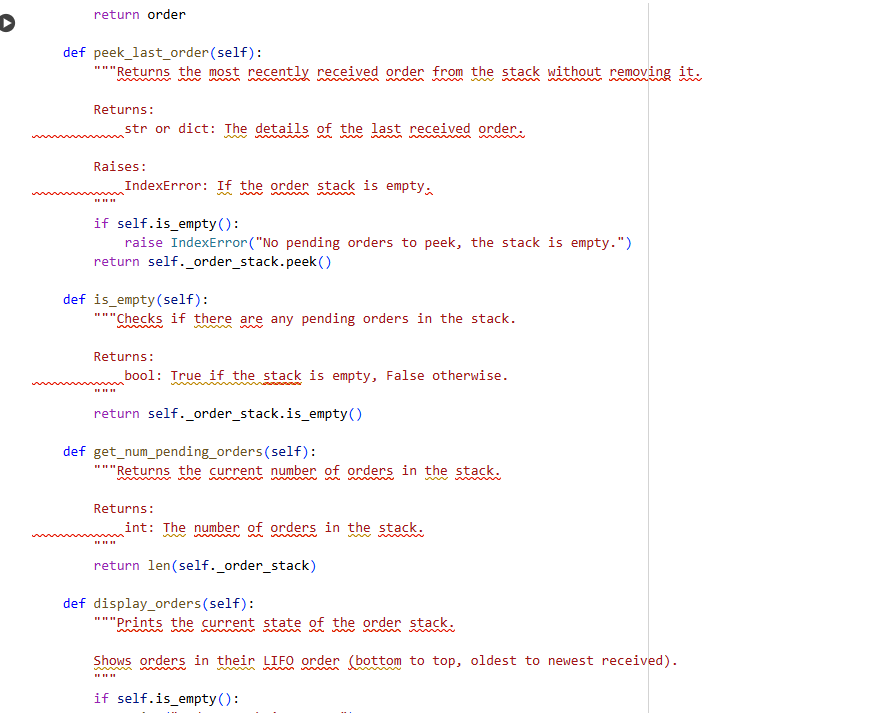
**Order Processing System (Stacks)**

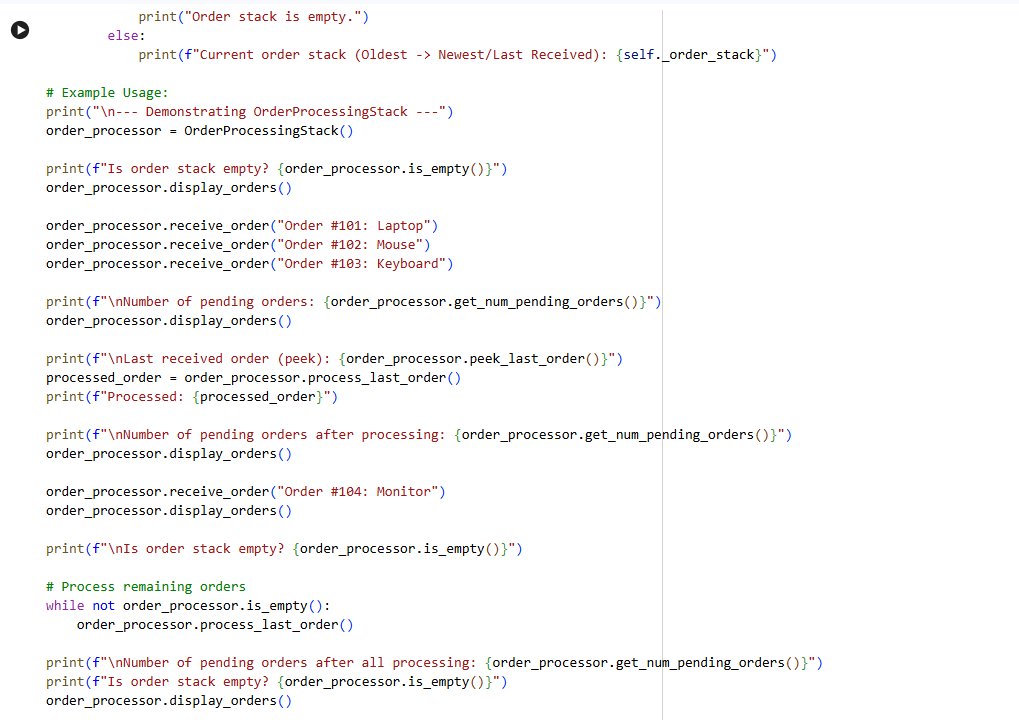
**Subtask:**

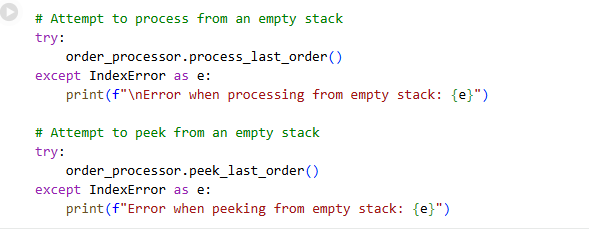
**Implement an OrderProcessingStack class using a Stack to manage incoming orders. This will demonstrate LIFO behavior, processing the most recently received order first.**

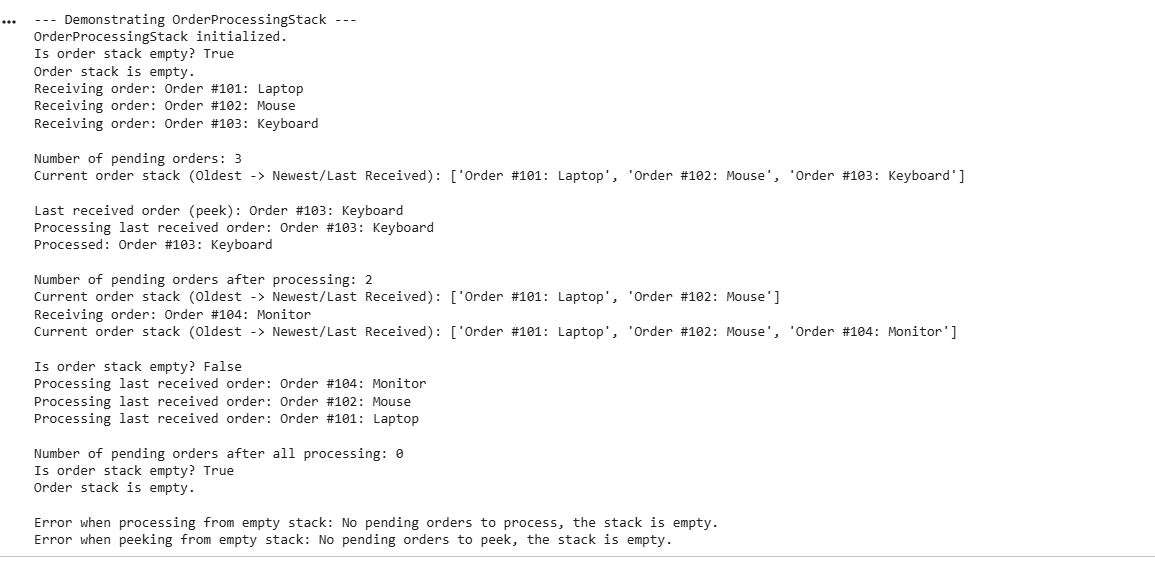
**Reasoning: I need to define the OrderProcessingStack class, leveraging the Stack class previously established in the notebook. This implementation will include methods for adding orders (receive\_order), processing them in LIFO order (process\_last\_order), peeking at the last order (peek\_last\_order), checking for pending orders (is\_empty), querying the stack size (get\_num\_pending\_orders), and displaying the stack's current state (display\_orders). All methods will feature docstrings, comments, and comprehensive example usage, including error handling for an empty stack.**

**Reasoning: I will implement the OrderProcessingStack class as instructed, utilizing the Stack class for its core functionality and ensuring all specified methods, docstrings, comments, and example usage are included. This will demonstrate the LIFO principle for managing orders.**

****





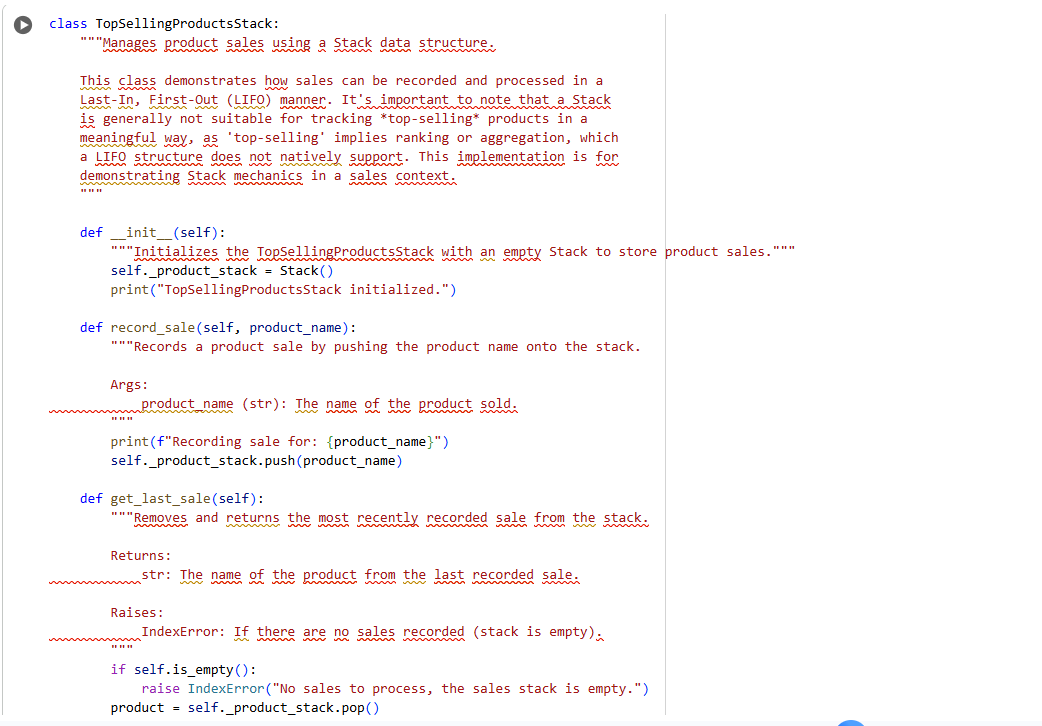


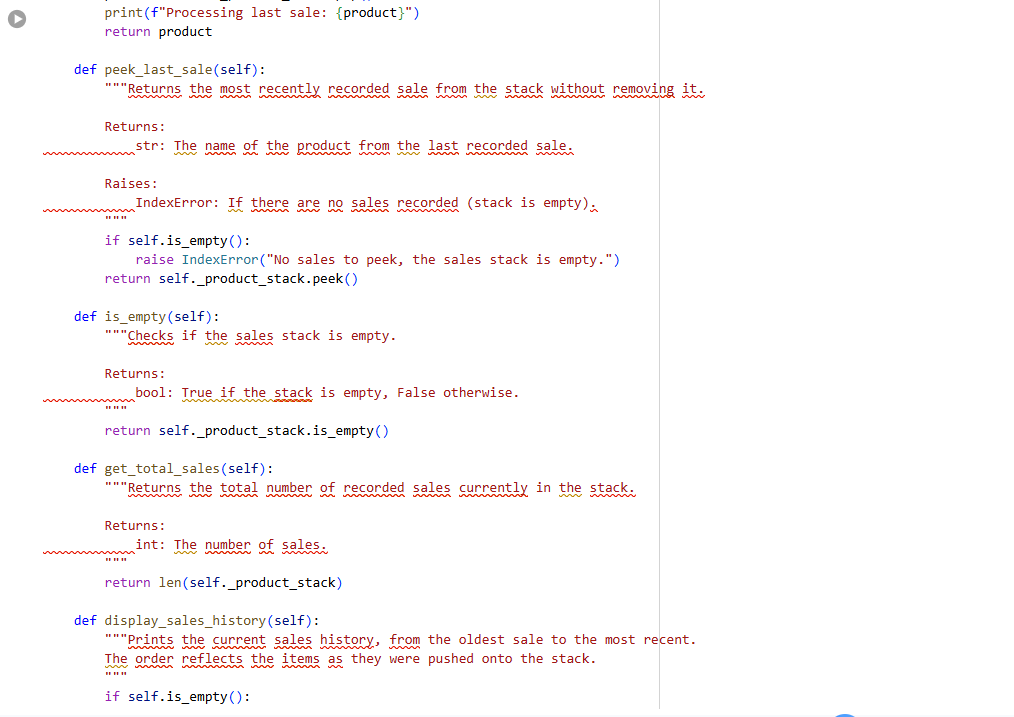
Top-Selling Products Tracker (Stacks)

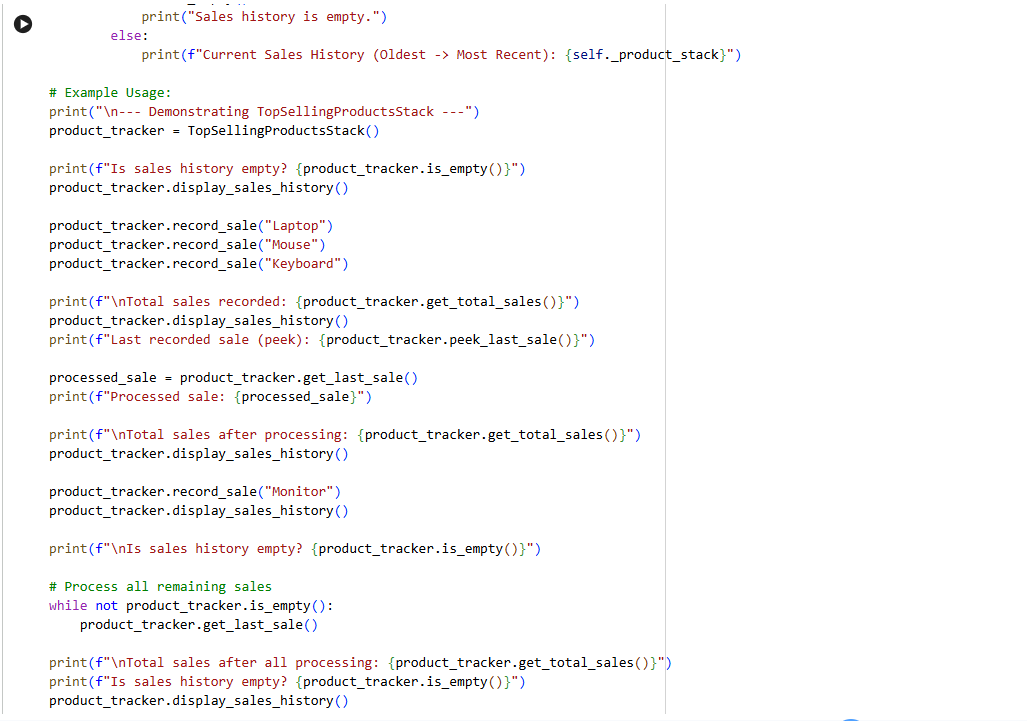
Subtask:

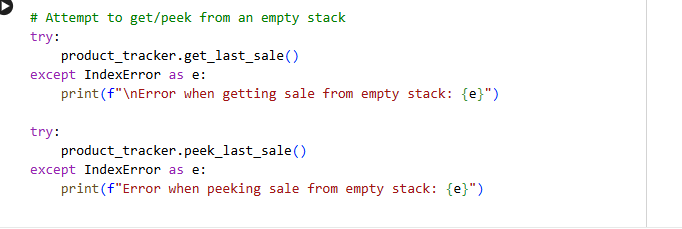
Implement a TopSellingProductsStack class using a Stack. Note that a Stack is generally not suitable for tracking *top-selling* products due to its LIFO nature, which doesn't facilitate ranking or aggregation needed for 'top-selling.' This implementation will demonstrate basic Stack operations in a context where its LIFO property might be misleading for the stated goal.

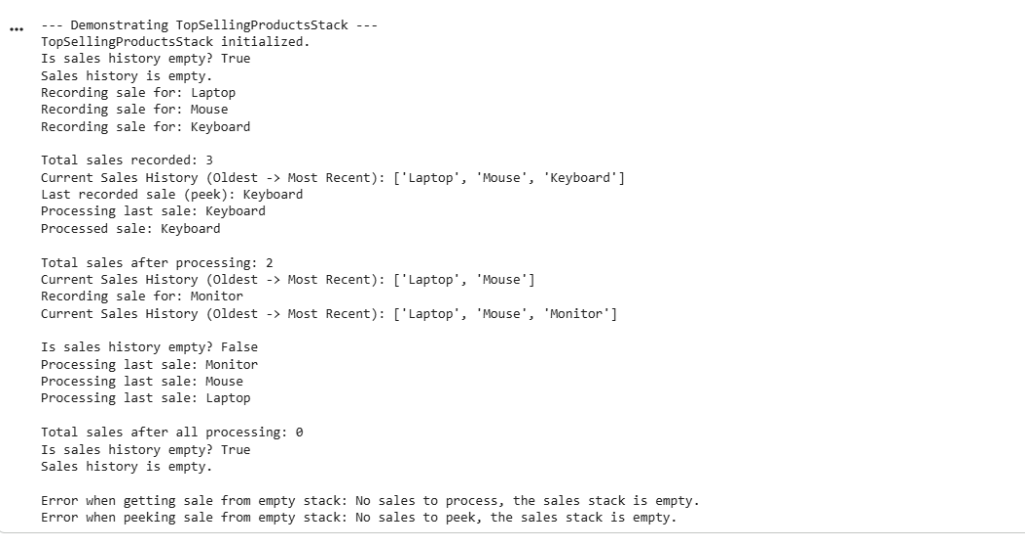
**Reasoning**: I need to define the TopSellingProductsStack class as instructed, utilizing the Stack class for its core functionality and ensuring all specified methods, docstrings, comments, and comprehensive example usage are included, especially error handling for empty stack scenarios.









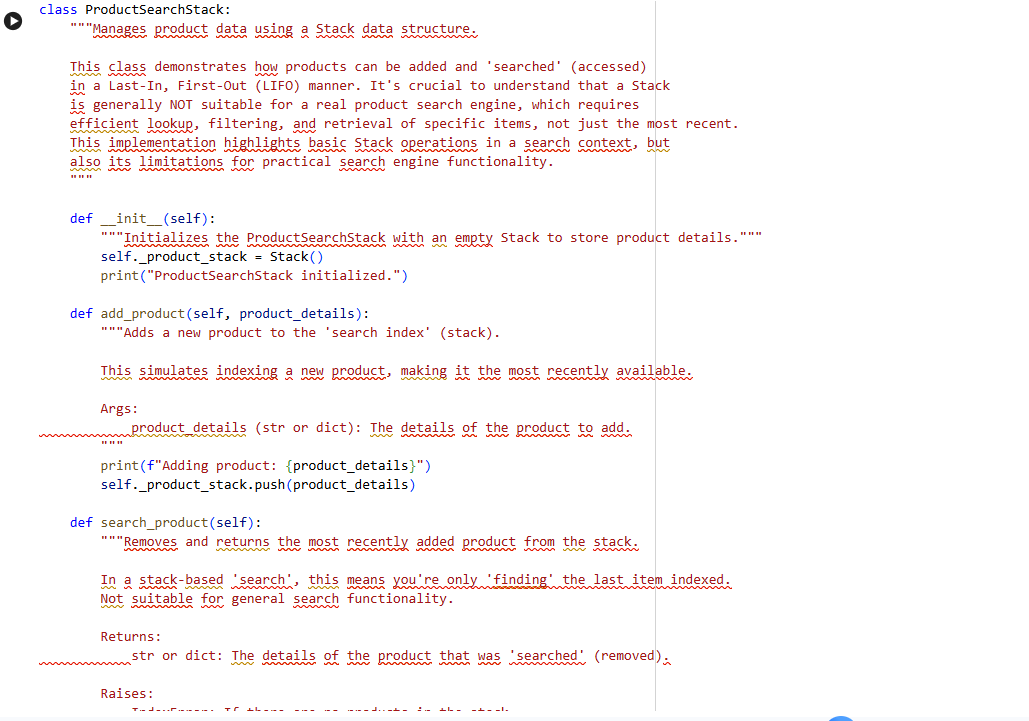


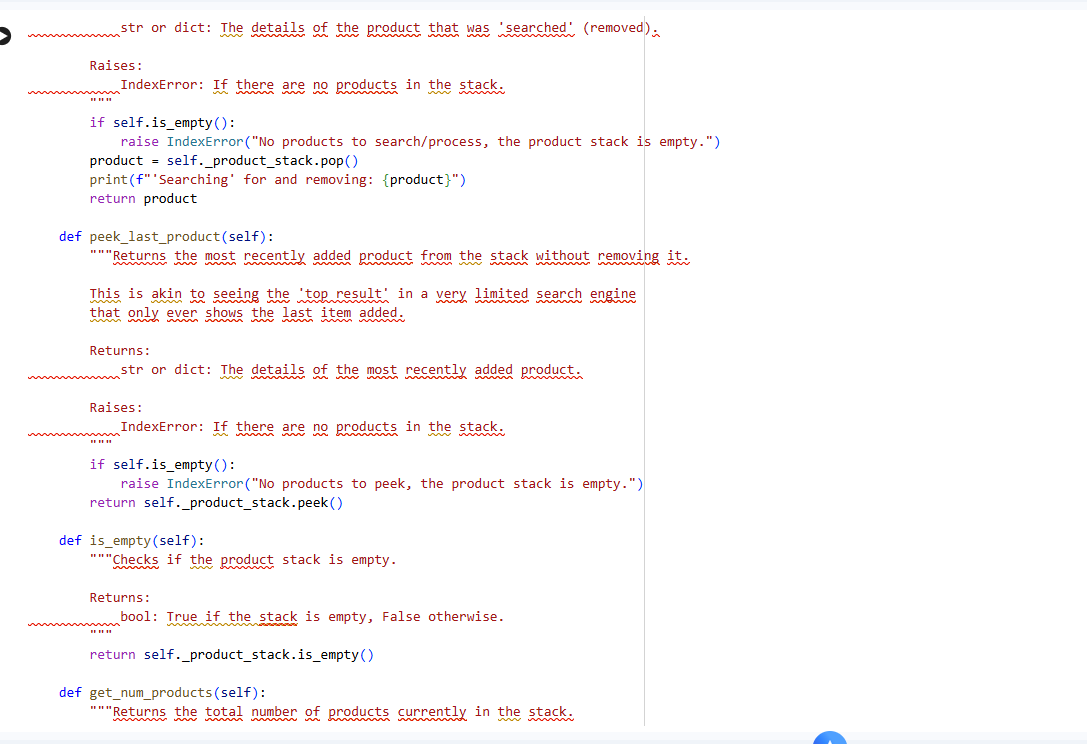
Product Search Engine (Stacks)

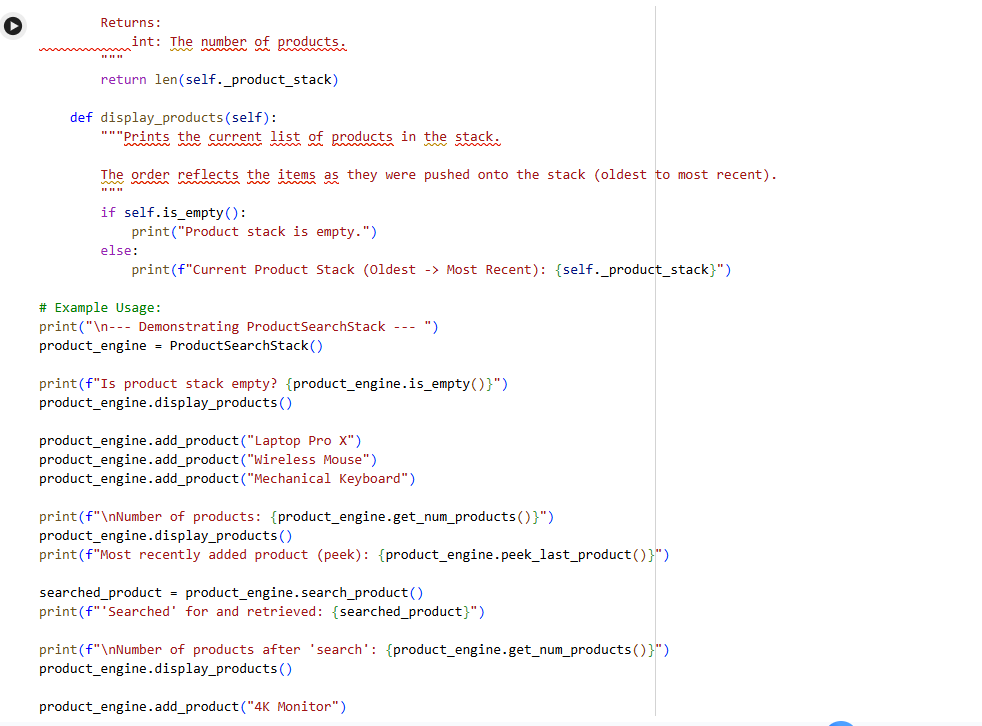
Subtask:

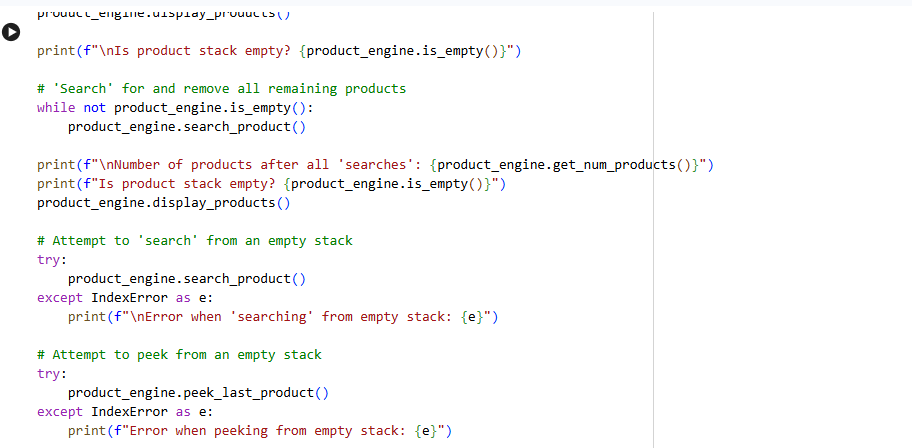
Implement a ProductSearchStack class using a Stack. It's important to note that a Stack is generally not appropriate for a search engine, which requires efficient lookup and retrieval, not LIFO access. This implementation will demonstrate basic Stack operations but will highlight its limitations for a real search engine.

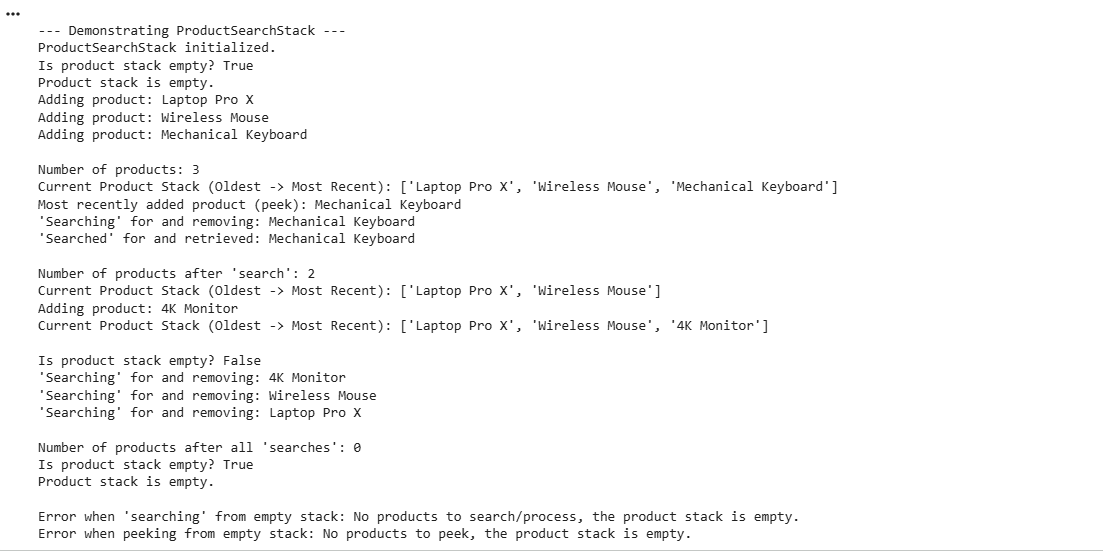
**Reasoning**: I need to define the ProductSearchStack class as instructed, utilizing the Stack class for its core functionality. This implementation will include all the specified methods, docstrings, comments, and comprehensive example usage, especially error handling for empty stack scenarios.











Delivery Route Planning (Linked Lists)

Subtask:

Implement a DeliveryRouteLinkedList class using a Linked List to represent a sequence of delivery stops.

**Reasoning**: I need to define the DeliveryRouteLinkedList class, leveraging the LinkedList class previously established in the notebook. This implementation will include all the specified methods, docstrings, comments, and comprehensive example usage, including error handling for an empty route.

**Reasoning**: I need to define the DeliveryRouteLinkedList class, leveraging the LinkedList class previously established in the notebook. This implementation will include all the specified methods, docstrings, comments, and comprehensive example usage, including error handling for an empty route. Since the provided LinkedList only supports insertion at the beginning (insert method), and the task specifically asks for add\_stop to add to the *end* of the list, I will implement a custom \_append\_to\_end helper method within DeliveryRouteLinkedList that manually traverses the internal LinkedList to add new nodes to its tail. This ensures that remove\_next\_stop (which will operate on the head) processes stops in the correct FIFO order for a route.







