

School of Computer Science and Artificial Intelligence

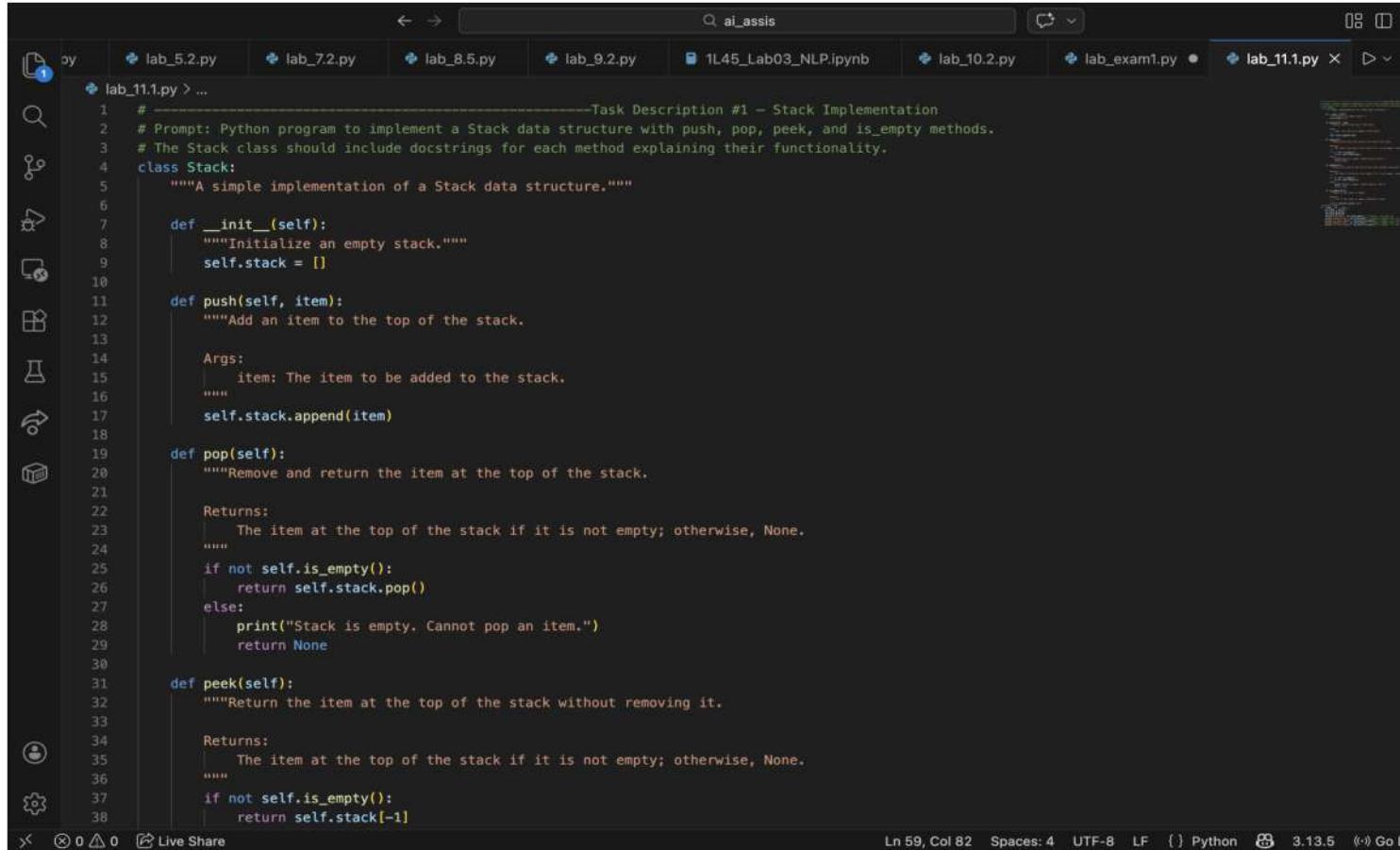
Lab Assignment # 11.1

Program : B. Tech (CSE)
Specialization : –
Course Title : AI Assisted coding
Course Code :
Semester II
Academic Session : 2025-2026
Name of Student : Akash Reddy
Enrollment No. : 2403A51L30
Batch No. : 51
Date : 17-02-2026

Task Description #1 – Stack Implementation

Prompt: Python program to implement a Stack data structure with push, pop, peek, and is_empty methods. The Stack class should include docstrings for each method explaining their functionality.

OUTPUT



The screenshot shows a Jupyter Notebook interface with multiple tabs at the top. The active tab is 'lab_11.1.py'. The code in the notebook is as follows:

```
1 # ----- Task Description #1 - Stack Implementation
2 # Prompt: Python program to implement a Stack data structure with push, pop, peek, and is_empty methods.
3 # The Stack class should include docstrings for each method explaining their functionality.
4 class Stack:
5     """A simple implementation of a Stack data structure."""
6
7     def __init__(self):
8         """Initialize an empty stack."""
9         self.stack = []
10
11    def push(self, item):
12        """Add an item to the top of the stack.
13
14        Args:
15            item: The item to be added to the stack.
16        """
17        self.stack.append(item)
18
19    def pop(self):
20        """Remove and return the item at the top of the stack.
21
22        Returns:
23            The item at the top of the stack if it is not empty; otherwise, None.
24        """
25        if not self.is_empty():
26            return self.stack.pop()
27        else:
28            print("Stack is empty. Cannot pop an item.")
29            return None
30
31    def peek(self):
32        """Return the item at the top of the stack without removing it.
33
34        Returns:
35            The item at the top of the stack if it is not empty; otherwise, None.
36        """
37        if not self.is_empty():
38            return self.stack[-1]
```

At the bottom of the interface, there are various icons for file operations, and the status bar shows 'Ln 59, Col 82 Spaces: 4 UTF-8 LF () Python 3.13.5 (0) Go'.

The screenshot shows a Python script named `lab_11.1.py` in a code editor. The script defines a `Stack` class with methods `push`, `pop`, `peek`, and `is_empty`. It includes example usage at the bottom. Below the editor is a terminal window showing the execution of the script and its output.

```
50     # Example usage
51     if __name__ == "__main__":
52         my_stack = Stack()
53         my_stack.push(10)
54         my_stack.push(20)
55         print("Top item:", my_stack.peek()) # Output: Top item: 20
56         print("Popped item:", my_stack.pop()) # Output: Popped item: 20
57         print("Is stack empty?", my_stack.is_empty()) # Output: Is stack empty? False
58         print("Popped item:", my_stack.pop()) # Output: Popped item: 10
59         print("Is stack empty?", my_stack.is_empty()) # Output: Is stack empty? True
60
61
62
63
64
```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASH-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Top item: 20
Popped item: 20
Is stack empty? False
Popped item: 10
Is stack empty? True
(base) akash@AKASH-MacBook-Air ai_assis %
```

Task Description #2 – Queue Implementation

Prompt: Python program to implement a Queue data structure with enqueue, dequeue, peek, and size methods. The Queue class should include docstrings for each method explaining their functionality.

OUTPUT:

The screenshot shows the `lab_11.1.py` file in a code editor. The file contains the implementation of a `Queue` class with methods `enqueue`, `dequeue`, `peek`, and `is_empty`. Each method has a detailed docstring explaining its functionality.

```
61
62     # -----
63     # Task Description #2 - Queue Implementation
64     # Prompt: Python program to implement a Queue data structure with enqueue, dequeue, peek, and size methods.
65     # The Queue class should include docstrings for each method explaining their functionality.
66
67     class Queue:
68         """A simple implementation of a Queue data structure."""
69
70         def __init__(self):
71             """Initialize an empty queue."""
72             self.queue = []
73
74         def enqueue(self, item):
75             """Add an item to the end of the queue.
76
77             Args:
78                 item: The item to be added to the queue.
79
80             Returns:
81                 None.
82
83             """
84             self.queue.append(item)
85
86         def dequeue(self):
87             """Remove and return the item at the front of the queue.
88
89             Returns:
90                 The item at the front of the queue if it is not empty; otherwise, None.
91             """
92             if not self.is_empty():
93                 return self.queue.pop(0)
94             else:
95                 print("Queue is empty. Cannot dequeue an item.")
96                 return None
97
98         def peek(self):
99             """Return the item at the front of the queue without removing it.
100
101            Returns:
102                The item at the front of the queue if it is not empty; otherwise, None.
103            """
104
```

Task Description #3 – Linked List

Prompt: Python program to implement a Singly Linked List data structure with insert and display methods. The LinkedList class should include docstrings for each method explaining their functionality.give code

Output:

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

- Title Bar:** The title bar displays "ai_assis".
- File List:** The left sidebar lists several files: "lab_5.2.py", "lab_7.2.py", "lab_8.5.py", "lab_9.2.py", "1L45_Lab03_NLP.ipynb", "lab_10.2.py", "lab_exam1.py", and "lab_11.1.py".
- Code Area:** The main area contains Python code for a singly linked list. The code includes docstrings for both the Node and LinkedList classes.

```
# Task Description #3 - Linked List
# Prompt: Python program to implement a Singly Linked List data structure with insert and display methods.
# The LinkedList class should include docstrings for each method explaining their functionality.give code

class Node:
    """A class representing a node in a singly linked list."""

    def __init__(self, data):
        """Initialize a node with the given data and a reference to the next node.

        Args:
            data: The data to be stored in the node.
        """
        self.data = data
        self.next = None

class LinkedList:
    """A class representing a singly linked list."""

    def __init__(self):
        """Initialize an empty linked list."""
        self.head = None

    def insert(self, data):
        """Insert a new node with the given data at the end of the linked list.

        Args:
            data: The data to be stored in the new node.
        """
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
            return
        last_node = self.head
        while last_node.next:
            last_node = last_node.next
        last_node.next = new_node
```

- Right Panel:** A vertical panel on the right side shows a list of recent files and a search bar.
- Bottom Bar:** The bottom bar shows the current file path as "1L45_Lab03_NLP.ipynb", and status indicators for line count (Ln 185), column count (Col 78), spaces (Spaces: 4), encoding (UTF-8), line feed (LF), Python language, version 3.13.5, and a "Go Live" button.

The screenshot shows a code editor with a Python file named `lab_11.py`. The code defines a `LinkedList` class with a `display` method to print the list's contents. It also includes an example usage block where a list is created, three values are inserted, and then `display` is called. The output window shows the command run in a terminal and the resulting list structure: `10 -> 20 -> 30 -> None`.

```
149     class LinkedList:
150
151         def display(self):
152             """Display the contents of the linked list."""
153             current_node = self.head
154             while current_node:
155                 print(current_node.data, end=" -> ")
156                 current_node = current_node.next
157             print("None")
158
159     # Example usage
160     if __name__ == "__main__":
161         my_list = LinkedList()
162         my_list.insert(10)
163         my_list.insert(20)
164         my_list.insert(30)
165         print("Linked List contents:")
166         my_list.display() # Output: Linked List contents: 10 -> 20 -> 30 -> None
167
168
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python + ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.py
(base) akash@AKASHs-MacBook-Air ~ % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.py
Linked List contents:
10 -> 20 -> 30 -> None
(base) akash@AKASHs-MacBook-Air ~ %
```

Task Description #4 – Binary Search Tree (BST)

Prompt: Python program to implement a Binary Search Tree (BST) data structure with insert and in-order traversal methods. The BST class should include docstrings for each method explaining their functionality.

OUTPUT:

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

- Title Bar:** The title bar displays "ai_assis".
- File List:** On the left, there is a sidebar with icons for file operations like Open, Save, Find, Copy, Paste, etc., and a list of files: "lab_5.2.py", "lab_7.2.py", "lab_8.5.py", "lab_9.2.py", "1L45_Lab03_NLP.ipynb" (highlighted in red), "lab_10.2.py", "lab_exam1.py", and "lab_11.1.py".
- Code Area:** The main area contains Python code for a Binary Search Tree (BST).

```
# -----Task Description #4 - Binary Search Tree (BST)
# Prompt: Python program to implement a Binary Search Tree (BST) data structure with insert and in-order traversal methods.
# The BST class should include docstrings for each method explaining their functionality.

187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224

class TreeNode:
    """A class representing a node in a binary search tree."""
    def __init__(self, data):
        """Initialize a tree node with the given data and references to left and right children.

        Args:
            data: The data to be stored in the tree node.
        """
        self.data = data
        self.left = None
        self.right = None

class BinarySearchTree:
    """A class representing a binary search tree."""
    def __init__(self):
        """Initialize an empty binary search tree."""
        self.root = None

    def insert(self, data):
        """Insert a new node with the given data into the binary search tree.

        Args:
            data: The data to be stored in the new node.
        """
        if self.root is None:
            self.root = TreeNode(data)
        else:
            self._insert_recursive(self.root, data)

    def _insert_recursive(self, node, data):
        """Helper method to insert a new node recursively.

        Args:
            node: The current node being traversed.
            data: The data to be inserted.
        """
        if data < node.data:
            if node.left is None:
                node.left = TreeNode(data)
            else:
                self._insert_recursive(node.left, data)
        else:
            if node.right is None:
                node.right = TreeNode(data)
            else:
                self._insert_recursive(node.right, data)
```
- Bottom Bar:** The bottom bar shows the status "Ln 267, Col 130" and various icons for file operations, including "Live Share".

```
244     def _in_order_recursive(self, node):
245         values.extend(self._in_order_recursive(node.left))
246         values.append(node.data)
247         values.extend(self._in_order_recursive(node.right))
248     return values
249 
250 # Example usage
251 if __name__ == "__main__":
252     bst = BinarySearchTree()
253     bst.insert(10)
254     bst.insert(5)
255     bst.insert(15)
256     bst.insert(3)
257     bst.insert(7)
258 
259     print("In-order traversal of the BST:", bst.in_order_traversal()) # Output: In-order traversal of the BST: [3, 5, 7, 10, 15]
260 
261 
262 
263 
264 
265 
266 
267 
268 
269 
270 
271 
272 
273 
274 
275 
276 
277 
278 
279 
280 
281 
282 
283 
284 
285 
286 
287 
288 
289 
290 
291 
292 
293 
294 
295 
296 
297 
298 
299 
300 
301 
302 
303 
304 
305 
306 
307 
308 
309 
310 
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
In-order traversal of the BST: [3, 5, 7, 10, 15]
(base) akash@AKASHs-MacBook-Air ai_assis %
```

Task Description #5 – Hash Table

Prompt: Python program to implement a Hash Table data structure with insert, search, and delete methods. The HashTable class should include docstrings for each method explaining their functionality, and it should handle collisions using chaining.

Output:

```
273 # -----
274 # Task Description #5 – Hash Table
275 # Prompt: Python program to implement a Hash Table data structure with insert, search, and delete methods.
276 # The HashTable class should include docstrings for each method explaining their functionality, and it should
277 # handle collisions using chaining.
278 
279 class HashTable:
280     """A simple implementation of a Hash Table data structure using chaining for collision resolution."""
281 
282     def __init__(self, size=10):
283         """Initialize the hash table with a specified size.
284 
285         Args:
286             size: The size of the hash table (default is 10).
287         """
288         self.size = size
289         self.table = [[] for _ in range(size)]
290 
291     def _hash(self, key):
292         """Generate a hash value for the given key.
293 
294         Args:
295             key: The key to be hashed.
296 
297         Returns:
298             The hash value corresponding to the key.
299         """
300         return hash(key) % self.size
301 
302     def insert(self, key, value):
303         """Insert a key-value pair into the hash table.
304 
305         Args:
306             key: The key to be inserted.
307             value: The value associated with the key.
308 
309         index = self._hash(key)
310         # Check if the key already exists and update it
311         for i, (k, v) in enumerate(self.table[index]):
```

The screenshot shows a code editor interface with a Python script named `HashTable.py`. The script defines a `HashTable` class with methods for insertion, deletion, and search. It includes example usage code. Below the code editor is a terminal window showing the execution of the script and its output.

```
278     class HashTable:
279         def delete(self, key):
280             if k == key:
281                 del self.table[index][i]
282             return True
283
284     # Example usage
285     if __name__ == "__main__":
286         hash_table = HashTable()
287         hash_table.insert("name", "Alice")
288         hash_table.insert("age", 30)
289
290         print("Search for 'name':", hash_table.search("name")) # Output: Search for 'name': Alice
291         print("Search for 'age':", hash_table.search("age")) # Output: Search for 'age': 30
292         print("Delete 'name':", hash_table.delete("name")) # Output: Delete 'name': True
293         print("Search for 'name' after deletion:", hash_table.search("name")) # Output: Search for 'name' after deletion: None
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
```

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASH-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Search for 'name': Alice
Search for 'age': 30
Delete 'name': True
Search for 'name' after deletion: None
(base) akash@AKASH-MacBook-Air ai_assis %
```

Task Description #6 – Graph Representation

Prompt: Python program to implement a Graph data structure using an adjacency list representation. The Graph class should include methods to add vertices, add edges, and display connections.

Output:

The screenshot shows a code editor interface with a Python script named `lab_11.1.py`. The script implements a `Graph` class using an adjacency list representation. It includes methods for adding vertices, adding edges, and displaying connections. The code is well-documented with docstrings and type hints.

```
359 #----- Task Description #6 – Graph Representation
360 # Prompt: Python program to implement a Graph data structure using an adjacency list representation.
361 # The Graph class should include methods to add vertices, add edges, and display connections.
362
363 class Graph:
364     """A simple implementation of a Graph data structure using an adjacency list representation."""
365
366     def __init__(self):
367         """Initialize an empty graph."""
368         self.graph = {}
369
370     def add_vertex(self, vertex):
371         """Add a vertex to the graph.
372
373         Args:
374             vertex: The vertex to be added to the graph.
375         """
376         if vertex not in self.graph:
377             self.graph[vertex] = []
378
379     def add_edge(self, vertex1, vertex2):
380         """Add an edge between two vertices in the graph.
381
382         Args:
383             vertex1: The first vertex of the edge.
384             vertex2: The second vertex of the edge.
385         """
386         if vertex1 in self.graph and vertex2 in self.graph:
387             self.graph[vertex1].append(vertex2)
388             self.graph[vertex2].append(vertex1) # For undirected graph
389
390     def display(self):
391         """Display the connections in the graph."""
392         for vertex, edges in self.graph.items():
393             print(f'{vertex}: {edges}')
394
395 # Example usage
396 if __name__ == "__main__":
397     my_graph = Graph()
```

The screenshot shows a code editor interface with a Python file named `lab_11.1.py` open. The code defines a `Graph` class with a `display` method that prints the connections between vertices. It also includes an example usage block. Below the code editor is a terminal window showing the execution of the script and its output, which displays the graph connections: A: B, C B: A, C C: A.

```
363     class Graph:
364         def display(self):
365             """Display the connections in the graph."""
366             for vertex, edges in self.graph.items():
367                 print(f'{vertex}: {edges}')
368
369     # Example usage
370     if __name__ == "__main__":
371         my_graph = Graph()
372         my_graph.add_vertex("A")
373         my_graph.add_vertex("B")
374         my_graph.add_vertex("C")
375         my_graph.add_edge("A", "B")
376         my_graph.add_edge("A", "C")
377         print("Graph connections:")
378         my_graph.display() # Output: Graph connections: A: B, C B: A, C C: A
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Graph connections:
A: B, C
B: A
C: A
(base) akash@AKASHs-MacBook-Air ai_assis %
```

Task Description #7 – Priority Queue

Prompt: Python program to implement a Priority Queue using Python's `heapq` module. The `PriorityQueue` class should include methods to enqueue items with a specified priority, dequeue the item with the highest priority

Output:

The screenshot shows a code editor interface with a Python file named `lab_11.1.py` open. The code implements a `PriorityQueue` class using Python's `heapq` module. It includes methods for enqueueing items with a specified priority and dequeuing the item with the highest priority. The code is well-documented with docstrings and type hints.

```
406
407     # -----
408     # Task Description #7 - Priority Queue
409     # Prompt: Python program to implement a Priority Queue using Python's heapq module. The PriorityQueue class should include methods
410     # to enqueue items with a specified priority, dequeue the item with the highest priority
411     # (lowest numerical value), and display the contents of the priority queue.
412
413     import heapq
414
415     class PriorityQueue:
416         """A simple implementation of a Priority Queue using Python's heapq module."""
417
418         def __init__(self):
419             """Initialize an empty priority queue."""
420             self.queue = []
421
422         def enqueue(self, item, priority):
423             """Add an item to the priority queue with a specified priority.
424
425             Args:
426                 item: The item to be added to the priority queue.
427                 priority: The priority of the item (lower values indicate higher priority).
428
429             heappush(self.queue, (priority, item))
430
431         def dequeue(self):
432             """Remove and return the item with the highest priority (lowest numerical value).
433
434             Returns:
435                 The item with the highest priority if the queue is not empty; otherwise, None.
436
437             if not self.is_empty():
438                 return heappop(self.queue)[1]
439             else:
440                 print("Priority Queue is empty. Cannot dequeue an item.")
441                 return None
442
443         def display(self):
444             """Display the contents of the priority queue."""
445
```

Task Description #8 – Deque

Prompt: Python program to implement a double-ended queue (deque) using the collections.deque module. The DequeDS class should include methods to insert and remove items from both ends of the deque, along with docstrings explaining their functionality.

Output:

The screenshot shows a Jupyter Notebook interface with a dark theme. The left sidebar contains various icons for file operations, search, and help. The top navigation bar has tabs for different notebooks: 'lab_5.2.py', 'lab_7.2.py', 'lab_8.5.py', 'lab_9.2.py', '1L45_Lab03_NLP.ipynb' (selected), 'lab_10.2.py', 'lab_exam1.py', and 'lab_11.1.py'. The main area displays a Python script for a 'DequeDS' class:

```
# Task Description #8 - Deque
# Prompt: Python program to implement a double-ended queue (deque) using the collections.deque
# module. The DequeDS class should include methods to insert and remove items from both ends of the deque,
# along with docstrings explaining their functionality.

from collections import deque

class DequeDS:
    """A simple implementation of a double-ended queue (deque) using the collections.deque module."""

    def __init__(self):
        """Initialize an empty deque."""
        self.deque = deque()

    def insert_front(self, item):
        """Insert an item at the front of the deque.

        Args:
            item: The item to be inserted at the front of the deque.
        """
        self.deque.appendleft(item)

    def insert_rear(self, item):
        """Insert an item at the rear of the deque.

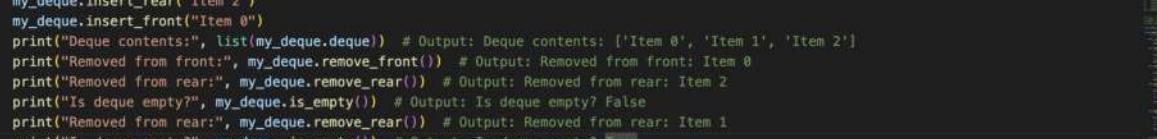
        Args:
            item: The item to be inserted at the rear of the deque.
        """
        self.deque.append(item)

    def remove_front(self):
        """Remove and return an item from the front of the deque.

        Returns:
            The item removed from the front of the deque if it is not empty; otherwise, None.
        """
        if not self.is_empty():
            return self.deque.popleft()

    def is_empty(self):
        """Check if the deque is empty.

        Returns:
            True if the deque is empty; False otherwise.
        """
        return len(self.deque) == 0
```



A screenshot of a Python code editor interface. The top part shows a code editor with the following Python script:

```
531 if __name__ == "__main__":
532     my_deque = DequeDS()
533     my_deque.insert_rear("Item 1")
534     my_deque.insert_rear("Item 2")
535     my_deque.insert_front("Item 0")
536     print("Deque contents:", list(my_deque.deque)) # Output: Deque contents: ['Item 0', 'Item 1', 'Item 2']
537     print("Removed from front:", my_deque.remove_front()) # Output: Removed from front: Item 0
538     print("Removed from rear:", my_deque.remove_rear()) # Output: Removed from rear: Item 2
539     print("Is deque empty?", my_deque.is_empty()) # Output: Is deque empty? False
540     print("Removed from rear:", my_deque.remove_rear()) # Output: Removed from rear: Item 1
541     print("Is deque empty?", my_deque.is_empty()) # Output: Is deque empty? True
542
```

The bottom part shows the terminal output of running the script:

```
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASH-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Deque contents: ['Item 0', 'Item 1', 'Item 2']
Removed from front: Item 0
Removed from rear: Item 2
Is deque empty? False
Removed from rear: Item 1
Is deque empty? True
(base) akash@AKASH-MacBook-Air ai_assis %
```

Task Description #9 Real-Time Application Challenge – Choose the Right Data Structure

Prompt: Python program to implement a Student Attendance Tracking system that uses a Queue data structure to log students entering and exiting the campus. The program should allow adding students to the queue when they enter and removing them when they exit, along with displaying the current attendance log.
from collections import deque

Output:

```
584 # Example usage
585 if __name__ == "__main__":
586     tracker = AttendanceTracker()
587     tracker.student_enters("Alice")
588     tracker.student_enters("Bob")
589     tracker.display_attendance() # Output: Current Attendance Log: Alice Bob
590     tracker.student_exits("Alice")
591     tracker.display_attendance() # Output: Current Attendance Log: Bob
592     tracker.student_exits("Charlie") # Output: Charlie is not in the attendance log.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
(base) akash@AKASHs-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Alice has entered the campus.
Bob has entered the campus.
Current Attendance Log:
Alice
Bob
Alice has exited the campus.
Current Attendance Log:
Bob
Charlie is not in the attendance log.
(base) akash@AKASHs-MacBook-Air ai_assis %
```

Task Description #10: Smart E-Commerce Platform – Data Structure Challenge

Prompt: Python program to implement a Shopping Cart Management system that allows adding and removing products dynamically using a Linked List data structure. The ShoppingCart class should include methods to add products, remove products, and display the current contents of the cart, along with docstrings explaining their functionality.

Output:

```
594
595 # -----Task Description #10: Smart E-Commerce Platform – Data Structure Challenge
596 # Prompt: Python program to implement a Shopping Cart Management system that allows adding and removing products dynamically using a
597 # Linked List data structure. The ShoppingCart class should include methods to add products, remove products, and display the current
598 # contents of the cart, along with docstrings explaining their functionality.
599
600 class ProductNode:
601     """A class representing a node in a linked list for a shopping cart."""
602
603     def __init__(self, product_name, price):
604         """Initialize a product node with the given product name and price.
605
606         Args:
607             product_name: The name of the product.
608             price: The price of the product.
609         """
610
611         self.product_name = product_name
612         self.price = price
613         self.next = None
614
615 class ShoppingCart:
616     """A class representing a shopping cart using a linked list data structure."""
617
618     def __init__(self):
619         """Initialize an empty shopping cart."""
620         self.head = None
621
622     def add_product(self, product_name, price):
623         """Add a product to the shopping cart.
624
625         Args:
626             product_name: The name of the product to be added.
627             price: The price of the product to be added.
628
629             new_product = ProductNode(product_name, price)
630             if self.head is None:
631                 self.head = new_product
632             else:
```

```
lab_11.1.py > ...
614     class ShoppingCart:
659         def display_cart(self):
660             return
665             print("Current Shopping Cart Contents:")
666             while current:
667                 print(f"Product: {current.product_name}, Price: ${current.price:.2f}")
668                 current = current.next
669     # Example usage
670     if __name__ == "__main__":
671         cart = ShoppingCart()
672         cart.add_product("Laptop", 999.99)
673         cart.add_product("Smartphone", 499.99)
674         cart.add_product("Headphones", 199.99)
675         cart.display_cart() # Output: Current Shopping Cart Contents: Product: Laptop, Price: $999.99 Product: Smartphone, Price: $499.99 Product: Headphones, Price: $199.99
676         cart.remove_product("Smartphone")
677         cart.display_cart() # Output: Current Shopping Cart Contents: Product: Laptop, Price: $999.99 Product: Headphones, Price: $199.99
678         cart.remove_product("Tablet") # Output: False (Product not found)
679
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
/usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
● (base) akash@AKASH-MacBook-Air ai_assis % /usr/local/bin/python3 /Users/akash/Desktop/ai_assis/lab_11.1.py
Current Shopping Cart Contents:
Product: Laptop, Price: $999.99
Product: Smartphone, Price: $499.99
Product: Headphones, Price: $199.99
Current Shopping Cart Contents:
Product: Laptop, Price: $999.99
Product: Headphones, Price: $199.99
● (base) akash@AKASH-MacBook-Air ai_assis %
```