

Assignment-11.1

Data Structures with AI: Implementing Fundamental Structures

K. Ruthwik Bhat

2403a51l08

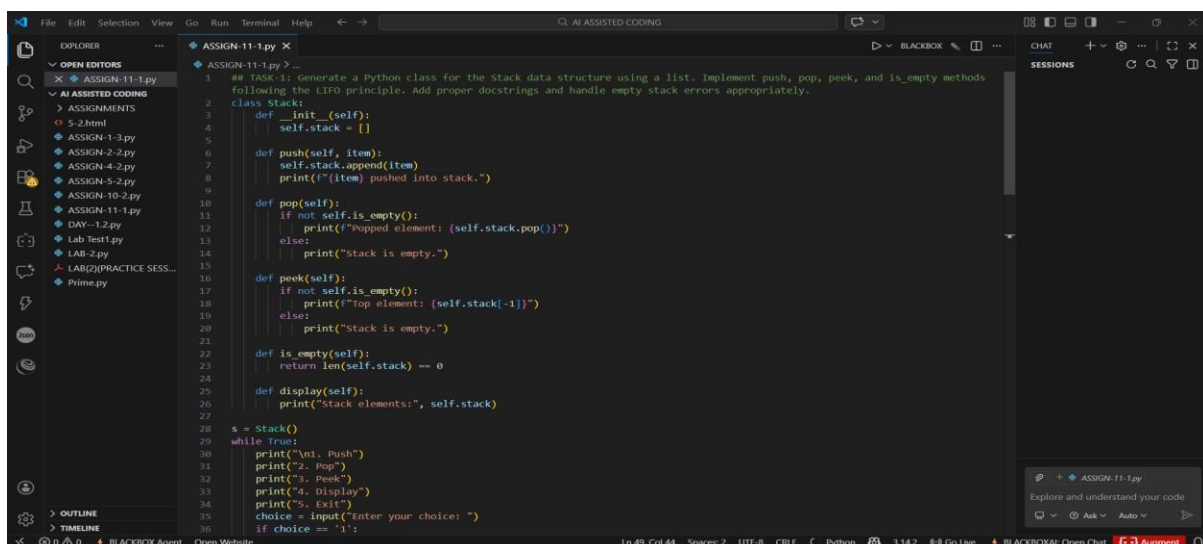
B-51

Task Description #1 - Stack Implementation:

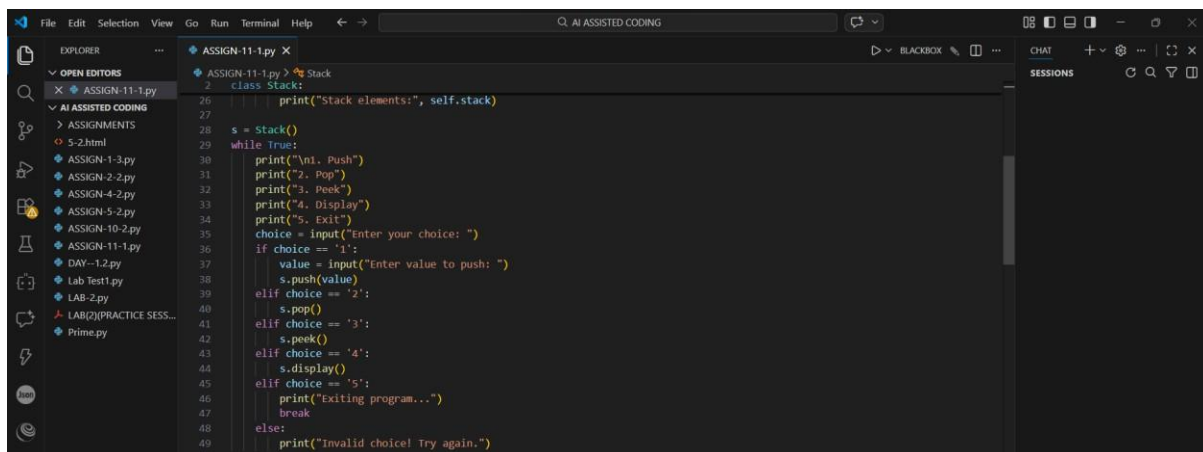
Task: Use AI to generate a Stack class with push, pop, peek, and is_empty methods.

PROMPT: Generate a Python class for the Stack data structure using a list. Implement push, pop, peek, and is_empty methods following the LIFO principle. Add proper docstrings and handle empty stack errors appropriately.

Sample Input Code:



```
1  ## TASK-1: Generate a Python class for the Stack data structure using a list. Implement push, pop, peek, and is_empty methods
2  following the LIFO principle. Add proper docstrings and handle empty stack errors appropriately.
3  class Stack:
4      def __init__(self):
5          self.stack = []
6
7      def push(self, item):
8          self.stack.append(item)
9          print(f"{item} pushed into stack.")
10
11     def pop(self):
12         if not self.is_empty():
13             print(f"Popped element: {self.stack.pop()}")
14         else:
15             print("Stack is empty.")
16
17     def peek(self):
18         if not self.is_empty():
19             print(f"Top element: {self.stack[-1]}")
20         else:
21             print("Stack is empty.")
22
23     def is_empty(self):
24         return len(self.stack) == 0
25
26     def display(self):
27         print("Stack elements:", self.stack)
28
29 s = Stack()
30 while True:
31     print("\n1. Push")
32     print("2. Pop")
33     print("3. Peek")
34     print("4. Display")
35     print("5. Exit")
36     choice = input("Enter your choice: ")
37     if choice == '1':
```



```
26     print("Stack elements:", self.stack)
27
28 s = Stack()
29 while True:
30     print("\n1. Push")
31     print("2. Pop")
32     print("3. Peek")
33     print("4. Display")
34     print("5. Exit")
35     choice = input("Enter your choice: ")
36     if choice == '1':
37         value = input("Enter value to push: ")
38         s.push(value)
39     elif choice == '2':
40         s.pop()
41     elif choice == '3':
42         s.peek()
43     elif choice == '4':
44         s.display()
45     elif choice == '5':
46         print("Exiting program...")
47         break
48     else:
49         print("Invalid choice! Try again.")
```

OUTPUT:

```
File Edit Selection View Go Run Terminal Help AI ASSISTED CODING
EXPLORER
  ASSIGN-11-1.py
  AI ASSISTED CODING
  ASSIGNMENTS
  5-2.html
  ASSIGN-1-3.py
  ASSIGN-2-2.py
  ASSIGN-4-2.py
  ASSIGN-5-2.py
  ASSIGN-10-2.py
  DAY-1-2.py
  Lab Test1.py
  LAB-2.py
  LAB(2)(PRACTICE SESS...
  Prime.py
  OUTLINE
  TIMELINE
  ASSIGN-11-1.py
  38 s.push(value)
  39 elif choice == '2':
  40     s.pop()
  41 elif choice == '3':
  42     s.peek()
  43 elif choice == '4':
  44     s.display()
  45 elif choice == '5':
  46     print("Exiting program...")
  47     break
  48 else:
  49     print("Invalid choice! Try again.")
  50
  PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AUGMENT NEXT EDIT TRUFFLE
  PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> & c:\Users\sarik\AppData\Local\Python\pythoncore\3.14-64\python.exe "c:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING\ASSIGN-11-1.py"
  1. Push
  2. Pop
  3. Peek
  4. Display
  5. Exit
  Enter your choice: 1
  Enter value to push: 24
  24 pushed into stack.
  1. Push
  2. Pop
  3. Peek
  4. Display
  5. Exit
  Enter your choice:
  CHAT
  SESSIONS
  ASSIGN-11-1.py
  Explore and understand your code
  Ask Auto
```

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 Python class for a Queue using a list. Implement enqueue, dequeue, peek, and size methods following the FIFO principle. Add proper documentation and error handling.

Sample Input Code:

```
File Edit Selection View Go Run Terminal Help AI ASSISTED CODING
EXPLORER
  ASSIGN-11-1.py
  AI ASSISTED CODING
  ASSIGNMENTS
  5-2.html
  ASSIGN-1-3.py
  ASSIGN-2-2.py
  ASSIGN-4-2.py
  ASSIGN-5-2.py
  ASSIGN-10-2.py
  DAY-1-2.py
  Lab Test1.py
  LAB-2.py
  LAB(2)(PRACTICE SESS...
  Prime.py
  OUTLINE
  TIMELINE
  ASSIGN-11-1.py
  51 ## TASK-2: Create a Python class for a Queue using a list. Implement enqueue, dequeue, peek, and size methods following the
  52 FIFO principle. Add proper documentation and error handling.
  53 class Queue:
  54     def __init__(self):
  55         self.queue = []
  56
  57     def enqueue(self, item):
  58         self.queue.append(item)
  59         print(f"{item} enqueued into queue.")
  60
  61     def dequeue(self):
  62         if not self.is_empty():
  63             print(f"Dequeued element: {self.queue.pop(0)}")
  64         else:
  65             print("Queue is empty.")
  66
  67     def peek(self):
  68         if not self.is_empty():
  69             print(f"Front element: {self.queue[0]}")
  70         else:
  71             print("Queue is empty.")
  72
  73     def is_empty(self):
  74         return len(self.queue) == 0
  75
  76     def display(self):
  77         print("Queue elements:", self.queue)
  78
  79 q = Queue()
  80 while True:
  81     print("\n1. Enqueue")
  82     print("2. Dequeue")
  83     print("3. Peek")
  84     print("4. Display")
  85     print("5. Exit")
  86     choice = input("Enter your choice: ")
  87     if choice == '1':
  CHAT
  SESSIONS
  ASSIGN-11-1.py
  Explore and understand your code
  Ask Auto
```

```

77
78 q = Queue()
79 while True:
80     print("\n1. Enqueue")
81     print("\n2. Dequeue")
82     print("\n3. Peek")
83     print("\n4. Display")
84     print("\n5. Exit")
85     choice = input("Enter your choice: ")
86     if choice == '1':
87         value = input("Enter value to enqueue: ")
88         q.enqueue(value)
89     elif choice == '2':
90         q.dequeue()
91     elif choice == '3':
92         q.peek()
93     elif choice == '4':
94         q.display()
95     elif choice == '5':
96         print("Exiting program...")
97         break
98     else:
99         print("Invalid choice! Try again.")
100
101

```

OUTPUT:

```

83     print("\n4. Display")
84     print("\n5. Exit")
85     choice = input("Enter your choice: ")
86     if choice == '1':
87         value = input("Enter value to enqueue: ")
88         q.enqueue(value)
89     elif choice == '2':
90         q.dequeue()
91     elif choice == '3':
92         q.peek()
93     elif choice == '4':
94         q.display()
95     elif choice == '5':

```

```

PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> & C:/Users/sarik/AppData/Local/python/pythoncore-3.14-64/python.exe "c:/Users/sarik/OneDrive/Desktop/AI ASSISTED CODING/ASSIGN-11-1.py"

1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice: 1
Enter value to enqueue: 24
24 enqueued into queue.

1. Enqueue
2. Dequeue
3. Peek
4. Display
5. Exit
Enter your choice:

```

EXPLANATION: A Queue is a linear data structure that follows the FIFO (First In First Out) principle, where the first inserted element is removed first. Elements are added at the rear and removed from the front. It is widely used in scheduling systems, buffering, and real-world waiting line applications.

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 including a Node class and LinkedList class. Implement insert and display methods with clear documentation.

Sample Input Code:

```
100
101 ## TASK-3: Generate a Python implementation of a Singly Linked List including a Node class and LinkedList class. Implement
102 insert and display methods with clear documentation.
103 class Node:
104     def __init__(self, data):
105         self.data = data
106         self.next = None
107
108 class LinkedList:
109     def __init__(self):
110         self.head = None
111
112     def insert(self, data):
113         new_node = Node(data)
114         if self.head is None:
115             self.head = new_node
116             print(f"{data} inserted as head of the list.")
117         else:
118             current = self.head
119             while current.next:
120                 current = current.next
121             current.next = new_node
122             print(f"{data} inserted into the list.")
123
124     def display(self):
125         if self.head is None:
126             print("The linked list is empty.")
127         else:
128             current = self.head
129             elements = []
130             while current:
131                 elements.append(current.data)
132                 current = current.next
133             print("linked list elements:", " ".join(map(str, elements)))
134
135 ll = LinkedList()
136 while True:
137     print("\n1. Insert")
138     print("2. Display")
139     print("3. Exit")
140     choice = input("Enter your choice: ")
141     if choice == '1':
142         value = input("Enter value to insert: ")
143         ll.insert(value)
144     elif choice == '2':
145         ll.display()
146     elif choice == '3':
147         print("Exiting program...")
148         break
149     else:
150         print("Invalid choice! Try again.")
```

```
107 class LinkedList:
108     def display(self):
109         print("linked list elements:", " ".join(map(str, elements)))
110
111 ll = LinkedList()
112 while True:
113     print("\n1. Insert")
114     print("2. Display")
115     print("3. Exit")
116     choice = input("Enter your choice: ")
117     if choice == '1':
118         value = input("Enter value to insert: ")
119         ll.insert(value)
120     elif choice == '2':
121         ll.display()
122     elif choice == '3':
123         print("Exiting program...")
124         break
125     else:
126         print("Invalid choice! Try again.")
```

OUTPUT:

```
1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter value to insert: 11
11 inserted as head of the list.

1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter value to insert: 14
14 inserted into the list.

1. Insert
2. Display
3. Exit
Enter your choice: 1
Enter value to insert: 24
24 inserted into the list.
```

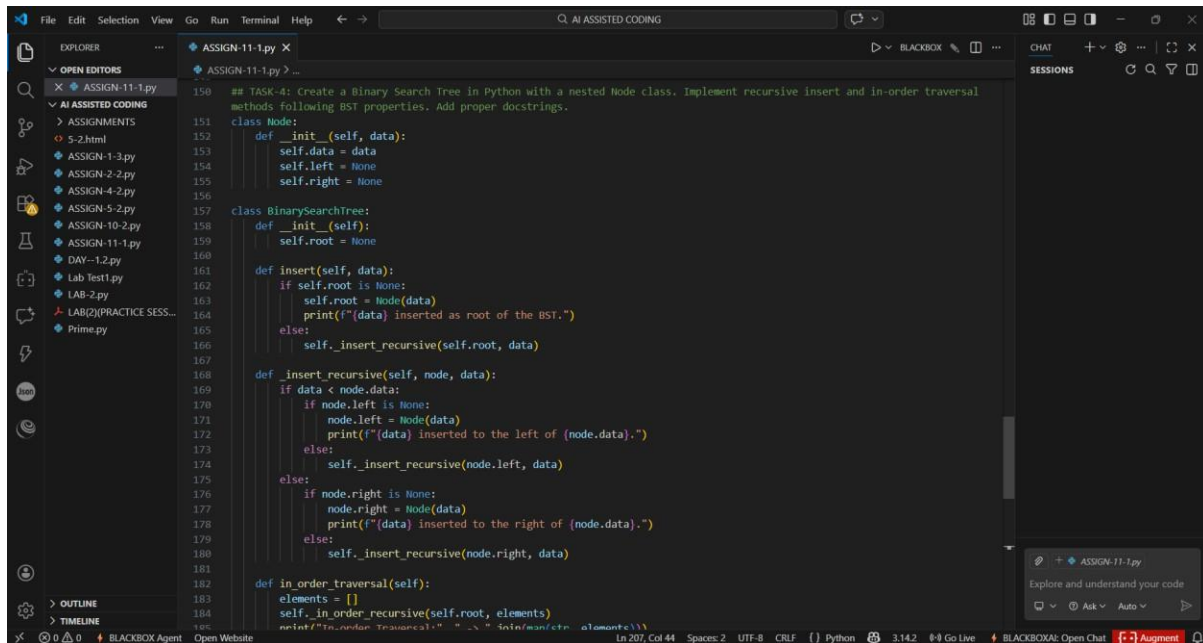
EXPLANATION: A Singly Linked List consists of nodes where each node contains data and a reference to the next node. Unlike arrays, it does not require contiguous memory, making it dynamic in size. It allows efficient insertions and deletions compared to fixed-size structures.

Task Description #4 - Binary Search Tree (BST):

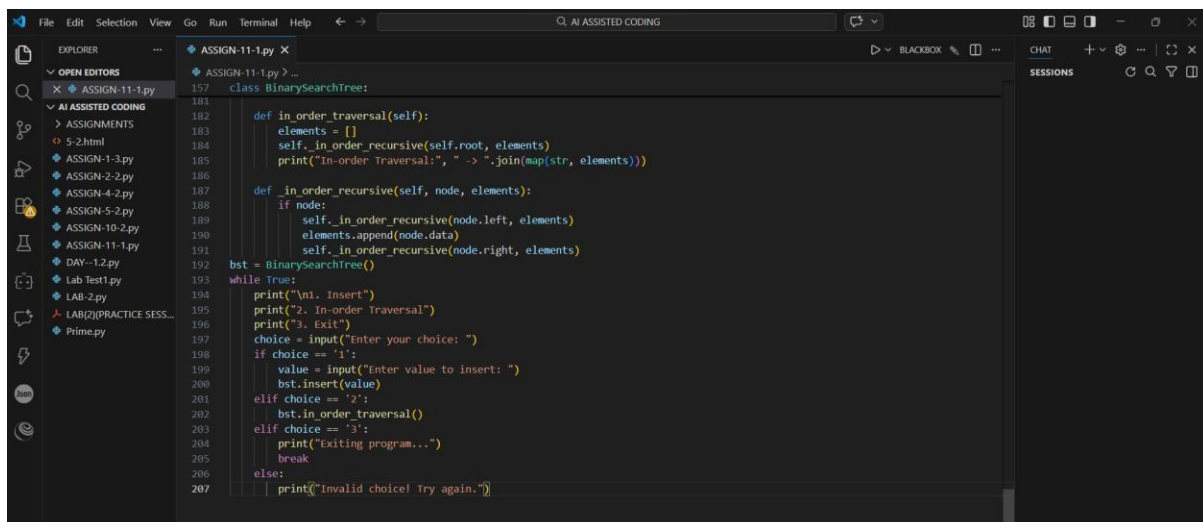
Task: Use AI to create a BST with insert and in-order traversal methods.

PROMPT: Create a Binary Search Tree in Python with a nested Node class. Implement recursive insert and in-order traversal methods following BST properties. Add proper docstrings.

Sample Input Code:

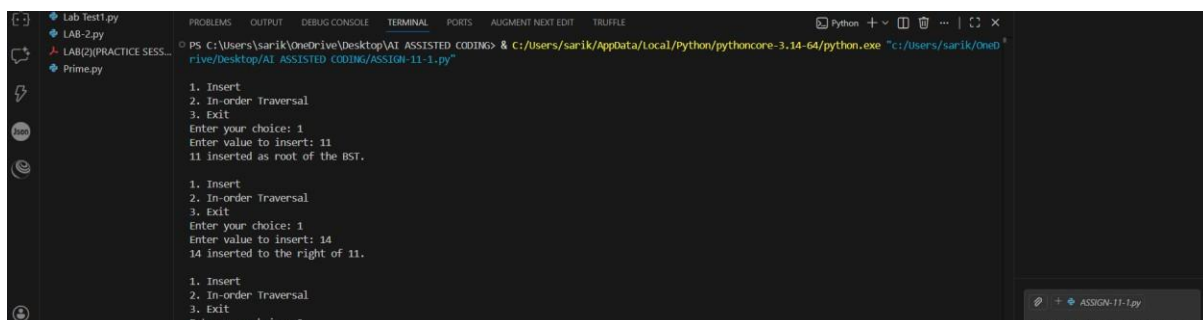


```
150 ## TASK-4: Create a Binary Search Tree in Python with a nested Node class. Implement recursive insert and in-order traversal
151 methods following BST properties. Add proper docstrings.
152 class Node:
153     def __init__(self, data):
154         self.data = data
155         self.left = None
156         self.right = None
157
158 class BinarySearchTree:
159     def __init__(self):
160         self.root = None
161
162     def insert(self, data):
163         if self.root is None:
164             self.root = Node(data)
165             print(f"{data} inserted as root of the BST.")
166         else:
167             self._insert_recursive(self.root, data)
168
169     def _insert_recursive(self, node, data):
170         if data < node.data:
171             if node.left is None:
172                 node.left = Node(data)
173                 print(f"{data} inserted to the left of {node.data}.")
174             else:
175                 self._insert_recursive(node.left, data)
176         else:
177             if node.right is None:
178                 node.right = Node(data)
179                 print(f"{data} inserted to the right of {node.data}.")
180             else:
181                 self._insert_recursive(node.right, data)
182
183     def in_order_traversal(self):
184         elements = []
185         self._in_order_recursive(self.root, elements)
186         print("In-order Traversal: ", " ".join(map(str, elements)))
```



```
187 class BinarySearchTree:
188
189     def in_order_traversal(self):
190         elements = []
191         self._in_order_recursive(self.root, elements)
192         print("In-order Traversal: ", " ".join(map(str, elements)))
193
194     def _in_order_recursive(self, node, elements):
195         if node:
196             self._in_order_recursive(node.left, elements)
197             elements.append(node.data)
198             self._in_order_recursive(node.right, elements)
199
200 bst = BinarySearchTree()
201 while True:
202     print("\n1. Insert")
203     print("2. In-order Traversal")
204     print("3. Exit")
205     choice = input("Enter your choice: ")
206     if choice == '1':
207         value = input("Enter value to insert: ")
208         bst.insert(value)
209     elif choice == '2':
210         bst.in_order_traversal()
211     elif choice == '3':
212         print("Exiting program...")
213         break
214     else:
215         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\OneDrive\Desktop\AI ASSISTED CODING\ASSIGN-11-1.py"

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: 1
Enter value to insert: 14
14 inserted to the right of 11.

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

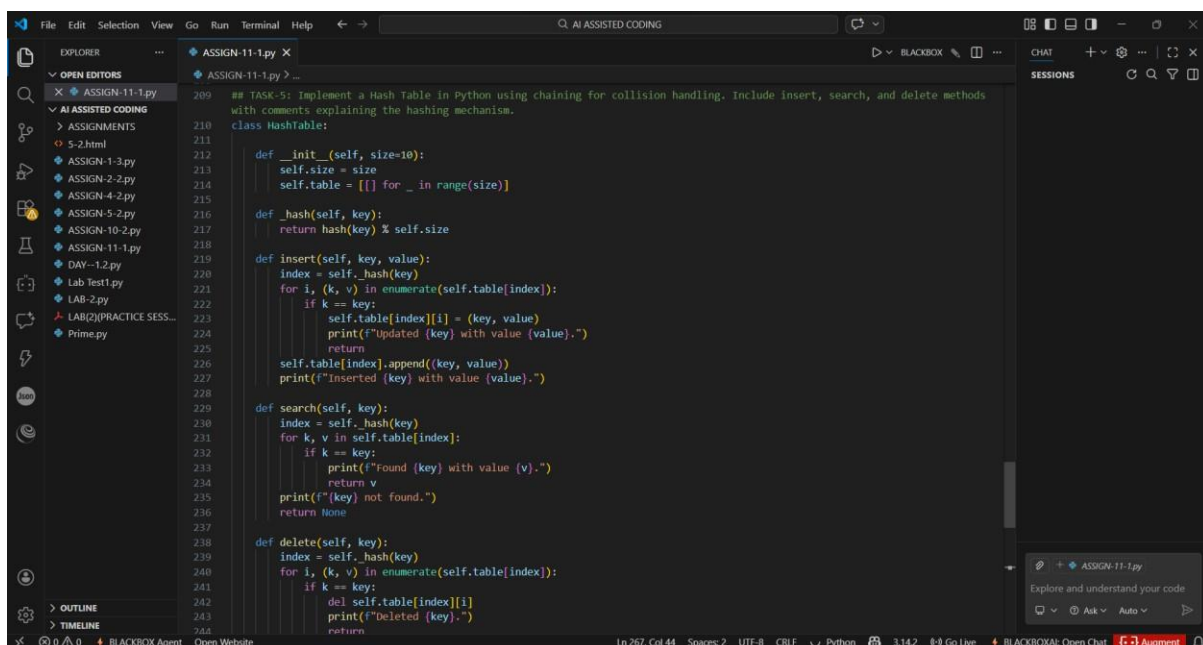

EXPLANATION: A Binary Search Tree is a hierarchical data structure where the left child contains smaller values and the right child contains larger values than the root. This property makes searching, insertion, and deletion efficient. In-order traversal of a BST produces sorted output.

Task Description #5 - Hash Table:

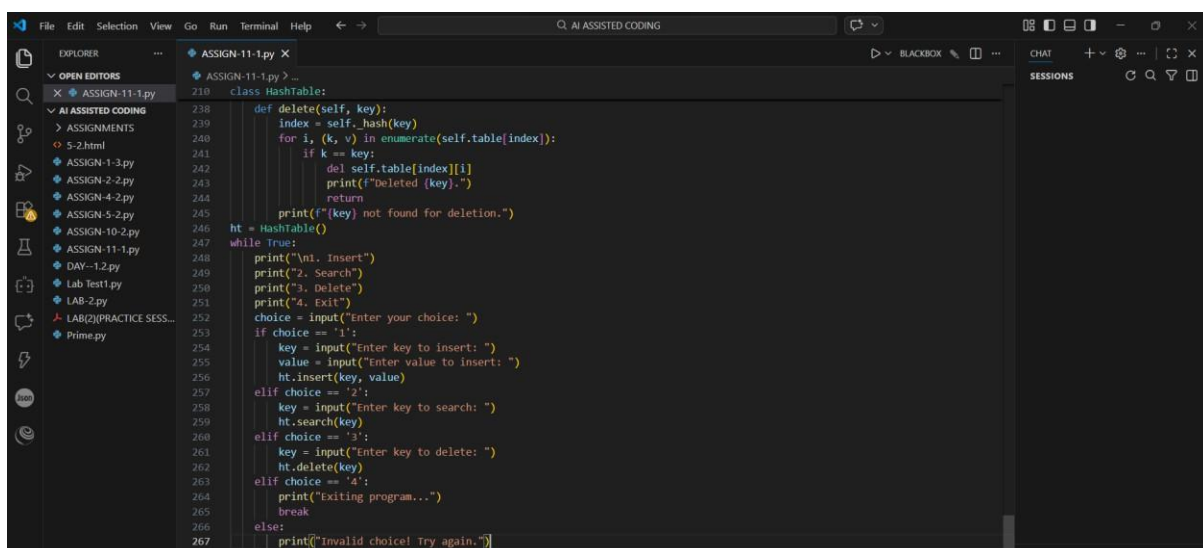
Task: Use AI to implement a hash table with basic insert, search, and delete methods.

PROMPT: Implement a Hash Table in Python using chaining for collision handling. Include insert, search, and delete methods with comments explaining the hashing mechanism.

Sample Input Code:

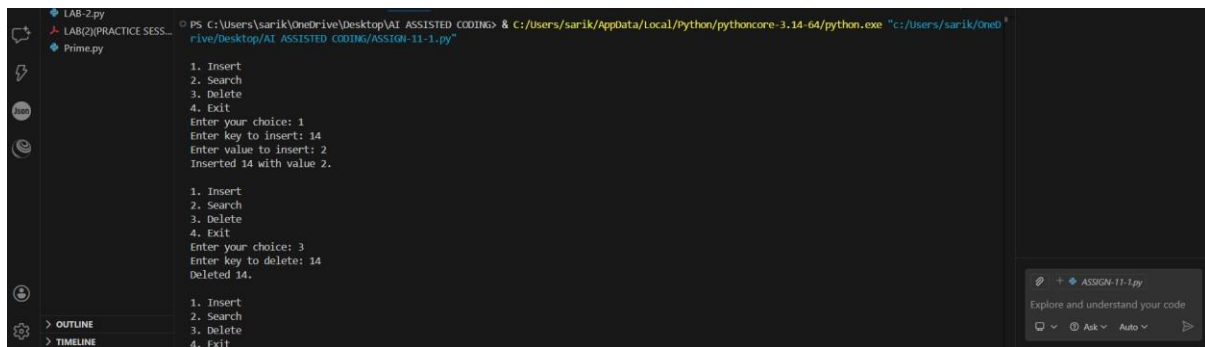


```
209 ## TASK-5: Implement a Hash Table in Python using chaining for collision handling. Include insert, search, and delete methods
210 with comments explaining the hashing mechanism.
211 class HashTable:
212     def __init__(self, size=10):
213         self.size = size
214         self.table = [[] for _ in range(size)]
215
216     def _hash(self, key):
217         return hash(key) % self.size
218
219     def insert(self, key, value):
220         index = self._hash(key)
221         for i, (k, v) in enumerate(self.table[index]):
222             if k == key:
223                 self.table[index][i] = (key, value)
224                 print(f"Updated {key} with value {value}.")
225                 return
226         self.table[index].append((key, value))
227         print(f"Inserted {key} with value {value}.")
228
229     def search(self, key):
230         index = self._hash(key)
231         for k, v in self.table[index]:
232             if k == key:
233                 print(f"Found {key} with value {v}.")
234                 return v
235         print(f"{key} not found.")
236         return None
237
238     def delete(self, key):
239         index = self._hash(key)
240         for i, (k, v) in enumerate(self.table[index]):
241             if k == key:
242                 del self.table[index][i]
243                 print(f"Deleted {key}.")
244                 return
245         print(f"{key} not found for deletion.")
```



```
238 ht = HashTable()
239 while True:
240     print("\n1. Insert")
241     print("2. Search")
242     print("3. Delete")
243     print("4. Exit")
244     choice = input("Enter your choice: ")
245     if choice == '1':
246         key = input("Enter key to insert: ")
247         value = input("Enter value to insert: ")
248         ht.insert(key, value)
249     elif choice == '2':
250         key = input("Enter key to search: ")
251         ht.search(key)
252     elif choice == '3':
253         key = input("Enter key to delete: ")
254         ht.delete(key)
255     elif choice == '4':
256         print("Exiting program...")
257         break
258     else:
259         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/OneDrive/Desktop/AI ASSISTED CODING/ASSIGN-11-1.py"

1. Insert
2. Search
3. Delete
4. Exit
Enter your choice: 1
Enter key to insert: 14
Enter value to insert: 2
Inserted 14 with value 2.

1. Insert
2. Search
3. Delete
4. Exit
Enter your choice: 3
Enter key to delete: 14
Deleted 14.

1. Insert
2. Search
3. Delete
4. Exit
```

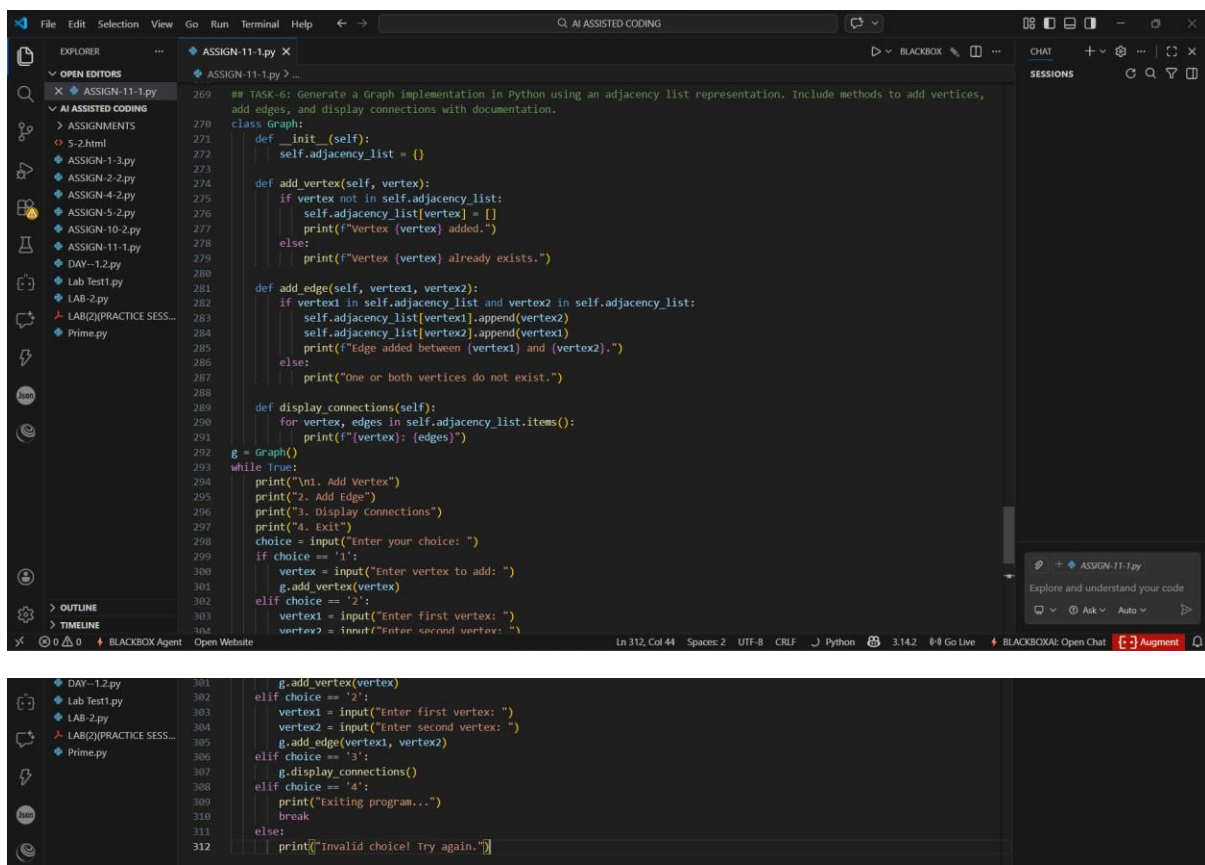
EXPLANATION: A Hash Table stores data in key-value pairs using a hash function to compute an index. It provides fast average-case time complexity for search, insertion, and deletion operations. Collisions are handled using techniques such as chaining.

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 representation. Include methods to add vertices, add edges, and display connections with documentation.

Sample Input Code:



```
## TASK-6: Generate a Graph Implementation in Python using an adjacency list representation. Include methods to add vertices,
add edges, and display connections with documentation.
class Graph:
    def __init__(self):
        self.adjacency_list = {}

    def add_vertex(self, vertex):
        if vertex not in self.adjacency_list:
            self.adjacency_list[vertex] = []
            print(f"Vertex {vertex} added.")
        else:
            print(f"Vertex {vertex} already exists.")

    def add_edge(self, vertex1, vertex2):
        if vertex1 in self.adjacency_list and vertex2 in self.adjacency_list:
            self.adjacency_list[vertex1].append(vertex2)
            self.adjacency_list[vertex2].append(vertex1)
            print(f"Edge added between {vertex1} and {vertex2}.")
        else:
            print("One or both vertices do not exist.")

    def display_connections(self):
        for vertex, edges in self.adjacency_list.items():
            print(f"{vertex}: {edges}")

g = Graph()
while True:
    print("\n1. Add Vertex")
    print("2. Add Edge")
    print("3. Display connections")
    print("4. Exit")
    choice = input("Enter your choice: ")
    if choice == '1':
        vertex = input("Enter vertex to add: ")
        g.add_vertex(vertex)
    elif choice == '2':
        vertex1 = input("Enter first vertex: ")
        vertex2 = input("Enter second vertex: ")
        g.add_edge(vertex1, vertex2)
    elif choice == '3':
        g.display_connections()
    elif choice == '4':
        print("Exiting program...")
        break
    else:
        print("Invalid choice! Try again.")
```

OUTPUT:



```
1. Add Vertex
2. Add Edge
3. Display Connections
4. Exit
Enter your choice: 1
Enter vertex to add: 14
Vertex 14 added.

1. Add Vertex
2. Add Edge
3. Display Connections
4. Exit
Enter your choice: 2
Enter first vertex: 11
Enter second vertex: 14
One or both vertices do not exist.

1. Add Vertex
2. Add Edge
3. Display Connections
4. Exit
```

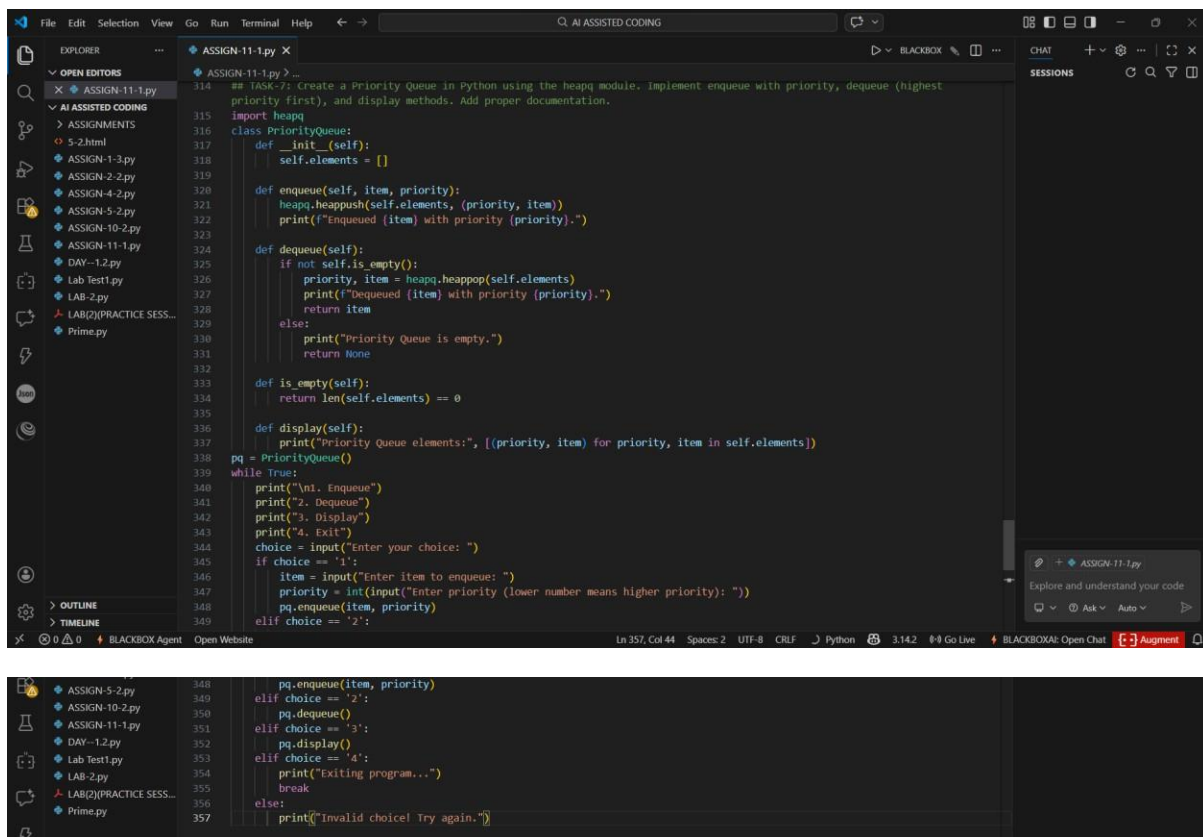
EXPLANATION: A Graph is a non-linear data structure used to represent relationships between entities. It consists of vertices (nodes) and edges (connections). Graphs are commonly used in networks, maps, and routing systems.

Task Description #7 - Priority Queue:

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

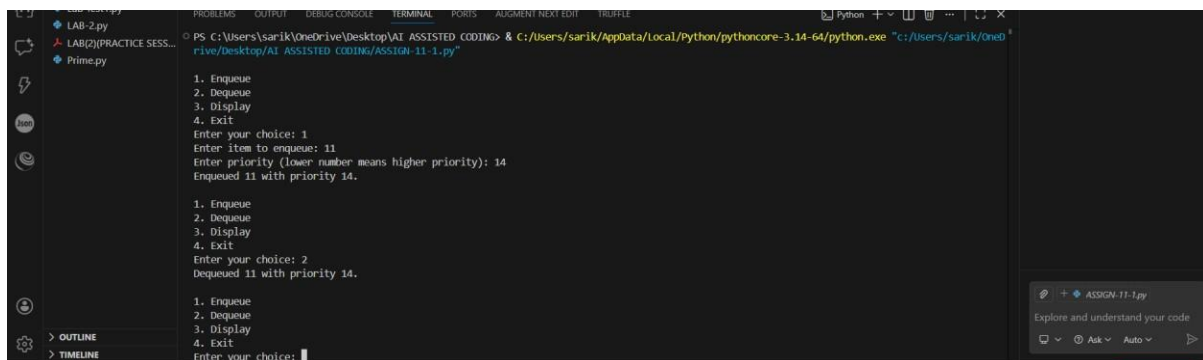
PROMPT: Create a Priority Queue in Python using the heapq module. Implement enqueue with priority, dequeue (highest priority first), and display methods. Add proper documentation.

Sample Input Code:



```
314 ## TASK-7: Create a Priority Queue in Python using the heapq module. Implement enqueue with priority, dequeue (highest
315 priority first), and display methods. Add proper documentation.
316 import heapq
317 class PriorityQueue:
318     def __init__(self):
319         self.elements = []
320
321     def enqueue(self, item, priority):
322         heapq.heappush(self.elements, (priority, item))
323         print(f"Enqueued {item} with priority {priority}.")
324
325     def dequeue(self):
326         if not self.is_empty():
327             priority, item = heapq.heappop(self.elements)
328             print(f"Dequeued {item} with priority {priority}.")
329             return item
330         else:
331             print("Priority Queue is empty.")
332             return None
333
334     def is_empty(self):
335         return len(self.elements) == 0
336
337     def display(self):
338         print("Priority Queue elements:", [(priority, item) for priority, item in self.elements])
339
340 pq = PriorityQueue()
341 while True:
342     print("\n1. Enqueue")
343     print("2. Dequeue")
344     print("3. Display")
345     print("4. Exit")
346     choice = input("Enter your choice: ")
347     if choice == '1':
348         item = input("Enter item to enqueue: ")
349         priority = int(input("Enter priority (lower number means higher priority): "))
350         pq.enqueue(item, priority)
351     elif choice == '2':
352         pq.dequeue()
353     elif choice == '3':
354         pq.display()
355     elif choice == '4':
356         print("Exiting program...")
357         break
358     else:
359         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/OneDrive/Desktop/AI ASSISTED CODING/ASSIGN-11-1.py"
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter item to enqueue: 11
Enter priority (lower number means higher priority): 14
Enqueued 11 with priority 14.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued 11 with priority 14.

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
```

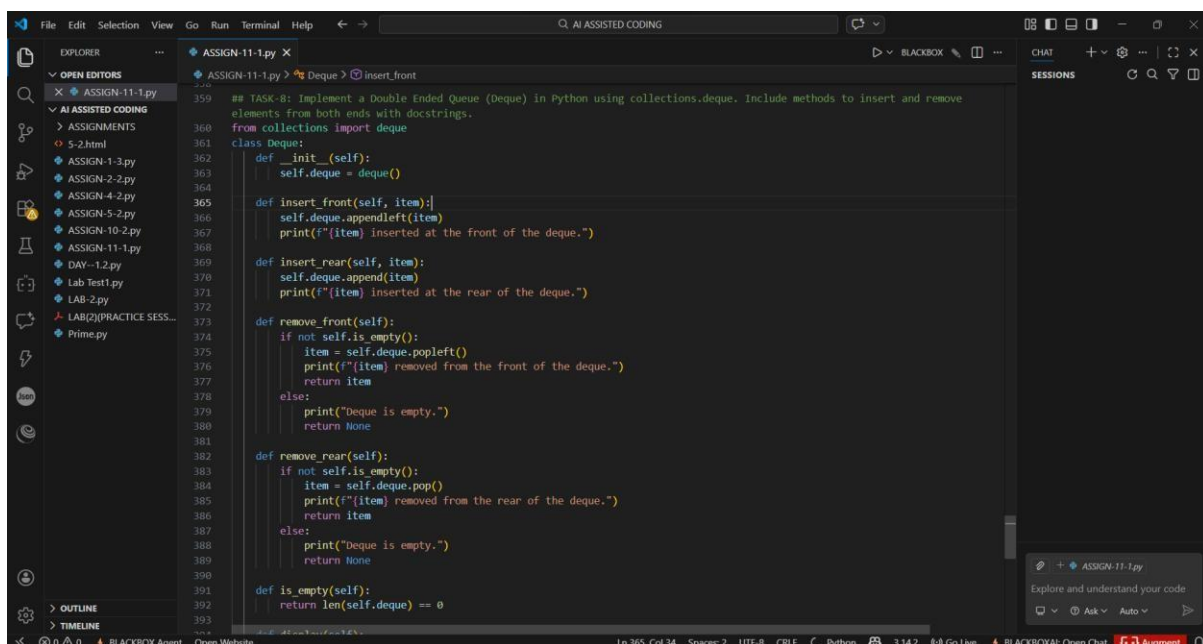
EXPLANATION: A Priority Queue is a special type of queue where elements are removed based on priority rather than order of insertion. Higher priority elements are processed first. It is typically implemented using a heap for efficiency.

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 elements from both ends with docstrings.

Sample Input Code:



```
## TASK-8: Implement a Double Ended Queue (Deque) in Python using collections.deque. Include methods to insert and remove
elements from both ends with docstrings.
from collections import deque

class Deque:
    def __init__(self):
        self.deque = deque()

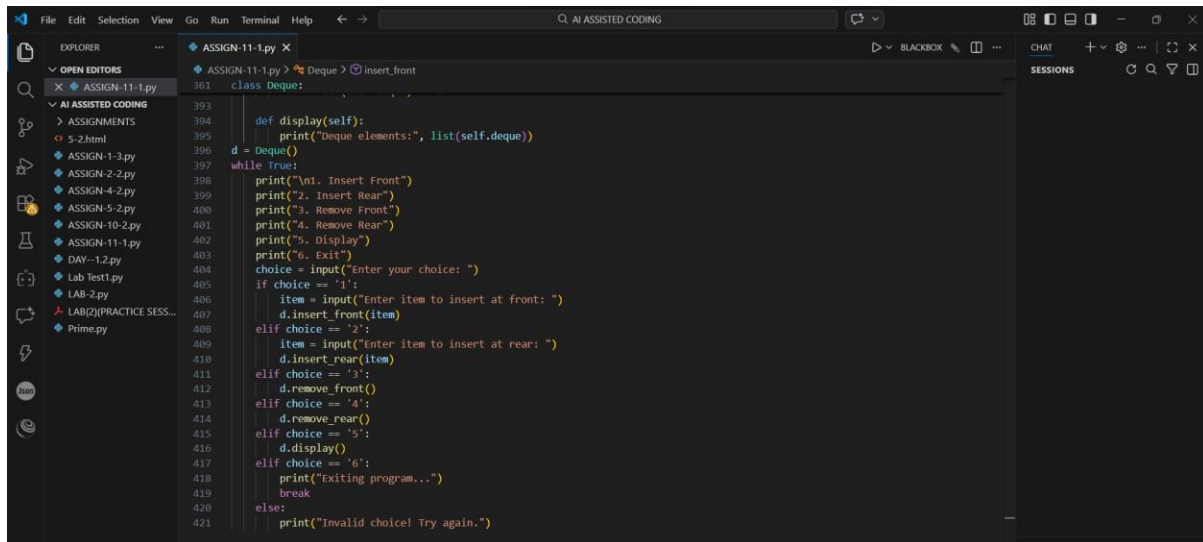
    def insert_front(self, item):
        self.deque.appendleft(item)
        print(f"{item} Inserted at the front of the deque.")

    def insert_rear(self, item):
        self.deque.append(item)
        print(f"{item} Inserted at the rear of the deque.")

    def remove_front(self):
        if not self.is_empty():
            item = self.deque.popleft()
            print(f"{item} removed from the front of the deque.")
            return item
        else:
            print("Deque is empty.")
            return None

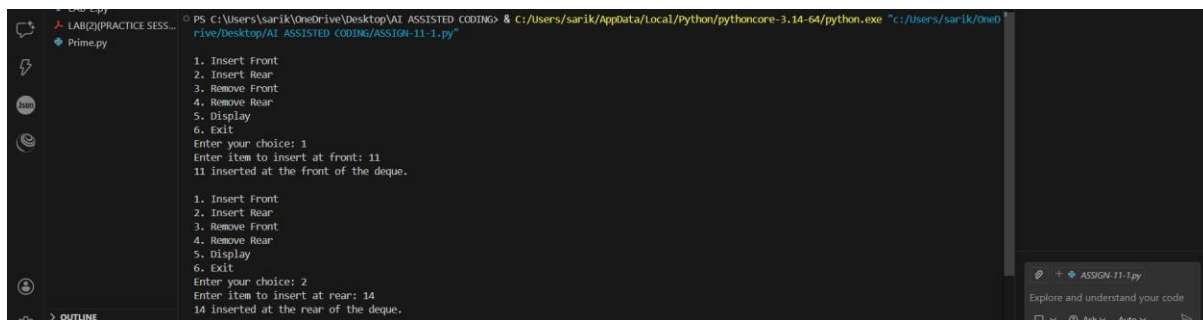
    def remove_rear(self):
        if not self.is_empty():
            item = self.deque.pop()
            print(f"{item} removed from the rear of the deque.")
            return item
        else:
            print("Deque is empty.")
            return None

    def is_empty(self):
        return len(self.deque) == 0
```



```
File Edit Selection View Go Run Terminal Help AI ASSISTED CODING
EXPLORER ASSIGN-11-1.py X
OPEN EDITORS
X ASSIGN-11-1.py Deque > Insert_front
AI ASSISTED CODING
ASSIGNMENTS
5-2.html
ASSIGN-1-3.py
ASSIGN-2-2.py
ASSIGN-5-2.py
ASSIGN-10-2.py
ASSIGN-11-1.py
DAY-1-2.py
Lab Test1.py
LAB-2.py
LAB(2)(PRACTICE SESS...
Prime.py
361 class Deque:
393
394     def display(self):
395         print("Deque elements:", list(self.deque))
396     d = Deque()
397     while True:
398         print("\n1. Insert Front")
399         print("2. Insert Rear")
400         print("3. Remove Front")
401         print("4. Remove Rear")
402         print("5. Display")
403         print("6. Exit")
404         choice = input("Enter your choice: ")
405         if choice == '1':
406             item = input("Enter item to insert at front: ")
407             d.insert_front(item)
408         elif choice == '2':
409             item = input("Enter item to insert at rear: ")
410             d.insert_rear(item)
411         elif choice == '3':
412             d.remove_front()
413         elif choice == '4':
414             d.remove_rear()
415         elif choice == '5':
416             d.display()
417         elif choice == '6':
418             print("Exiting program...")
419             break
420         else:
421             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
rive/Desktop/AI ASSISTED CODING/ASSIGN-11-1.py"
1. Insert Front
2. Insert Rear
3. Remove Front
4. Remove Rear
5. Display
6. Exit
Enter your choice: 1
Enter item to insert at front: 11
11 inserted at the front of the deque.

1. Insert Front
2. Insert Rear
3. Remove Front
4. Remove Rear
5. Display
6. Exit
Enter your choice: 2
Enter item to insert at rear: 14
14 inserted at the rear of the deque.
```

EXPLANATION: A Deque (Double Ended Queue) allows insertion and deletion of elements from both the front and rear ends. It combines features of both stacks and queues. It is useful in applications like sliding window algorithms and task scheduling.