

# Assignment-11.1

## Data Structures with AI: Implementing Fundamental Structures

K. Santhosh Kumar

2403a51l21

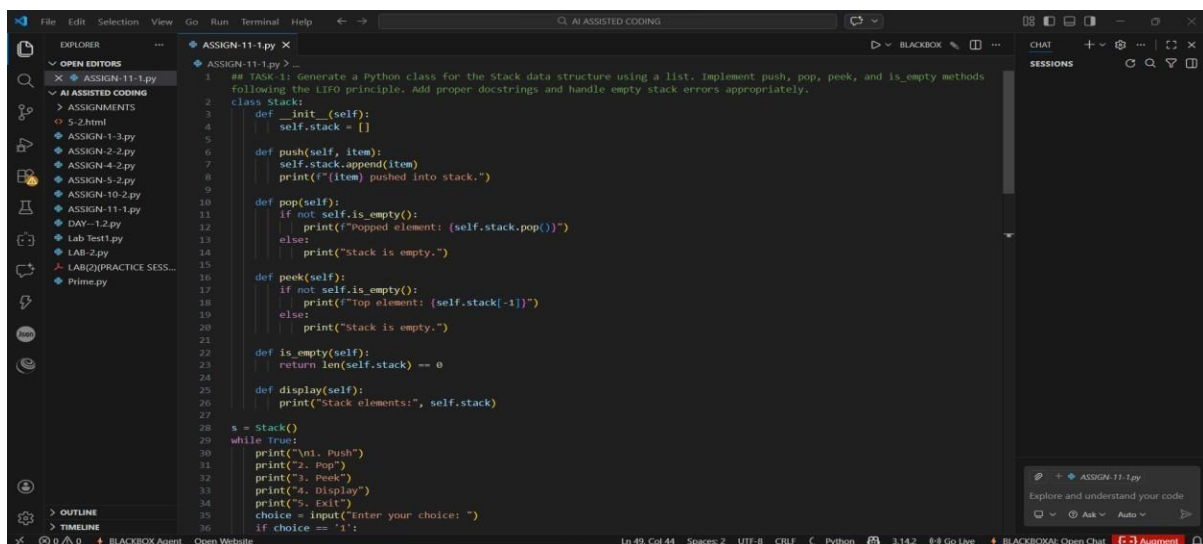
B-51

### Task Description #1 - Stack Implementation:

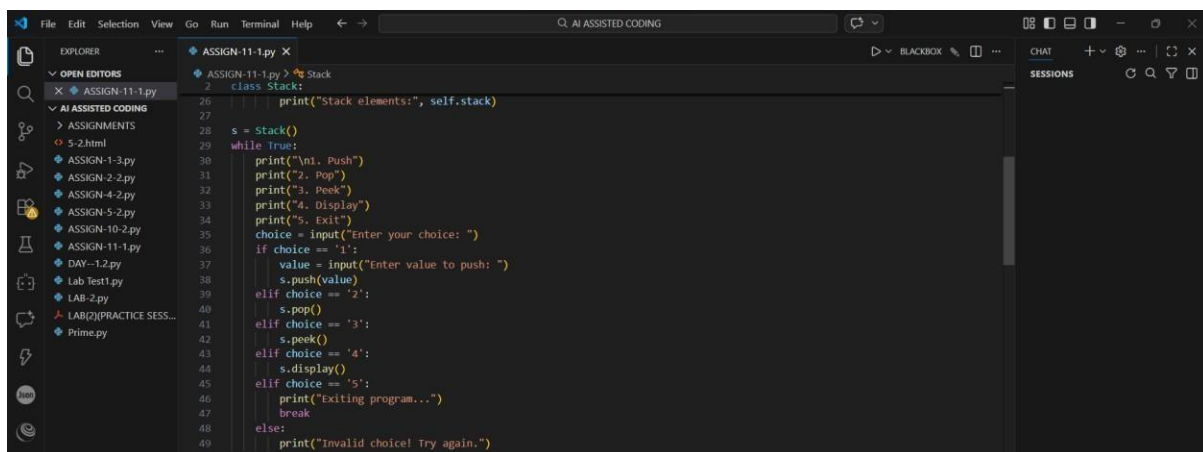
**Task:** Use AI to generate a Stack class with push, pop, peek, and is\_empty methods.

**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:

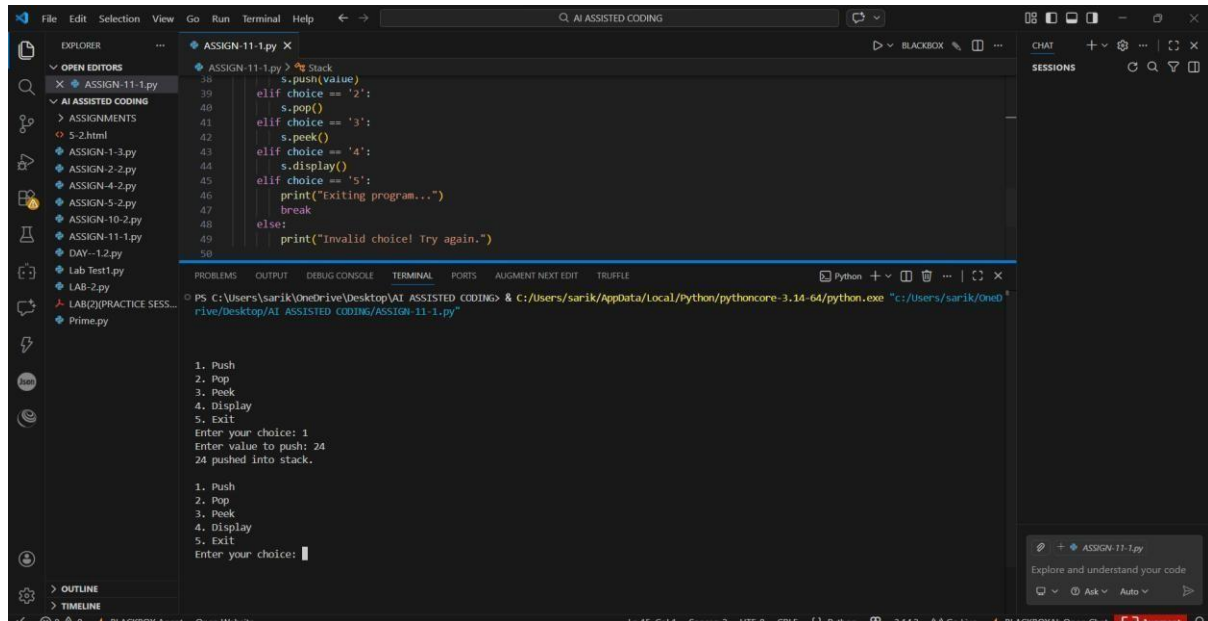


```
1  ## TASK 1: Generate a Python class for the Stack data structure using a list. Implement push, pop, peek, and is_empty methods
2  following the LIFO principle. Add proper docstrings and handle empty stack errors appropriately.
3  class Stack:
4      def __init__(self):
5          self.stack = []
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 class Stack:
27     def __init__(self):
28         self.stack = []
29
30     def push(self, item):
31         self.stack.append(item)
32         print(f"{item} pushed into stack.")
33
34     def pop(self):
35         if not self.is_empty():
36             print(f"Popped element: {self.stack.pop()}")
37         else:
38             print("Stack is empty.")
39
40     def peek(self):
41         if not self.is_empty():
42             print(f"Top element: {self.stack[-1]}")
43         else:
44             print("Stack is empty.")
45
46     def is_empty(self):
47         return len(self.stack) == 0
48
49     def display(self):
50         print("Stack elements:", self.stack)
51
52 s = Stack()
53 while True:
54     print("\n1. Push")
55     print("2. Pop")
56     print("3. Peek")
57     print("4. Display")
58     print("5. Exit")
59     choice = input("Enter your choice: ")
60     if choice == '1':
61         value = input("Enter value to push: ")
62         s.push(value)
63     elif choice == '2':
64         s.pop()
65     elif choice == '3':
66         s.peek()
67     elif choice == '4':
68         s.display()
69     elif choice == '5':
70         print("Exiting program...")
71         break
72     else:
73         print("Invalid choice! Try again.")
```

## OUTPUT:



```
File Edit Selection View Go Run Terminal Help
ASSIGN-11-1.py X
EXPLORER
OPEN EDITORS
X 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
ASSIGN-11-1.py
DAY-1-2.py
Lab Test1.py
LAB-2.py
LAB2(PRACTICE SESS...
Prime.py
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AUGMENT NEXT EDIT TRUFFLE
Python + Python - Python Python
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: |
```

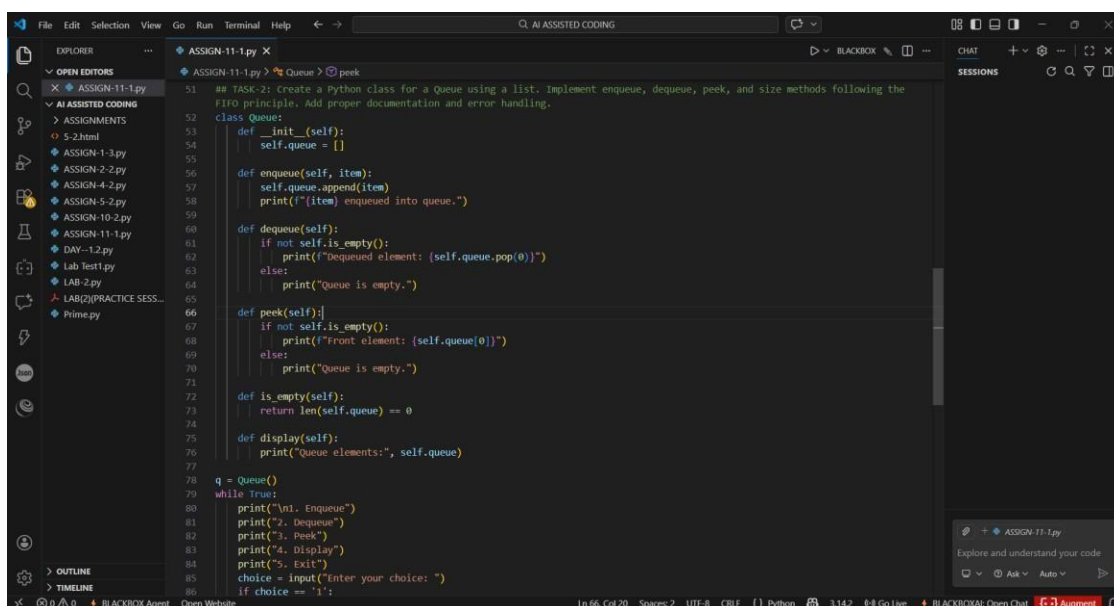
**EXPLANATION:** A Stack is a linear data structure that follows the LIFO (Last In First Out) principle, where the last element inserted is the first one removed. Operations such as push, pop, and peek are performed at one end called the top. It is commonly used in function calls, undo operations, and 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
ASSIGN-11-1.py X
EXPLORER
OPEN EDITORS
X 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
ASSIGN-11-1.py
DAY-1-2.py
Lab Test1.py
LAB-2.py
LAB2(PRACTICE SESS...
Prime.py
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AUGMENT NEXT EDIT TRUFFLE
Python + Python - Python Python
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: |
```

```
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\OneD
rive\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
104 class Node:
105     def __init__(self, data):
106         self.data = data
107         self.next = None
108
109 class LinkedList:
110     def __init__(self):
111         self.head = None
112
113     def insert(self, data):
114         new_node = Node(data)
115         if self.head is None:
116             self.head = new_node
117             print(f"{data} inserted as head of the list.")
118         else:
119             current = self.head
120             while current.next:
121                 current = current.next
122             current.next = new_node
123             print(f"{data} inserted into the list.")
124
125     def display(self):
126         if self.head is None:
127             print("The linked list is empty.")
128         else:
129             current = self.head
130             elements = []
131             while current:
132                 elements.append(current.data)
133                 current = current.next
134             print("Linked List elements:", " -> ".join(map(str, elements)))
135
136 ll = LinkedList()
137 while True:
138     print("\n1. Insert")
139     print("2. Display")
140     print("3. Exit")
141     choice = input("Enter your choice: ")
142     if choice == '1':
143         value = input("Enter value to insert: ")
144         ll.insert(value)
145     elif choice == '2':
146         ll.display()
147     elif choice == '3':
148         print("Exiting program...")
149         break
150     else:
151         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. 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.
```

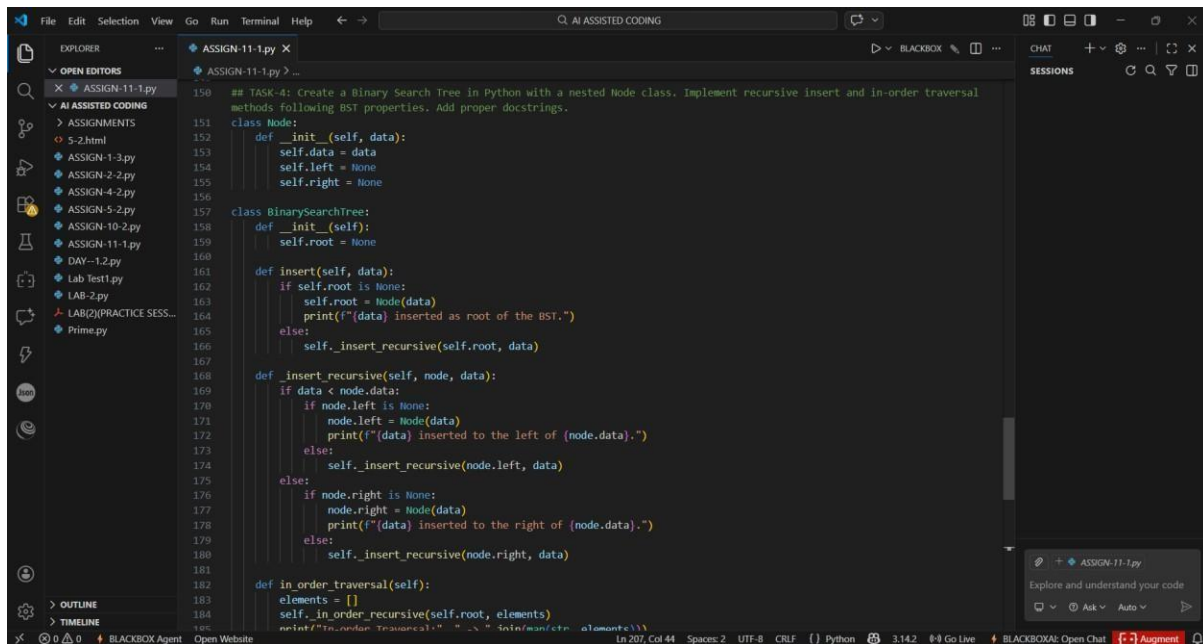
**EXPLANATION:** A Singly Linked List consists of nodes where each node contains data and a reference to the next node. Unlike arrays, it does not require contiguous memory, making it dynamic in size. It allows efficient insertions and deletions compared to fixed-size structures.

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

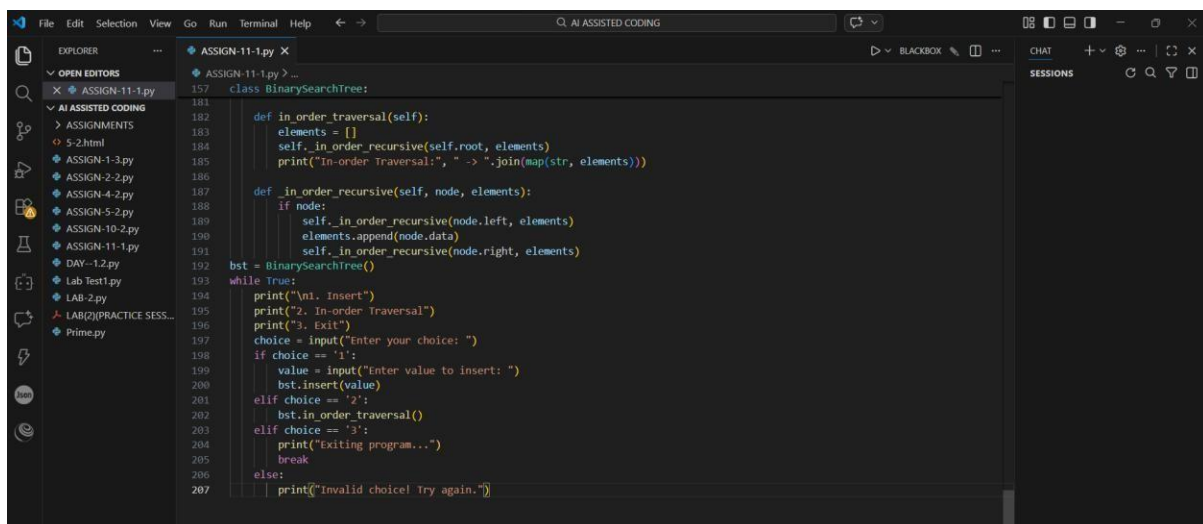
**Task:** Use AI to create a BST with insert and in-order traversal methods.

**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:**

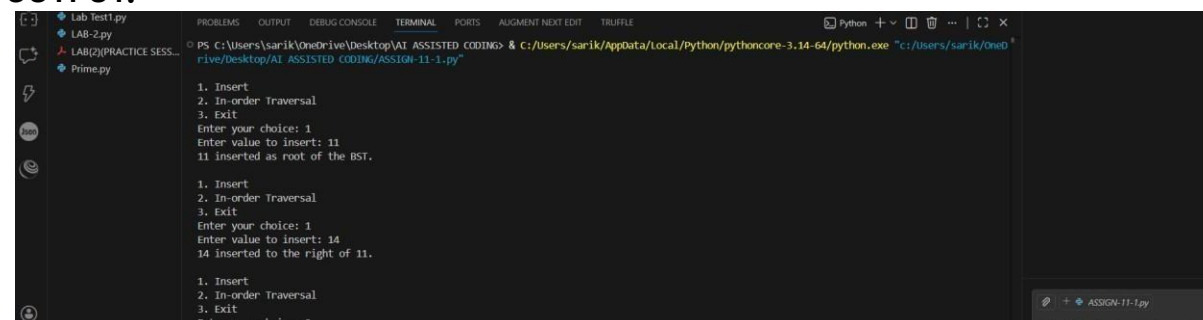


```
150 ## TASK-4: Create a Binary Search Tree in Python with a nested Node class. Implement recursive insert and in-order traversal
151 methods following BST properties. Add proper docstrings.
152
153 class Node:
154     def __init__(self, data):
155         self.data = data
156         self.left = None
157         self.right = None
158
159 class BinarySearchTree:
160     def __init__(self):
161         self.root = None
162
163     def insert(self, data):
164         if self.root is None:
165             self.root = Node(data)
166             print(f"{data} inserted as root of the BST.")
167         else:
168             self._insert_recursive(self.root, data)
169
170     def _insert_recursive(self, node, data):
171         if data < node.data:
172             if node.left is None:
173                 node.left = Node(data)
174                 print(f"{data} inserted to the left of {node.data}.")
175             else:
176                 self._insert_recursive(node.left, data)
177         else:
178             if node.right is None:
179                 node.right = Node(data)
180                 print(f"{data} inserted to the right of {node.data}.")
181             else:
182                 self._insert_recursive(node.right, data)
183
184     def in_order_traversal(self):
185         elements = []
186         self._in_order_recursive(self.root, elements)
187         print("In-order Traversal: ", " ".join(map(str, elements)))
188
189     def _in_order_recursive(self, node, elements):
190         if node:
191             self._in_order_recursive(node.left, elements)
192             elements.append(node.data)
193             self._in_order_recursive(node.right, elements)
194
195 bst = BinarySearchTree()
196 while True:
197     print("\n1. Insert")
198     print("2. In-order Traversal")
199     print("3. Exit")
200     choice = input("Enter your choice: ")
201     if choice == '1':
202         value = input("Enter value to insert: ")
203         bst.insert(value)
204     elif choice == '2':
205         bst.in_order_traversal()
206     elif choice == '3':
207         print("Exiting program...")
208         break
209     else:
210         print("Invalid choice! Try again.")
```



```
157 class BinarySearchTree:
158
159     def in_order_traversal(self):
160         elements = []
161         self._in_order_recursive(self.root, elements)
162         print("In-order Traversal: ", " ".join(map(str, elements)))
163
164     def _in_order_recursive(self, node, elements):
165         if node:
166             self._in_order_recursive(node.left, elements)
167             elements.append(node.data)
168             self._in_order_recursive(node.right, elements)
169
170 bst = BinarySearchTree()
171 while True:
172     print("\n1. Insert")
173     print("2. In-order Traversal")
174     print("3. Exit")
175     choice = input("Enter your choice: ")
176     if choice == '1':
177         value = input("Enter value to insert: ")
178         bst.insert(value)
179     elif choice == '2':
180         bst.in_order_traversal()
181     elif choice == '3':
182         print("Exiting program...")
183         break
184     else:
185         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
```



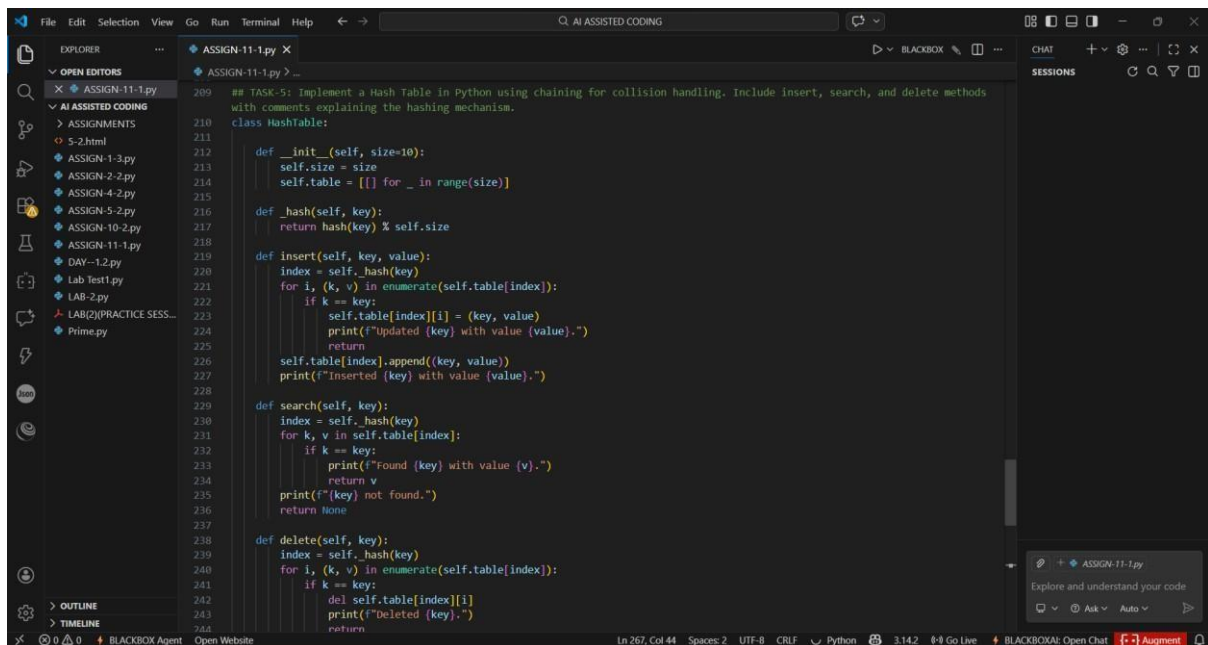
**EXPLANATION:** A Binary Search Tree is a hierarchical data structure where the left child contains smaller values and the right child contains larger values than the root. This property makes searching, insertion, and deletion efficient. In-order traversal of a BST produces sorted output.

## Task Description #5 - Hash Table:

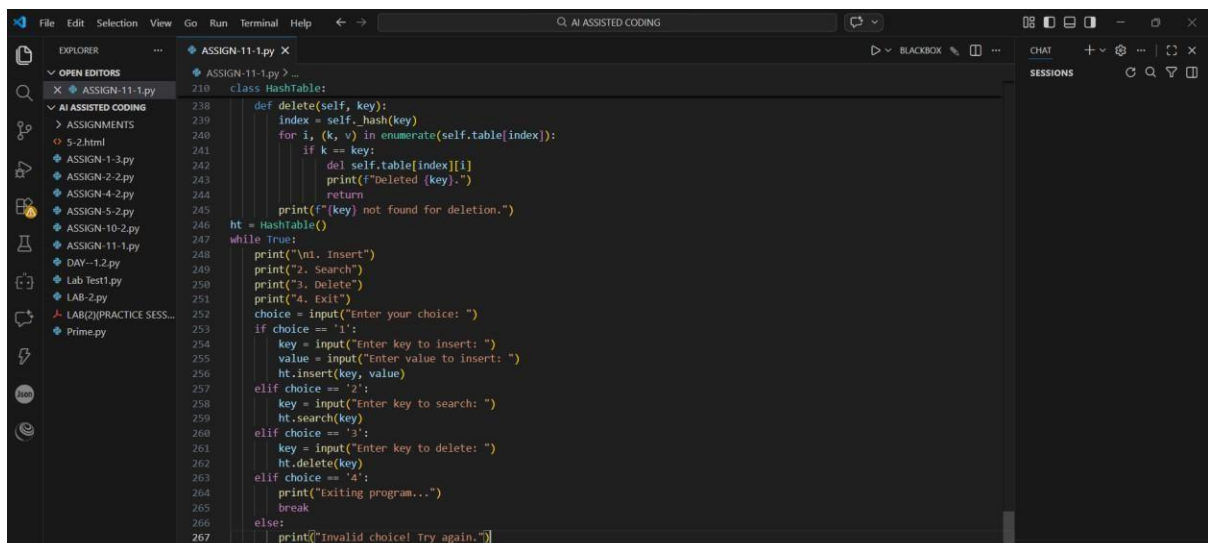
**Task:** Use AI to implement a hash table with basic insert, search, and delete methods.

**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:

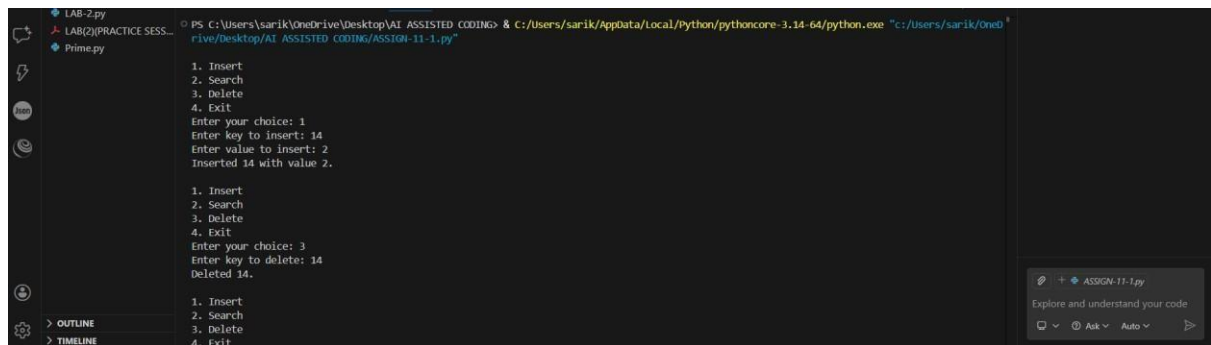


```
209 ## TASK-5: Implement a Hash Table in Python using chaining for collision handling. Include insert, search, and delete methods
210 with comments explaining the hashing mechanism.
211 class HashTable:
212     def __init__(self, size=10):
213         self.size = size
214         self.table = [[] 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.")
```



```
246 ht = HashTable()
247 while True:
248     print("\n1. Insert")
249     print("2. Search")
250     print("3. Delete")
251     print("4. Exit")
252     choice = input("Enter your choice: ")
253     if choice == '1':
254         key = input("Enter key to insert: ")
255         value = input("Enter value to insert: ")
256         ht.insert(key, value)
257     elif choice == '2':
258         key = input("Enter key to search: ")
259         ht.search(key)
260     elif choice == '3':
261         key = input("Enter key to delete: ")
262         ht.delete(key)
263     elif choice == '4':
264         print("Exiting program...")
265         break
266     else:
267         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.

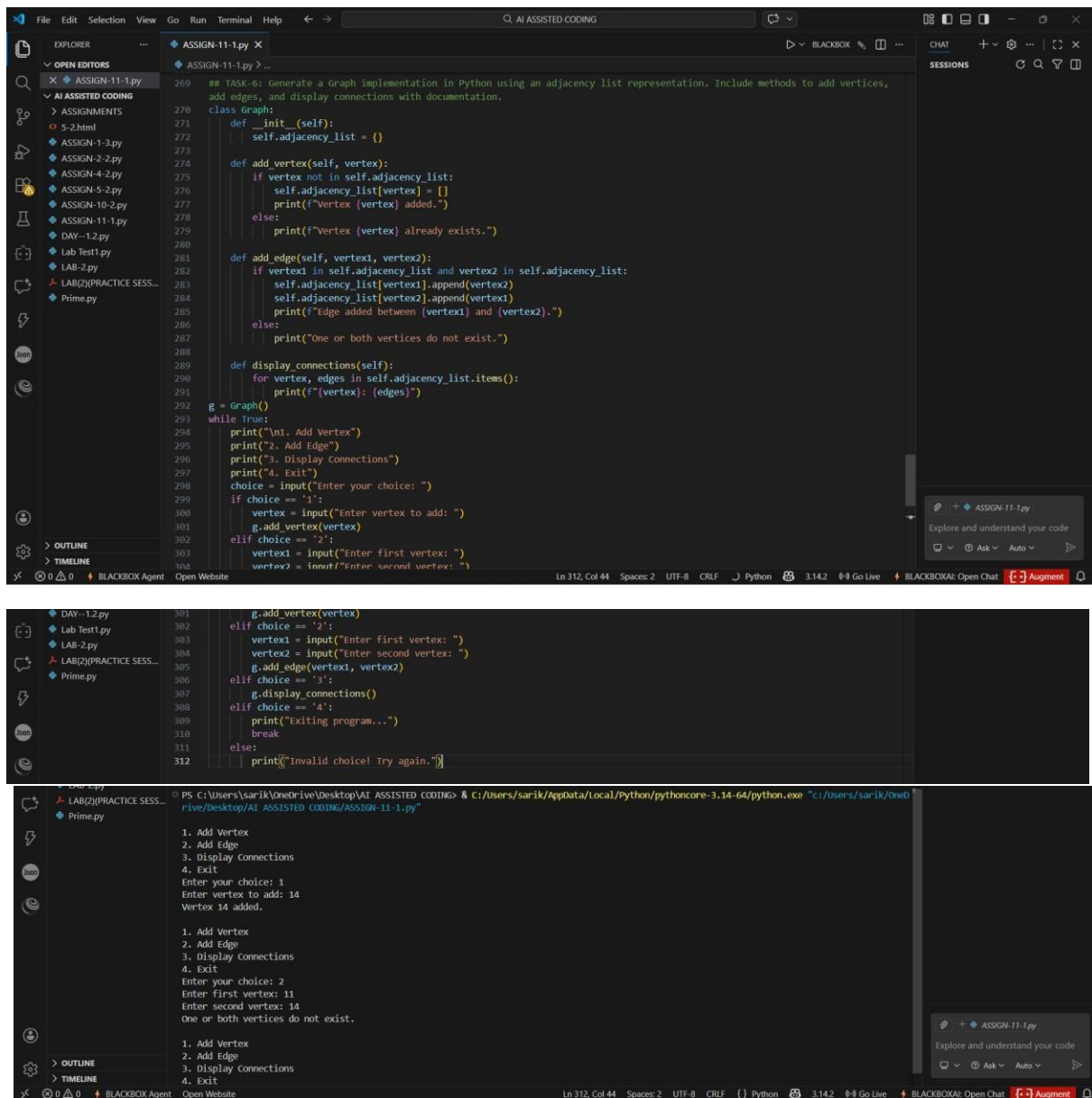
### Task Description #6 - Graph Representation:

**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:

## OUTPUT:



```
269 ## TASK-6: Generate a Graph implementation in Python using an adjacency list representation. Include methods to add vertices,
270 add edges, and display connections with documentation.
271 class Graph:
272     def __init__(self):
273         self.adjacency_list = {}
274
275     def add_vertex(self, vertex):
276         if vertex not in self.adjacency_list:
277             self.adjacency_list[vertex] = []
278             print(f"Vertex {vertex} added.")
279         else:
280             print(f"Vertex {vertex} already exists.")
281
282     def add_edge(self, vertex1, vertex2):
283         if vertex1 in self.adjacency_list and vertex2 in self.adjacency_list:
284             self.adjacency_list[vertex1].append(vertex2)
285             self.adjacency_list[vertex2].append(vertex1)
286             print(f"Edge added between {vertex1} and {vertex2}.")
287         else:
288             print("One or both vertices do not exist.")
289
290     def display_connections(self):
291         for vertex, edges in self.adjacency_list.items():
292             print(f"{vertex}: {edges}")
293
294 g = Graph()
295 while True:
296     print("\n1. Add Vertex")
297     print("2. Add Edge")
298     print("3. Display connections")
299     print("4. Exit")
300     choice = input("Enter your choice: ")
301     if choice == '1':
302         vertex = input("Enter vertex to add: ")
303         g.add_vertex(vertex)
304     elif choice == '2':
305         vertex1 = input("Enter first vertex: ")
306         vertex2 = input("Enter second vertex: ")
307         g.add_edge(vertex1, vertex2)
308     elif choice == '3':
309         g.display_connections()
310     elif choice == '4':
311         print("Exiting program...")
312         break
313     else:
314         print("Invalid choice! Try again.")
```

```
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:

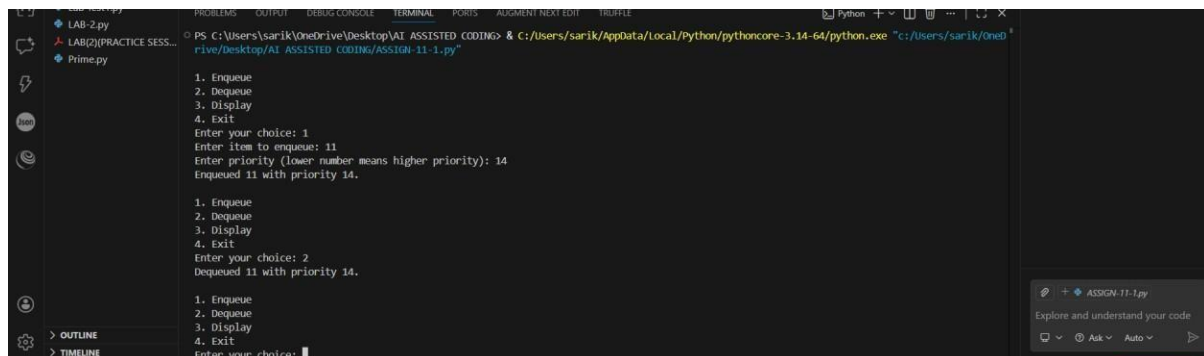


## OUTPUT:

**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:**

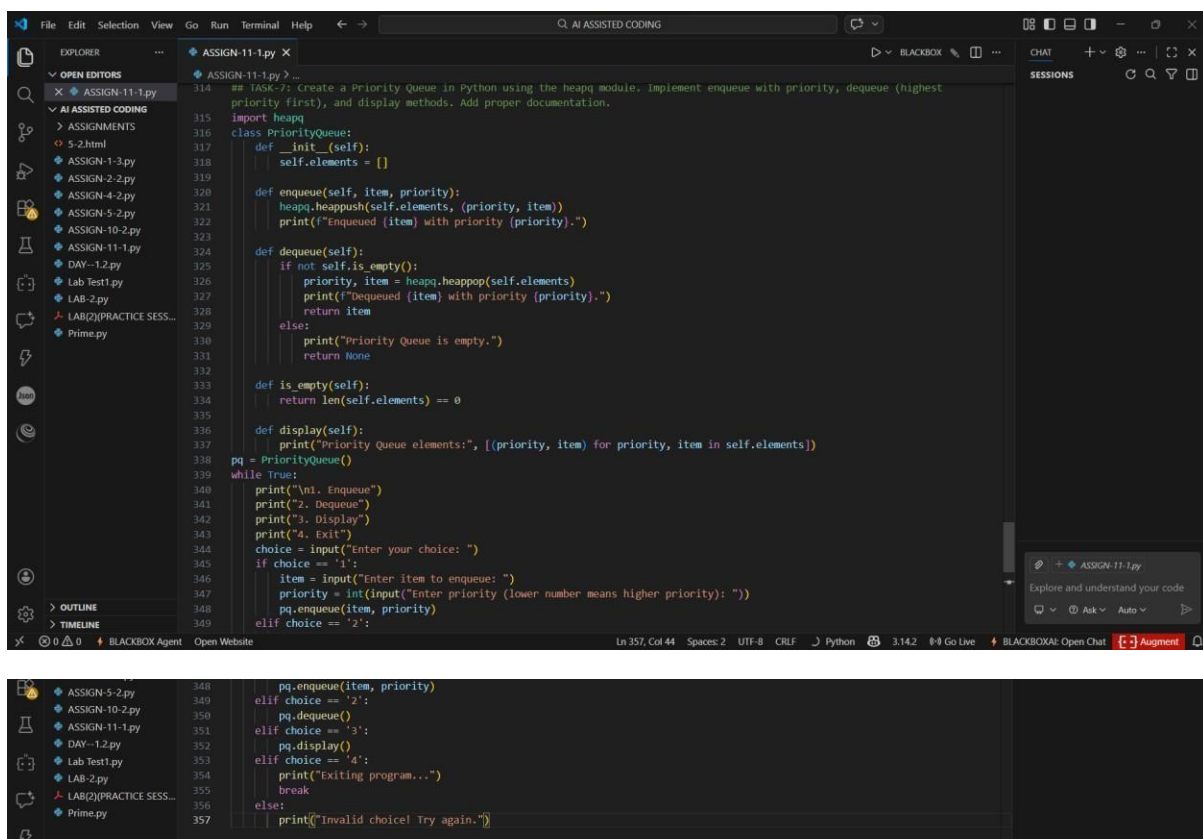


```
1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter item to enqueue: 11
Enter priority (lower number means higher priority): 14
Enqueued 11 with priority 14.

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: 1
```

**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.



```
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  """
317  import heapq
318  class PriorityQueue:
319      def __init__(self):
320          self.elements = []
321
322      def enqueue(self, item, priority):
323          heapq.heappush(self.elements, (priority, item))
324          print(f"Enqueued {item} with priority {priority}.")
325
326      def dequeue(self):
327          if not self.is_empty():
328              priority, item = heapq.heappop(self.elements)
329              print(f"Dequeued {item} with priority {priority}.")
330              return item
331          else:
332              print("Priority Queue is empty.")
333              return None
334
335      def is_empty(self):
336          return len(self.elements) == 0
337
338      def display(self):
339          print("Priority Queue elements:", [(priority, item) for priority, item in self.elements])
340
341  pq = PriorityQueue()
342  while True:
343      print("\n1. Enqueue")
344      print("2. Dequeue")
345      print("3. Display")
346      print("4. Exit")
347      choice = input("Enter your choice: ")
348      if choice == '1':
349          item = input("Enter item to enqueue: ")
350          priority = int(input("Enter priority (lower number means higher priority): "))
351          pq.enqueue(item, priority)
352      elif choice == '2':
353          pq.dequeue()
354      elif choice == '3':
355          pq.display()
356      elif choice == '4':
357          print("Exiting program...")
358          break
359      else:
360          print("Invalid choice! Try again.")
```

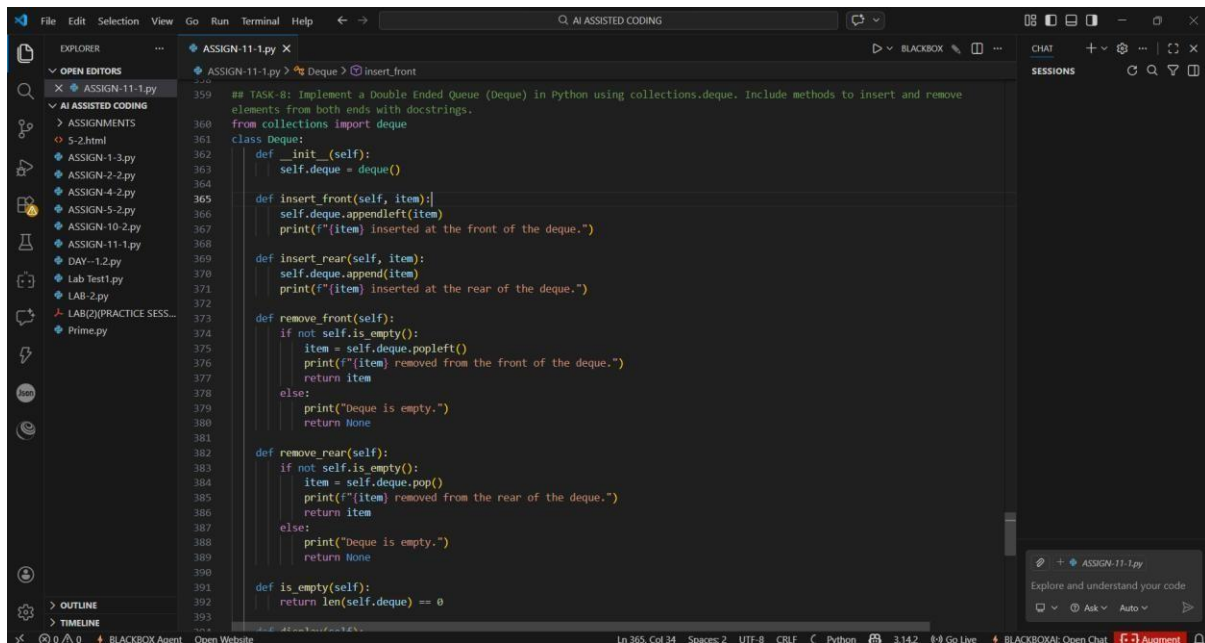
## OUTPUT:

### 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:



```
359 ## TASK-8: Implement a Double Ended Queue (Deque) in Python using collections.deque. Include methods to insert and remove
360 elements from both ends with docstrings.
361 from collections import deque
362 class Deque:
363     def __init__(self):
364         self.deque = deque()
365
366     def insert_front(self, item):
367         self.deque.appendleft(item)
368         print(f'{item} inserted at the front of the deque.')
369
370     def insert_rear(self, item):
371         self.deque.append(item)
372         print(f'{item} inserted at the rear of the deque.')
373
374     def remove_front(self):
375         if not self.is_empty():
376             item = self.deque.popleft()
377             print(f'{item} removed from the front of the deque.')
378             return item
379         else:
380             print("Deque is empty.")
381             return None
382
383     def remove_rear(self):
384         if not self.is_empty():
385             item = self.deque.pop()
386             print(f'{item} removed from the rear of the deque.')
387             return item
388         else:
389             print("Deque is empty.")
390             return None
391
392     def is_empty(self):
393         return len(self.deque) == 0
```

```
361 class Deque:
362
363     def display(self):
364         print("Deque elements:", list(self.deque))
365
366     d = Deque()
367     while True:
368         print("\n1. Insert Front")
369         print("2. Insert Rear")
370         print("3. Remove Front")
371         print("4. Remove Rear")
372         print("5. Display")
373         print("6. Exit")
374         choice = input("Enter your choice: ")
375         if choice == '1':
376             item = input("Enter item to insert at front: ")
377             d.insert_front(item)
378         elif choice == '2':
379             item = input("Enter item to insert at rear: ")
380             d.insert_rear(item)
381         elif choice == '3':
382             d.remove_front()
383         elif choice == '4':
384             d.remove_rear()
385         elif choice == '5':
386             d.display()
387         elif choice == '6':
388             print("Exiting program...")
389             break
390         else:
391             print("Invalid choice! Try again.")
```

OUTPUT:

```
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.