

# ASSIGNMENT 11.1

## Data Structures with AI: Implementing Fundamental Structures

Vyshnavi . P

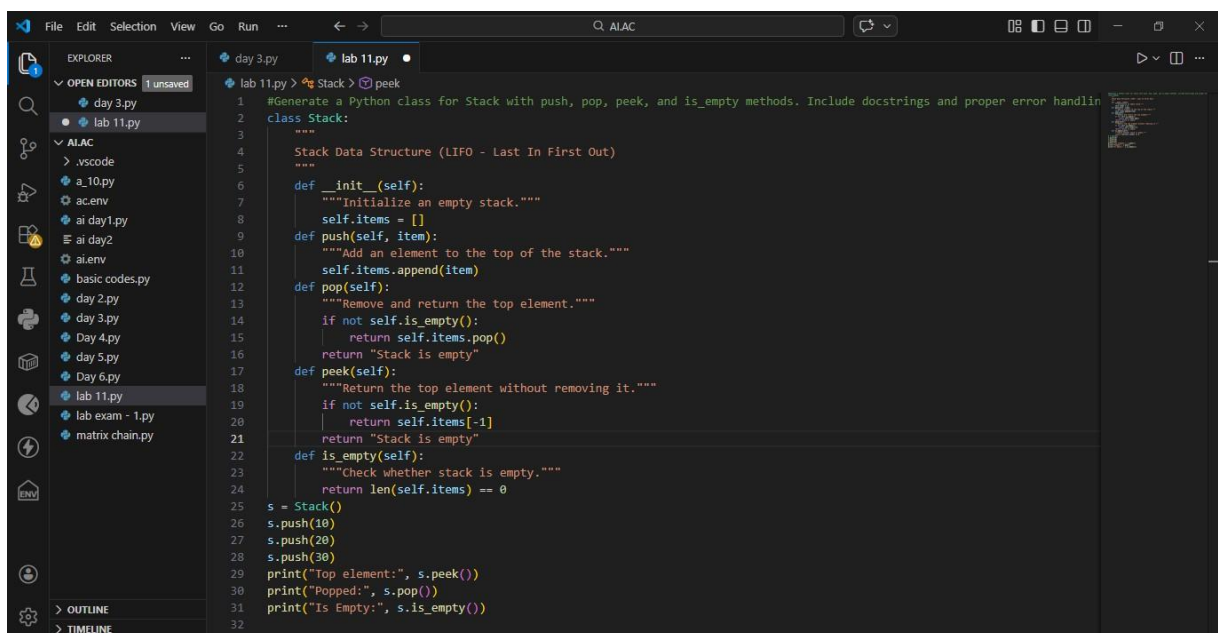
2403A51L34

B-52

### Task 1: Stack Implementation

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

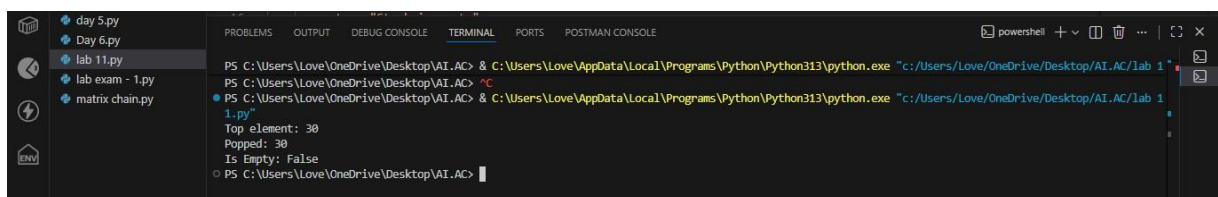
**Prompt:** Generate a Python class for Stack with push, pop, peek, and is\_empty methods. Include docstrings and proper error handling.



The screenshot shows a VS Code editor with a file explorer on the left and a code editor in the center. The file explorer shows a project named 'AIAC' with several files, including 'lab 11.py'. The code editor displays the following Python code for a Stack class:

```
1 #Generate a Python class for Stack with push, pop, peek, and is_empty methods. Include docstrings and proper error handling
2 class Stack:
3     """
4     Stack Data Structure (LIFO - Last In First Out)
5     """
6     def __init__(self):
7         """Initialize an empty stack."""
8         self.items = []
9     def push(self, item):
10        """Add an element to the top of the stack."""
11        self.items.append(item)
12    def pop(self):
13        """Remove and return the top element."""
14        if not self.is_empty():
15            return self.items.pop()
16        return "Stack is empty"
17    def peek(self):
18        """Return the top element without removing it."""
19        if not self.is_empty():
20            return self.items[-1]
21        return "Stack is empty"
22    def is_empty(self):
23        """Check whether stack is empty."""
24        return len(self.items) == 0
25
26 s = Stack()
27 s.push(10)
28 s.push(20)
29 s.push(30)
30 print("Top element:", s.peek())
31 print("Popped:", s.pop())
32 print("Is Empty:", s.is_empty())
```

### OUTPUT:



The screenshot shows a terminal window with the following output:

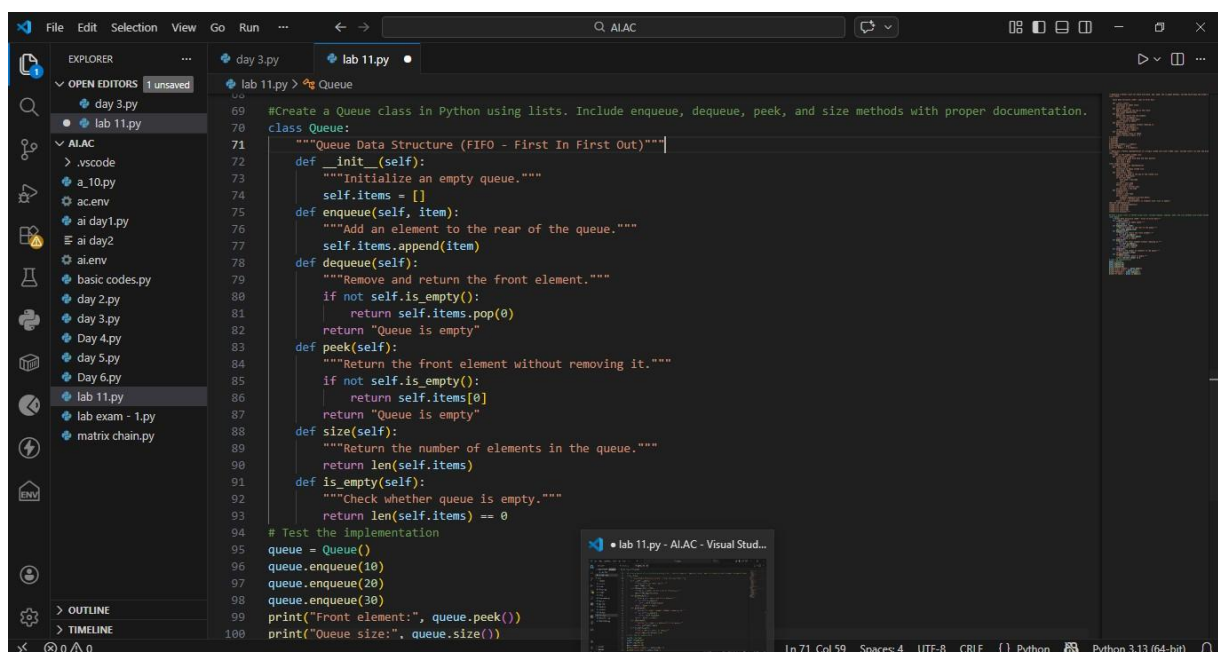
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 11.py"
Top element: 30
Popped: 30
Is Empty: False
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

**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

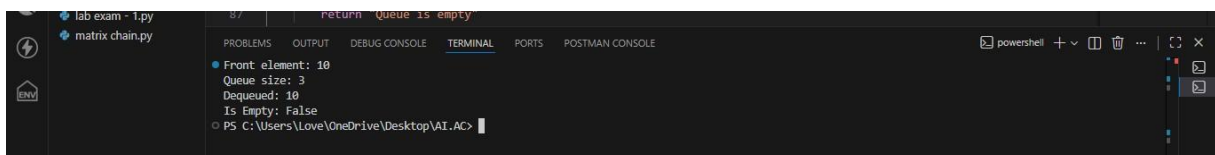
**Task:** Use AI to implement a Queue using Python lists.

**Prompt:** Create a Queue class in Python using lists. Include enqueue, dequeue, peek, and size methods with proper documentation.



```
69 #Create a Queue class in Python using lists. Include enqueue, dequeue, peek, and size methods with proper documentation.
70 class Queue:
71     """Queue Data Structure (FIFO - First In First Out)"""
72     def __init__(self):
73         """Initialize an empty queue."""
74         self.items = []
75     def enqueue(self, item):
76         """Add an element to the rear of the queue."""
77         self.items.append(item)
78     def dequeue(self):
79         """Remove and return the front element."""
80         if not self.is_empty():
81             return self.items.pop(0)
82         return "Queue is empty"
83     def peek(self):
84         """Return the front element without removing it."""
85         if not self.is_empty():
86             return self.items[0]
87         return "Queue is empty"
88     def size(self):
89         """Return the number of elements in the queue."""
90         return len(self.items)
91     def is_empty(self):
92         """Check whether queue is empty."""
93         return len(self.items) == 0
94
95 # Test the implementation
96 queue = Queue()
97 queue.enqueue(10)
98 queue.enqueue(20)
99 queue.enqueue(30)
100 print("Front element:", queue.peek())
101 print("Queue size:", queue.size())
```

## OUTPUT:



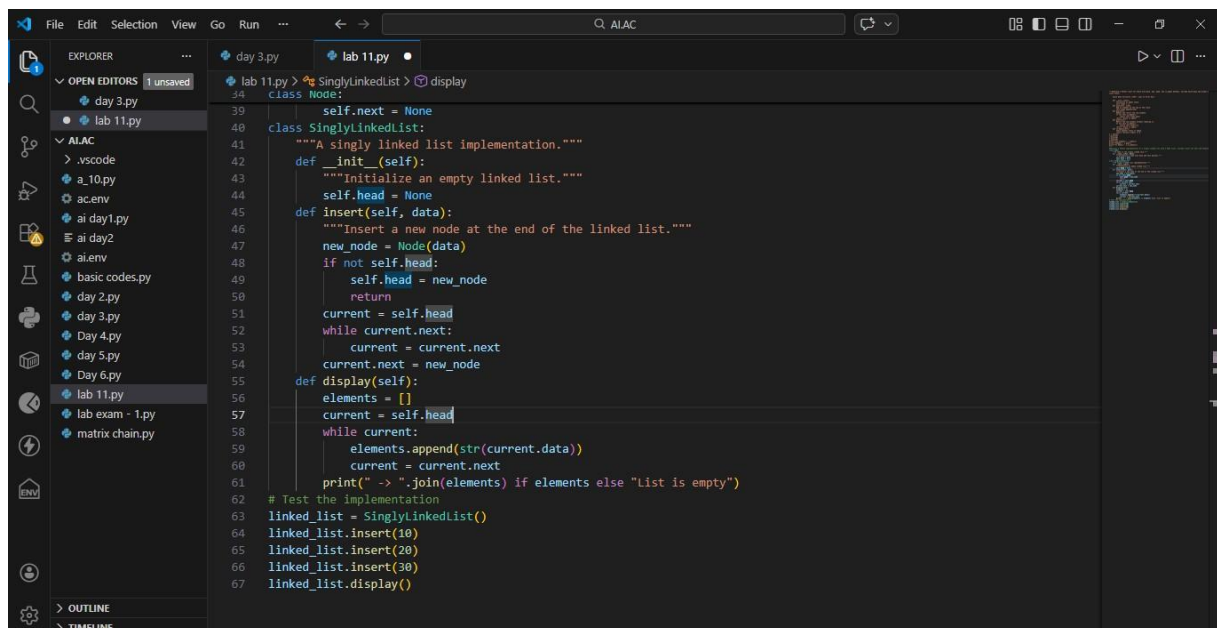
```
Front element: 10
Queue size: 3
Is Empty: False
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

**Explanation:** A Queue is a linear data structure that follows the FIFO (First In First Out) principle. This means the first element inserted is the first one removed.

## Task Description #3: Linked List

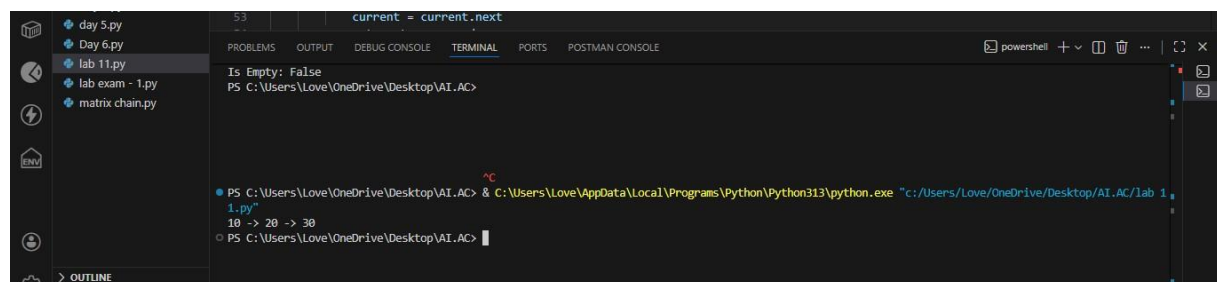
**Task:** Use AI to generate a Singly Linked List with insert and display methods.

**Prompt :** Generate a Python implementation of a Singly Linked List with a Node class. Include insert (at end) and display methods with docstrings.



```
File Edit Selection View Go Run ...
lab 11.py > SinglyLinkedList > display
class Node:
    39     self.next = None
    40
class SinglyLinkedList:
    41     """A singly linked list implementation."""
    42     def __init__(self):
    43         """Initialize an empty linked list."""
    44         self.head = None
    45     def insert(self, data):
    46         """Insert a new node at the end of the linked list."""
    47         new_node = Node(data)
    48         if not self.head:
    49             self.head = new_node
    50             return
    51         current = self.head
    52         while current.next:
    53             current = current.next
    54         current.next = new_node
    55     def display(self):
    56         elements = []
    57         current = self.head
    58         while current:
    59             elements.append(str(current.data))
    60             current = current.next
    61         print("-> ".join(elements) if elements else "List is empty")
    62
# Test the implementation
    63 linked_list = SinglyLinkedList()
    64 linked_list.insert(10)
    65 linked_list.insert(20)
    66 linked_list.insert(30)
    67 linked_list.display()
```

**OUTPUT:**



```
53     current = current.next
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
Is Empty: False
PS C:\Users\Love\OneDrive\Desktop\AI.AC>

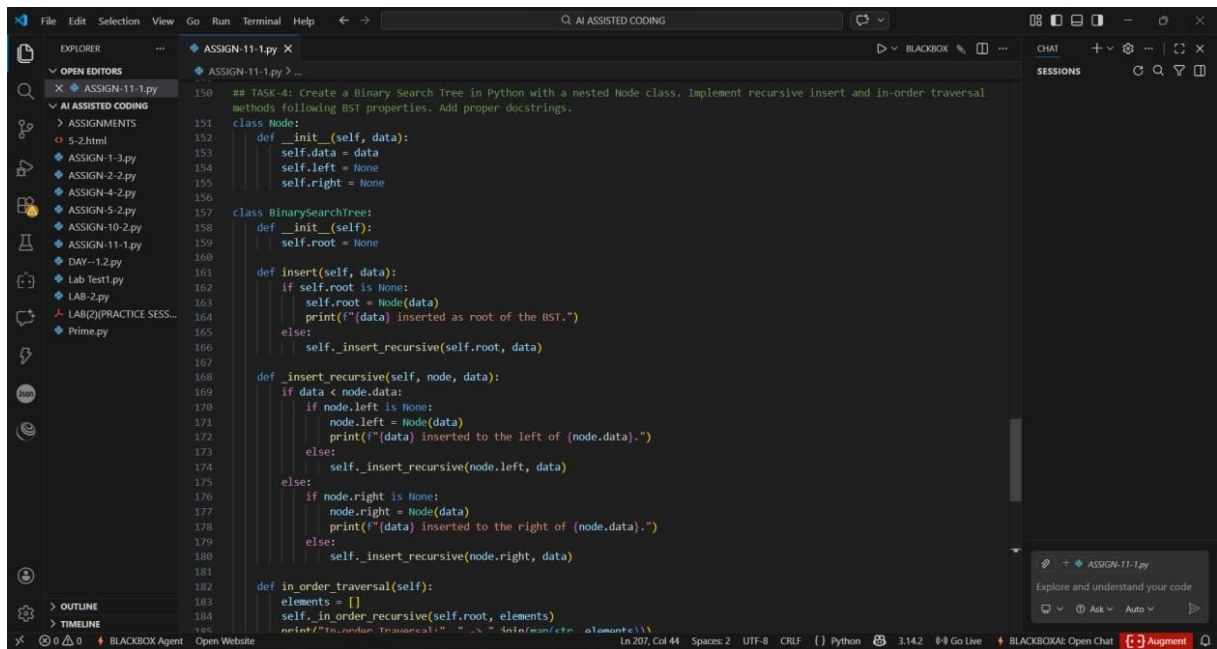
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 11.py"
10 -> 20 -> 30
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

**Explanation:** A Singly Linked List is a dynamic data structure where elements (nodes) are connected using pointers. Linked Lists are useful when frequent insertions and deletions are required, as they do not require shifting elements like arrays.

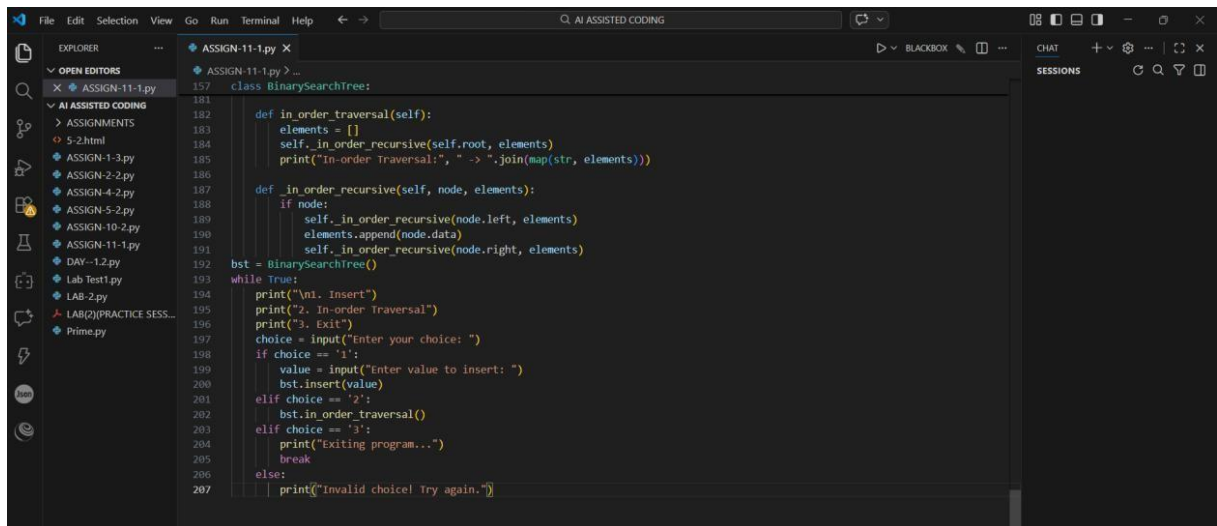
## Task Description #4: Binary Search Tree (BST)

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

**Prompt:** Create a Binary Search Tree in Python with recursive insert and inorder traversal methods. Include proper class structure and documentation.

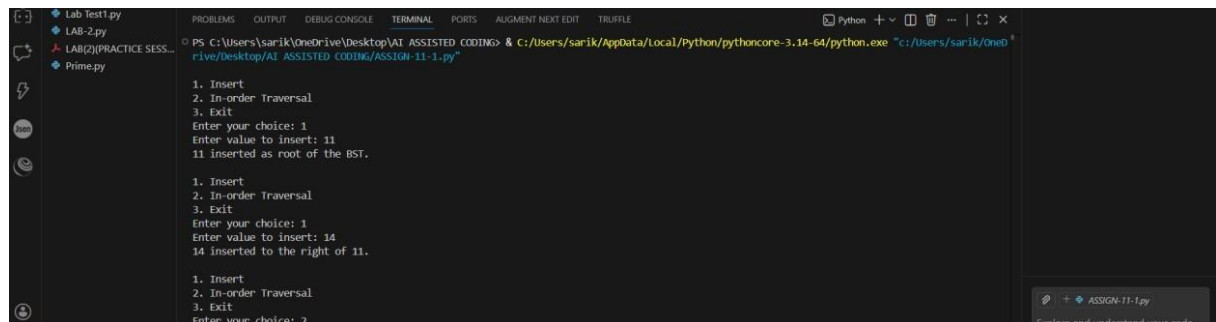


```
150 ## TASK-4: Create a Binary Search Tree in Python with a nested Node class. Implement recursive insert and in-order traversal
151 ## methods following BST properties. Add proper docstrings.
152 class Node:
153     def __init__(self, data):
154         self.data = data
155         self.left = None
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
188     def _in_order_recursive(self, node, elements):
189         if node:
190             self._in_order_recursive(node.left, elements)
191             elements.append(node.data)
192             self._in_order_recursive(node.right, elements)
193
194 bst = BinarySearchTree()
195 while True:
196     print("\n1. Insert")
197     print("2. In-order Traversal")
198     print("3. Exit")
199     choice = input("Enter your choice: ")
200     if choice == '1':
201         value = input("Enter value to insert: ")
202         bst.insert(value)
203     elif choice == '2':
204         bst.in_order_traversal()
205     elif choice == '3':
206         print("Exiting program...")
207         break
208     else:
209         print("Invalid choice! Try again.")
```



```
182 class BinarySearchTree:
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
188     def _in_order_recursive(self, node, elements):
189         if node:
190             self._in_order_recursive(node.left, elements)
191             elements.append(node.data)
192             self._in_order_recursive(node.right, elements)
193
194 bst = BinarySearchTree()
195 while True:
196     print("\n1. Insert")
197     print("2. In-order Traversal")
198     print("3. Exit")
199     choice = input("Enter your choice: ")
200     if choice == '1':
201         value = input("Enter value to insert: ")
202         bst.insert(value)
203     elif choice == '2':
204         bst.in_order_traversal()
205     elif choice == '3':
206         print("Exiting program...")
207         break
208     else:
209         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.

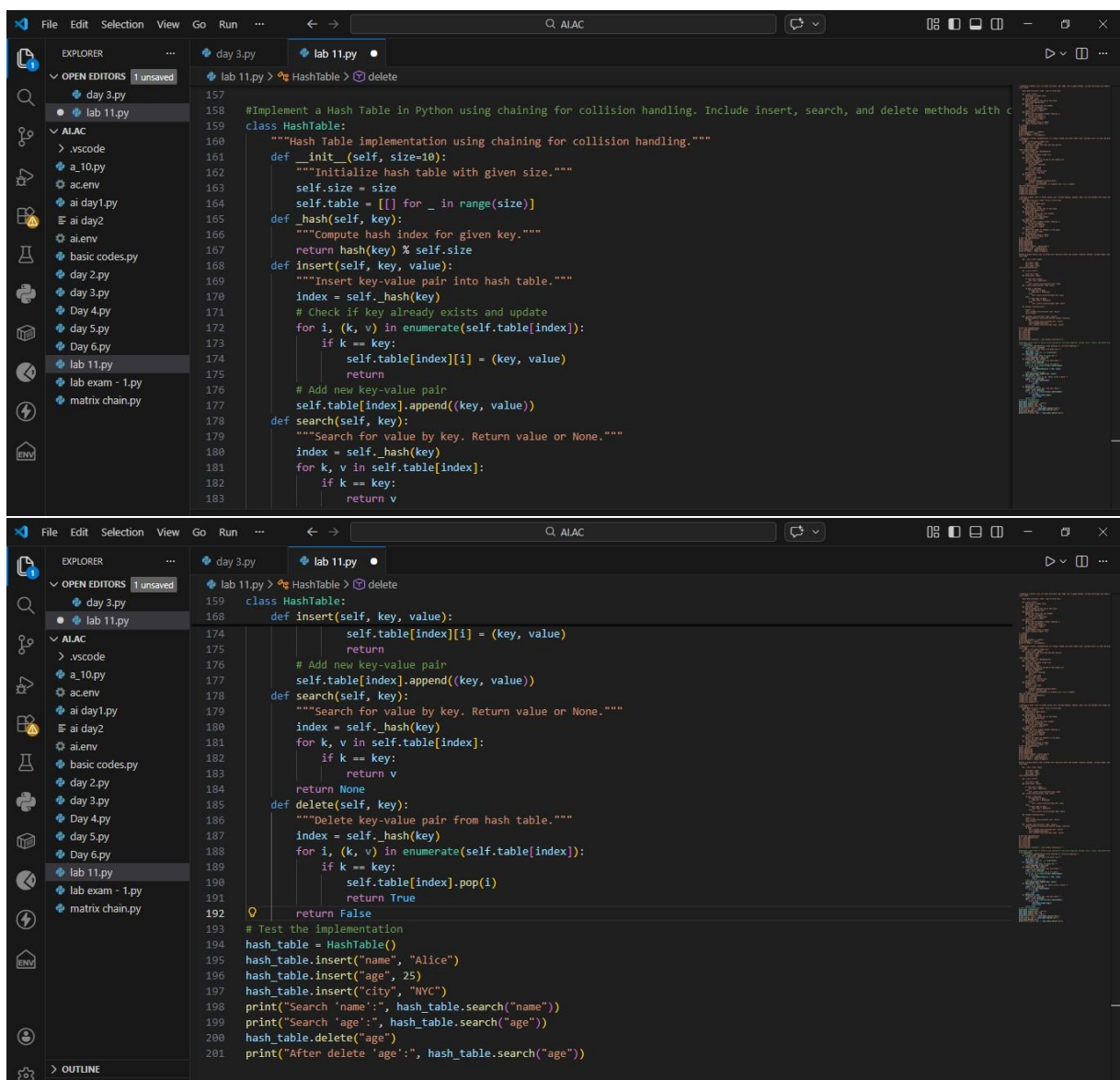
1. Insert
2. In-order Traversal
3. Exit
Enter your choice: 2
```

**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.

## Task Description #5: Hash Table

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

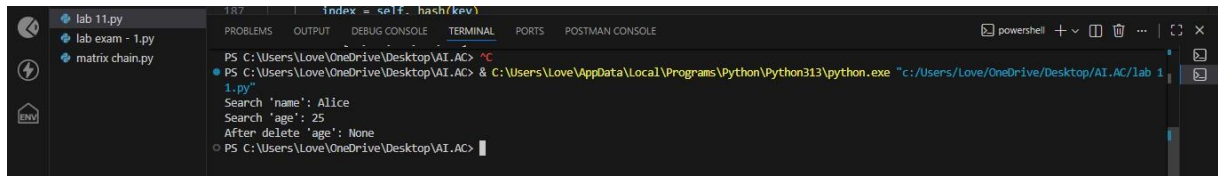
**Prompt:** Implement a Hash Table in Python using chaining for collision handling. Include insert, search, and delete methods with comments.



```
157
158 #Implement a Hash Table in Python using chaining for collision handling. Include insert, search, and delete methods with c
159 class HashTable:
160     """Hash Table implementation using chaining for collision handling."""
161     def __init__(self, size=10):
162         """Initialize hash table with given size."""
163         self.size = size
164         self.table = [[] for _ in range(size)]
165     def _hash(self, key):
166         """Compute hash index for given key."""
167         return hash(key) % self.size
168     def insert(self, key, value):
169         """Insert key-value pair into hash table."""
170         index = self._hash(key)
171         # Check if key already exists and update
172         for i, (k, v) in enumerate(self.table[index]):
173             if k == key:
174                 self.table[index][i] = (key, value)
175                 return
176         # Add new key-value pair
177         self.table[index].append((key, value))
178     def search(self, key):
179         """Search for value by key. Return value or None."""
180         index = self._hash(key)
181         for k, v in self.table[index]:
182             if k == key:
183                 return v
184
185
186
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
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
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
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
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
```



## OUTPUT:



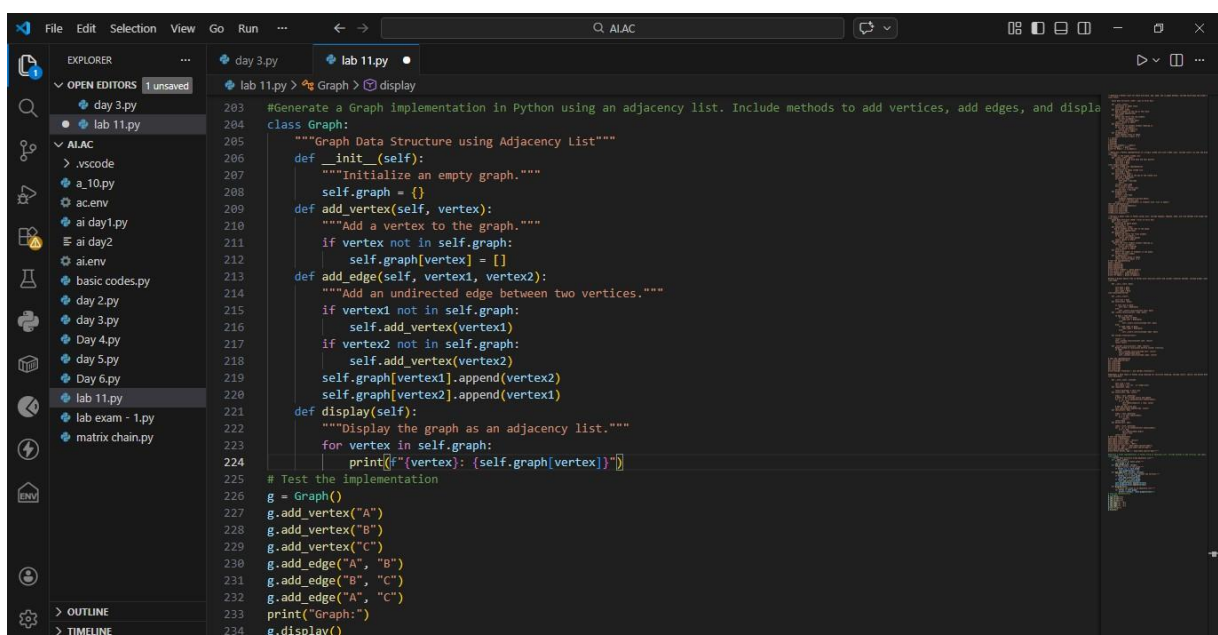
```
187         index = self.hash(key)
PS C:\Users\Love\OneDrive\Desktop\AI.AC> ^C
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1
1.py"
Search 'name': Alice
Search 'age': 25
After delete 'age': None
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

**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.

## Task Description #6: Graph Representation

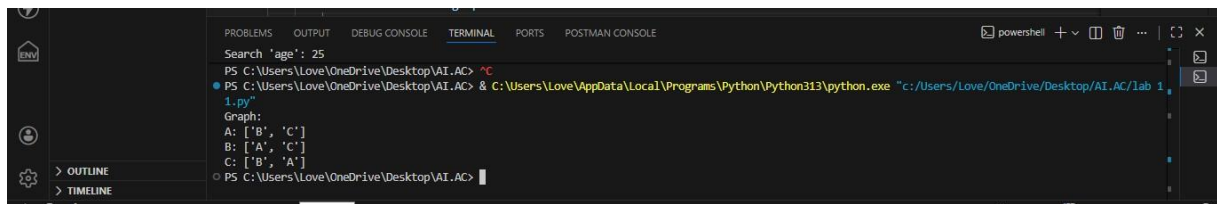
**Task:** Use AI to implement a graph using an adjacency list.

**Prompt:** Generate a Graph implementation in Python using an adjacency list. Include methods to add vertices, add edges, and display the graph.



```
File Edit Selection View Go Run ...
Q ALAC
EXPLORER
day 3.py
lab 11.py
day 3.py
a_10.py
ac.env
ai day1.py
ai day2
ai.env
basic codes.py
day 2.py
day 3.py
Day 4.py
day 5.py
Day 6.py
lab 11.py
lab exam - 1.py
matrix chain.py
OUTLINE
TIMELINE
lab 11.py > Graph > display
#Generate a Graph implementation in Python using an adjacency list. Include methods to add vertices, add edges, and display the graph.
class Graph:
    """Graph Data Structure using Adjacency List"""
    def __init__(self):
        """Initialize an empty graph."""
        self.graph = {}
    def add_vertex(self, vertex):
        """Add a vertex to the graph."""
        if vertex not in self.graph:
            self.graph[vertex] = []
    def add_edge(self, vertex1, vertex2):
        """Add an undirected edge between two vertices."""
        if vertex1 not in self.graph:
            self.add_vertex(vertex1)
        if vertex2 not in self.graph:
            self.add_vertex(vertex2)
        self.graph[vertex1].append(vertex2)
        self.graph[vertex2].append(vertex1)
    def display(self):
        """Display the graph as an adjacency list."""
        for vertex in self.graph:
            print(f"{vertex}: {self.graph[vertex]}")
# Test the implementation
g = Graph()
g.add_vertex("A")
g.add_vertex("B")
g.add_vertex("C")
g.add_edge("A", "B")
g.add_edge("B", "C")
g.add_edge("A", "C")
print("Graph:")
g.display()
```

## Output:

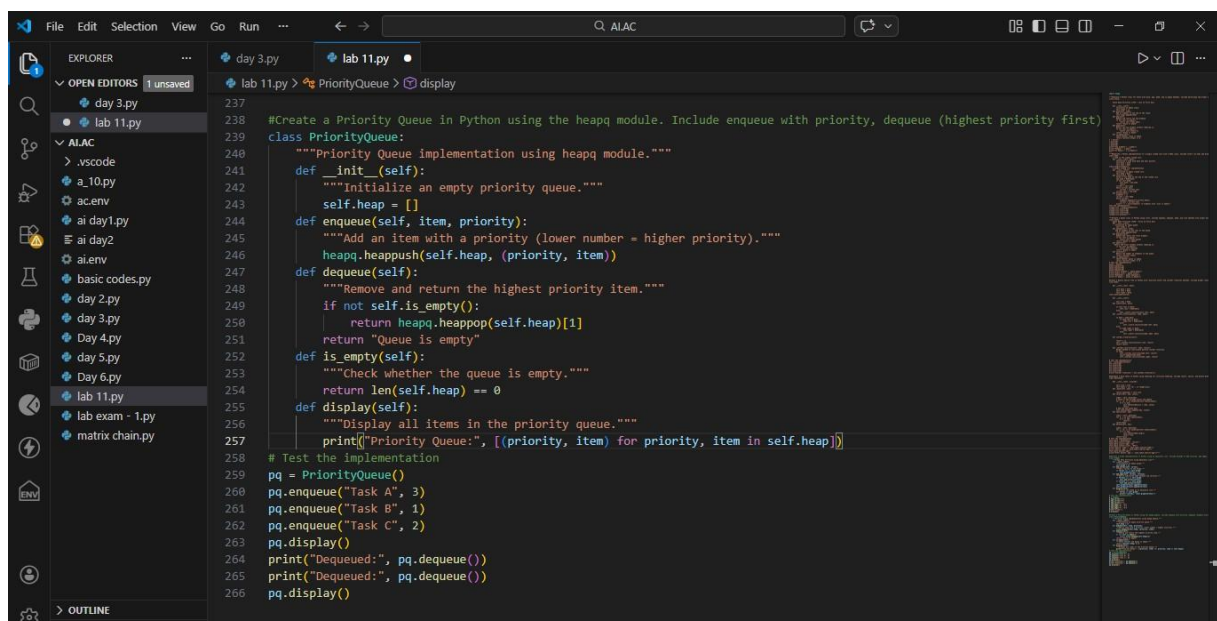


**Explanation:** A Graph is a non-linear data structure used to represent relationships between entities. It consists of vertices (nodes) and edges (connections).

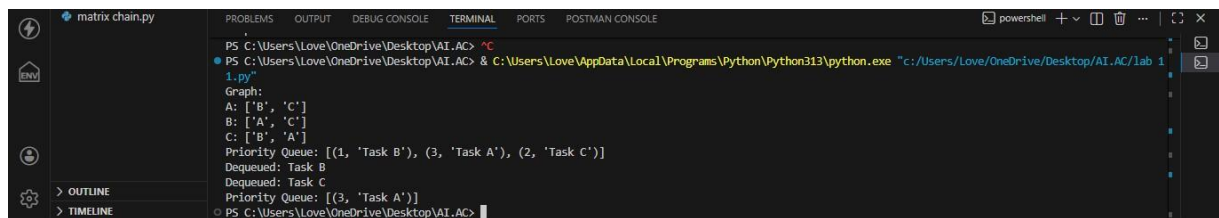
## Task Description #7: Priority Queue

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

**Prompt:** Create a Priority Queue in Python using the heapq module. Include enqueue with priority, dequeue (highest priority first), and display methods.



## Output:



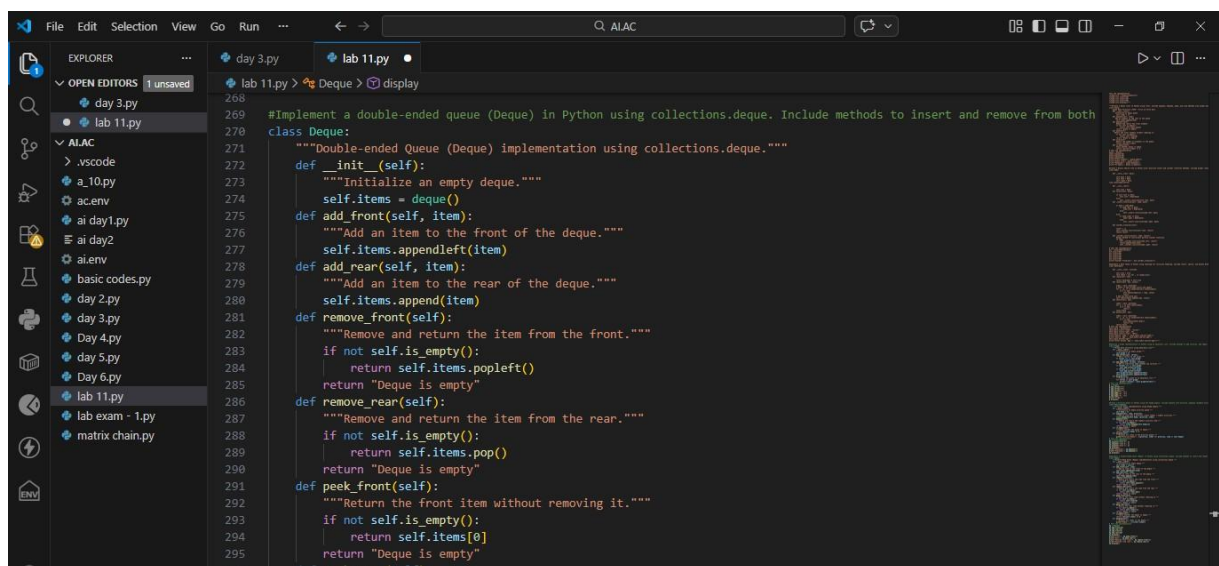
```
PS C:\Users\Love\OneDrive\Desktop\AI.AC> ^C
PS C:\Users\Love\OneDrive\Desktop\AI.AC> & C:\Users\Love\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Love/OneDrive/Desktop/AI.AC/lab 1
1.py"
Graph:
A: ['B', 'C']
B: ['A', 'C']
C: ['B', 'A']
Priority Queue: [(1, 'Task B'), (3, 'Task A'), (2, 'Task C')]
Dequeued: Task B
Dequeued: Task C
Priority Queue: [(3, 'Task A')]
PS C:\Users\Love\OneDrive\Desktop\AI.AC>
```

**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 from both ends with documentation.



```
268
269
270 #Implement a double-ended queue (Deque) in Python using collections.deque. Include methods to insert and remove from both
class Deque:
271     """Double-ended Queue (Deque) implementation using collections.deque."""
272     def __init__(self):
273         """Initialize an empty deque."""
274         self.items = deque()
275     def add_front(self, item):
276         """Add an item to the front of the deque."""
277         self.items.appendleft(item)
278     def add_rear(self, item):
279         """Add an item to the rear of the deque."""
280         self.items.append(item)
281     def remove_front(self):
282         """Remove and return the item from the front."""
283         if not self.is_empty():
284             return self.items.popleft()
285         return "Deque is empty"
286     def remove_rear(self):
287         """Remove and return the item from the rear."""
288         if not self.is_empty():
289             return self.items.pop()
290         return "Deque is empty"
291     def peek_front(self):
292         """Return the front item without removing it."""
293         if not self.is_empty():
294             return self.items[0]
295         return "Deque is empty"
296     def peek_rear(self):
297         """Return the rear item without removing it."""
```



```
270 class Deque:
286     def remove_rear(self):
290         return "Deque is empty"
291     def peek_front(self):
292         """Return the front item without removing it."""
293         if not self.is_empty():
294             return self.items[0]
295         return "Deque is empty"
296     def peek_rear(self):
297         """Return the rear item without removing it."""
298         if not self.is_empty():
299             return self.items[-1]
300         return "Deque is empty"
301     def is_empty(self):
302         """Check whether the deque is empty."""
303         return len(self.items) == 0
304     def display(self):
305         """Display all items in the deque."""
306         print("Deque:", list(self.items))
307
308 # Test the implementation
309 dq = Deque()
310 dq.add_front(10)
311 dq.add_rear(20)
312 dq.add_front(5)
313 dq.add_rear(30)
314 dq.display()
315 print("Front:", dq.peek_front())
316 print("Rear:", dq.peek_rear())
317 print("Removed from front:", dq.remove_front())
318 print("Removed from rear:", dq.remove_rear())
319 dq.display()
```

## Output:

```
Priority Queue: [(3, 'Task A')]
Deque: [5, 10, 20, 30]
Front: 5
Rear: 30
Removed from front: 5
Removed from rear: 30
Deque: [10, 20]
```

**Explanation:** A Deque (Double Ended Queue) allows insertion and deletion of elements from both the front and rear ends.