

ASSIGNMENT-11.1
AMGOTH VIKAS NAYAK
2403A51410

Lab 11-Data Structures with AI: Implementing Fundamental Structures.

Lab Objectives:

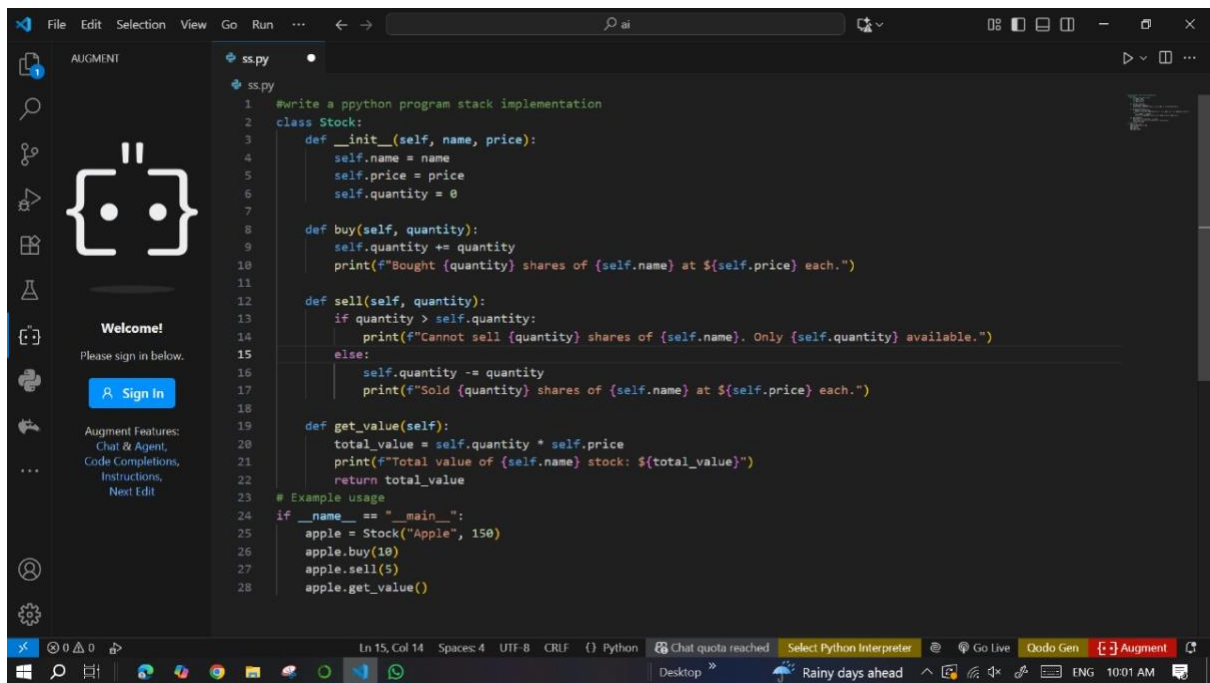
- **Use AI to assist in designing and implementing fundamental data structures in Python.**
- **Learn how to prompt AI for structure creation, optimization, and documentation.**
- **Improve understanding of Lists, Stacks, Queues, Linked Lists, Trees, Graphs, and Hash Tables.**
- **Enhance code quality with AI-generated comments and performance suggestions.**

Task Description #1 – Stack Implementation

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

Sample Input Code:

```
class Stack:  
  
    pass
```



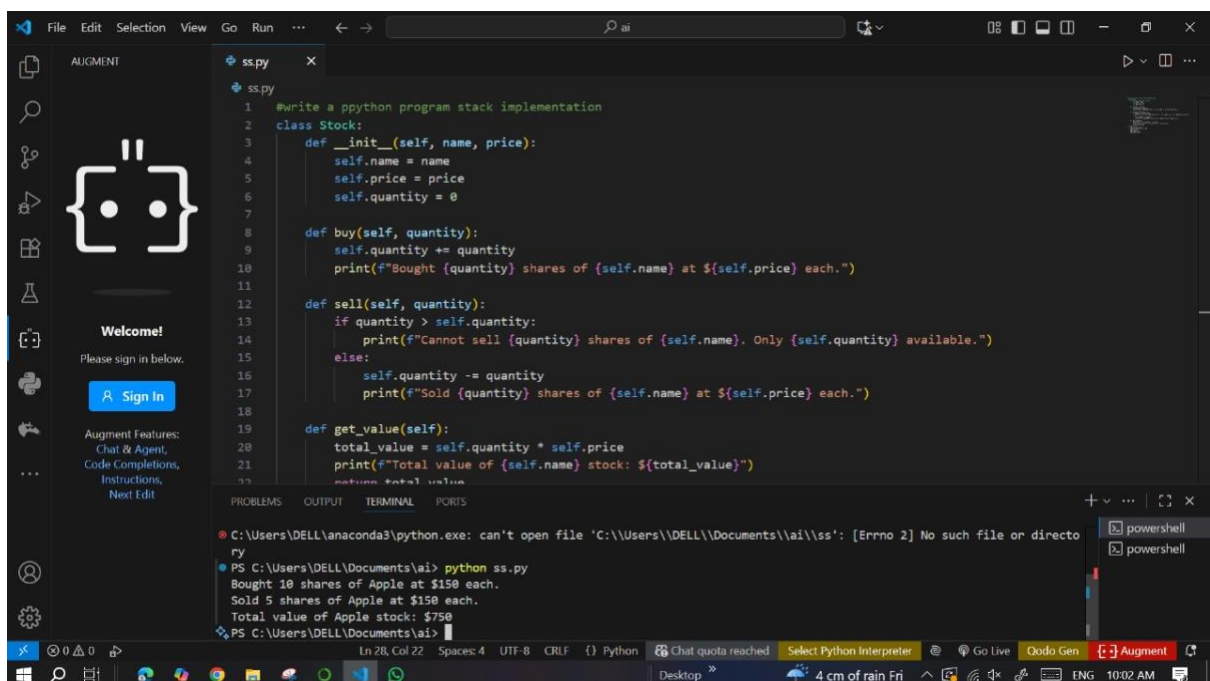
The screenshot shows the AUGMENT IDE interface. On the left is a sidebar with a 'Welcome!' message and a 'Sign In' button. The main editor displays a Python file named 'ss.py' with the following code:

```
1 #Write a ppython program stack implementation
2 class Stock:
3     def __init__(self, name, price):
4         self.name = name
5         self.price = price
6         self.quantity = 0
7
8     def buy(self, quantity):
9         self.quantity += quantity
10        print(f"Bought {quantity} shares of {self.name} at ${self.price} each.")
11
12    def sell(self, quantity):
13        if quantity > self.quantity:
14            print(f"Cannot sell {quantity} shares of {self.name}. Only {self.quantity} available.")
15        else:
16            self.quantity -= quantity
17            print(f"Sold {quantity} shares of {self.name} at ${self.price} each.")
18
19    def get_value(self):
20        total_value = self.quantity * self.price
21        print(f"Total value of {self.name} stock: ${total_value}")
22        return total_value
23
24 # Example usage
25 if __name__ == "__main__":
26     apple = Stock("Apple", 150)
27     apple.buy(10)
28     apple.sell(5)
29     apple.get_value()
```

The status bar at the bottom indicates 'Ln 15, Col 14', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Python'. It also shows a 'Chat quota reached' message and a 'Select Python Interpreter' button.

Expected Output:

- A functional stack implementation with all required methods and docstrings.



The screenshot shows the AUGMENT IDE interface with the same Python code as the previous image. The 'TERMINAL' tab is active, displaying the output of the program:

```
PS C:\Users\DELL\Documents\ai> python ss.py
Bought 10 shares of Apple at $150 each.
Sold 5 shares of Apple at $150 each.
Total value of Apple stock: $750
```

The status bar at the bottom indicates 'Ln 28, Col 22', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Python'. It also shows a 'Chat quota reached' message and a 'Select Python Interpreter' button.

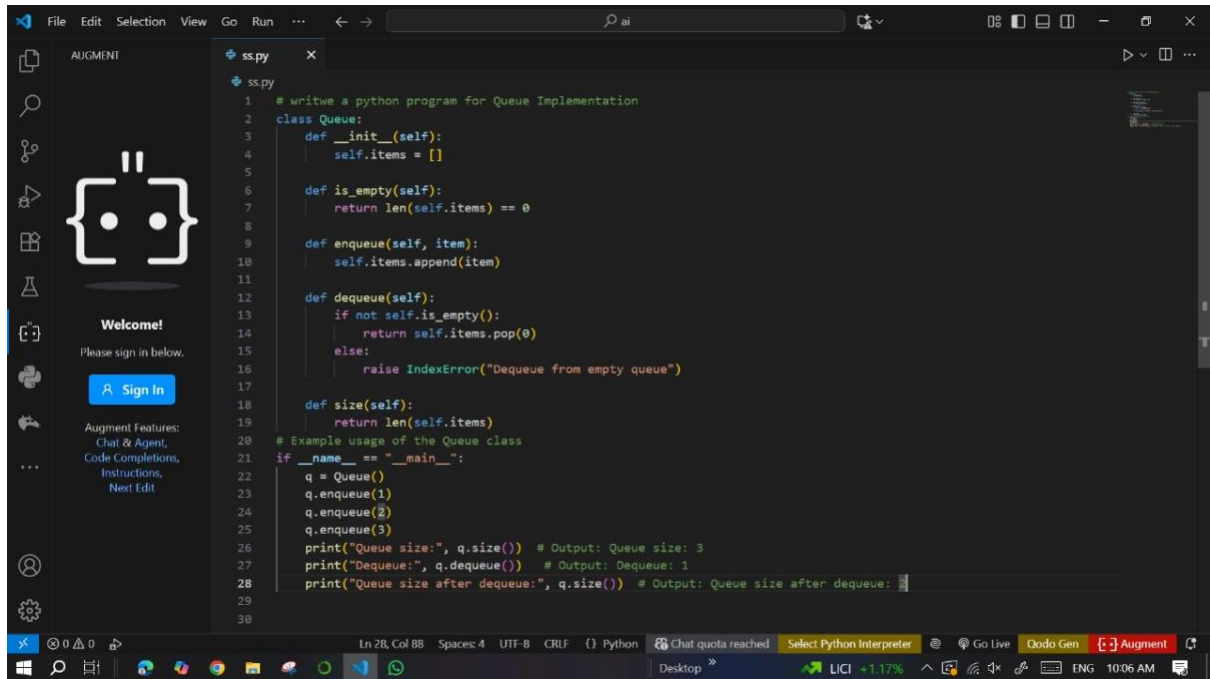
Task Description #2 – Queue Implementation

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

Sample Input Code:

class Queue:

pass



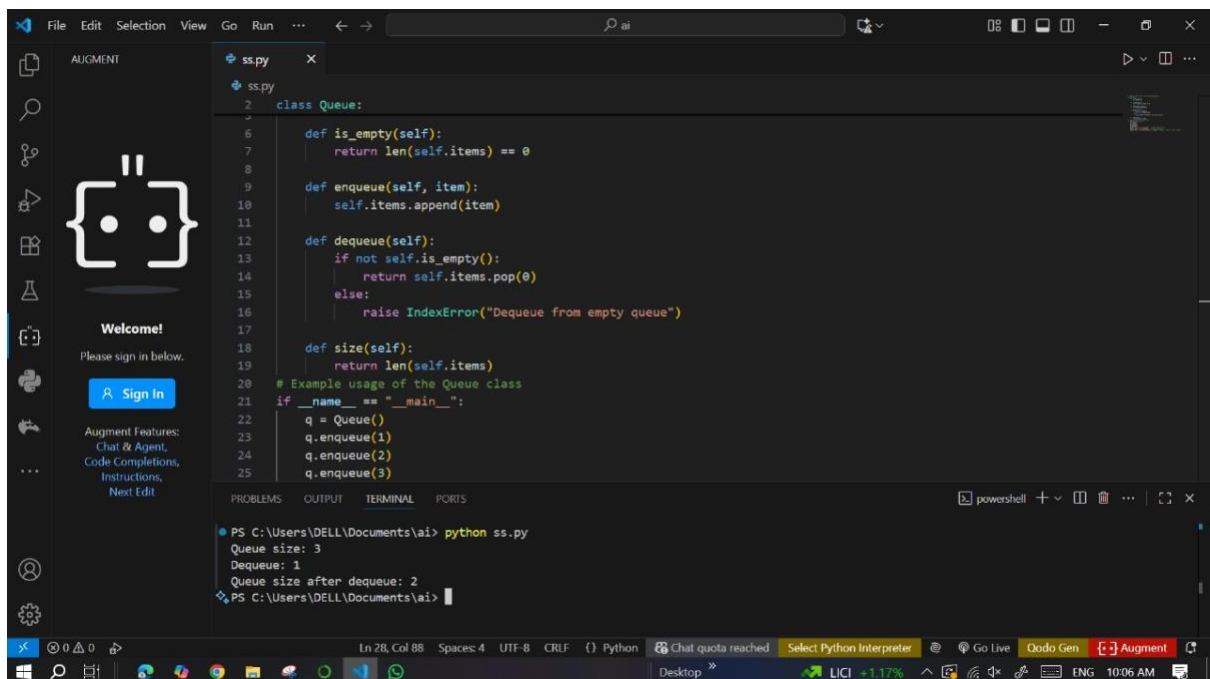
The screenshot shows a code editor with a sidebar on the left containing a 'Welcome!' message and a 'Sign In' button. The main editor area displays a Python file named 'ss.py' with the following code:

```
1 # write a python program for Queue Implementation
2 class Queue:
3     def __init__(self):
4         self.items = []
5
6     def is_empty(self):
7         return len(self.items) == 0
8
9     def enqueue(self, item):
10        self.items.append(item)
11
12    def dequeue(self):
13        if not self.is_empty():
14            return self.items.pop(0)
15        else:
16            raise IndexError("Dequeue from empty queue")
17
18    def size(self):
19        return len(self.items)
20
21 # Example usage of the Queue class
22 if __name__ == "__main__":
23     q = Queue()
24     q.enqueue(1)
25     q.enqueue(2)
26     q.enqueue(3)
27     print("Queue size:", q.size()) # Output: Queue size: 3
28     print("Dequeue:", q.dequeue()) # Output: Dequeue: 1
29     print("Queue size after dequeue:", q.size()) # Output: Queue size after dequeue: 2
30
```

The status bar at the bottom indicates 'Ln 28, Col 88', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Python'. It also shows a 'Chat quota reached' message and a 'Select Python Interpreter' button.

Expected Output:

- FIFO-based queue class with enqueue, dequeue, peek, and size methods.



The screenshot shows the same code editor as before, but with the code executed. The output is displayed in the terminal window at the bottom:

```
PS C:\Users\DELL\Documents\ai> python ss.py
Queue size: 3
Dequeue: 1
Queue size after dequeue: 2
PS C:\Users\DELL\Documents\ai>
```

The status bar at the bottom indicates 'Ln 28, Col 88', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Python'. It also shows a 'Chat quota reached' message and a 'Select Python Interpreter' button.

Task Description #3 – Linked List

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

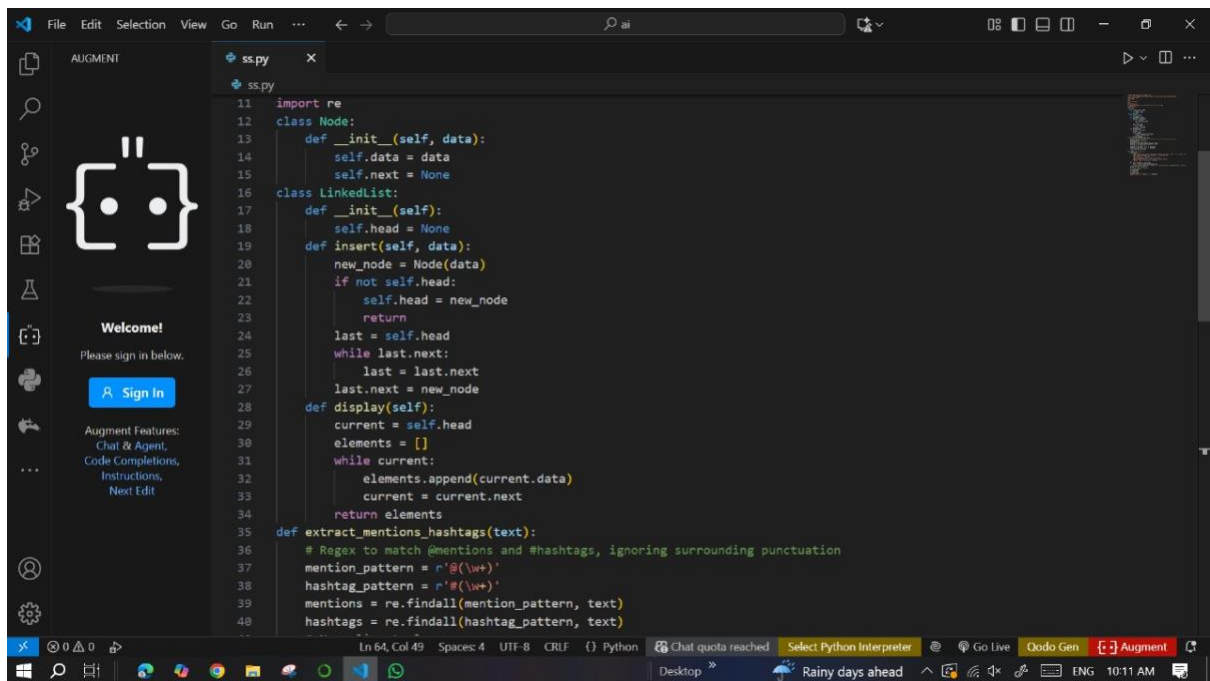
Sample Input Code:

class Node:

pass

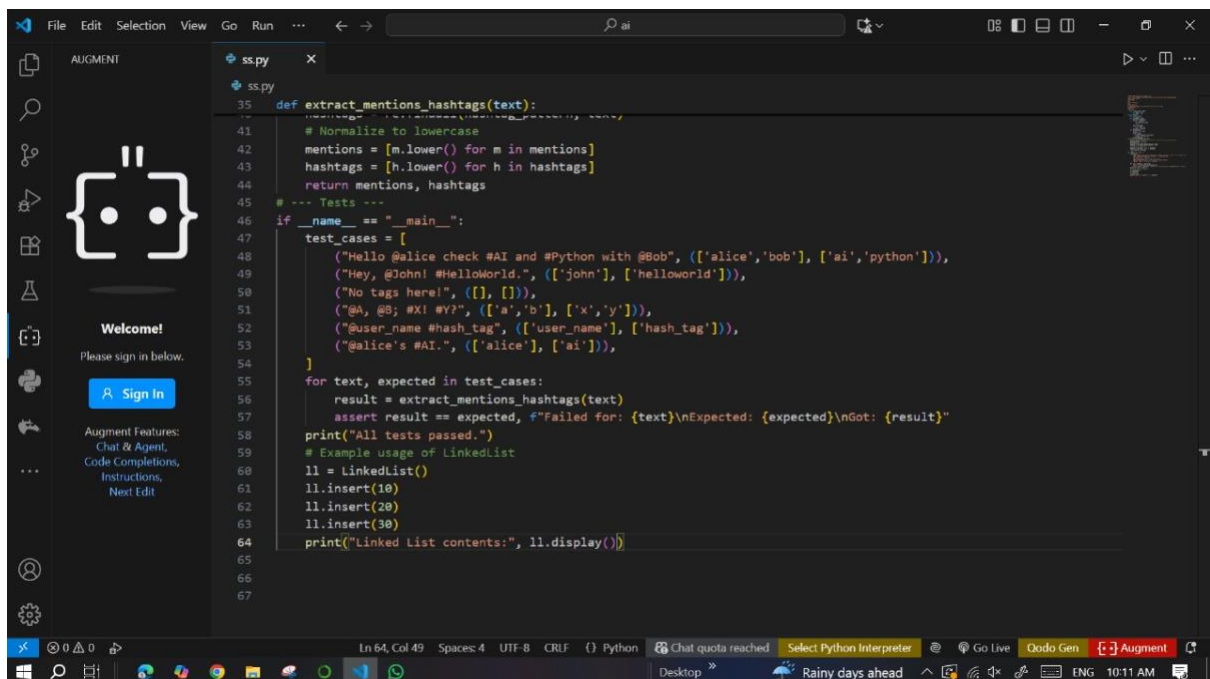
class LinkedList:

pass



The screenshot shows a code editor with a sidebar on the left containing a 'Welcome!' message and a 'Sign In' button. The main editor area displays a Python file named 'ss.py' with the following code:

```
11 import re
12 class Node:
13     def __init__(self, data):
14         self.data = data
15         self.next = None
16 class LinkedList:
17     def __init__(self):
18         self.head = None
19     def insert(self, data):
20         new_node = Node(data)
21         if not self.head:
22             self.head = new_node
23             return
24         last = self.head
25         while last.next:
26             last = last.next
27         last.next = new_node
28     def display(self):
29         current = self.head
30         elements = []
31         while current:
32             elements.append(current.data)
33             current = current.next
34         return elements
35 def extract_mentions_hashtags(text):
36     # Regex to match @mentions and #hashtags, ignoring surrounding punctuation
37     mention_pattern = r'@(\w+)'
38     hashtag_pattern = r'#(\w+)'
39     mentions = re.findall(mention_pattern, text)
40     hashtags = re.findall(hashtag_pattern, text)
```

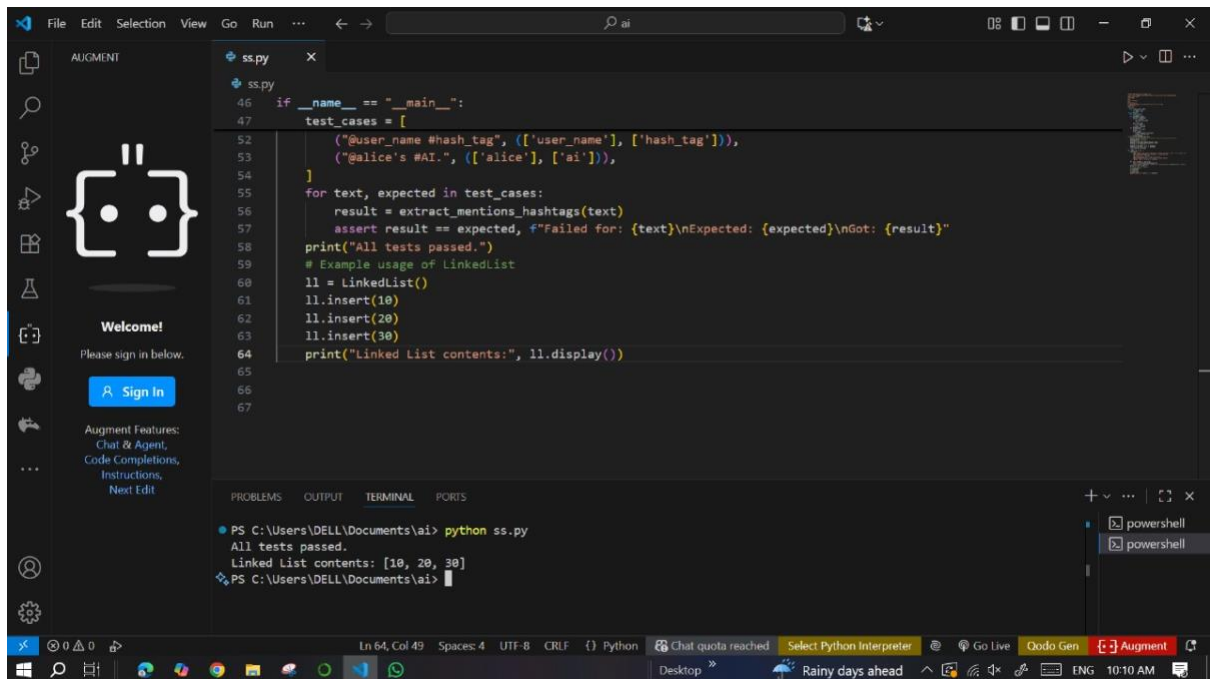


The screenshot shows the same code editor with the 'ss.py' file. The code has been extended to include tests and a main function. The code is as follows:

```
35 def extract_mentions_hashtags(text):
36     # Normalize to lowercase
37     mentions = [m.lower() for m in mentions]
38     hashtags = [h.lower() for h in hashtags]
39     return mentions, hashtags
40 # --- Tests ---
41 if __name__ == "__main__":
42     test_cases = [
43         ("Hello @alice check #AI and #Python with @Bob", ([ 'alice', 'bob' ], [ 'ai', 'python' ])),
44         ("Hey, @John! #HelloWorld.", ([ 'john' ], [ 'helloworld' ])),
45         ("No tags here!", ([], [])),
46         ("@A, @B; #X! #Y?", ([ 'a', 'b' ], [ 'x', 'y' ])),
47         ("@user_name #hash_tag", ([ 'user_name' ], [ 'hash_tag' ])),
48         ("@alice's #AI.", ([ 'alice' ], [ 'ai' ])),
49     ]
50     for text, expected in test_cases:
51         result = extract_mentions_hashtags(text)
52         assert result == expected, f"Failed for: {text}\nExpected: {expected}\nGot: {result}"
53     print("All tests passed.")
54     # Example usage of LinkedList
55     ll = LinkedList()
56     ll.insert(10)
57     ll.insert(20)
58     ll.insert(30)
59     print("Linked List contents:", ll.display())
```

Expected Output:

- A working linked list implementation with clear method documentation.



The screenshot shows a Visual Studio Code editor window with a Python file named `ss.py`. The code defines a `LinkedList` class and includes test cases. The terminal output shows that all tests passed and the linked list contains the values `[10, 20, 30]`.

```
ss.py
46 if __name__ == "__main__":
47     test_cases = [
52         ("@user_name #hash_tag", (['user_name'], ['hash_tag'])),
53         ("@alice's #AI.", (['alice'], ['ai'])),
54     ]
55     for text, expected in test_cases:
56         result = extract_mentions_hashtags(text)
57         assert result == expected, f"Failed for: {text}\nExpected: {expected}\nGot: {result}"
58     print("All tests passed.")
59     # Example usage of LinkedList
60     ll = LinkedList()
61     ll.insert(10)
62     ll.insert(20)
63     ll.insert(30)
64     print("Linked List contents:", ll.display())
65
66
67
```

Terminal Output:

```
PS C:\Users\DELL\Documents\ai> python ss.py
All tests passed.
Linked List contents: [10, 20, 30]
PS C:\Users\DELL\Documents\ai>
```

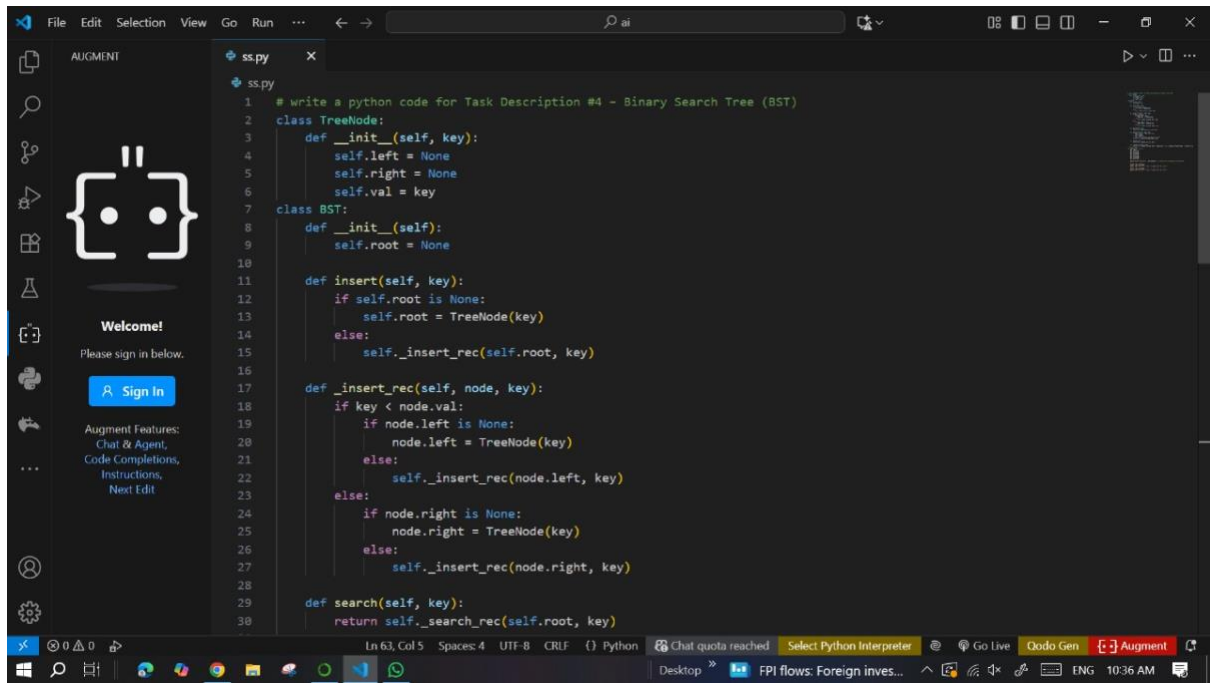
Task Description #4 – Binary Search Tree (BST)

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

Sample Input Code:

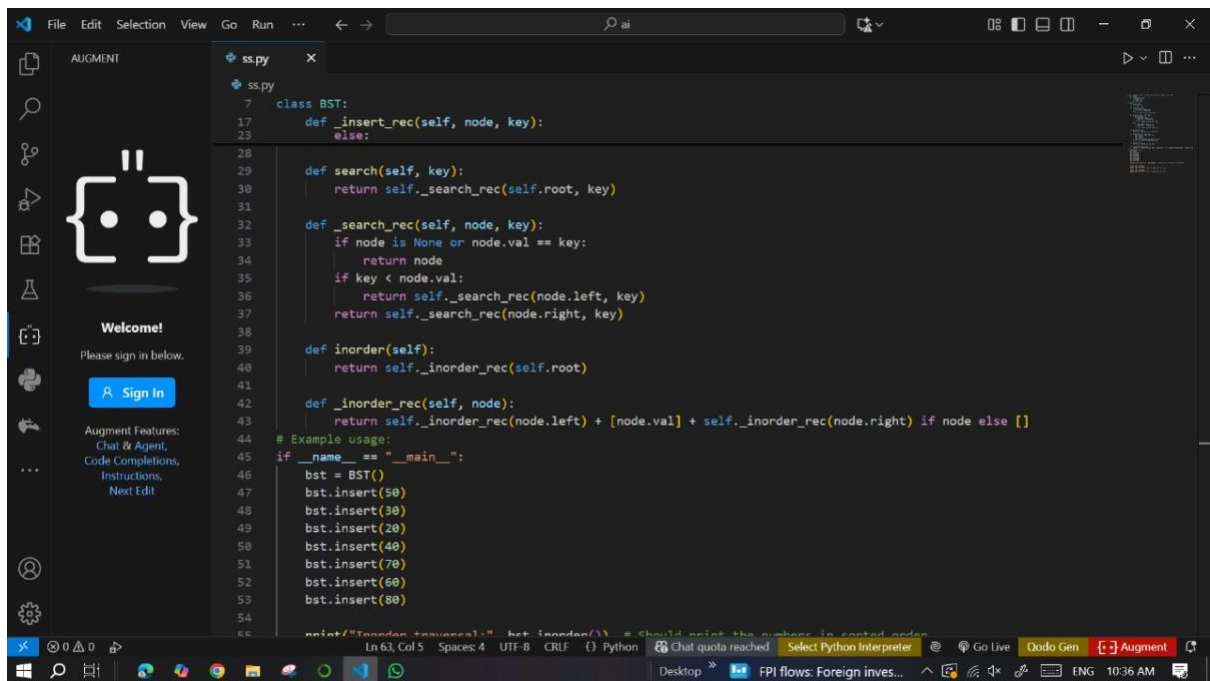
```
class BST:

    pass
```



The screenshot shows a VS Code editor window with a file named `ss.py`. The code defines a `TreeNode` class and a `BST` class. The `BST` class has methods for inserting a key and searching for a key. The code is as follows:

```
1 # write a python code for Task Description #4 - Binary Search Tree (BST)
2 class TreeNode:
3     def __init__(self, key):
4         self.left = None
5         self.right = None
6         self.val = key
7 class BST:
8     def __init__(self):
9         self.root = None
10
11     def insert(self, key):
12         if self.root is None:
13             self.root = TreeNode(key)
14         else:
15             self._insert_rec(self.root, key)
16
17     def _insert_rec(self, node, key):
18         if key < node.val:
19             if node.left is None:
20                 node.left = TreeNode(key)
21             else:
22                 self._insert_rec(node.left, key)
23         else:
24             if node.right is None:
25                 node.right = TreeNode(key)
26             else:
27                 self._insert_rec(node.right, key)
28
29     def search(self, key):
30         return self._search_rec(self.root, key)
```



The screenshot shows the same VS Code editor window with the `ss.py` file. The code is now complete, including methods for searching and inorder traversal, and an example usage section. The code is as follows:

```
7 class BST:
17     def _insert_rec(self, node, key):
23         else:
28
29     def search(self, key):
30         return self._search_rec(self.root, key)
31
32     def _search_rec(self, node, key):
33         if node is None or node.val == key:
34             return node
35         if key < node.val:
36             return self._search_rec(node.left, key)
37         return self._search_rec(node.right, key)
38
39     def inorder(self):
40         return self._inorder_rec(self.root)
41
42     def _inorder_rec(self, node):
43         return self._inorder_rec(node.left) + [node.val] + self._inorder_rec(node.right) if node else []
44
45 # Example usage:
46 if __name__ == "__main__":
47     bst = BST()
48     bst.insert(50)
49     bst.insert(30)
50     bst.insert(20)
51     bst.insert(40)
52     bst.insert(70)
53     bst.insert(60)
54     bst.insert(80)
55
56 print("Inorder traversal: ", bst.inorder()) # Should print the numbers in sorted order
```



```

class BST:
    def __init__(self):
        self.root = None

    def insert(self, val):
        if self.root is None:
            self.root = Node(val)
        else:
            self._insert(self.root, val)

    def _insert(self, node, val):
        if val < node.val:
            if node.left is None:
                node.left = Node(val)
            else:
                self._insert(node.left, val)
        else:
            if node.right is None:
                node.right = Node(val)
            else:
                self._insert(node.right, val)

    def inorder(self):
        return self._inorder_rec(self.root)

    def _inorder_rec(self, node):
        if node is None:
            return []
        return self._inorder_rec(node.left) + [node.val] + self._inorder_rec(node.right)

    def search(self, val):
        return self._search(self.root, val)

    def _search(self, node, val):
        if node is None:
            return False
        if node.val == val:
            return True
        if val < node.val:
            return self._search(node.left, val)
        return self._search(node.right, val)

# Example usage:
if __name__ == "__main__":
    bst = BST()
    bst.insert(50)
    bst.insert(30)
    bst.insert(20)
    bst.insert(40)
    bst.insert(70)
    bst.insert(60)
    bst.insert(80)

    print("Inorder traversal:", bst.inorder()) # Should print the numbers in sorted order

    # Search for a value
    result = bst.search(40)
    print("Search for 40:", "Found" if result else "Not Found")

    result = bst.search(90)
    print("Search for 90:", "Found" if result else "Not Found")

```

Expected Output:

- BST implementation with recursive insert and traversal methods.

```

class HashTable:
    def __init__(self):
        self.table = {}

    def remove(self, key):
        if key in self.table:
            del self.table[key]
        else:
            return False

    def __str__(self):
        return str(self.table)

import re
def extract_mentions_hashtags(text):
    # Regex to match @mentions and #hashtags, ignoring surrounding punctuation
    mention_pattern = r'@(\w+)'
    hashtag_pattern = r'#(\w+)'

    mentions = re.findall(mention_pattern, text)
    hashtags = re.findall(hashtag_pattern, text)

    # Normalize to lowercase
    mentions = [m.lower() for m in mentions]
    hashtags = [h.lower() for h in hashtags]

```

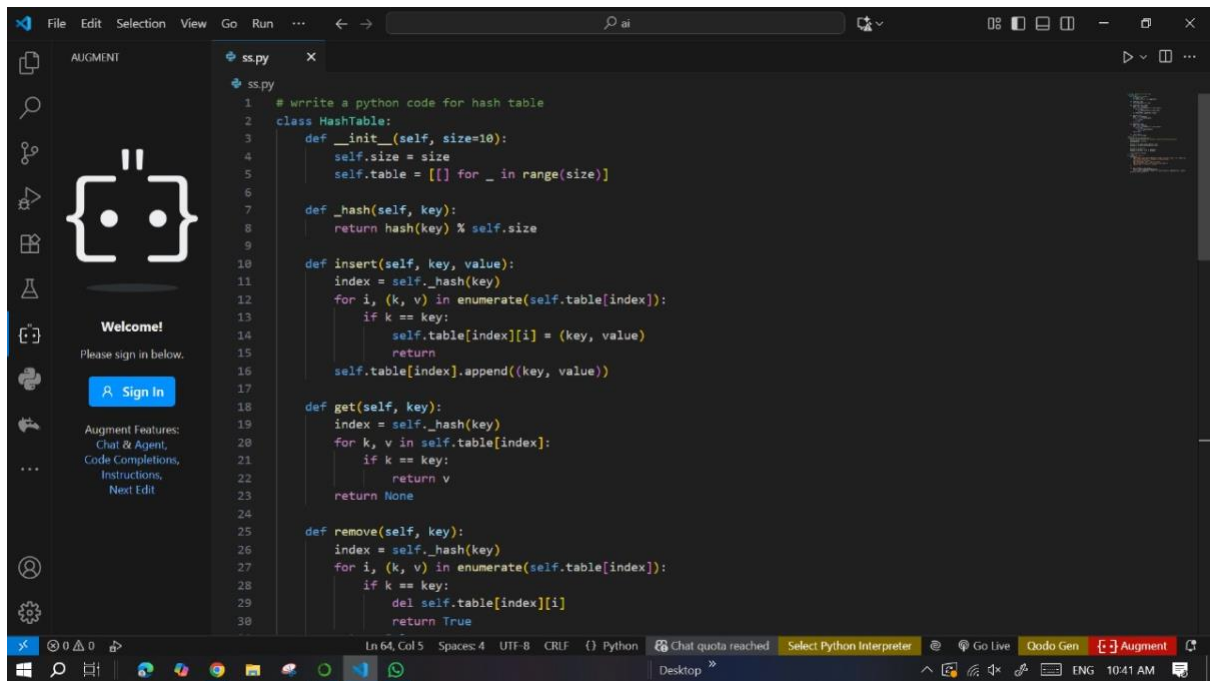
Task Description #5 – Hash Table

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

Sample Input Code:

class HashTable:

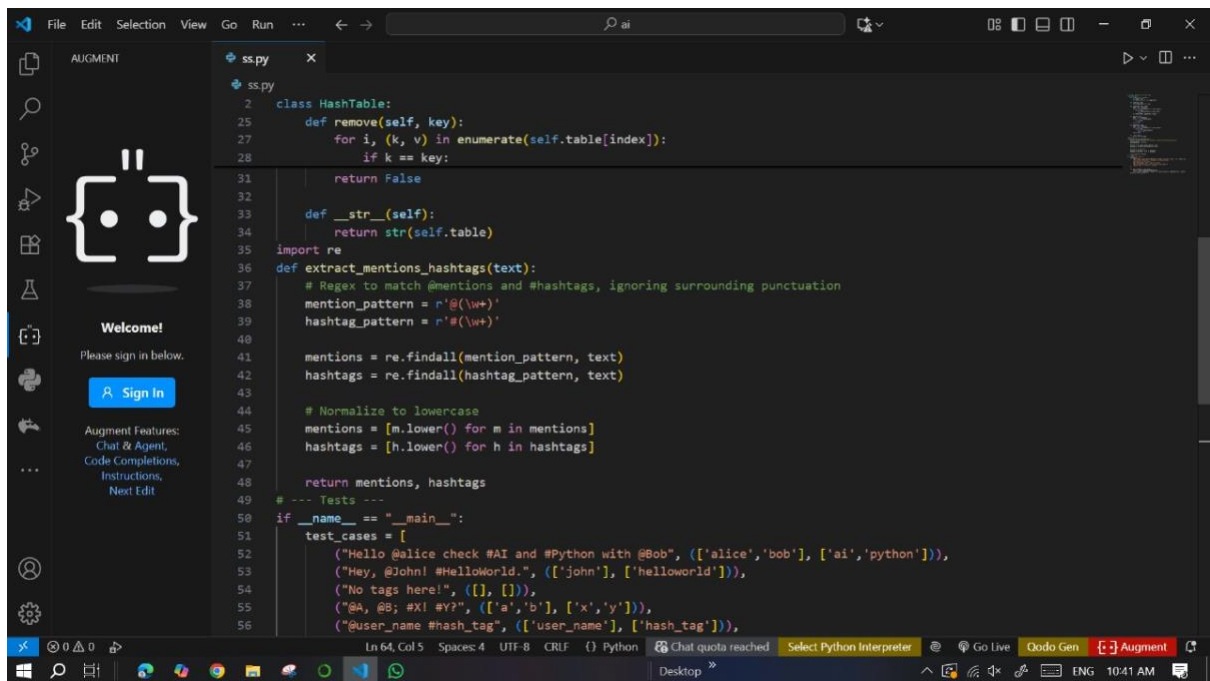
pass



The screenshot shows a VS Code editor window with a file named `ss.py`. The code implements a `HashTable` class with the following methods:

- `__init__(self, size=10)`: Initializes the hash table with a given size.
- `_hash(self, key)`: Returns the hash value for a given key.
- `insert(self, key, value)`: Inserts a key-value pair into the hash table.
- `get(self, key)`: Returns the value for a given key.
- `remove(self, key)`: Removes a key-value pair from the hash table.

The status bar at the bottom indicates the current position is Line 64, Column 5, with 4 spaces, UTF-8 encoding, and CR/LF line endings. The Python interpreter is set to the default.



The screenshot shows a VS Code editor window with a file named `ss.py`. The code implements a script for extracting mentions and hashtags from a text string. The script includes the following code:

```
class HashTable:
    def remove(self, key):
        for i, (k, v) in enumerate(self.table[index]):
            if k == key:
                return False

    def __str__(self):
        return str(self.table)

import re

def extract_mentions_hashtags(text):
    # Regex to match @mentions and #hashtags, ignoring surrounding punctuation
    mention_pattern = r'@(\w+)'
    hashtag_pattern = r'#(\w+)'

    mentions = re.findall(mention_pattern, text)
    hashtags = re.findall(hashtag_pattern, text)

    # Normalize to lowercase
    mentions = [m.lower() for m in mentions]
    hashtags = [h.lower() for h in hashtags]

    return mentions, hashtags

# --- Tests ---
if __name__ == "__main__":
    test_cases = [
        ("Hello @alice check #AI and #Python with @Bob", ([ 'alice', 'bob' ], [ 'ai', 'python' ])),
        ("Hey, @John! #HelloWorld.", ([ 'john' ], [ 'helloworld' ])),
        ("No tags here!", ([ ], [ ])),
        ("@A, @B; #X! #Y?", ([ 'a', 'b' ], [ 'x', 'y' ])),
        ("@user_name #hash_tag", ([ 'user_name' ], [ 'hash_tag' ])),
    ]
```

The status bar at the bottom indicates the current position is Line 64, Column 5, with 4 spaces, UTF-8 encoding, and CR/LF line endings. The Python interpreter is set to the default.


```

36 def extract_mentions_hashtags(text):
37     return mentions, hashtags
38
39 # --- Tests ---
40 if __name__ == "__main__":
41     test_cases = [
42         ("Hello @alice check #AI and #Python with @Bob", (['alice', 'bob'], ['ai', 'python'])),
43         ("Hey, @John! #HelloWorld.", (['john'], ['helloworld'])),
44         ("No tags here!", ([], [])),
45         ("@A, @B; #X! #Y?", (['a', 'b'], ['x', 'y'])),
46         ("@user_name #hash_tag", (['user_name'], ['hash_tag'])),
47         ("@alice's #AI.", (['alice'], ['ai'])),
48     ]
49
50     for text, expected in test_cases:
51         result = extract_mentions_hashtags(text)
52         assert result == expected, f"Failed for: {text}\nExpected: {expected}\nGot: {result}"
53     print("All tests passed.")

```

Expected Output:

- Collision handling using chaining, with well-commented methods.

```

42     hashtags = re.findall(hashtag_pattern, text)
43
44     # Normalize to lowercase
45     mentions = [m.lower() for m in mentions]
46     hashtags = [h.lower() for h in hashtags]
47
48     return mentions, hashtags
49
50 # --- Tests ---
51 if __name__ == "__main__":
52     test_cases = [
53         ("Hello @alice check #AI and #Python with @Bob", (['alice', 'bob'], ['ai', 'python'])),
54         ("Hey, @John! #HelloWorld.", (['john'], ['helloworld'])),
55         ("No tags here!", ([], [])),
56         ("@A, @B; #X! #Y?", (['a', 'b'], ['x', 'y'])),
57         ("@user_name #hash_tag", (['user_name'], ['hash_tag'])),
58         ("@alice's #AI.", (['alice'], ['ai'])),
59     ]
60
61     for text, expected in test_cases:
62         result = extract_mentions_hashtags(text)

```

PS C:\Users\DELL\Documents\ai> python ss.py
All tests passed.

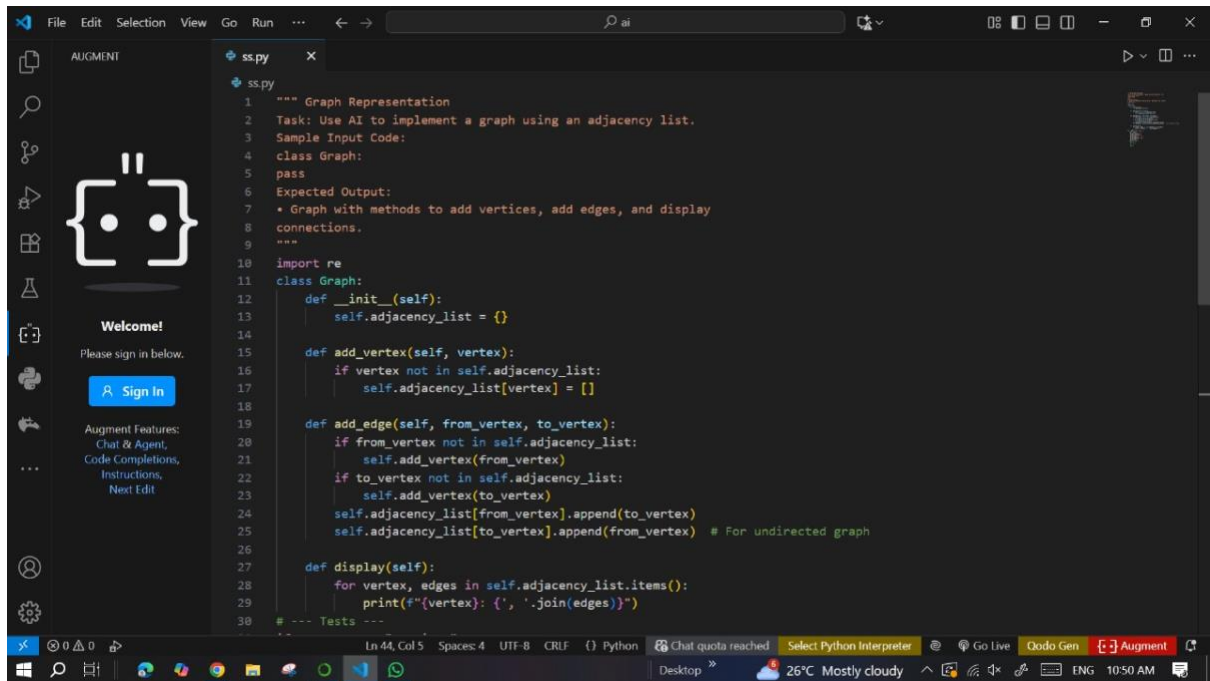
Task Description #6 – Graph Representation

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

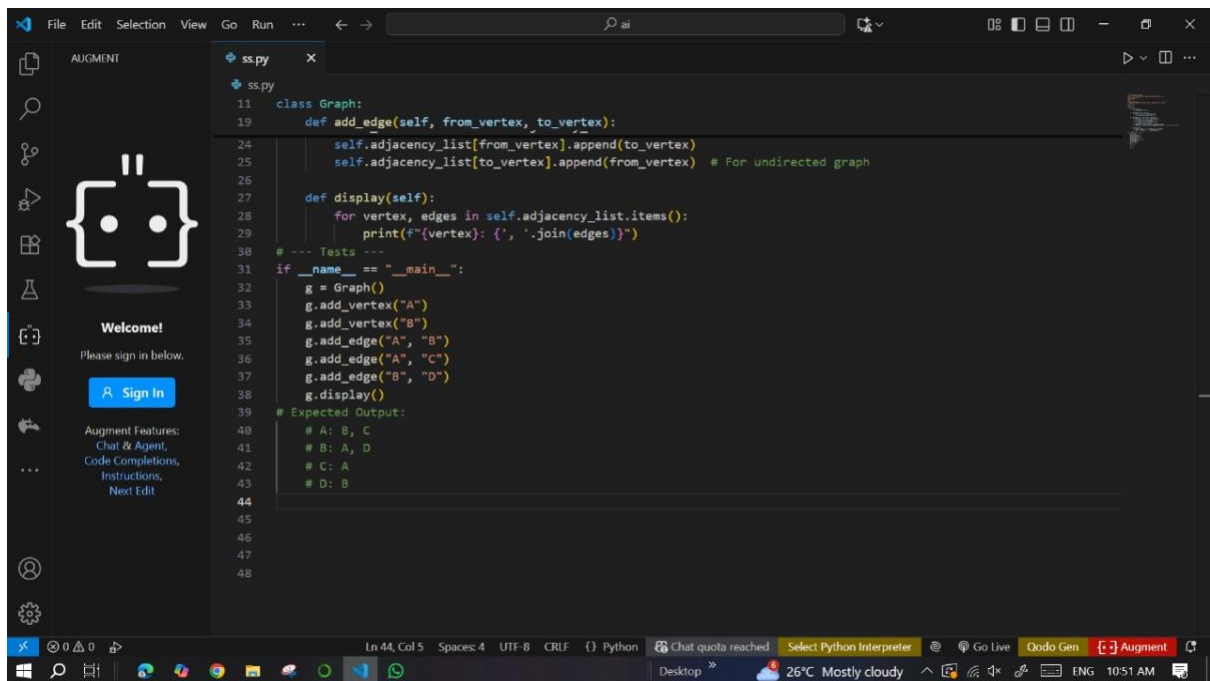
Sample Input Code:

class Graph:

pass



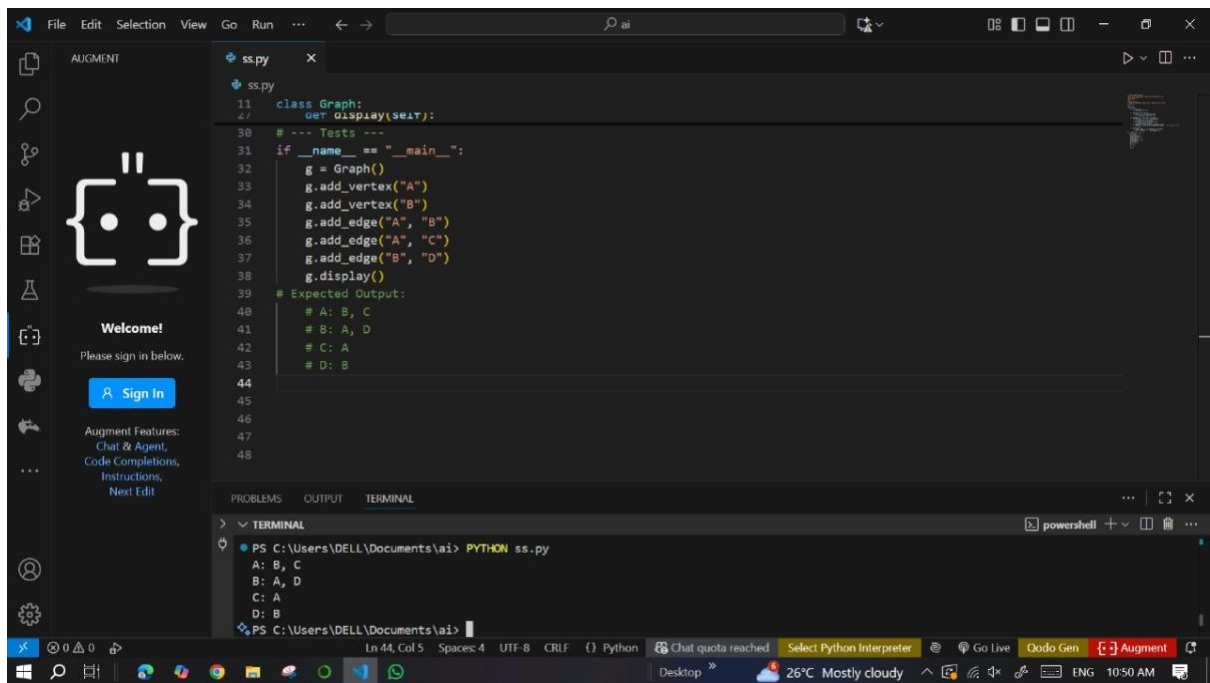
```
1 """ Graph Representation
2 Task: Use AI to implement a graph using an adjacency list.
3 Sample Input Code:
4 class Graph:
5     pass
6 Expected Output:
7 • Graph with methods to add vertices, add edges, and display
8   connections.
9 """
10 import re
11 class Graph:
12     def __init__(self):
13         self.adjacency_list = {}
14
15     def add_vertex(self, vertex):
16         if vertex not in self.adjacency_list:
17             self.adjacency_list[vertex] = []
18
19     def add_edge(self, from_vertex, to_vertex):
20         if from_vertex not in self.adjacency_list:
21             self.add_vertex(from_vertex)
22         if to_vertex not in self.adjacency_list:
23             self.add_vertex(to_vertex)
24         self.adjacency_list[from_vertex].append(to_vertex)
25         self.adjacency_list[to_vertex].append(from_vertex) # For undirected graph
26
27     def display(self):
28         for vertex, edges in self.adjacency_list.items():
29             print(f"{vertex}: {' '.join(edges)}")
30
31 # --- Tests ---
```



```
11 class Graph:
12     def add_edge(self, from_vertex, to_vertex):
13         self.adjacency_list[from_vertex].append(to_vertex)
14         self.adjacency_list[to_vertex].append(from_vertex) # For undirected graph
15
16     def display(self):
17         for vertex, edges in self.adjacency_list.items():
18             print(f"{vertex}: {' '.join(edges)}")
19
20 # --- Tests ---
21 if __name__ == "__main__":
22     g = Graph()
23     g.add_vertex("A")
24     g.add_vertex("B")
25     g.add_edge("A", "B")
26     g.add_edge("A", "C")
27     g.add_edge("B", "D")
28     g.display()
29
30 # Expected Output:
31 # A: B, C
32 # B: A, D
33 # C: A
34 # D: B
```

Expected Output:

- **Graph with methods to add vertices, add edges, and display connections.**



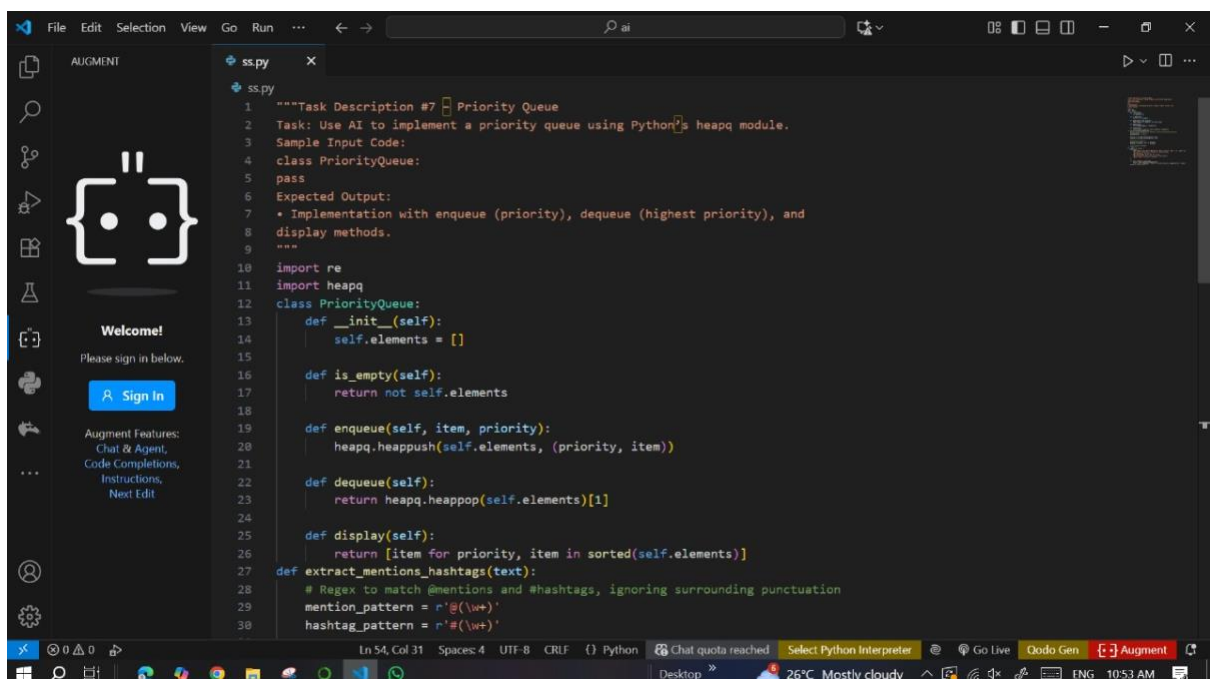
Task Description #7 – Priority Queue

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

Sample Input Code:

class PriorityQueue:

pass



```
File Edit Selection View Go Run ...  
ss.py  
27 def extract_mentions_hashtags(text):  
31     mentions = re.findall(mention_pattern, text)  
32     hashtags = re.findall(hashtag_pattern, text)  
34  
35     # Normalize to lowercase  
36     mentions = [m.lower() for m in mentions]  
37     hashtags = [h.lower() for h in hashtags]  
38  
39     return mentions, hashtags  
40  
41 # --- Tests ---  
42 if __name__ == "__main__":  
43     test_cases = [  
44         ("Hello @alice check #AI and #Python with @Bob", (['alice', 'bob'], ['ai', 'python'])),  
45         ("Hey, @John! #HelloWorld.", (['john'], ['helloworld'])),  
46         ("No tags here!", ([], [])),  
47         ("@A, @B; #X! #Y?", (['a', 'b'], ['x', 'y'])),  
48         ("user_name #hash_tag", (['user_name'], ['hash_tag'])),  
49         ("@alice's #AI.", (['alice'], ['ai'])),  
50     ]  
51  
52     for text, expected in test_cases:  
53         result = extract_mentions_hashtags(text)  
54         assert result == expected, f"Failed for: {text}\nExpected: {expected}\nGot: {result}"  
55     print("All tests passed.")  
56  
57  
58  
59  
Ln 54, Col 31 Spaces: 4 UTF-8 CRLF Python Chat quota reached Select Python Interpreter Go Live Qodo Gen AUGMENT  
Desktop 26°C Mostly cloudy ENG 10:53 AM
```

Expected Output:

- Implementation with enqueue (priority), dequeue (highest priority), and display methods.

```
File Edit Selection View Go Run ...  
ss.py  
41 if __name__ == "__main__":  
42     test_cases = [  
43         ("Hello @alice check #AI and #Python with @Bob", (['alice', 'bob'], ['ai', 'python'])),  
44         ("Hey, @John! #HelloWorld.", (['john'], ['helloworld'])),  
45         ("No tags here!", ([], [])),  
46         ("@A, @B; #X! #Y?", (['a', 'b'], ['x', 'y'])),  
47         ("user_name #hash_tag", (['user_name'], ['hash_tag'])),  
48         ("@alice's #AI.", (['alice'], ['ai'])),  
49     ]  
50  
51     for text, expected in test_cases:  
52         result = extract_mentions_hashtags(text)  
53         assert result == expected, f"Failed for: {text}\nExpected: {expected}\nGot: {result}"  
54     print("All tests passed.")  
55  
56  
57  
58  
59  
PROBLEMS OUTPUT TERMINAL  
TERMINAL  
PS C:\Users\DELL\Documents\ai> python ss.py  
All tests passed.  
PS C:\Users\DELL\Documents\ai>  
Ln 54, Col 31 Spaces: 4 UTF-8 CRLF Python Chat quota reached Select Python Interpreter Go Live Qodo Gen AUGMENT  
Desktop 26°C Mostly cloudy ENG 10:53 AM
```

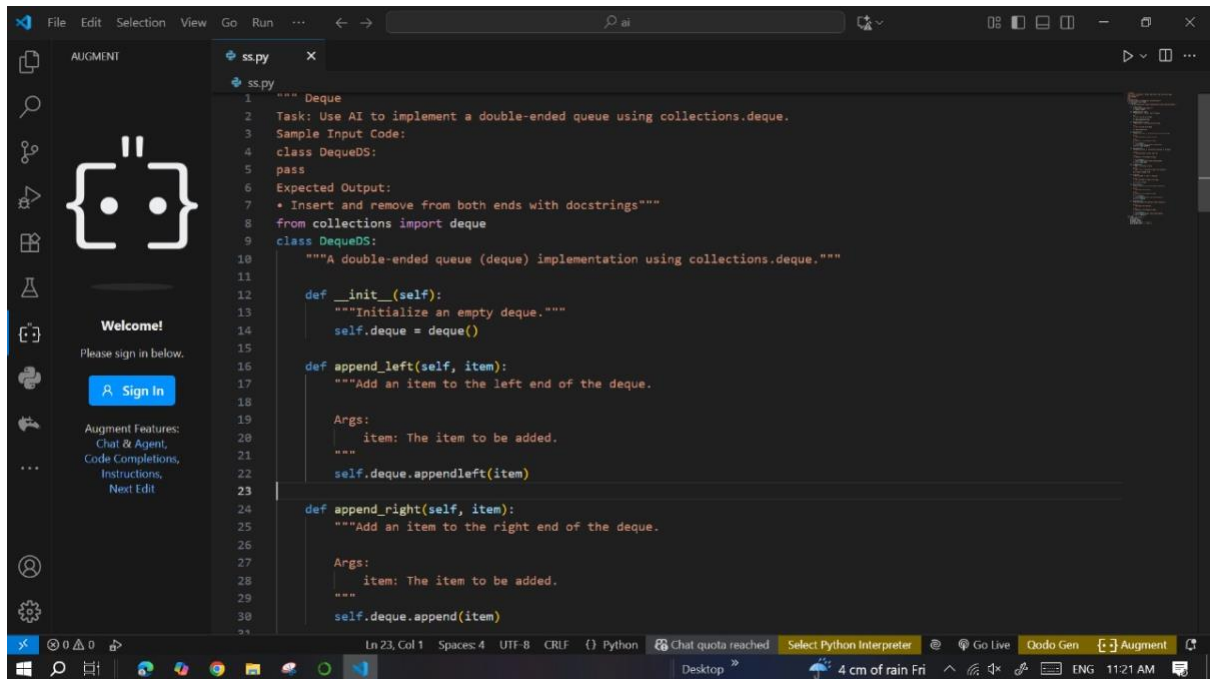
Task Description #8 – Deque

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

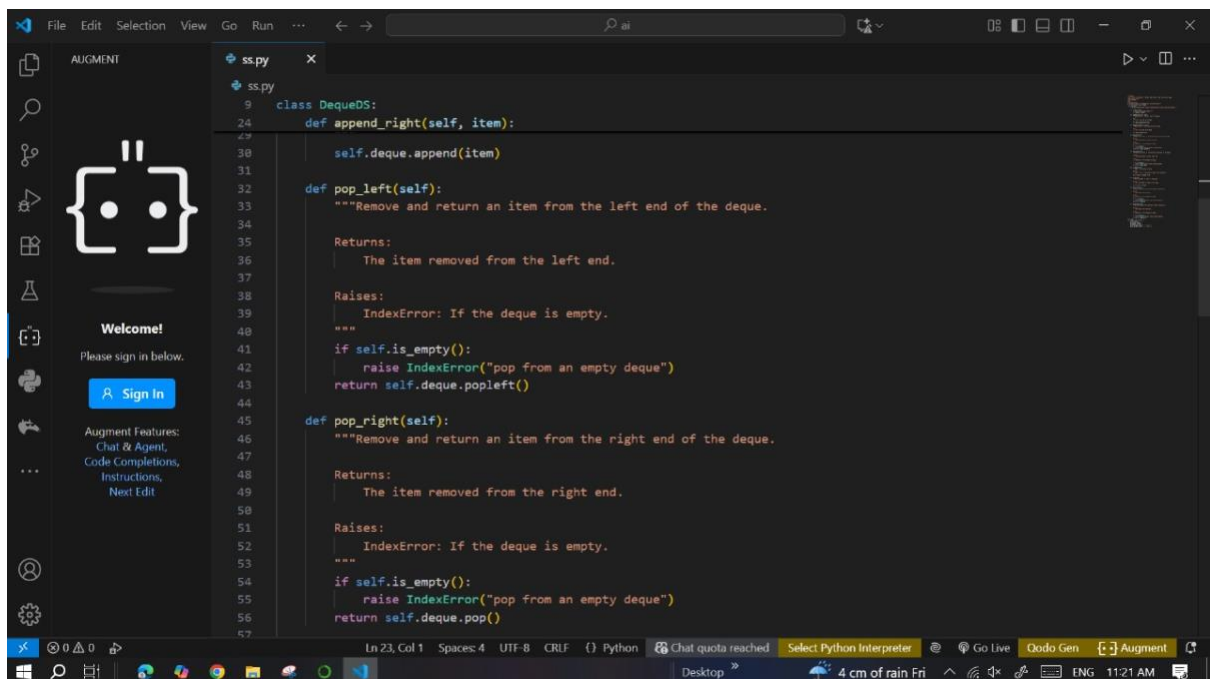
Sample Input Code:

class DequeDS:

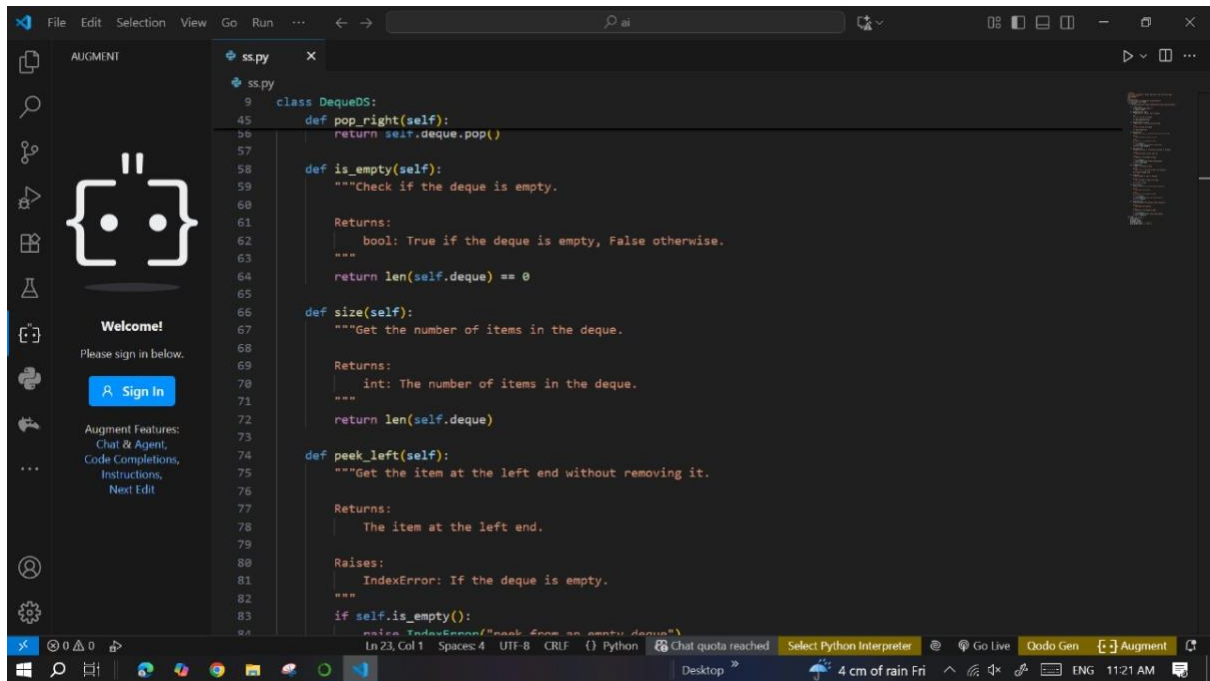
pass



```
1 """ Deque
2 Task: Use AI to implement a double-ended queue using collections.deque.
3 Sample Input Code:
4 class DequeDS:
5     pass
6 Expected Output:
7 * Insert and remove from both ends with docstrings"""
8 from collections import deque
9 class DequeDS:
10     """A double-ended queue (deque) implementation using collections.deque."""
11
12     def __init__(self):
13         """Initialize an empty deque."""
14         self.deque = deque()
15
16     def append_left(self, item):
17         """Add an item to the left end of the deque.
18
19         Args:
20             item: The item to be added.
21         """
22         self.deque.appendleft(item)
23
24     def append_right(self, item):
25         """Add an item to the right end of the deque.
26
27         Args:
28             item: The item to be added.
29         """
30         self.deque.append(item)
```



```
9 class DequeDS:
24     def append_right(self, item):
25         self.deque.append(item)
26
27     def pop_left(self):
28         """Remove and return an item from the left end of the deque.
29
30         Returns:
31             The item removed from the left end.
32
33         Raises:
34             IndexError: If the deque is empty.
35         """
36         if self.is_empty():
37             raise IndexError("pop from an empty deque")
38         return self.deque.popleft()
39
40     def pop_right(self):
41         """Remove and return an item from the right end of the deque.
42
43         Returns:
44             The item removed from the right end.
45
46         Raises:
47             IndexError: If the deque is empty.
48         """
49         if self.is_empty():
50             raise IndexError("pop from an empty deque")
51         return self.deque.pop()
```

```
File Edit Selection View Go Run ... < ->
AUGMENT
Welcome!
Please sign in below.
Sign In
Augment Features:
Chat & Agent,
Code Completions,
Instructions,
Next Edit

ss.py
class DequeDS:
    def pop_right(self):
        return self.deque.pop()

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

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

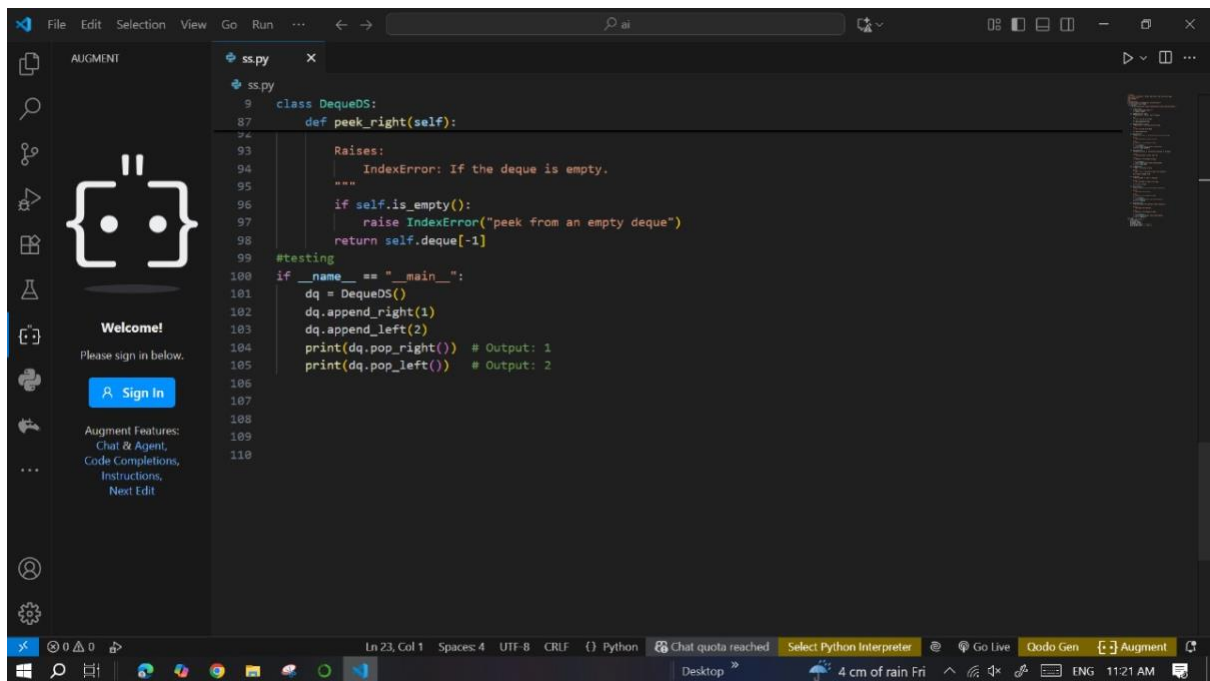
    def size(self):
        """Get the number of items in the deque.

        Returns:
            int: The number of items in the deque.
        """
        return len(self.deque)

    def peek_left(self):
        """Get the item at the left end without removing it.

        Returns:
            The item at the left end.

        Raises:
            IndexError: If the deque is empty.
        """
        if self.is_empty():
            raise IndexError("peek from an empty deque")
```



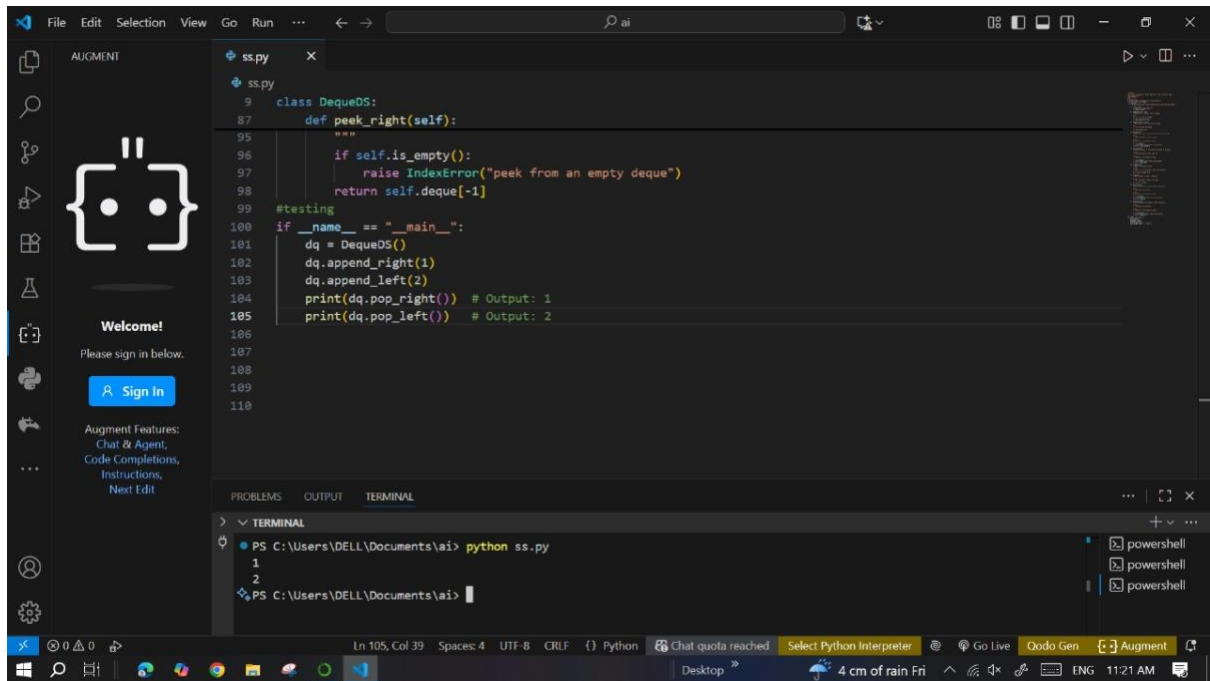
```
File Edit Selection View Go Run ... < ->
AUGMENT
Welcome!
Please sign in below.
Sign In
Augment Features:
Chat & Agent,
Code Completions,
Instructions,
Next Edit

ss.py
class DequeDS:
    def peek_right(self):
        """
        Raises:
            IndexError: If the deque is empty.
        """
        if self.is_empty():
            raise IndexError("peek from an empty deque")
        return self.deque[-1]

#testing
if __name__ == "__main__":
    dq = DequeDS()
    dq.append_right(1)
    dq.append_left(2)
    print(dq.pop_right()) # Output: 1
    print(dq.pop_left()) # Output: 2
```

Expected Output:

- Insert and remove from both ends with docstrings.

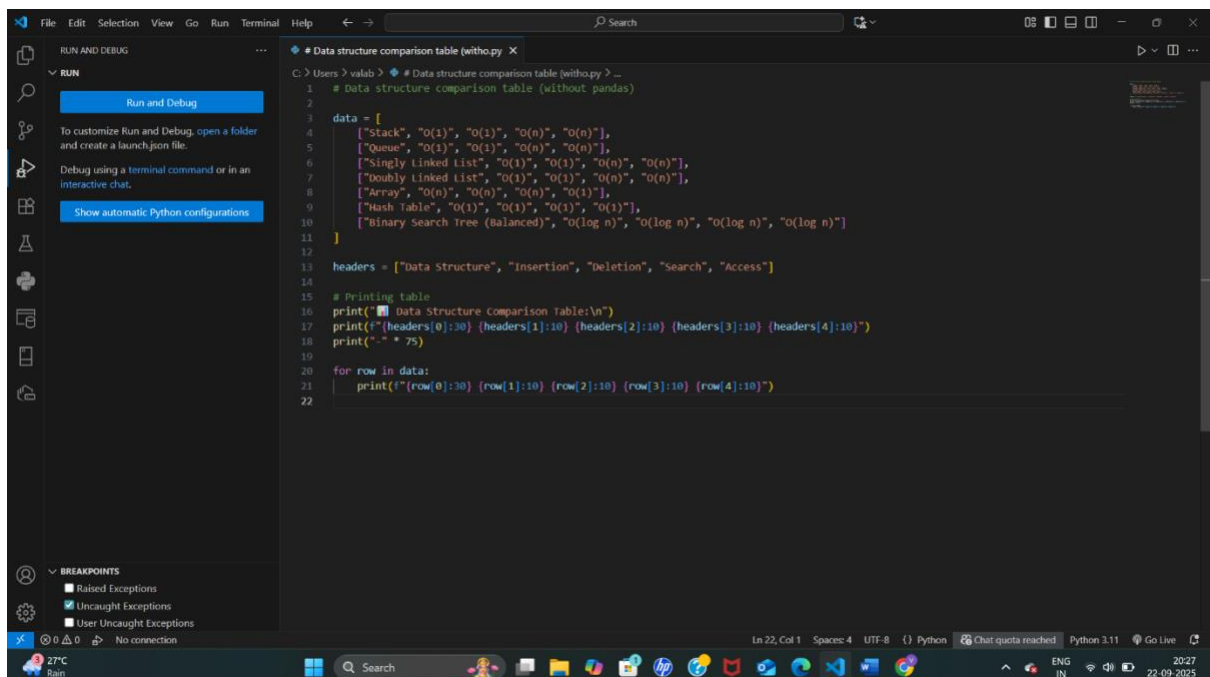


Task Description #9 – AI-Generated Data Structure Comparisons

Task: Use AI to generate a comparison table of different data structures (stack, queue, linked list, etc.) including time complexities.

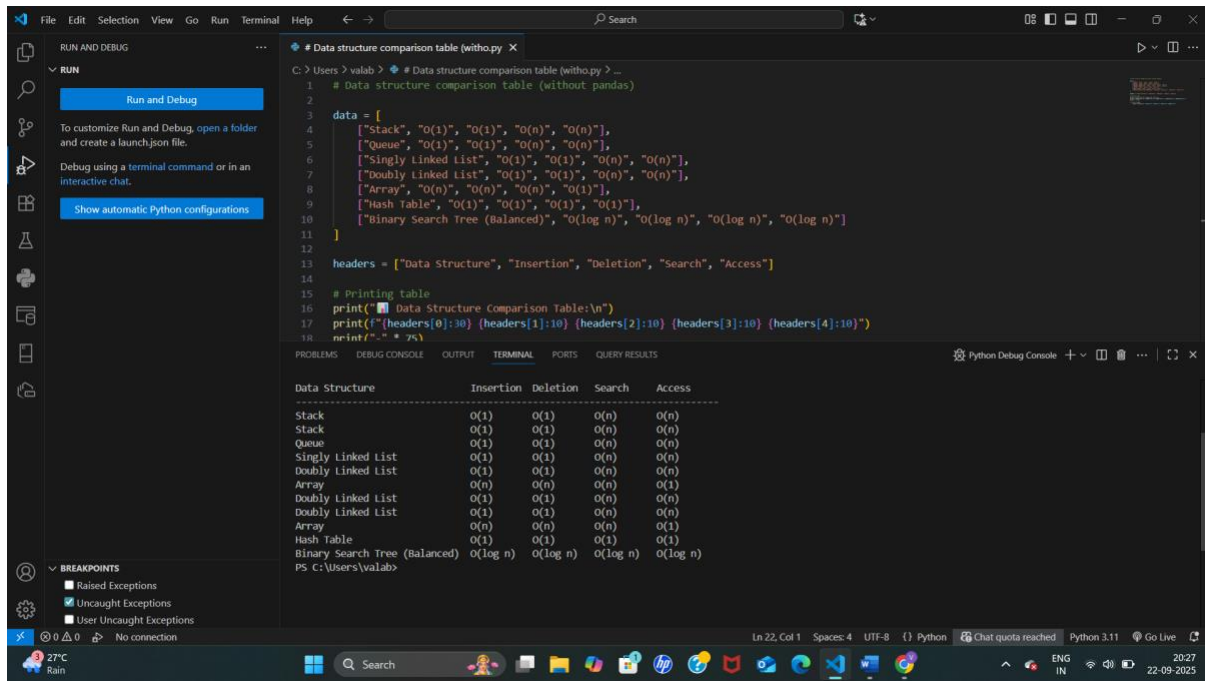
Sample Input Code:

No code, prompt AI for a data structure comparison table.



Expected Output:

- A markdown table with structure names, operations, and complexities.



```
1 # Data structure comparison table (without pandas)
2
3 data = [
4     ["Stack", "O(1)", "O(1)", "O(n)", "O(n)"],
5     ["Queue", "O(1)", "O(1)", "O(n)", "O(n)"],
6     ["Singly Linked List", "O(1)", "O(1)", "O(n)", "O(n)"],
7     ["Doubly Linked List", "O(1)", "O(1)", "O(n)", "O(n)"],
8     ["Array", "O(n)", "O(n)", "O(1)", "O(1)"],
9     ["Hash Table", "O(1)", "O(1)", "O(1)", "O(1)"],
10    ["Binary Search Tree (Balanced)", "O(log n)", "O(log n)", "O(log n)", "O(log n)"]
11]
12
13 headers = ["Data Structure", "Insertion", "Deletion", "Search", "Access"]
14
15 # Printing Table
16 print("\n Data Structure Comparison Table:\n")
17 print(f"{headers[0]:30} {headers[1]:10} {headers[2]:10} {headers[3]:10} {headers[4]:10}")
18 print("-" * 75)
```

Data Structure	Insertion	Deletion	Search	Access
Stack	O(1)	O(1)	O(n)	O(n)
Queue	O(1)	O(1)	O(n)	O(n)
Singly Linked List	O(1)	O(1)	O(n)	O(n)
Doubly Linked List	O(1)	O(1)	O(n)	O(n)
Array	O(n)	O(n)	O(1)	O(1)
Hash Table	O(1)	O(1)	O(1)	O(1)
Binary Search Tree (Balanced)	O(log n)	O(log n)	O(log n)	O(log n)

Task Description #10 Real-Time Application Challenge – Choose the Right Data Structure

Scenario:

Your college wants to develop a Campus Resource Management System that handles:

1. Student Attendance Tracking – Daily log of students entering/exiting the campus.
2. Event Registration System – Manage participants in events with quick search and removal.
3. Library Book Borrowing – Keep track of available books and their due dates.
4. Bus Scheduling System – Maintain bus routes and stop connections.
5. Cafeteria Order Queue – Serve students in the order they arrive.

Student Task:

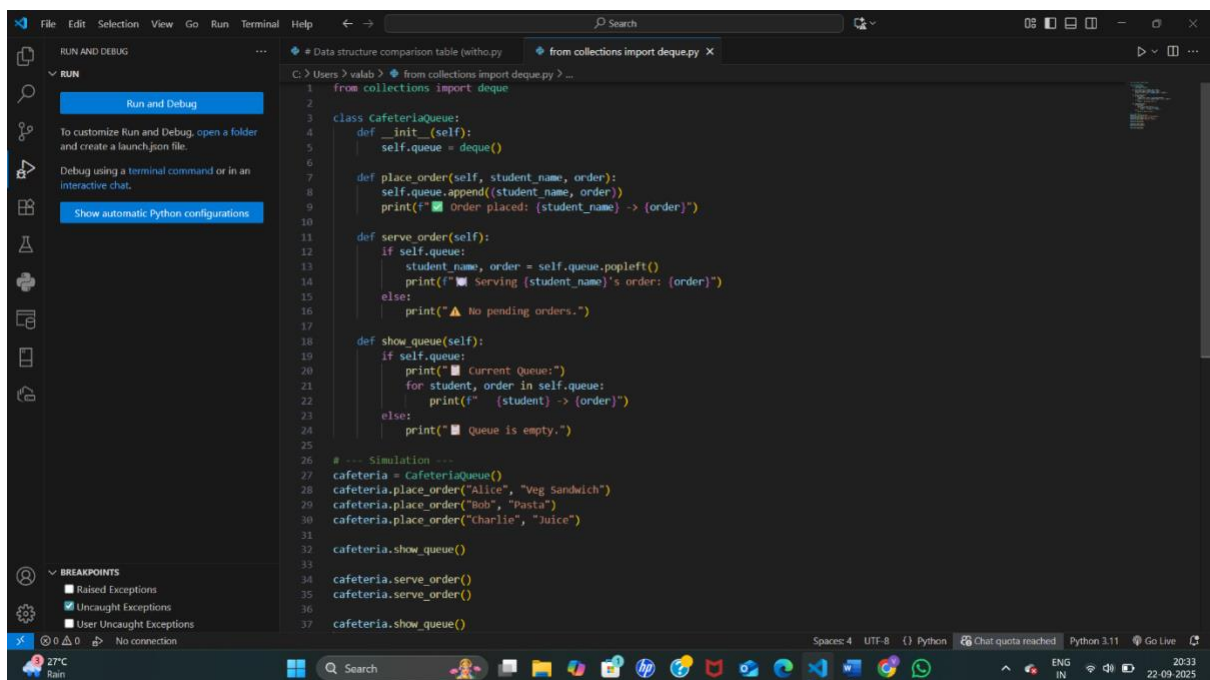
- For each feature, select the most appropriate data structure from the list

below:

- o Stack
- o Queue
- o Priority Queue
- o Linked List
- o Binary Search Tree (BST)
- o Graph
- o Hash Table
- o Deque

- Justify your choice in 2–3 sentences per feature.

- Implement one selected feature as a working Python program with AI-assisted code generation.



The screenshot shows a Python IDE with a file named 'witho.py'. The code implements a 'CafeteriaQueue' class using a 'deque' from the 'collections' module. The class has three methods: 'place_order' to add orders to the queue, 'serve_order' to remove and serve orders, and 'show_queue' to display the current queue. A simulation at the bottom demonstrates the queue's behavior with three orders: 'Alice' (Veg Sandwich), 'Bob' (Pasta), and 'Charlie' (Juice). The output shows the queue being served in order.

```
1 from collections import deque
2
3 class CafeteriaQueue:
4     def __init__(self):
5         self.queue = deque()
6
7     def place_order(self, student_name, order):
8         self.queue.append((student_name, order))
9         print(f"Order placed: {student_name} -> {order}")
10
11     def serve_order(self):
12         if self.queue:
13             student_name, order = self.queue.popleft()
14             print(f"Serving {student_name}'s order: {order}")
15         else:
16             print("No pending orders.")
17
18     def show_queue(self):
19         if self.queue:
20             print("Current Queue:")
21             for student, order in self.queue:
22                 print(f"    {student} -> {order}")
23         else:
24             print("Queue is empty.")
25
26 # --- Simulation ---
27 cafeteria = CafeteriaQueue()
28 cafeteria.place_order("Alice", "Veg Sandwich")
29 cafeteria.place_order("Bob", "Pasta")
30 cafeteria.place_order("Charlie", "Juice")
31 cafeteria.show_queue()
32
33 cafeteria.serve_order()
34 cafeteria.serve_order()
35 cafeteria.show_queue()
```

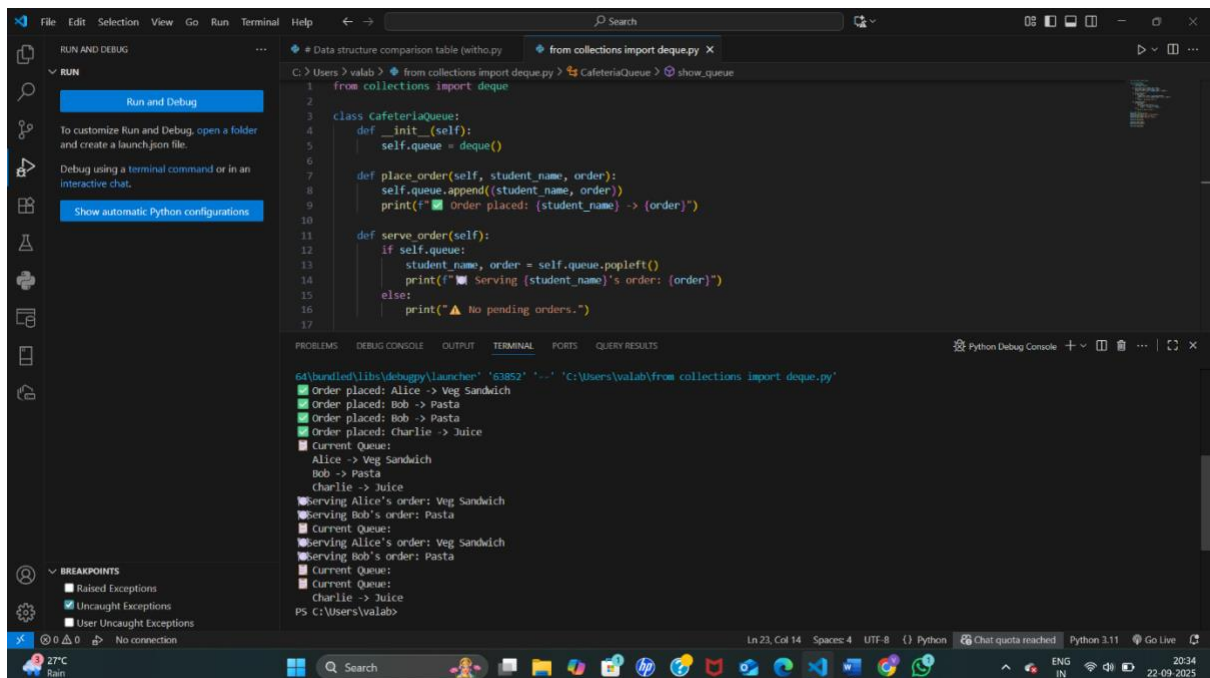
Expected Output:

- A table mapping feature → chosen data structure → justification.
- A functional Python program implementing the chosen feature with

comments and docstrings.

Deliverables (For All Tasks)

1. AI-generated prompts for code and test case generation.
2. At least 3 assert test cases for each task.
3. AI-generated initial code and execution screenshots.
4. Analysis of whether code passes all tests.
5. Improved final version with inline comments and explanation.
6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.



The screenshot shows a Python IDE with a dark theme. The main editor displays a Python script for a `CafeteriaQueue` class. The class uses a `deque` from the `collections` module to manage orders. The `__init__` method initializes the queue. The `place_order` method appends orders to the queue and prints them. The `serve_order` method serves orders from the queue, printing the student name and order, and prints "No pending orders." if the queue is empty.

```
1 from collections import deque
2
3 class CafeteriaQueue:
4     def __init__(self):
5         self.queue = deque()
6
7     def place_order(self, student_name, order):
8         self.queue.append((student_name, order))
9         print(f"Order placed: {student_name} -> {order}")
10
11     def serve_order(self):
12         if self.queue:
13             student_name, order = self.queue.popleft()
14             print(f"Serving {student_name}'s order: {order}")
15         else:
16             print("No pending orders.")
17
```

The terminal window at the bottom shows the execution output. It lists orders placed by Alice, Bob, and Charlie, followed by the current queue state and the orders being served.

```
64\bundled\libs\debugpy\launcher '63852' '-' 'C:\Users\valab\from collections import deque.py'
Order placed: Alice -> Veg Sandwich
Order placed: Bob -> Pasta
Order placed: Bob -> Pasta
Order placed: Charlie -> Juice
Current Queue:
Alice -> Veg Sandwich
Bob -> Pasta
Charlie -> Juice
Serving Alice's order: Veg Sandwich
Serving Bob's order: Pasta
Current Queue:
Serving Alice's order: Veg Sandwich
Serving Bob's order: Pasta
Current Queue:
Charlie -> Juice
PS C:\Users\valab>
```