

# Assignment-11.1

## Data Structures with AI: Implementing Fundamental Structures

K. Ruthwik Bhat

2403a51l08

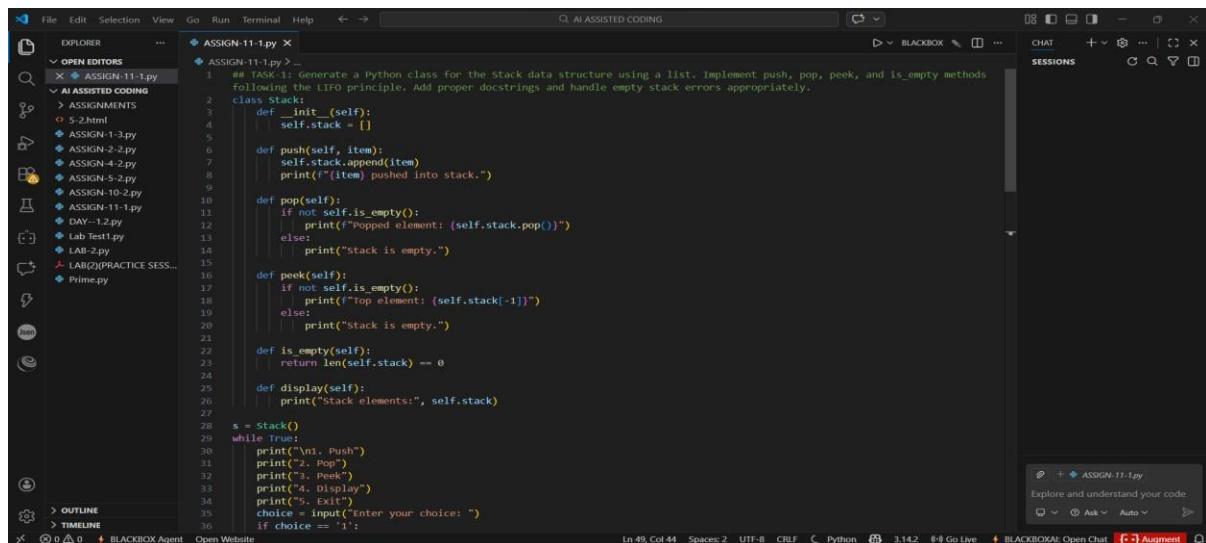
B-51

### Task Description #1 - Stack Implementation:

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

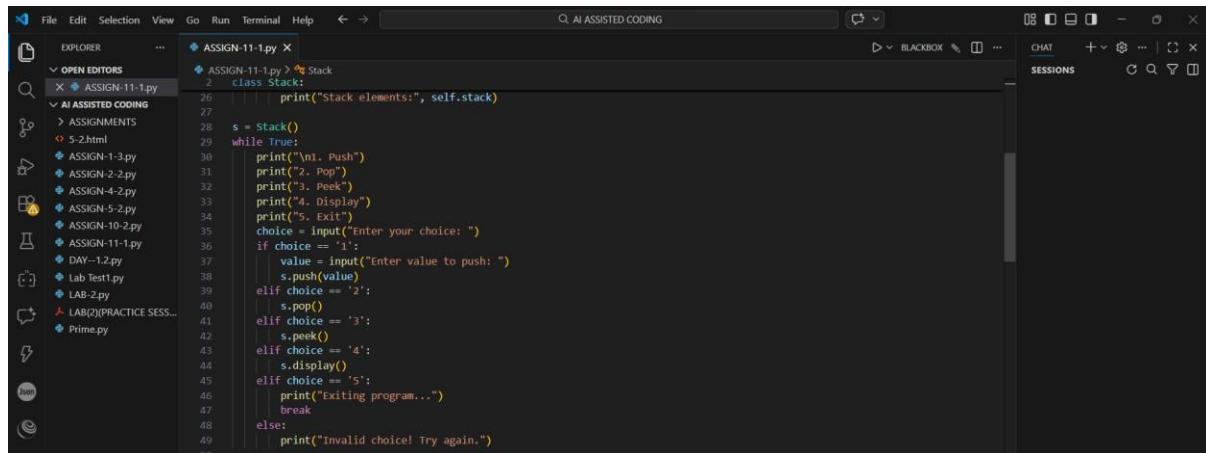
**PROMPT:** Generate a Python class for the Stack data structure using a list. Implement push, pop, peek, and is\_empty methods following the LIFO principle. Add proper docstrings and handle empty stack errors appropriately.

### Sample Input Code:



```
File Edit Selection View Go Run Terminal Help <- > AI ASSISTED CODING
EXPLORER OPEN EDITORS ASSIGN-11-1.py
AI ASSISTED CODING ASSIGNMENTS S-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
LABQ2(PRACTICE SESS...
Prime.py
ASSIGN-11-1.py ~
# TASK-1: 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.
1
2 class Stack:
3     def __init__(self):
4         self.stack = []
5
6     def push(self, item):
7         self.stack.append(item)
8         print(f"{item} pushed into stack.")
9
10    def pop(self):
11        if not self.is_empty():
12            print(f"Popped element: {self.stack.pop()}")
13        else:
14            print("Stack is empty.")
15
16    def peek(self):
17        if not self.is_empty():
18            print(f"Top element: {self.stack[-1]}")
19        else:
20            print("Stack is empty.")
21
22    def is_empty(self):
23        return len(self.stack) == 0
24
25    def display(self):
26        print("Stack elements:", self.stack)
27
28 s = Stack()
29 while True:
30     print("\n1. Push")
31     print("2. Pop")
32     print("3. Peek")
33     print("4. Display")
34     print("5. Exit")
35     choice = input("Enter your choice: ")
36     if choice == '1':
37         value = input("Enter value to push: ")
38         s.push(value)
39     elif choice == '2':
40         s.pop()
41     elif choice == '3':
42         s.peek()
43     elif choice == '4':
44         s.display()
45     elif choice == '5':
46         print("Exiting program...")
47         break
48     else:
49         print("Invalid choice! Try again.")

In 49, Col 44 Spaces: 2 UFT-B CRLF C Python 3.14.2 8:0 Go Live BLACKBOXAI: Open Chat [C] Alignent
```



```
File Edit Selection View Go Run Terminal Help <- > AI ASSISTED CODING
EXPLORER OPEN EDITORS ASSIGN-11-1.py
AI ASSISTED CODING ASSIGNMENTS S-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
LABQ2(PRACTICE SESS...
Prime.py
ASSIGN-11-1.py > Stack
2 class Stack:
3     def __init__(self):
4         self.stack = []
5
6     def push(self, item):
7         self.stack.append(item)
8         print(f"{item} pushed into stack.")
9
10    def pop(self):
11        if not self.is_empty():
12            print(f"Popped element: {self.stack.pop()}")
13        else:
14            print("Stack is empty.")
15
16    def peek(self):
17        if not self.is_empty():
18            print(f"Top element: {self.stack[-1]}")
19        else:
20            print("Stack is empty.")
21
22    def is_empty(self):
23        return len(self.stack) == 0
24
25    def display(self):
26        print("Stack elements:", self.stack)
27
28 s = Stack()
29 while True:
30     print("\n1. Push")
31     print("2. Pop")
32     print("3. Peek")
33     print("4. Display")
34     print("5. Exit")
35     choice = input("Enter your choice: ")
36     if choice == '1':
37         value = input("Enter value to push: ")
38         s.push(value)
39     elif choice == '2':
40         s.pop()
41     elif choice == '3':
42         s.peek()
43     elif choice == '4':
44         s.display()
45     elif choice == '5':
46         print("Exiting program...")
47         break
48     else:
49         print("Invalid choice! Try again.")

In 49, Col 44 Spaces: 2 UFT-B CRLF C Python 3.14.2 8:0 Go Live BLACKBOXAI: Open Chat [C] Alignent
```

### OUTPUT:

```

class Stack:
    def __init__(self):
        self.stack = []

    def push(self, value):
        self.stack.append(value)

    def pop(self):
        if len(self.stack) > 0:
            return self.stack.pop()
        else:
            return "Stack is empty"

    def peek(self):
        if len(self.stack) > 0:
            return self.stack[-1]
        else:
            return "Stack is empty"

    def display(self):
        print("Stack elements: ", self.stack)

    def exit(self):
        print("Exiting program...")
        exit()

def main():
    stack = Stack()
    choice = None

    while choice != '5':
        print("\n1. Push")
        print("2. Pop")
        print("3. Peek")
        print("4. Display")
        print("5. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            value = input("Enter value to push: ")
            stack.push(value)
            print(f'{value} pushed into stack.')
        elif choice == '2':
            print(stack.pop())
        elif choice == '3':
            print(stack.peek())
        elif choice == '4':
            stack.display()
        elif choice == '5':
            break
        else:
            print("Invalid choice! Try again.")

if __name__ == "__main__":
    main()

```

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:

```

class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, item):
        self.queue.append(item)
        print(f"({item}) enqueued into queue.")

    def dequeue(self):
        if not self.is_empty():
            print(f"Dequeued element: ({self.queue.pop(0)})")
        else:
            print("Queue is empty.")

    def peek(self):
        if not self.is_empty():
            print(f"Front element: ({self.queue[0]})")
        else:
            print("Queue is empty.")

    def is_empty(self):
        return len(self.queue) == 0

    def display(self):
        print("Queue elements: ", self.queue)

q = Queue()
while True:
    print("\n1. Enqueue")
    print("2. Dequeue")
    print("3. Peek")
    print("4. Display")
    print("5. Exit")
    choice = input("Enter your choice: ")
    if choice == '1':
        item = input("Enter item to enqueue: ")
        q.enqueue(item)
    elif choice == '2':
        q.dequeue()
    elif choice == '3':
        q.peek()
    elif choice == '4':
        q.display()
    elif choice == '5':
        break
    else:
        print("Invalid choice! Try again.")

```

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

1. Enqueue  
2. Dequeue  
3. Peek  
4. Display  
5. Exit  
Enter your choice: 1  
Enter item to enqueue: 10  
(10) enqueued into queue.  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Display  
5. Exit  
Enter your choice: 2  
Dequeued element: (10)  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Display  
5. Exit  
Enter your choice: 3  
Front element: (10)  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Display  
5. Exit  
Enter your choice: 4  
Queue elements: [10]  
1. Enqueue  
2. Dequeue  
3. Peek  
4. Display  
5. Exit  
Enter your choice: 5

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files like `ASSIGN-11-1.py`, `ASSIGN-1-3.py`, `ASSIGN-2-2.py`, etc.
- Code Editor:** Displays the `ASSIGN-11-1.py` file content:

```
77 q = Queue()
78 while True:
79     print("1. Enqueue")
80     print("2. Dequeue")
81     print("3. Peek")
82     print("4. Display")
83     print("5. Exit")
84     choice = input("Enter your choice: ")
85     if choice == '1':
86         value = input("Enter value to enqueue: ")
87         q.enqueue(value)
88     elif choice == '2':
89         q.dequeue()
90     elif choice == '3':
91         q.peek()
92     elif choice == '4':
93         q.display()
94     elif choice == '5':
95         print("Exiting program...")
96         break
97     else:
98         print("Invalid choice! Try again.")
```
- Terminal:** Shows the command `python ASSIGN-11-1.py` running.
- Sessions:** Shows a session named `BLACKBOX`.

## OUTPUT:

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files like `ASSIGN-11-1.py`, `ASSIGN-1-3.py`, `ASSIGN-2-2.py`, etc.
- Terminal:** Shows the output of the `ASSIGN-11-1.py` program:

```
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: 4
[Output redacted]
```
- Sessions:** Shows a session named `BLACKBOX`.

**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:

```

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

```

```

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

```

## OUTPUT:

```

PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> python3.11\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.

```

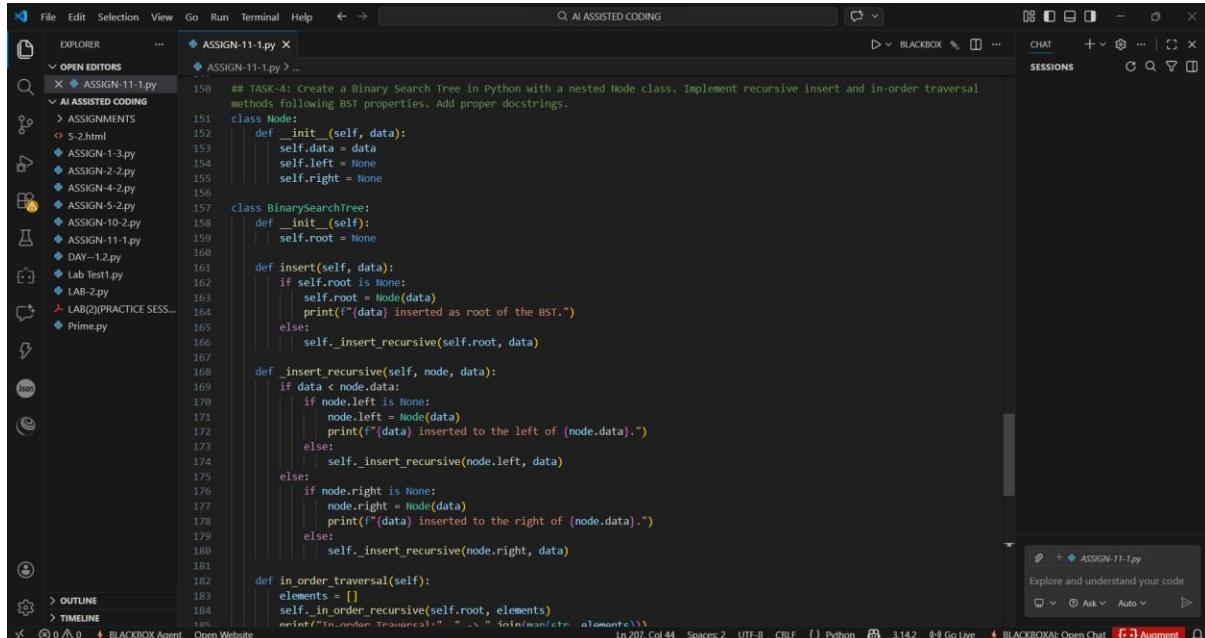
**EXPLANATION:** A Singly Linked List consists of nodes where each node contains data and a reference to the next node. Unlike arrays, it does not require contiguous memory, making it dynamic in size. It allows efficient insertions and deletions compared to fixed-size structures.

## 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:



```
ASSIGN-11-1.py
# TASK-4: 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.

class Node:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

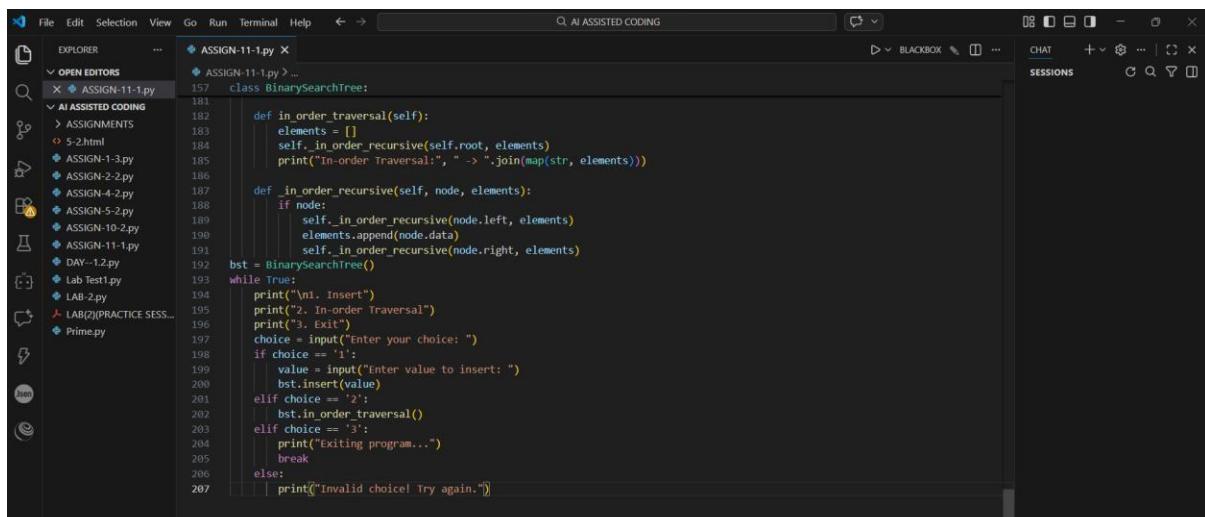
class BinarySearchTree:
    def __init__(self):
        self.root = None

    def insert(self, data):
        if self.root is None:
            self.root = Node(data)
            print(f"{data} inserted as root of the BST.")
        else:
            self._insert_recursive(self.root, data)

    def _insert_recursive(self, node, data):
        if data < node.data:
            if node.left is None:
                node.left = Node(data)
                print(f"{data} inserted to the left of {node.data}.")
            else:
                self._insert_recursive(node.left, data)
        else:
            if node.right is None:
                node.right = Node(data)
                print(f"{data} inserted to the right of {node.data}.")
            else:
                self._insert_recursive(node.right, data)

    def in_order_traversal(self):
        elements = []
        self._in_order_recursive(self.root, elements)
        print("In-order Traversal: " + " ".join(map(str, elements)))

    def _in_order_recursive(self, node, elements):
        if node:
            self._in_order_recursive(node.left, elements)
            elements.append(node.data)
            self._in_order_recursive(node.right, elements)
```

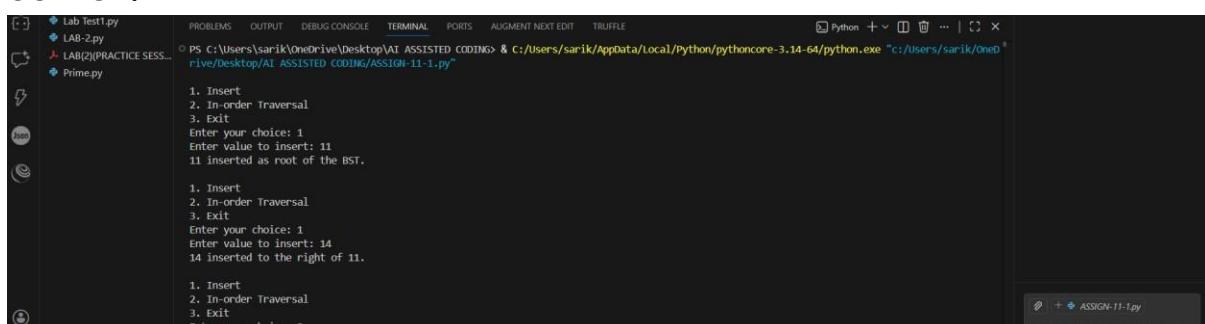


```
ASSIGN-11-1.py
class BinarySearchTree:
    def in_order_traversal(self):
        elements = []
        self._in_order_recursive(self.root, elements)
        print("In-order Traversal: " + " ".join(map(str, elements)))

    def _in_order_recursive(self, node, elements):
        if node:
            self._in_order_recursive(node.left, elements)
            elements.append(node.data)
            self._in_order_recursive(node.right, elements)

bst = BinarySearchTree()
while True:
    print("1. Insert")
    print("2. In-order Traversal")
    print("3. Exit")
    choice = input("Enter your choice: ")
    if choice == '1':
        value = input("Enter value to insert: ")
        bst.insert(value)
    elif choice == '2':
        bst.in_order_traversal()
    elif choice == '3':
        print("Exiting program...")
        break
    else:
        print("Invalid choice! Try again.")
```

### OUTPUT:



```
PS C:\Users\sarik\OneDrive\Desktop\AI ASSISTED CODING> python 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: 2
11 14
Enter your choice: 3
```

**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:

```
File Edit Selection View Go Run Terminal Help ← → Q AI ASSISTED CODING EXPLORE OPEN EDITORS ASSIGN-11-1.py < ... ASSIGN-11-1.py > ... 209 ## TASK-5: Implement a Hash Table in Python using chaining for collision handling. Include insert, search, and delete methods with comments explaining the hashing mechanism. 210 class HashTable: 211     def __init__(self, size=10): 212         self.size = size 213         self.table = [[ ] for _ in range(size)] 214 215     def _hash(self, key): 216         return hash(key) % self.size 217 218     def insert(self, key, value): 219         index = self._hash(key) 220         for i, (k, v) in enumerate(self.table[index]): 221             if k == key: 222                 self.table[index][i] = (key, value) 223                 print(f"Updated {key} with value {value}.") 224                 return 225         self.table[index].append((key, value)) 226         print(f"Inserted {key} with value {value}.") 227 228     def search(self, key): 229         index = self._hash(key) 230         for k, v in self.table[index]: 231             if k == key: 232                 print(f"Found {key} with value {v}.") 233                 return v 234         print(f"{key} not found.") 235         return None 236 237     def delete(self, key): 238         index = self._hash(key) 239         for i, (k, v) in enumerate(self.table[index]): 240             if k == key: 241                 del self.table[index][i] 242                 print(f"Deleted {key}.") 243             else: 244                 return 245 246 247 248 249 250 251 252 253 254
```

The screenshot shows a Microsoft Visual Studio Code (VS Code) window. The title bar includes the standard File, Edit, Selection, View, Go, Run, Terminal, Help, and a back/forward navigation icon. To the right of the title bar is a search bar containing "AI ASSISTED CODING". The left sidebar, titled "EXPLORER", lists various project files under sections like "OPEN EDITORS" and "AI ASSISTED CODING". The main editor area displays a Python script named "ASSIGN-11-1.py". The code defines a `HashTable` class with methods for insertion, search, deletion, and exiting. An AI-assisted code editor interface is overlaid on the right side of the editor area, showing suggestions and annotations for the current line of code.

```
ASSIGN-11-1.py
ASSIGN-11-1.py ...
210 class HashTable:
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("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/OneD...rive/Desktop/AI ASSISTED CODING/ASSIGN-11-1.py"
1. Insert
2. Search
3. Delete
4. Exit
Enter your choice: 1
Enter key to insert: 14
Enter value to insert: 2
Inserted 14 with value 2.

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

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

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

## Task Description #6 - Graph Representation:

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

**PROMPT:** Generate a Graph implementation in Python using an adjacency list representation. Include methods to add vertices, add edges, and display connections with documentation.

## Sample Input Code:

```
## TASK-6: Generate a Graph implementation in Python using an adjacency list representation. Include methods to add vertices, add edges, and display connections with documentation.

class Graph:
    def __init__(self):
        self.adjacency_list = {}

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

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

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

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

```
g.add_vertex(vertex)
elif choice == '2':
    vertex1 = input("Enter first vertex: ")
    vertex2 = input("Enter second vertex: ")
    g.add_edge(vertex1, vertex2)
elif choice == '3':
    g.display_connections()
elif choice == '4':
    print("Exiting program...")
    break
else:
    print("Invalid choice! Try again.")
```

## OUTPUT:

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

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

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

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

## Task Description #7 - Priority Queue:

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

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

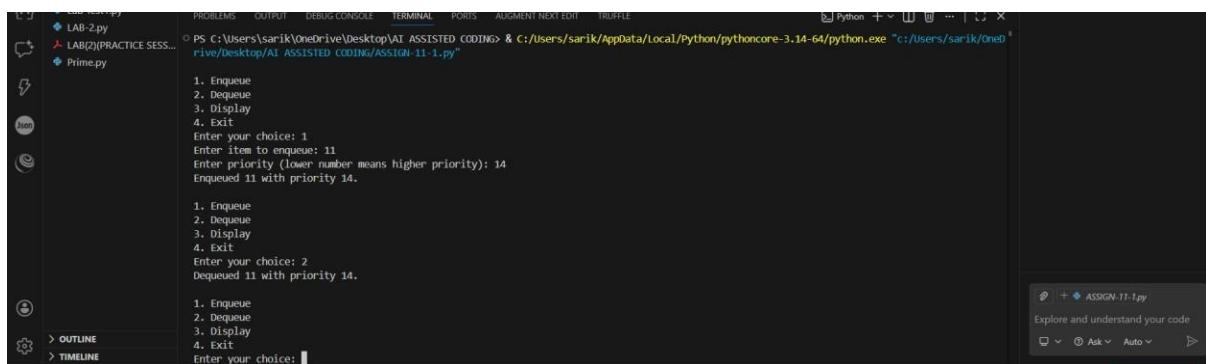
## Sample Input Code:

```
File Edit Selection View Go Run Terminal Help ← → ⌂ AI ASSISTED CODING
OPEN EDITORS
ASSIGN-11-1.py
AI ASSISTED CODING
ASSIGNMENTS
5-2.html
ASSIGN-1-3.py
ASSIGN-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 SESSIONS)
Prime.py

ASSIGN-11-1.py > ...
314  ## TASK-7: Create a Priority Queue in Python using the heapq module. Implement enqueue with priority, dequeue (highest priority first), and display methods. Add proper documentation.
315
316  import heapq
317  class PriorityQueue:
318      def __init__(self):
319          self.elements = []
320
321      def enqueue(self, item, priority):
322          heapq.heappush(self.elements, (priority, item))
323          print("Enqueued {} with priority {}".format(item, priority))
324
325      def dequeue(self):
326          if not self.is_empty():
327              priority, item = heapq.heappop(self.elements)
328              print("Dequeued {} with priority {}".format(item, priority))
329          else:
330              print("Priority Queue is empty.")
331
332      def is_empty(self):
333          return len(self.elements) == 0
334
335      def display(self):
336          print("Priority Queue elements:", [(priority, item) for priority, item in self.elements])
337
338  pq = PriorityQueue()
339  while True:
340      print("\n1. Enqueue")
341      print("2. Dequeue")
342      print("3. Display")
343      print("4. Exit")
344      choice = input("Enter your choice: ")
345
346      if choice == '1':
347          item = input("Enter item to enqueue: ")
348          priority = int(input("Enter priority (lower number means higher priority): "))
349          pq.enqueue(item, priority)
350
351      elif choice == '2':
352          pq.dequeue()
353
354      elif choice == '3':
355          pq.display()
356
357      elif choice == '4':
358          print("Exiting program...")
359          break
360
361      else:
362          print("Invalid choice! Try again.")

ASSIGN-5-2.py
ASSIGN-10-2.py
ASSIGN-11-1.py
DAY-1-2.py
Lab Test1.py
LAB-2.py
LAB2(PRACTICE SESSIONS)
Prime.py
```

## OUTPUT:



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

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

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

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

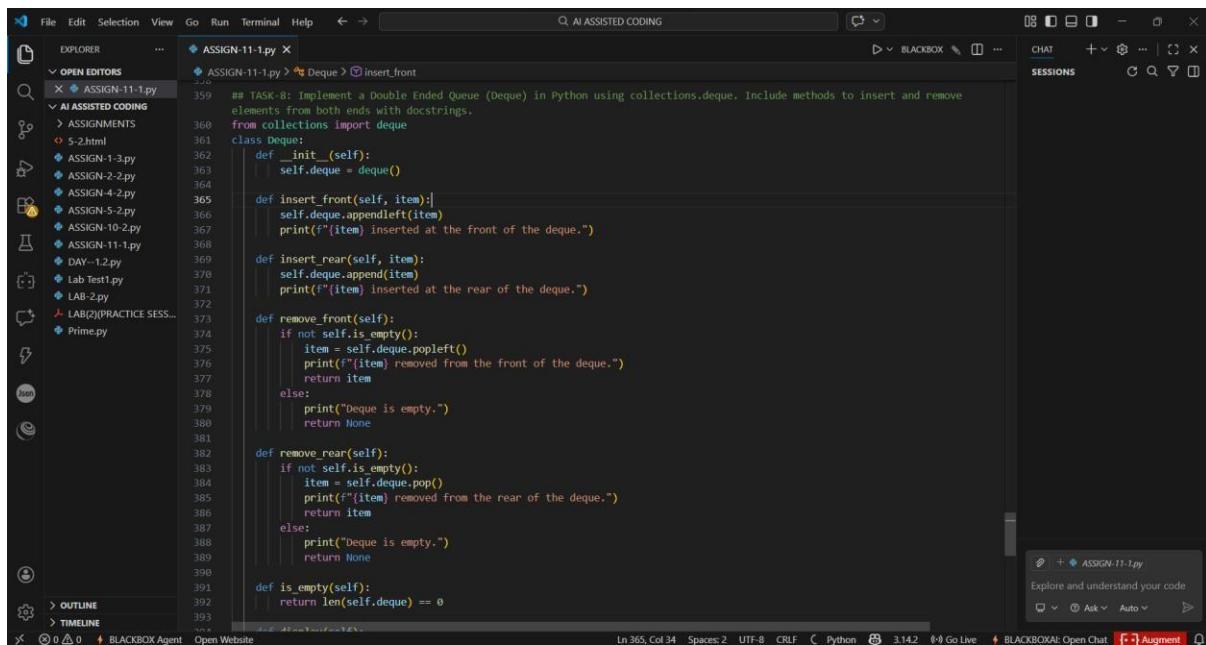
**EXPLANATION:** A Priority Queue is a special type of queue where elements are removed based on priority rather than order of insertion. Higher priority elements are processed first. It is typically implemented using a heap for efficiency.

### Task Description #8 - Deque:

**Task:** Use AI to implement a double-ended queue using `Collections.deque`.

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

### Sample Input Code:



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

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

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

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

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

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

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

```
File Edit Selection View Go Run Terminal Help ⌘ ⌘ AI ASSISTED CODING EXPLORER OPEN EDITORS ... ASSIGN-11-1.py X AI ASSISTED CODING 361 class Deque:
```

```
393     def display(self):
394         print("Deque elements:", list(self.deque))
395
396     d = deque()
397     while True:
398         print("1. Insert Front")
399         print("2. Insert Rear")
400         print("3. Remove Front")
401         print("4. Remove Rear")
402         print("5. Display")
403         print("6. Exit")
404
405         choice = input("Enter your choice: ")
406         if choice == '1':
407             item = input("Enter item to insert at front: ")
408             d.insert_front(item)
409         elif choice == '2':
410             item = input("Enter item to insert at rear: ")
411             d.insert_rear(item)
412         elif choice == '3':
413             d.remove_front()
414         elif choice == '4':
415             d.remove_rear()
416         elif choice == '5':
417             d.display()
418         elif choice == '6':
419             print("Exiting program...")
420             break
421         else:
422             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 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.