
Lab Assignment # 11.1

Program : B. Tech (CSE)
Specialization : -
Course Title : AI Assisted Coding
Course Code : 23CS002PC304
Semester II
Academic Session : 2025-2026
Name of Student : P.Eshwar
Enrollment No. : 2403A51L26
Batch No. 51
Date : 06/02/26

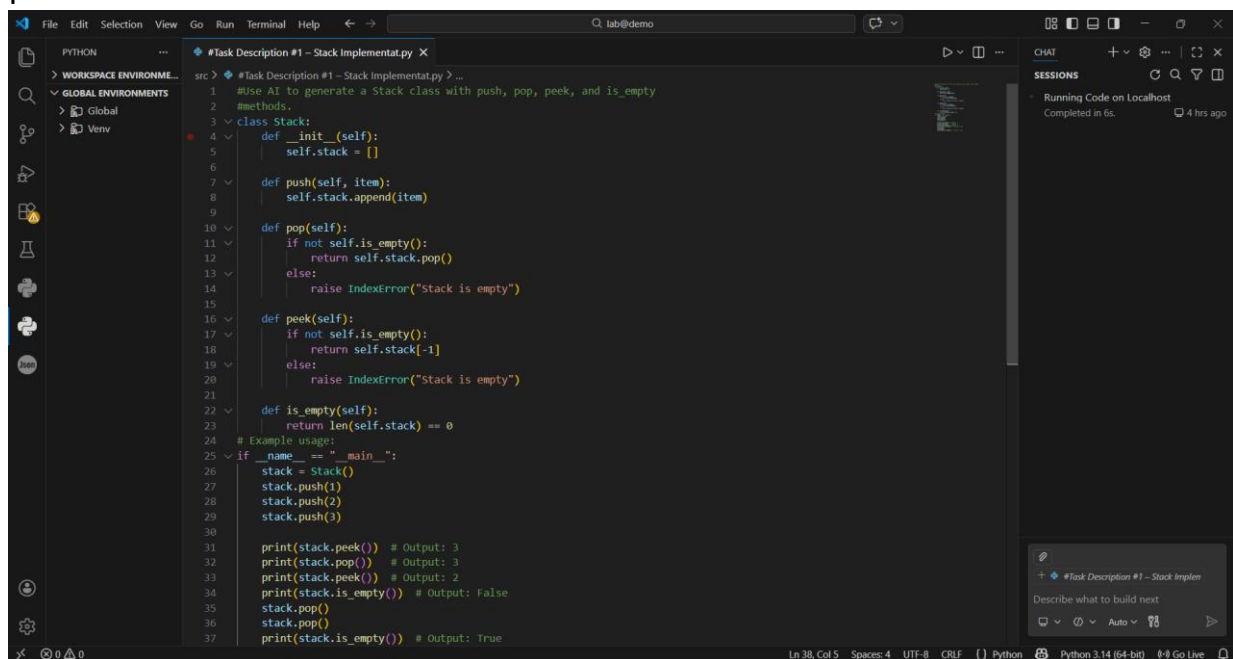
Submission Starts here**Screenshots:****Task Description #1**

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

Sample Input Code:

class Stack:

pass



```
# Task Description #1 - Stack Implementation.py
1 #Use AI to generate a Stack class with push, pop, peek, and is_empty
2 #methods.
3 class Stack:
4     def __init__(self):
5         self.stack = []
6
7     def push(self, item):
8         self.stack.append(item)
9
10    def pop(self):
11        if not self.is_empty():
12            return self.stack.pop()
13        else:
14            raise IndexError("Stack is empty")
15
16    def peek(self):
17        if not self.is_empty():
18            return self.stack[-1]
19        else:
20            raise IndexError("Stack is empty")
21
22    def is_empty(self):
23        return len(self.stack) == 0
24
25    # Example usage:
26    if __name__ == "__main__":
27        stack = Stack()
28        stack.push(1)
29        stack.push(2)
30        stack.push(3)
31
32        print(stack.peek()) # Output: 3
33        print(stack.pop()) # Output: 3
34        print(stack.pop()) # Output: 2
35        print(stack.is_empty()) # Output: False
36        stack.pop()
37        print(stack.is_empty()) # Output: True
```

```
PS D:\lab@demo> & "C:\Program Files\Python314\python.exe" "d:/lab@demo/src/#Task Description #1 - Stack Implementat.py"
3
3
2
False
True
PS D:\lab@demo> |
```

Task Description #2 Queue Implementation

Task:

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

Sample Input Code:

class Queue:

pass

```
rc > #Task Description #1 - Stack Implementat.py > ...
1  #Use AI to implement a Queue using Python lists.
2  #Sample Input Code:
3  #class Queue:
4  #pass
5  class Queue:
6      def __init__(self):
7          self.items = []
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("Queue is empty")
17
18     def is_empty(self):
19         return len(self.items) == 0
20
21     def size(self):
22         return len(self.items)
23
24     def __str__(self):
25         return str(self.items)
26 # Example usage:
27 if __name__ == "__main__":
28     queue = Queue()
29     queue.enqueue(1)
30     queue.enqueue(2)
31     queue.enqueue(3)
32     print(queue) # Output: [1, 2, 3]
33
34     print(queue.dequeue()) # Output: 1
35     print(queue) # Output: [2, 3]
36
37     print(queue.is_empty()) # Output: False
```

```
src > #Task Description #1 – Stack Implementat.py > ...
5  class Queue:
23
24      def __str__(self):
25          return str(self.items)
26  # Example usage:
27  if __name__ == "__main__":
28      queue = Queue()
29      queue.enqueue(1)
30      queue.enqueue(2)
31      queue.enqueue(3)
32      print(queue) # Output: [1, 2, 3]
33
34      print(queue.dequeue()) # Output: 1
35      print(queue) # Output: [2, 3]
36
37      print(queue.is_empty()) # Output: False
38      print(queue.size()) # Output: 2
39      queue.dequeue()
40      queue.dequeue()
41      print(queue.is_empty()) # Output: True
42
43
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\lab@demo> & "C:\Program Files\Python314\python.exe" "d:/lab@demo/src/#Task Description #1 – Stack Implementat.py"
[1, 2, 3]
1
[2, 3]
False
2
True
PS D:\lab@demo> |
```

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
```

```
#Task Description #1 – Stack Implementat.py X
src > #Task Description #1 – Stack Implementat.py > ...
1  # - Linked List
2  #Task: Use AI to generate a Singly Linked List with insert and display
3  #methods.
4  #Sample Input Code:
5  #class Node:
6  #pass
7  #class LinkedList:
8  #pass
9  class Node:
10     def __init__(self, data):
11         self.data = data
12         self.next = None
13
14     class LinkedList:
15         def __init__(self):
16             self.head = None
17
18         def insert(self, data):
19             new_node = Node(data)
20             if self.head is None:
21                 self.head = new_node
22                 return
23             last_node = self.head
24             while last_node.next:
25                 last_node = last_node.next
26             last_node.next = new_node
27
28         def display(self):
29             current_node = self.head
30             while current_node:
31                 print(current_node.data, end=' ')
32                 current_node = current_node.next
33             print()
34     # Example usage:
35     linked_list = LinkedList()
36     linked_list.insert(10)
37     linked_list.insert(20)
```

```
PS D:\lab@demo> & "C:\Program Files\Python314\python.exe" "d:/lab@demo/src/#Task Description #1 – Stack Implementat.py"
10 20 30
PS D:\lab@demo> █
```

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
```

#Task Description #1 – Stack Implementat.py X

src > #Task Description #1 – Stack Implementat.py > ...

```
1  #Binary Search Tree (BST)
2  #Task: Use AI to create a BST with insert and in-order traversal methods.
3  #Sample Input Code:
4  #class BST:
5  #pass
6  class Node:
7      def __init__(self, key):
8          self.left = None
9          self.right = None
10         self.val = key
11
12     class BST:
13         def __init__(self):
14             self.root = None
15
16         def insert(self, key):
17             if self.root is None:
18                 self.root = Node(key)
19             else:
20                 self._insert_recursively(self.root, key)
21
22         def _insert_recursively(self, node, key):
23             if key < node.val:
24                 if node.left is None:
25                     node.left = Node(key)
26                 else:
27                     self._insert_recursively(node.left, key)
28             else:
29                 if node.right is None:
30                     node.right = Node(key)
31                 else:
32                     self._insert_recursively(node.right, key)
33
34         def in_order_traversal(self):
35             return self._in_order_recursively(self.root)
36
37         def _in_order_recursively(self, node):
```

```
src > #Task Description #1 – Stack Implementat.py > ...
12 class BST:
36
37     def _in_order_recursively(self, node):
38         res = []
39         if node:
40             res = self._in_order_recursively(node.left)
41             res.append(node.val)
42             res = res + self._in_order_recursively(node.right)
43         return res
44 # Example usage:
45 bst = BST()
46 bst.insert(5)
47 bst.insert(3)
48 bst.insert(7)
49 print(bst.in_order_traversal()) # Output: [3, 5, 7]
50
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\lab@demo> & "C:\Program Files\Python314\python.exe" "d:/lab@demo/src/#Task Description #1 - St
[3, 5, 7]
PS D:\lab@demo> █
```

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

```
src > #Task Description #1 – Stack Implementat.py > ...
1  # – Hash Table
2  #Task: Use AI to implement a hash table with basic insert, search, and
3  #delete methods.
4  #Sample Input Code:
5  #class HashTable:
6  #pass
7  class HashTable:
8      def __init__(self, size=10):
9          self.size = size
10         self.table = [None] * self.size
11
12     def _hash(self, key):
13         return hash(key) % self.size
14
15     def insert(self, key, value):
16         index = self._hash(key)
17         if self.table[index] is None:
18             self.table[index] = [(key, value)]
19         else:
20             for i, (k, v) in enumerate(self.table[index]):
21                 if k == key:
22                     self.table[index][i] = (key, value)
23                 return
24             self.table[index].append((key, value))
25
26     def search(self, key):
27         index = self._hash(key)
28         if self.table[index] is not None:
29             for k, v in self.table[index]:
30                 if k == key:
31                     return v
32         return None
33
34     def delete(self, key):
35         index = self._hash(key)
36         if self.table[index] is not None:
37             for i, (k, v) in enumerate(self.table[index]):
```

```
        if k == key:
            del self.table[index][i]
            return True
    return False

# Example usage:
hash_table = HashTable()
hash_table.insert("name", "Alice")
hash_table.insert("age", 30)
print(hash_table.search("name")) # Output: Alice
print(hash_table.search("age")) # Output: 30
hash_table.delete("name")
print(hash_table.search("name")) # Output: None
```

```
PS D:\lab@demo> & "C:\Program Files\Python314\python.exe" "d:/lab@demo/src/#Task Desc
Alice
30
None
PS D:\lab@demo> █
```

Task Description #6 – Graph Representation

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

Sample Input Code:

```
class Graph:
    pass
```

```
#Task Description #1 – Stack Implementat.py X
src > #Task Description #1 – Stack Implementat.py > ...
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  class Graph:
7      def __init__(self):
8          self.adjacency_list = {}
9
10     def add_vertex(self, vertex):
11         if vertex not in self.adjacency_list:
12             self.adjacency_list[vertex] = []
13
14     def add_edge(self, vertex1, vertex2):
15         if vertex1 in self.adjacency_list and vertex2 in self.adjacency_list:
16             self.adjacency_list[vertex1].append(vertex2)
17             self.adjacency_list[vertex2].append(vertex1)
18
19     def __str__(self):
20         return str(self.adjacency_list)
21 # Example usage:
22 graph = Graph()
23 graph.add_vertex('A')
24 graph.add_vertex('B')
25 graph.add_edge('A', 'B')
26 print(graph)
27 # Output: {'A': ['B'], 'B': ['A']}
28 |
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\lab@demo> & "C:\Program Files\Python314\python.exe" "d:/lab@demo/src/#Task Descr:
{'A': ['B'], 'B': ['A']}
PS D:\lab@demo> |
```

Task Description #7 – Priority Queue

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

Sample Input Code:

```
class PriorityQueue:
    pass
```

```

src > #Task Description #1 – Stack Implementation.py > ...
1  # Priority Queue
2  #Task: Use AI to implement a priority queue using Python's heapq
3  #module.
4  #Sample Input Code:
5  #class PriorityQueue:
6  #pass
7  import heapq
8
9  class PriorityQueue:
10     def __init__(self):
11         self.elements = []
12
13     def is_empty(self):
14         return not self.elements
15
16     def put(self, item, priority):
17         heapq.heappush(self.elements, (priority, item))
18
19     def get(self):
20         return heapq.heappop(self.elements)[1]
21
22     def peek(self):
23         return self.elements[0][1] if self.elements else None
24
25     def size(self):
26         return len(self.elements)
27 # Example usage:
28 if __name__ == "__main__":
29     pq = PriorityQueue()
30     pq.put("task1", priority=2)
31     pq.put("task2", priority=1)
32     pq.put("task3", priority=3)
33
34     print(pq.get()) # Output: task2 (highest priority)
35     print(pq.peek()) # Output: task1 (next highest priority)
36     print(pq.size()) # Output: 2
37     print(pq.get()) # Output: task1

```

Ln 2

```

PS D:\lab@demo> & "C:\Program Files\Python314\python.exe" "d:/lab@demo/src/#Task Descript
task2
task1
2
task1
task3
True
PS D:\lab@demo>

```

Task Description #8 -Deque

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

Sample Input Code:

class DequeDS:

pass

```
src > #Task Description #1 = Stack Implementation.py > ...
1  #- Deque
2  #Task: Use AI to implement a double-ended queue using
3  #collections.deque.
4  #Sample Input Code:
5  #class DequeDS:
6  #pass
7  from collections import deque
8
9  class DequeDS:
10     def __init__(self):
11         self.deque = deque()
12
13     def add_front(self, item):
14         self.deque.appendleft(item)
15
16     def add_rear(self, item):
17         self.deque.append(item)
18
19     def remove_front(self):
20         if not self.is_empty():
21             return self.deque.popleft()
22         else:
23             raise IndexError("Deque is empty")
24
25     def remove_rear(self):
26         if not self.is_empty():
27             return self.deque.pop()
28         else:
29             raise IndexError("Deque is empty")
30
31     def is_empty(self):
32         return len(self.deque) == 0
33
34     def size(self):
35         return len(self.deque)
36 # Example usage:
37 if __name__ == "__main__":
```

```
src > #Task Description #1 – Stack Implementat.py > ...
36 # Example usage:
37 if __name__ == "__main__":
38     deque_ds = DequeDS()
39     deque_ds.add_rear(1)
40     deque_ds.add_rear(2)
41     deque_ds.add_front(0)
42     print(deque_ds.deque) # Output: deque([0, 1, 2])
43     print(deque_ds.remove_front()) # Output: 0
44     print(deque_ds.remove_rear()) # Output: 2
45     print(deque_ds.deque) # Output: deque([1])
46     print(deque_ds.is_empty()) # Output: False
47     print(deque_ds.size()) # Output: 1
48
```

```
PS D:\lab@demo> & "C:\Program Files\Python314\python.exe" "d:/lab@demo/src/#Task Descri
deque([0, 1, 2])
0
2
deque([1])
False
1
PS D:\lab@demo>
```

Task Description #9- Real-Time Application Challenge – Choose the Right Data Structure

Prompt:

"""

Event Registration System

This program manages participants using a hash table (Python dictionary).

It supports:

1. Register participant
2. Search participant
3. Remove participant
4. Display all participants

"""

```
#Task Description #1 – Stack Implementat.py > ...
src > #Task Description #1 – Stack Implementat.py > ...
1  """
2  Event Registration System
3  -----
4  This program manages participants using a hash table (Python dictionary).
5  It supports:
6  1. Register participant
7  2. Search participant
8  3. Remove participant
9  4. Display all participants
10 """
11
12 class EventRegistration:
13     def __init__(self):
14         """Initialize empty participant registry."""
15         self.participants = {} # Hash table
16
17     def register(self, student_id, name):
18         """Add a participant."""
19         if student_id in self.participants:
20             print("Student already registered.")
21         else:
22             self.participants[student_id] = name
23             print("Registration successful.")
24
25     def search(self, student_id):
26         """Search participant by ID."""
27         if student_id in self.participants:
28             print(f"Found: {self.participants[student_id]}")
29         else:
30             print("Participant not found.")
31
32     def remove(self, student_id):
33         """Remove participant."""
34         if student_id in self.participants:
35             del self.participants[student_id]
36             print("Participant removed.")
37         else:
```

```

12 class EventRegistration:
40     def display(self):
41         # -----
45         print("\nRegistered Participants:")
46         for sid, name in self.participants.items():
47             print(sid, "-", name)
48
49
50 # ----- MAIN PROGRAM -----
51 system = EventRegistration()
52
53 while True:
54     print("\n1.Register 2.Search 3.Remove 4.Display 5.Exit")
55     choice = input("Enter choice: ")
56
57     if choice == "1":
58         sid = input("Enter Student ID: ")
59         name = input("Enter Name: ")
60         system.register(sid, name)
61
62     elif choice == "2":
63         sid = input("Enter Student ID to search: ")
64         system.search(sid)
65
66     elif choice == "3":
67         sid = input("Enter Student ID to remove: ")
68         system.remove(sid)
69
70     elif choice == "4":
71         system.display()
72
73     elif choice == "5":
74         print("Exiting system.")
75         break
76
77     else:
78         print("Invalid choice.")
79

```

```

1.Register 2.Search 3.Remove 4.Display 5.Exit
Enter choice: 

```

Task Description #10- Smart E-Commerce Platform – Data Structure Challenge

"""

Order Processing System

This program simulates an e-commerce order processing system using a Queue data structure (FIFO principle).

"""

```
src > #Task Description #1 – Stack Implementat.py > ...
1  from collections import deque
2
3  class OrderQueue:
4      """Queue-based order management system."""
5
6      def __init__(self):
7          """Initialize empty order queue."""
8          self.orders = deque()
9
10     def place_order(self, order_id, customer):
11         """
12         Add a new order to the queue.
13         Args:
14             order_id (str): Unique order ID
15             customer (str): Customer name
16         """
17         self.orders.append((order_id, customer))
18         print(f"Order {order_id} placed successfully.")
19
20     def process_order(self):
21         """
22         Process the next order in queue.
23         Removes and returns first order.
24         """
25         if self.orders:
26             order = self.orders.popleft()
27             print(f"Processing Order {order[0]} for {order[1]}")
28         else:
29             print("No orders to process.")
30
31     def display_orders(self):
32         """Display all pending orders."""
33         if not self.orders:
34             print("No pending orders.")
35         else:
36             print("\nPending Orders:")
37             for order in self.orders:
```

```
35     else:
36         print("\nPending Orders:")
37         for order in self.orders:
38             print(f"Order ID: {order[0]}, Customer: {order[1]}")
39
40
41     # ----- MAIN PROGRAM -----
42
43     queue = OrderQueue()
44
45     while True:
46         print("\n1.Place Order  2.Process Order  3.Display Orders  4.Exit")
47         choice = input("Enter choice: ")
48
49         if choice == "1":
50             oid = input("Enter Order ID: ")
51             name = input("Enter Customer Name: ")
52             queue.place_order(oid, name)
53
54         elif choice == "2":
55             queue.process_order()
56
57         elif choice == "3":
58             queue.display_orders()
59
60         elif choice == "4":
61             print("Exiting system.")
62             break
63
64         else:
65             print("Invalid choice. Try again.")
```

```
1.Place Order  2.Process Order  3.Display Orders  4.Exit
Enter choice: 1
Enter Order ID: 1
Enter Customer Name:
Order 1 placed successfully.
```