

# AI ASSISTED CODING

## LAB-11.1

VIGHNESH BACHWAL

2303A51795

BATCH – 12

### **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
```

**Expected Output:**

- A functional stack implementation with all required methods and docstrings

### **PROMPT:**

```
#Write a code to generate a stack class with push, pop, peek and isEmpty methods
```

**CODE:**

```

• lab6.py > ...
1  #Write a code to generate a stack class with push,pop,peek and isEmpty methods
2  class Stack:
3      def __init__(self):
4          self.stack = []
5      def push(self, item):
6          self.stack.append(item)
7      def pop(self):
8          if not self.isEmpty():
9              return self.stack.pop()
10         else:
11             raise IndexError("Stack is empty")
12     def peek(self):
13         if not self.isEmpty():
14             return self.stack[-1]
15         else:
16             raise IndexError("Stack is empty")
17     def isEmpty(self):
18         return len(self.stack) == 0
19 # Example usage
20 if __name__ == "__main__":
21     stack = Stack()
22     stack.push(1)
23     stack.push(2)
24     stack.push(3)
25     print(stack.peek()) # Output: 3
26     print(stack.pop()) # Output: 3
27     print(stack.isEmpty()) # Output: False
28     print(stack.pop()) # Output: 2
29     print(stack.pop()) # Output: 1
30     print(stack.isEmpty()) # Output: True
31

```

#### OUTPUT:

```

False ...
● PS C:\Users\thota\OneDrive\Desktop\AIAC> & c:/Users/thota/AppData/Local/Programs/Python/Python
/AIAC/lab6.py
3
3
False
2
1
True
○ PS C:\Users\thota\OneDrive\Desktop\AIAC> []

```

#### Task Description #2 – Queue Implementation Task:

Use AI to implement a Queue using Python lists.

**Sample Input Code: class Queue:**

pass

## **Expected Output:**

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

#Write a code to generate a queue class with enqueue,dequeue,peek and size methods **CODE:**

```
⚡ palindrome.py > ...
1  #Write a code to generate a queue class with enqueue,dequeue,peek and size methods
2  class Queue:
3      def __init__(self):
4          self.queue = []
5      def enqueue(self, item):
6          self.queue.append(item)
7      def dequeue(self):
8          if not self.isEmpty():
9              return self.queue.pop(0)
10         else:
11             raise IndexError("Queue is empty")
12     def peek(self):
13         if not self.isEmpty():
14             return self.queue[0]
15         else:
16             raise IndexError("Queue is empty")
17     def size(self):
18         return len(self.queue)
19     def isEmpty(self):
20         return len(self.queue) == 0
21 # Example usage
22 if __name__ == "__main__":
23     queue = Queue()
24     queue.enqueue(1)
25     queue.enqueue(2)
26     queue.enqueue(3)
27     print(queue.peek()) # Output: 1
28     print(queue.dequeue()) # Output: 1
29     print(queue.size()) # Output: 2
30     print(queue.dequeue()) # Output: 2
31     print(queue.dequeue()) # Output: 3
32     print(queue.isEmpty()) # Output: True
```

## **OUTPUT:**

```
22 if __name__ == "__main__":
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python
/AIAC/palindrome.py
1
1
1
2
2
3
True
PS C:\Users\thota\OneDrive\Desktop\AIAC>
```

### 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

**Expected Output:**

- A working linked list implementation with clear method documentation

#### PROMPT:

#Write a code to generate a singly linkedlist with insert and display methods

**CODE AND OUTPUT:**

```

palindrome.py > SinglyLinkedList > display
 2   class Node:
 3     def __init__(self, data):
 4       self.data = data
 5       self.next = None
 6   class SinglyLinkedList:
 7     def __init__(self):
 8       self.head = None
 9     def insert(self, data):
10       new_node = Node(data)
11       if not self.head:
12         self.head = new_node
13         return
14       last_node = self.head
15       while last_node.next:
16         last_node = last_node.next
17       last_node.next = new_node
18     def display(self):
19       current_node = self.head
20       while current_node:
21         print(current_node.data, end=' ')
22         current_node = current_node.next
23       print()
24   # Example usage
25   if __name__ == "__main__":
26     linked_list = SinglyLinkedList()
27     linked_list.insert(10)
28     linked_list.insert(20)
29     linked_list.insert(30)
30     print("Singly Linked List:")
31     linked_list.display()
32   # This program defines a Node class for the elements of the linked

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```

Singly Linked List:
10 20 30
PS C:\Users\thota\OneDrive\Desktop\AIAC> []

```

### 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

**Expected Output:**

- **BST implementation with recursive insert and traversal methods. PROMPT:**

#Write a code to create a binary search tree and inorder traversal methods using recursive insert and traversal methods

**CODE AND OUTPUT:**

The screenshot shows a code editor window with multiple tabs at the top: palindrome.py (closed), lab6.py, lab1exam.py, lab4.py, lab2.py (marked with a '1' indicating changes), and lab5.py. The main area displays the following Python code:

```
palindrome.py > BinarySearchTree > _insert_recursive
1  #Write a code to create a binary search tree and inorder traversal methods using recursive
2  class TreeNode:
3      def __init__(self, value):
4          self.value = value
5          self.left = None
6          self.right = None
7  class BinarySearchTree:
8      def __init__(self):
9          self.root = None
10     def insert(self, value):
11         if self.root is None:
12             self.root = TreeNode(value)
13         else:
14             self._insert_recursive(self.root, value)
15     def _insert_recursive(self, node, value):
16         if value < node.value:
17             if node.left is None:
18                 node.left = TreeNode(value)
19             else:
20                 self._insert_recursive(node.left, value)
21         else:
22             if node.right is None:
23                 node.right = TreeNode(value)
24             else:
25                 self._insert_recursive(node.right, value)
26     def inorder_traversal(self):
27         return self._inorder_recursive(self.root)
28     def _inorder_recursive(self, node):
29         result = []
30         if node:
31             result.extend(self._inorder_recursive(node.left))
32             result.append(node.value)
33             result.extend(self._inorder_recursive(node.right))
34         return result
35 # Example usage
36 if __name__ == "__main__":
37     bst = BinarySearchTree()
```

The screenshot shows a code editor with multiple tabs open, all titled 'palindrome.py'. The active tab displays Python code for a binary search tree. The code defines a class `BinarySearchTree` with an `_inorder_recursive` method that performs an inorder traversal and prints the result. It also includes an example usage section where values 5 through 8 are inserted into the tree, and the resulting inorder traversal output is printed as [2, 3, 4, 5, 6, 7, 8]. Below the code editor, a terminal window is visible with the following output:

```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
True ...
● PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/Users/ /AIAC/palindrome.py
/AIAC/palindrome.py
○ Inorder Traversal: [2, 3, 4, 5, 6, 7, 8]
PS C:\Users\thota\OneDrive\Desktop\AIAC> []
```

## 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
```

**Expected Output:**

- Collision handling using chaining, with wellcommented methods.

### PROMPT:

#Write a code to implement a hash table with basic operations like insert, delete and search methods using chaining for collision handling with well commented methods

### CODE AND OUTPUT:

```

palindrome.py > HashTable > hash_function
1  #Write a code to implement a hash table with basic operations like insert, delete and search methods using chaining
2  class HashTable:
3      def __init__(self, size=10):
4          """Initialize the hash table with a specified size."""
5          self.size = size
6          self.table = [[] for _ in range(size)] # Create a list of empty lists for chaining
7      def hash_function(self, key):
8          """Generate a hash for the given key."""
9          return hash(key) % self.size
10     def insert(self, key, value):
11         """Insert a key-value pair into the hash table."""
12         index = self.hash_function(key)
13         # Check if the key already exists and update it
14         for i, (k, v) in enumerate(self.table[index]):
15             if k == key:
16                 self.table[index][i] = (key, value) # Update existing key
17                 return
18         # If the key does not exist, add a new key-value pair
19         self.table[index].append((key, value))
20     def delete(self, key):
21         """Delete a key-value pair from the hash table."""
22         index = self.hash_function(key)
23         for i, (k, v) in enumerate(self.table[index]):
24             if k == key:
25                 del self.table[index][i] # Remove the key-value pair
26                 return True
27         return False # Key not found
28     def search(self, key):
29         """Search for a value by its key in the hash table."""
30         index = self.hash_function(key)
31         for k, v in self.table[index]:
32             if k == key:
33                 return v # Return the value associated with the key
34

```

```

palindrome.py > HashTable > hash_function
2  class HashTable:
20      def delete(self, key):
21          if k == key:
22              del self.table[index][i] # Remove the key-value pair
23              return True
24          return False # Key not found
25      def search(self, key):
26          """Search for a value by its key in the hash table."""
27          index = self.hash_function(key)
28          for k, v in self.table[index]:
29              if k == key:
30                  return v # Return the value associated with the key
31          return None # Key not found
32
# Example usage
33 if __name__ == "__main__":
34     hash_table = HashTable()
35     hash_table.insert("name", "Alice")
36     hash_table.insert("age", 30)
37     print(hash_table.search("name")) # Output: Alice
38     print(hash_table.search("age")) # Output: 30
39     hash_table.delete("name")
40     print(hash_table.search("name")) # Output: None
41
# This program implements a hash table using chaining for collision handling. It includes methods for

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

PS C:\Users\thota\OneDrive\Desktop\AIAC> ^C

- PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.exe c:/User/ /AIAC/palindrome.py
 

```
Alice
30
None
```

PS C:\Users\thota\OneDrive\Desktop\AIAC> []

## Task Description #6 – Graph Representation

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

**Sample Input Code:** class Graph:

```
pass
```

**Expected Output:**

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

### PROMPT:

#Write a code to implement a graph using an adjacency list and perform methods like add\_vertices,add\_edges and display connections CODE AND OUTPUT:

```
palindrome.py > ...
1  write a code to implement a graph using an adjacency list and perform methods like add_vertices
2  iss Graph:
3  def __init__(self):
4      self.adjacency_list = {}
5  def add_vertex(self, vertex):
6      if vertex not in self.adjacency_list:
7          self.adjacency_list[vertex] = []
8  def add_edge(self, vertex1, vertex2):
9      if vertex1 in self.adjacency_list and vertex2 in self.adjacency_list:
10         self.adjacency_list[vertex1].append(vertex2)
11         self.adjacency_list[vertex2].append(vertex1) # For undirected graph
12  def display_connections(self):
13      for vertex, edges in self.adjacency_list.items():
14          print(f"{vertex}: {', '.join(edges)}")
15 example usage
16 __name__ == "__main__":
17 graph = Graph()
18 graph.add_vertex("A")
19 graph.add_vertex("B")
20 graph.add_vertex("C")
21 graph.add_edge("A", "B")
22 graph.add_edge("A", "C")
23 graph.add_edge(["B", "C"])
24 graph.display_connections()
25
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

/AIAC/palindrome.py

● PS C:\Users\thota\OneDrive\Desktop\AIAC & C:/Users/thota/AppData/Local/Programs/Python/Python313/python.

/AIAC/palindrome.py

A: B, C

B: A, C

C: A, B

○ PS C:\Users\thota\OneDrive\Desktop\AIAC> []

## Task Description #7 – Priority Queue

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

**Sample Input Code: class PriorityQueue:**

```
palindrome.py X lab6.py lab1exam.py lab4.py lab2.py 1 lab5.py
palindrome.py > PriorityQueue > is_empty
1 #Write a code to implement a priority queue using python's heapq module and implement
2 import heapq
3 class PriorityQueue:
4     def __init__(self):
5         self.elements = []
6     def enqueue(self, item, priority):
7         heapq.heappush(self.elements, (priority, item))
8     def dequeue(self):
9         if not self.is_empty():
10            return heapq.heappop(self.elements)[1]
11        else:
12            raise IndexError("Priority Queue is empty")
13    def display(self):
14        print("Priority Queue:")
15        for priority, item in sorted(self.elements):
16            print(f"Item: {item}, Priority: {priority}")
17    def is_empty(self):
18        return len(self.elements) == 0
19 # Example usage
20 if __name__ == "__main__":
21    pq = PriorityQueue()
22    pq.enqueue("Task 1", priority=3)
23    pq.enqueue("Task 2", priority=1)
24    pq.enqueue("Task 3", priority=2)
25    pq.display()
```

```
palindrome.py X lab6.py lab1exam.py lab4.py lab2.py 1 lab5.py
palindrome.py > PriorityQueue > is_empty
1 #Write a code to implement a priority queue using python's heapq module and implement
2 import heapq
3 class PriorityQueue:
4     def __init__(self):
5         self.elements = []
6     def enqueue(self, item, priority):
7         heapq.heappush(self.elements, (priority, item))
8     def dequeue(self):
9         if not self.is_empty():
10            return heapq.heappop(self.elements)[1]
11        else:
12            raise IndexError("Priority Queue is empty")
13    def display(self):
14        print("Priority Queue:")
15        for priority, item in sorted(self.elements):
16            print(f"Item: {item}, Priority: {priority}")
17    def is_empty(self):
18        return len(self.elements) == 0
19 # Example usage
20 if __name__ == "__main__":
21    pq = PriorityQueue()
22    pq.enqueue("Task 1", priority=3)
23    pq.enqueue("Task 2", priority=1)
24    pq.enqueue("Task 3", priority=2)
25    pq.display()
```

## Task Description #8 – Deque

**Task:** Use AI to implement a double-ended queue using collections.deque. Sample Input  
**Code:** class DequeDS:

pass

**Expected Output:**

- Insert and remove from both ends with docstrings.

### PROMPT:

#Write a code to implement a double ended queue using collections.deque using insert and remove from both ends with docstring

### CODE AND OUTPUT:

```
❸ palindrome.py > ↵ DoubleEndedQueue > ↵ is_empty
1  #Write a code to implement a double ended queue using collections.deque using insert and remove from both ends with docstr
2  from collections import deque
3  class DoubleEndedQueue:
4      def __init__(self):
5          """Initialize an empty double-ended queue."""
6          self.deque = deque()
7      def insert_front(self, item):
8          """Insert an item at the front of the deque."""
9          self.deque.appendleft(item)
10     def insert_rear(self, item):
11         """Insert an item at the rear of the deque."""
12         self.deque.append(item)
13     def remove_front(self):
14         """Remove and return an item from the front of the deque. Raises IndexError if the deque is empty."""
15         if not self.is_empty():
16             return self.deque.popleft()
17         else:
18             raise IndexError("Deque is empty")
19     def remove_rear(self):
20         """Remove and return an item from the rear of the deque. Raises IndexError if the deque is empty."""
21         if not self.is_empty():
22             return self.deque.pop()
23         else:
24             raise IndexError("Deque is empty")
25     def is_empty(self):
26         """Check if the deque is empty."""
27         return len(self.deque) == 0
28     # Example usage
29     if name == " main ":
```

```
3   class DoubleEndedQueue:
19      def remove_rear(self):
22          return self.deque.pop()
23      else:
24          raise IndexError("Deque is empty")
25      def is_empty(self):
26          """Check if the deque is empty."""
27          return len(self.deque) == 0
28  # Example usage
29  if __name__ == "__main__":
30      deque = DoubleEndedQueue()
31      deque.insert_rear(1)
32      deque.insert_rear(2)
33      deque.insert_front(0)
34      print(deque.deque)  # Output: deque([0, 1, 2])
35      print(deque.remove_front())  # Output: 0
36      print(deque.remove_rear())  # Output: 2
37      print(deque.is_empty())  # Output: False
38      print(deque.remove_front())  # Output: 1
39      print(deque.is_empty())  # Output: True
40  # This code implements a double-ended queue (deque) using the collections.d
41
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

/AIAC/palindrome.py

1

True

○ PS C:\Users\thota\OneDrive\Desktop\AIAC> ^C

● PS C:\Users\thota\OneDrive\Desktop\AIAC> & C:/Users/thota/AppData/Local/Programs/Python /AIAC/palindrome.py

deque([0, 1, 2])

0

2

False

1

True

○ PS C:\Users\thota\OneDrive\Desktop\AIAC> □