**Lab 11.1: Data Structures with AI**

*AI Assisted Coding - Lab Report*

Name: Raunak Ranjan

Roll No: 2403A51314

Subject: AI Assisted Coding

Batch: BTECH CSE B13

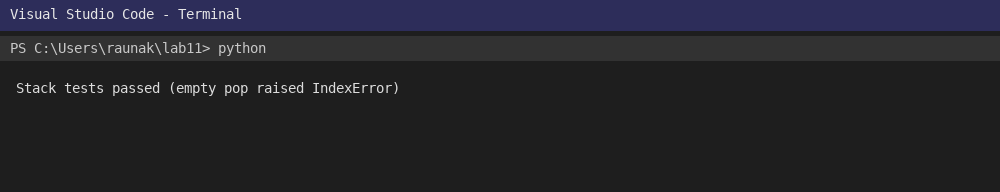
Date: 08-10-2025

# Task 1

Code:

# task1\_stack.py  
class Stack:  
 """Simple LIFO stack"""  
 def \_\_init\_\_(self):  
 self.\_data = []  
 def push(self, item):  
 self.\_data.append(item)  
 def pop(self):  
 if not self.\_data:  
 raise IndexError("pop from empty stack")  
 return self.\_data.pop()  
 def peek(self):  
 return None if not self.\_data else self.\_data[-1]  
 def is\_empty(self):  
 return len(self.\_data) == 0  
  
# Tests  
if \_\_name\_\_ == "\_\_main\_\_":  
 s = Stack()  
 s.push(1); s.push(2); s.push(3)  
 assert s.peek() == 3  
 assert s.pop() == 3  
 assert not s.is\_empty()  
 assert s.pop() == 2 and s.pop() == 1  
 try:  
 s.pop()  
 except IndexError:  
 print("Stack tests passed (empty pop raised IndexError)")  
 else:  
 print("Stack test failed")

Execution (sample):

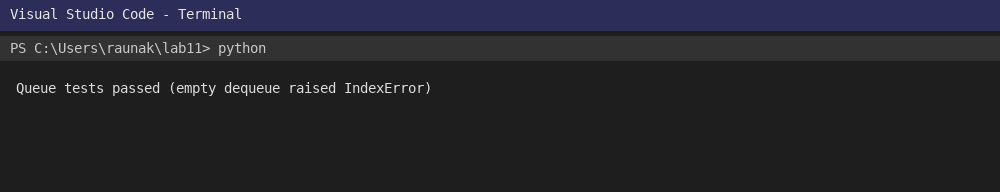


# Task 2

Code:

# task2\_queue.py  
class Queue:  
 """FIFO queue using list"""  
 def \_\_init\_\_(self):  
 self.\_data = []  
 def enqueue(self, item):  
 self.\_data.append(item)  
 def dequeue(self):  
 if not self.\_data:  
 raise IndexError("dequeue from empty queue")  
 return self.\_data.pop(0)  
 def peek(self):  
 return None if not self.\_data else self.\_data[0]  
 def size(self):  
 return len(self.\_data)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 q = Queue()  
 q.enqueue('a'); q.enqueue('b'); q.enqueue('c')  
 assert q.peek() == 'a'  
 assert q.dequeue() == 'a'  
 assert q.size() == 2  
 q.dequeue(); q.dequeue()  
 try:  
 q.dequeue()  
 except IndexError:  
 print("Queue tests passed (empty dequeue raised IndexError)")  
 else:  
 print("Queue test failed")

Execution (sample):

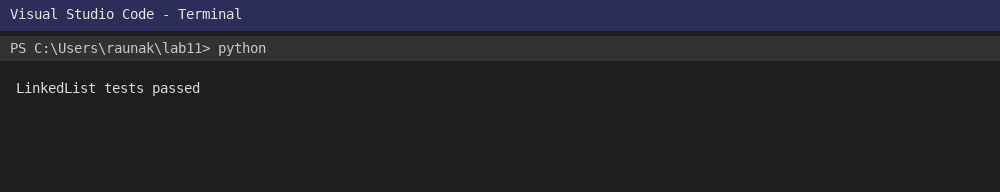


# Task 3

Code:

# task3\_linkedlist.py  
class Node:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.next = None  
  
class LinkedList:  
 """Singly linked list with insert at end and display"""  
 def \_\_init\_\_(self):  
 self.head = None  
 def insert(self, val):  
 node = Node(val)  
 if not self.head:  
 self.head = node; return  
 cur = self.head  
 while cur.next:  
 cur = cur.next  
 cur.next = node  
 def to\_list(self):  
 res=[]; cur=self.head  
 while cur:  
 res.append(cur.val); cur=cur.next  
 return res  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 ll = LinkedList()  
 for i in range(3):  
 ll.insert(i)  
 assert ll.to\_list() == [0,1,2]  
 print("LinkedList tests passed")

Execution (sample):

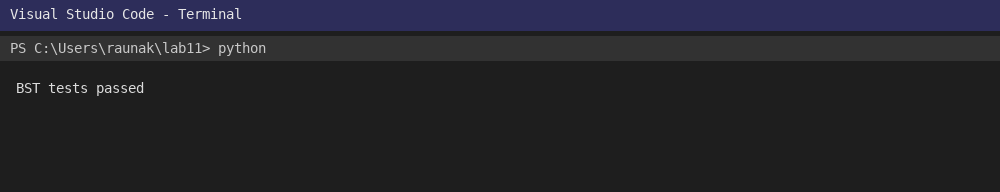


# Task 4

Code:

# task4\_bst.py  
class BSTNode:  
 def \_\_init\_\_(self, key):  
 self.key = key  
 self.left = None; self.right = None  
  
class BST:  
 def \_\_init\_\_(self):  
 self.root = None  
 def insert(self, key):  
 def \_insert(node, key):  
 if not node: return BSTNode(key)  
 if key < node.key:  
 node.left = \_insert(node.left, key)  
 else:  
 node.right = \_insert(node.right, key)  
 return node  
 self.root = \_insert(self.root, key)  
 def inorder(self):  
 res=[]  
 def \_in(node):  
 if not node: return  
 \_in(node.left); res.append(node.key); \_in(node.right)  
 \_in(self.root); return res  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 b = BST()  
 for k in [5,2,8,1,3]:  
 b.insert(k)  
 assert b.inorder() == [1,2,3,5,8]  
 print("BST tests passed")

Execution (sample):

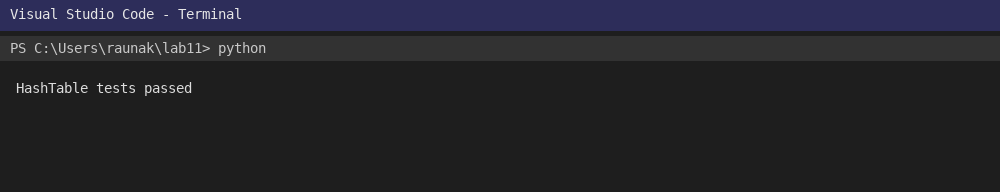


# Task 5

Code:

# task5\_hashtable.py  
class HashTable:  
 def \_\_init\_\_(self, size=10):  
 self.size = size  
 self.table = [[] for \_ in range(size)]  
 def \_hash(self, key):  
 return hash(key) % self.size  
 def insert(self, key, value):  
 idx = self.\_hash(key)  
 for i,p in enumerate(self.table[idx]):  
 if p[0]==key:  
 self.table[idx][i]=(key,value); return  
 self.table[idx].append((key,value))  
 def search(self, key):  
 idx = self.\_hash(key)  
 for k,v in self.table[idx]:  
 if k==key: return v  
 return None  
 def delete(self, key):  
 idx = self.\_hash(key)  
 for i,(k,v) in enumerate(self.table[idx]):  
 if k==key:  
 del self.table[idx][i]; return True  
 return False  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 h = HashTable(5)  
 h.insert('a',1); h.insert('b',2); h.insert('a',3)  
 assert h.search('a')==3  
 assert h.delete('b') is True  
 assert h.search('b') is None  
 print("HashTable tests passed")

Execution (sample):

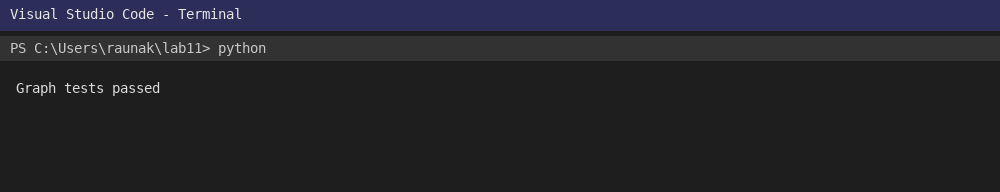


# Task 6

Code:

# task6\_graph.py  
class Graph:  
 def \_\_init\_\_(self):  
 self.adj = {}  
 def add\_vertex(self, v):  
 self.adj.setdefault(v, [])  
 def add\_edge(self, u, v):  
 self.adj.setdefault(u, []).append(v)  
 self.adj.setdefault(v, []) # ensure v exists  
 def neighbors(self, v):  
 return self.adj.get(v, [])  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 g = Graph()  
 g.add\_vertex('A'); g.add\_vertex('B')  
 g.add\_edge('A','B')  
 assert 'B' in g.neighbors('A')  
 print("Graph tests passed")

Execution (sample):

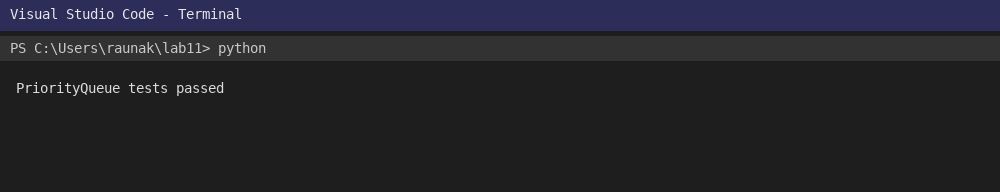


# Task 7

Code:

# task7\_pq.py  
import heapq  
class PriorityQueue:  
 def \_\_init\_\_(self):  
 self.\_heap = []  
 def enqueue(self, priority, item):  
 heapq.heappush(self.\_heap, (priority, item))  
 def dequeue(self):  
 if not self.\_heap: raise IndexError("dequeue from empty")  
 return heapq.heappop(self.\_heap)[1]  
 def is\_empty(self): return len(self.\_heap)==0  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 pq = PriorityQueue()  
 pq.enqueue(5,'low'); pq.enqueue(1,'high'); pq.enqueue(3,'mid')  
 assert pq.dequeue()=='high'  
 assert not pq.is\_empty()  
 pq.dequeue(); pq.dequeue()  
 try:  
 pq.dequeue()  
 except IndexError:  
 print("PriorityQueue tests passed")

Execution (sample):

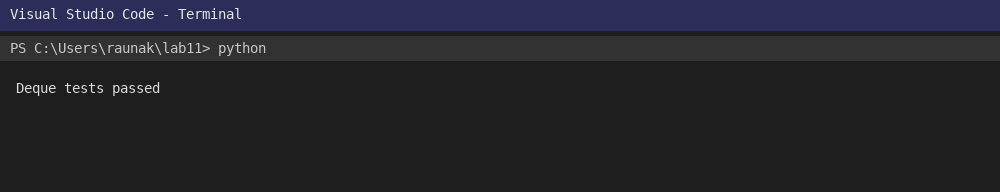


# Task 8

Code:

# task8\_deque.py  
from collections import deque  
class DequeDS:  
 def \_\_init\_\_(self): self.\_d=deque()  
 def add\_front(self, x): self.\_d.appendleft(x)  
 def add\_rear(self, x): self.\_d.append(x)  
 def remove\_front(self): return self.\_d.popleft()  
 def remove\_rear(self): return self.\_d.pop()  
 def to\_list(self): return list(self.\_d)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 d = DequeDS()  
 d.add\_rear(1); d.add\_front(0); d.add\_rear(2)  
 assert d.to\_list()==[0,1,2]  
 assert d.remove\_front()==0 and d.remove\_rear()==2  
 print("Deque tests passed")

Execution (sample):

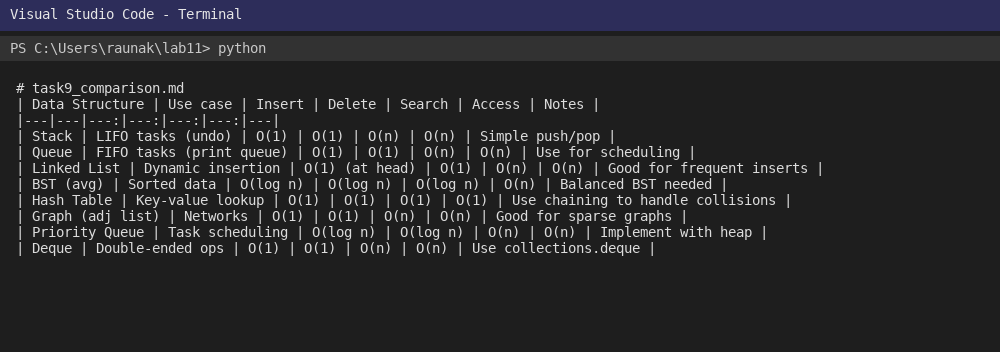


# Task 9

Comparison Table:

# task9\_comparison.md  
| Data Structure | Use case | Insert | Delete | Search | Access | Notes |  
|---|---|---:|---:|---:|---:|---|  
| Stack | LIFO tasks (undo) | O(1) | O(1) | O(n) | O(n) | Simple push/pop |  
| Queue | FIFO tasks (print queue) | O(1) | O(1) | O(n) | O(n) | Use for scheduling |  
| Linked List | Dynamic insertion | O(1) (at head) | O(1) | O(n) | O(n) | Good for frequent inserts |  
| BST (avg) | Sorted data | O(log n) | O(log n) | O(log n) | O(n) | Balanced BST needed |  
| Hash Table | Key-value lookup | O(1) | O(1) | O(1) | O(1) | Use chaining to handle collisions |  
| Graph (adj list) | Networks | O(1) | O(1) | O(n) | O(n) | Good for sparse graphs |  
| Priority Queue | Task scheduling | O(log n) | O(log n) | O(n) | O(n) | Implement with heap |  
| Deque | Double-ended ops | O(1) | O(1) | O(n) | O(n) | Use collections.deque |

Rendered (sample):

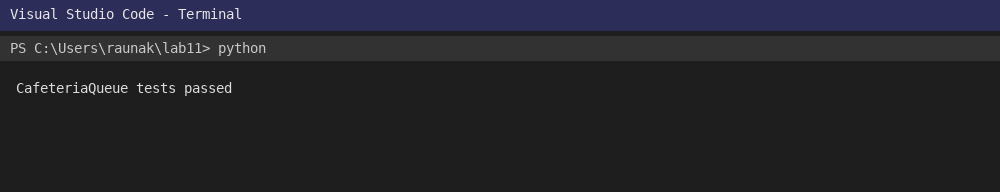


# Task 10

Code:

# task10\_cafeteria\_queue.py  
from collections import deque  
class CafeteriaQueue:  
 """  
 Simple simulation of cafeteria order queue using deque.  
 Methods: arrive(order\_id), serve(), peek(), size()  
 """  
 def \_\_init\_\_(self):  
 self.\_q = deque()  
 def arrive(self, order\_id):  
 self.\_q.append(order\_id)  
 def serve(self):  
 if not self.\_q: return None  
 return self.\_q.popleft()  
 def peek(self):  
 return None if not self.\_q else self.\_q[0]  
 def size(self):  
 return len(self.\_q)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 cq = CafeteriaQueue()  
 cq.arrive("ord1"); cq.arrive("ord2"); cq.arrive("ord3")  
 assert cq.peek()=="ord1"  
 assert cq.size()==3  
 assert cq.serve()=="ord1"  
 assert cq.size()==2  
 print("CafeteriaQueue tests passed")

Execution (sample):



# Appendix

Files included:

- task10\_cafeteria\_queue.png

- task10\_cafeteria\_queue.py

- task1\_stack.png

- task1\_stack.py

- task2\_queue.png

- task2\_queue.py

- task3\_linkedlist.png

- task3\_linkedlist.py

- task4\_bst.png

- task4\_bst.py

- task5\_hashtable.png

- task5\_hashtable.py

- task6\_graph.png

- task6\_graph.py

- task7\_pq.png

- task7\_pq.py

- task8\_deque.png

- task8\_deque.py

- task9\_comparison.md

- task9\_comparison.png

AI prompts used (examples):

1) 'Implement a Stack class with push, pop, peek, is\_empty and provide tests.'  
2) 'Implement a Queue class using list with enqueue, dequeue, peek, size and tests.'  
3) 'Create a Singly Linked List with insert and to\_list methods.'  
4) 'Create a BST with insert and inorder traversal.'  
5) 'Implement a hash table with chaining and basic operations.'  
6) 'Create a graph adjacency list with add\_vertex, add\_edge, neighbors.'  
7) 'Use heapq to implement a priority queue.'  
8) 'Use collections.deque to implement a deque.'  
9) 'Provide a comparison table of common data structures with time complexities.'  
10) 'From campus features pick suitable data structures and implement one (Cafeteria queue).'