AI Assisted Coding (III Year)
Assignment

Name: B.Mahanand

HT NO: 2303A52462

Batch: 35

Assignment Number: 11.5 / 24

Lab 11 – Data Structures with AI: Implementing Fundamental Structures

Week 6 – Friday

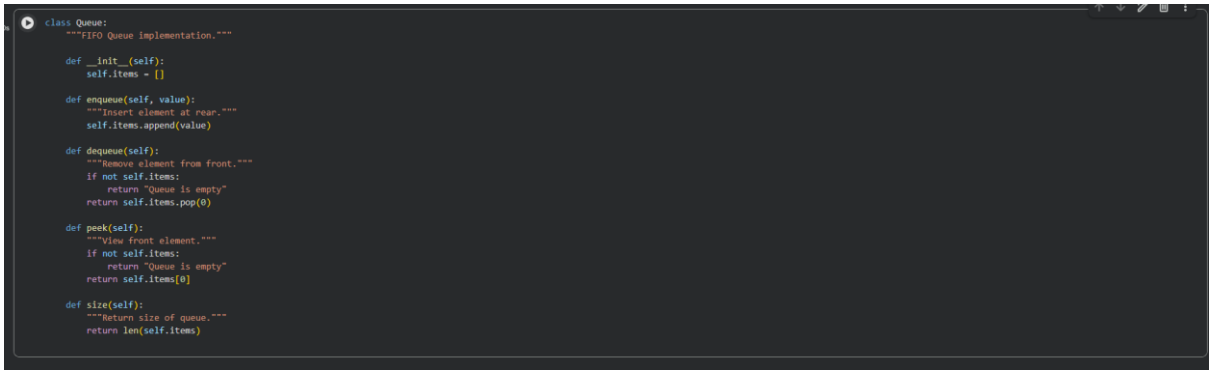
Lab Objectives

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

Task #1 – Stack Implementation

```
class Stack:
    """Implementation of Stack using Python list."""

    def __init__(self):
        self.items = []

    def push(self, value):
        """Add an element to the top of the stack."""
        self.items.append(value)

    def pop(self):
        """Remove and return the top element."""
        if self.is_empty():
            return "Stack is empty"
        return self.items.pop()

    def peek(self):
        """Return the top element without removing it."""
        if self.is_empty():
            return "Stack is empty"
        return self.items[-1]

    def is_empty(self):
        """Check whether stack is empty."""
        return len(self.items) == 0
```

Task #2 – Queue Implementation



```python
class Queue:
    """FIFO Queue implementation."""

    def __init__(self):
        self.items = []

    def enqueue(self, value):
        """Insert element at rear."""
        self.items.append(value)

    def dequeue(self):
        """Remove element from front."""
        if not self.items:
            return "Queue is empty"
        return self.items.pop(0)

    def peek(self):
        """View front element."""
        if not self.items:
            return "Queue is empty"
        return self.items[0]

    def size(self):
        """Return size of queue."""
        return len(self.items)
```

Task #3 – Singly Linked List

```python
class Node:
    """Node of a singly linked list."""

    def __init__(self, data):
        self.data = data
        self.next = None


class LinkedList:
    """Singly linked list implementation."""

    def __init__(self):
        self.head = None

    def insert(self, value):
        """Insert at end."""
        new_node = Node(value)
        if not self.head:
            self.head = new_node
            return

        temp = self.head
        while temp.next:
            temp = temp.next
        temp.next = new_node

    def display(self):
        """Display linked list."""
        temp = self.head
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
```

Task #4 – Hash Table

```python
class HashTable:
    """Hash table using chaining."""

    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):
        """Insert key-value."""
        index = self._hash(key)
        for pair in self.table[index]:
            if pair[0] == key:
                pair[1] = value
                return
        self.table[index].append([key, value])

    def search(self, key):
        """Search value."""
        index = self._hash(key)
        for pair in self.table[index]:
            if pair[0] == key:
                return pair[1]
        return None

    def delete(self, key):
        """Delete key."""
        index = self._hash(key)
        for i, pair in enumerate(self.table[index]):
            if pair[0] == key:
                self.table[index].pop(i)
                return
```

Task #5 – Graph (Adjacency List)

```python
class Graph:
    """Graph representation."""

    def __init__(self):
        self.graph = {}

    def add_vertex(self, vertex):
        if vertex not in self.graph:
            self.graph[vertex] = []

    def add_edge(self, v1, v2):
        self.graph.setdefault(v1, []).append(v2)

    def display(self):
        for v in self.graph:
            print(v, "->", self.graph[v])
```

Task #6 – Smart Hospital Management System

Feature Mapping

Feature	Data Structure	Justification
Patient Check-In	Queue	Patients are handled in arrival order which ensures fairness and efficiency.
Emergency Cases	Priority Queue	Critical patients must be treated first. This allows priority-based processing.
Medical Records	Hash Table	Provides fast search and update using patient ID.
Appointment Scheduling	Binary Search Tree	Appointments are sorted by time, making retrieval faster.
Room Navigation	Graph	Hospital structure is a network of connected locations.

Implementation – Emergency Case Handling

```python
import heapq


class EmergencyQueue:
    """Priority-based emergency patient system."""

    def __init__(self):
        self.heap = []

    def add_patient(self, name, priority):
        """Add patient with priority."""
        heapq.heappush(self.heap, (priority, name))

    def treat_patient(self):
        """Treat highest priority patient."""
        if not self.heap:
            return "No patients"
        return heapq.heappop(self.heap)[1]


# Example
eq = EmergencyQueue()
eq.add_patient("Ravi", 2)
eq.add_patient("Anita", 1)
print("Treated:", eq.treat_patient())
```

```
Treated: Anita
```

Task #7 – Smart City Traffic Control System

Feature Mapping

Feature	Data Structure	Justification
Traffic Signal	Queue	Vehicles pass in order of arrival.
Emergency Priority	Priority Queue	Ambulances and fire trucks handled first.
Vehicle Lookup	Hash Table	Fast access using registration number.
Road Network	Graph	Roads and intersections form a connected network.
Parking Slots	Deque	Efficient insertion and removal from both ends.

Implementation – Vehicle Lookup

```python
class VehicleRegistry:
    """Vehicle registration lookup."""

    def __init__(self):
        self.vehicles = {}

    def register(self, number, owner):
        """Register vehicle."""
        self.vehicles[number] = owner

    def search(self, number):
        """Find vehicle owner."""
        return self.vehicles.get(number, "Not found")

    def delete(self, number):
        """Remove vehicle."""
        if number in self.vehicles:
            del self.vehicles[number]


# Example
vr = VehicleRegistry()
vr.register("AP09AB1234", "Akshay")
print(vr.search("AP09AB1234"))
```

```
Akshay
```

Task #8 – Smart E-Commerce Platform

Feature	Data Structure	Justification
Shopping Cart	Linked List	Dynamic insertion and deletion of products.
Order Processing	Queue	Orders processed sequentially.
Top Products	Priority Queue	Ranking based on sales.
Product Search	Hash Table	Fast lookup by ID.
Delivery Routes	Graph	Efficient route representation.

Implementation – Product Search using Hash Table

```
class ProductSearch:
    """Fast lookup of products."""

    def __init__(self):
        self.products = {}

    def add_product(self, product_id, name):
        """Add product."""
        self.products[product_id] = name

    def search_product(self, product_id):
        """Search by ID."""
        return self.products.get(product_id, "Not found")

    def remove_product(self, product_id):
        """Delete product."""
        if product_id in self.products:
            del self.products[product_id]


# Example
ps = ProductSearch()
ps.add_product(101, "Laptop")
ps.add_product(102, "Mobile")
print(ps.search_product(101))
```

```
Laptop
```

Conclusion

This lab demonstrated how AI-assisted development improves data structure implementation, readability, performance, and real-world system design.