Subset 8 – Data Structures with AI for Student Records

NAME: J.ABHIRAM **END EXAM:AI-ASSISTED-CODING**

BATCH:06

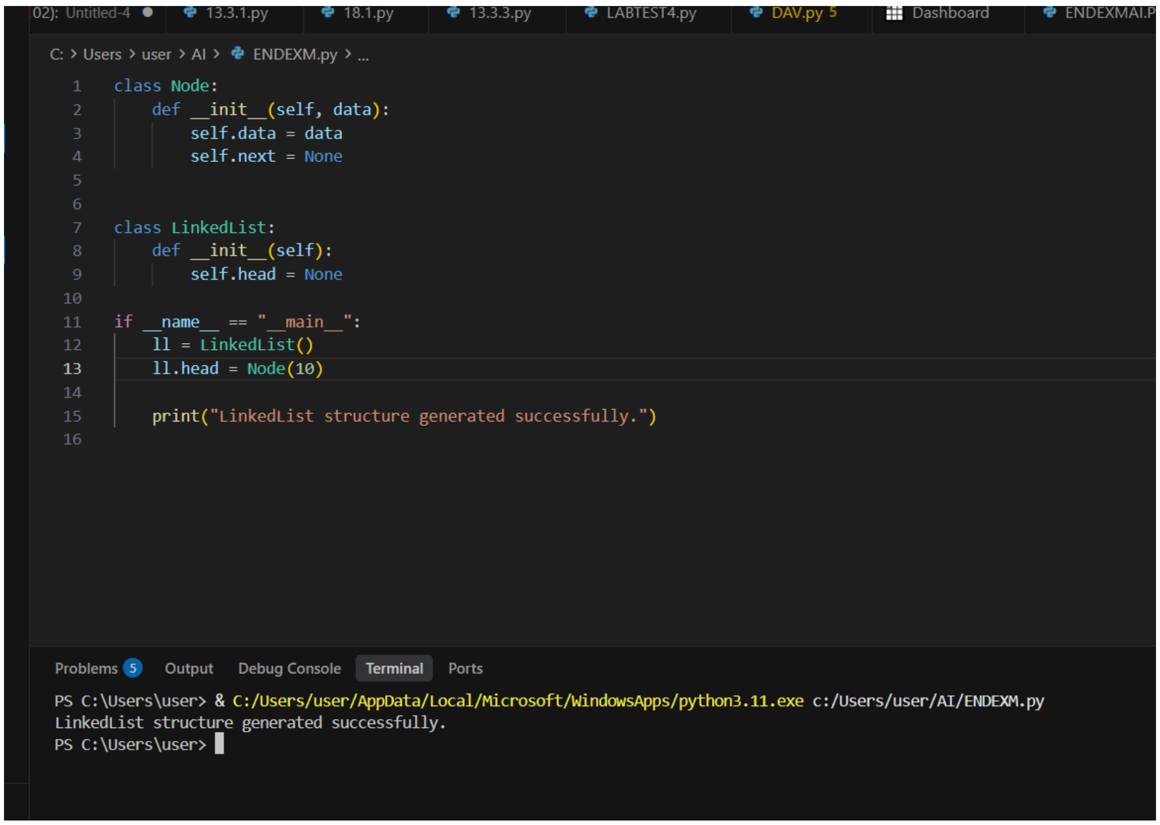
ROLLNO:2403A51342

Q1 – Task 1: Linked List Class Structure

Prompt Used:

- Generate Python class for Node and LinkedList.
- Include head initialization.
- Provide extendable DS foundation.

Code and Output:



```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class LinkedList:
    def __init__(self):
        self.head = None

if __name__ == "__main__":
    ll = LinkedList()
    ll.head = Node(10)

    print("LinkedList structure generated successfully.")
```

```
PS C:\Users\user> & C:/Users/user/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/user/AI/ENDEXM.py
LinkedList structure generated successfully.
PS C:\Users\user>
```

Observation:

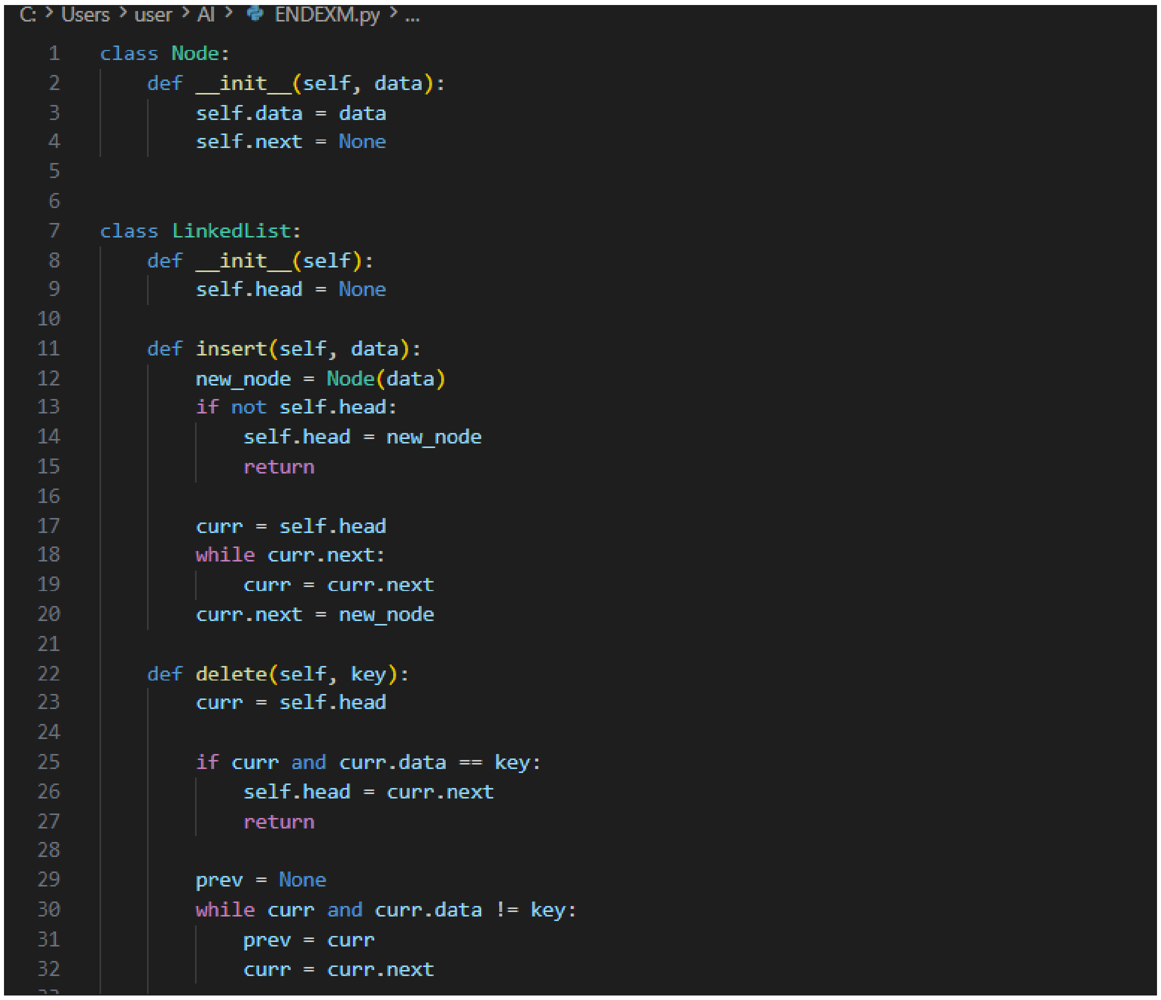
- Node and LinkedList classes initialized.
- Ready for adding operations.
- Memory-efficient for dynamic records.

Q1 – Task 2: Insert, Delete, Search Methods

Prompt Used:

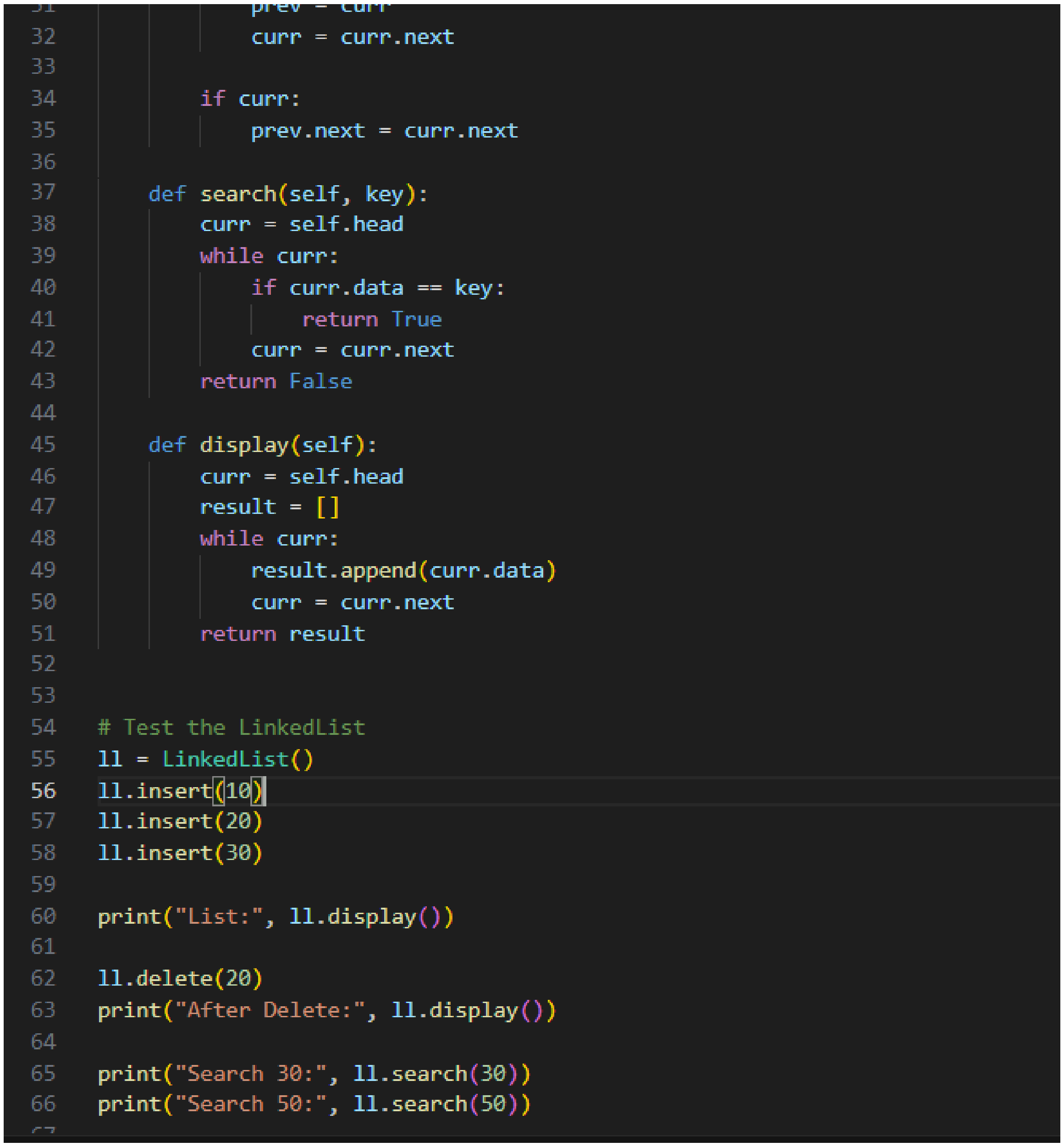
- Add insert(), delete(), search() methods.
- Traverse list and handle edge cases.
- Maintain readability.

Code and Output:

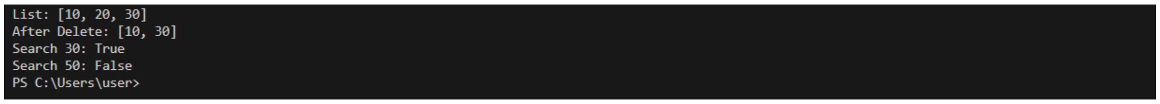


C: > Users > user > AI > ENDEXM.py > ...

```python
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


class LinkedList:
    def __init__(self):
        self.head = None

    def insert(self, data):
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return

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

    def delete(self, key):
        curr = self.head

        if curr and curr.data == key:
            self.head = curr.next
            return

        prev = None
        while curr and curr.data != key:
            prev = curr
            curr = curr.next
```



```python
            prev = curr
            curr = curr.next

        if curr:
            prev.next = curr.next

    def search(self, key):
        curr = self.head
        while curr:
            if curr.data == key:
                return True
            curr = curr.next
        return False

    def display(self):
        curr = self.head
        result = []
        while curr:
            result.append(curr.data)
            curr = curr.next
        return result


# Test the LinkedList
ll = LinkedList()
ll.insert(10)
ll.insert(20)
ll.insert(30)

print("List:", ll.display())

ll.delete(20)
print("After Delete:", ll.display())

print("Search 30:", ll.search(30))
print("Search 50:", ll.search(50))
```



```
List: [10, 20, 30]
After Delete: [10, 30]
Search 30: True
Search 50: False
PS C:\Users\user>
```

Observation:

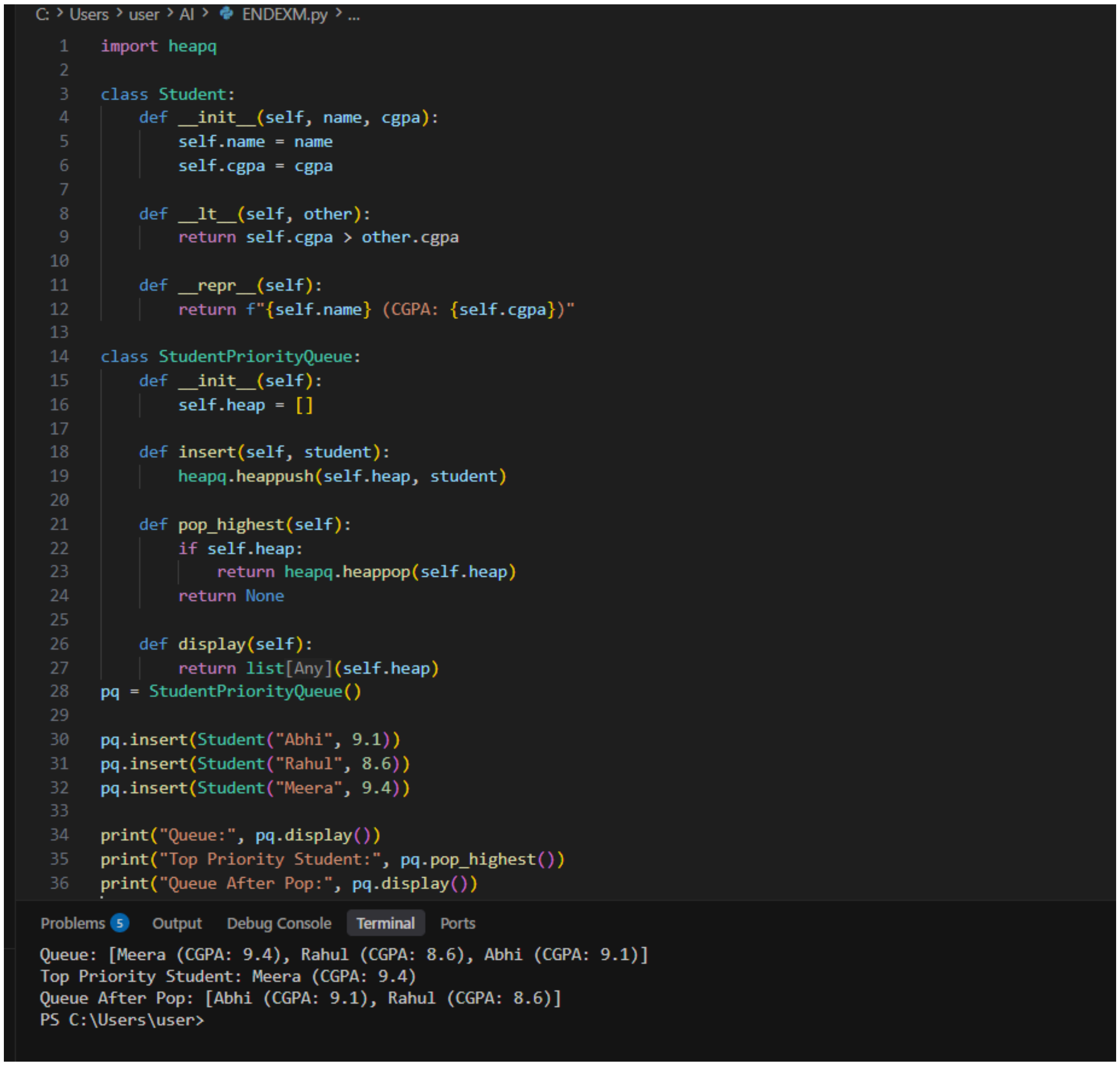
- Insert adds values correctly.
- Delete works for head and middle nodes.
- Search performs linear lookup.
- Linked list operations validated.

Q2 –Student Priority Queue Logic

Prompt Used:

- Use heapq for priority queue.
- Highest CGPA = highest priority.
- Override __lt__ for max-heap behavior.

Code and output:



```python
import heapq

class Student:
    def __init__(self, name, cgpa):
        self.name = name
        self.cgpa = cgpa

    def __lt__(self, other):
        return self.cgpa > other.cgpa

    def __repr__(self):
        return f"{self.name} (CGPA: {self.cgpa})"

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

    def insert(self, student):
        heapq.heappush(self.heap, student)

    def pop_highest(self):
        if self.heap:
            return heapq.heappop(self.heap)
        return None

    def display(self):
        return list[Any](self.heap)
pq = StudentPriorityQueue()

pq.insert(Student("Abhi", 9.1))
pq.insert(Student("Rahul", 8.6))
pq.insert(Student("Meera", 9.4))

print("Queue:", pq.display())
print("Top Priority Student:", pq.pop_highest())
print("Queue After Pop:", pq.display())
```

```
Queue: [Meera (CGPA: 9.4), Rahul (CGPA: 8.6), Abhi (CGPA: 9.1)]
Top Priority Student: Meera (CGPA: 9.4)
Queue After Pop: [Abhi (CGPA: 9.1), Rahul (CGPA: 8.6)]
PS C:\Users\user>
```

Observation:

- Highest CGPA student popped first.
- Queue updates automatically.
- Demonstrates correct heap-based priority behavior.