

Subset 8 – Data Structures with AI for Student Records

NAME: J.ABHIRAM

END EXAM:AI-ASSISTED-CODING

BATCH:06

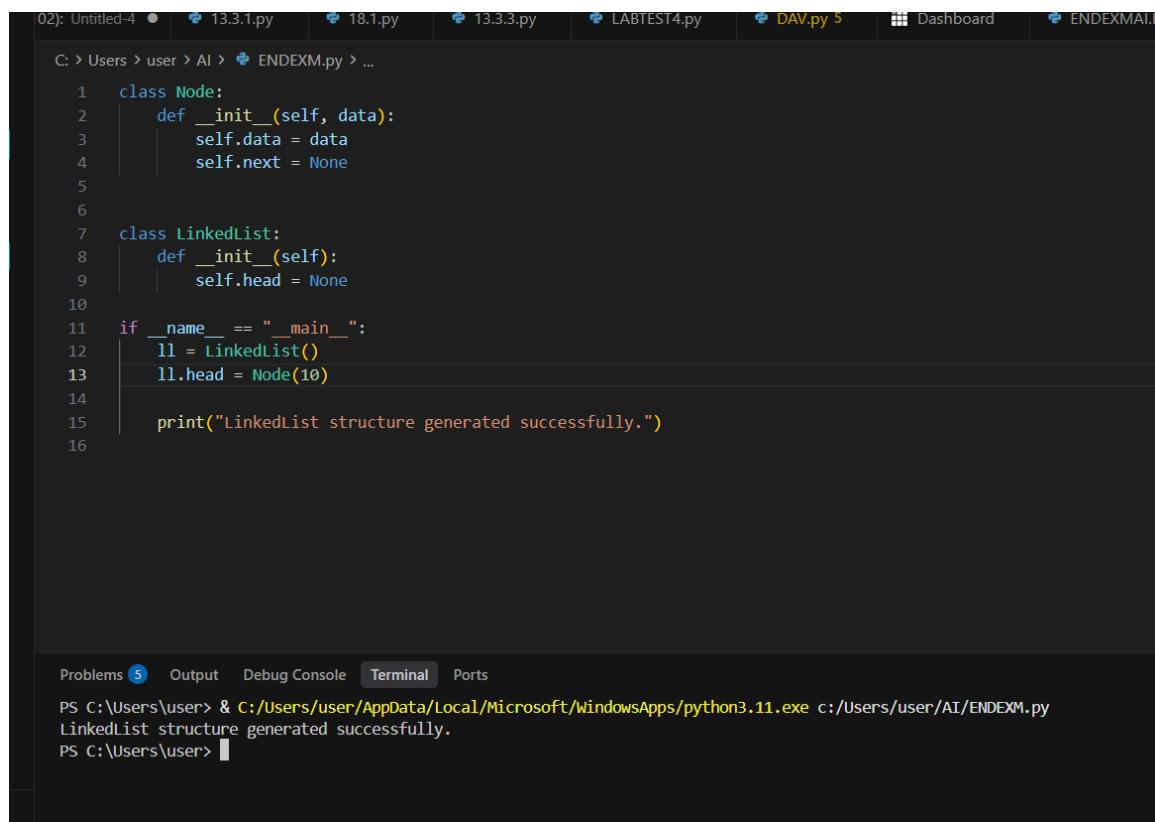
ROLLNO:2403A51342

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:



The screenshot shows a code editor interface with a dark theme. The main area displays a Python script named `ENDEXM.py`. The code defines two classes: `Node` and `LinkedList`. The `Node` class has an `__init__` method that initializes `data` and `next` attributes. The `LinkedList` class has an `__init__` method that initializes `head`. If the script is run (`__name__ == "__main__"`), it creates a `LinkedList` object with a single node containing the value 10, and prints a success message. Below the code editor, a terminal window shows the command `python3.11.exe c:/Users/user/AI/ENDEXM.py` being run, followed by the output "LinkedList structure generated successfully."

```
02: Untitled-4 ● 13.3.1.py 18.1.py 13.3.3.py LABTEST4.py DAV.py 5 Dashboard ENDEXMAI.P
C: > Users > user > AI > ENDEXM.py > ...
1  class Node:
2      def __init__(self, data):
3          self.data = data
4          self.next = None
5
6
7  class LinkedList:
8      def __init__(self):
9          self.head = None
10
11 if __name__ == "__main__":
12     ll = LinkedList()
13     ll.head = Node(10)
14
15     print("LinkedList structure generated successfully.")
16

Problems 5 Output Debug Console Terminal Ports
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 > ...
1  class Node:
2      def __init__(self, data):
3          self.data = data
4          self.next = None
5
6
7  class LinkedList:
8      def __init__(self):
9          self.head = None
10
11     def insert(self, data):
12         new_node = Node(data)
13         if not self.head:
14             self.head = new_node
15             return
16
17         curr = self.head
18         while curr.next:
19             curr = curr.next
20         curr.next = new_node
21
22     def delete(self, key):
23         curr = self.head
24
25         if curr and curr.data == key:
26             self.head = curr.next
27             return
28
29         prev = None
30         while curr and curr.data != key:
31             prev = curr
32             curr = curr.next
```

```

51     prev = curr
52     curr = curr.next
53
54     if curr:
55         prev.next = curr.next
56
57     def search(self, key):
58         curr = self.head
59         while curr:
60             if curr.data == key:
61                 return True
62             curr = curr.next
63
64     return False
65
66     def display(self):
67         curr = self.head
68         result = []
69         while curr:
70             result.append(curr.data)
71             curr = curr.next
72
73         return result
74
75
76     # Test the LinkedList
77     ll = LinkedList()
78     ll.insert(10)
79     ll.insert(20)
80     ll.insert(30)
81
82     print("List:", ll.display())
83
84     ll.delete(20)
85     print("After Delete:", ll.display())
86
87     print("Search 30:", ll.search(30))
88     print("Search 50:", ll.search(50))
89

```

```

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:

```
C: > Users > user > AI > ENDEXM.py > ...
1 import heapq
2
3 class Student:
4     def __init__(self, name, cgpa):
5         self.name = name
6         self.cgpa = cgpa
7
8     def __lt__(self, other):
9         return self.cgpa > other.cgpa
10
11    def __repr__(self):
12        return f"{self.name} (CGPA: {self.cgpa})"
13
14 class StudentPriorityQueue:
15     def __init__(self):
16         self.heap = []
17
18     def insert(self, student):
19         heapq.heappush(self.heap, student)
20
21     def pop_highest(self):
22         if self.heap:
23             return heapq.heappop(self.heap)
24         return None
25
26     def display(self):
27         return list[Any](self.heap)
28 pq = StudentPriorityQueue()
29
30 pq.insert(Student("Abhi", 9.1))
31 pq.insert(Student("Rahul", 8.6))
32 pq.insert(Student("Meera", 9.4))
33
34 print("Queue:", pq.display())
35 print("Top Priority Student:", pq.pop_highest())
36 print("Queue After Pop:", pq.display())

Problems 5 Output Debug Console Terminal Ports

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.