Course 2 — Data Structures with Python

Week 4 — Linked Lists

🎯 Learning Objectives

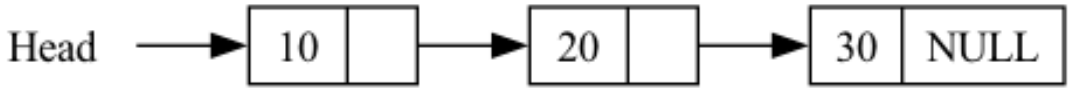
- Understand the concept and structure of linked lists
- Differentiate between singly, doubly, and circular linked lists
- Implement linked list operations in Python (insert, delete, traverse)
- Analyze time complexity of linked list operations
- Compare linked lists with arrays

🧠 Detailed Explanation

1. What is a Linked List?

A **linked list** is a linear data structure where each element (called a **node**) contains two parts: - **Data** — stores the actual value. - **Next** — a reference (pointer) to the next node in the sequence.

Unlike arrays, linked lists are not stored in contiguous memory locations. Each node "points" to the next, forming a chain-like structure.



Singly Linked List Diagram

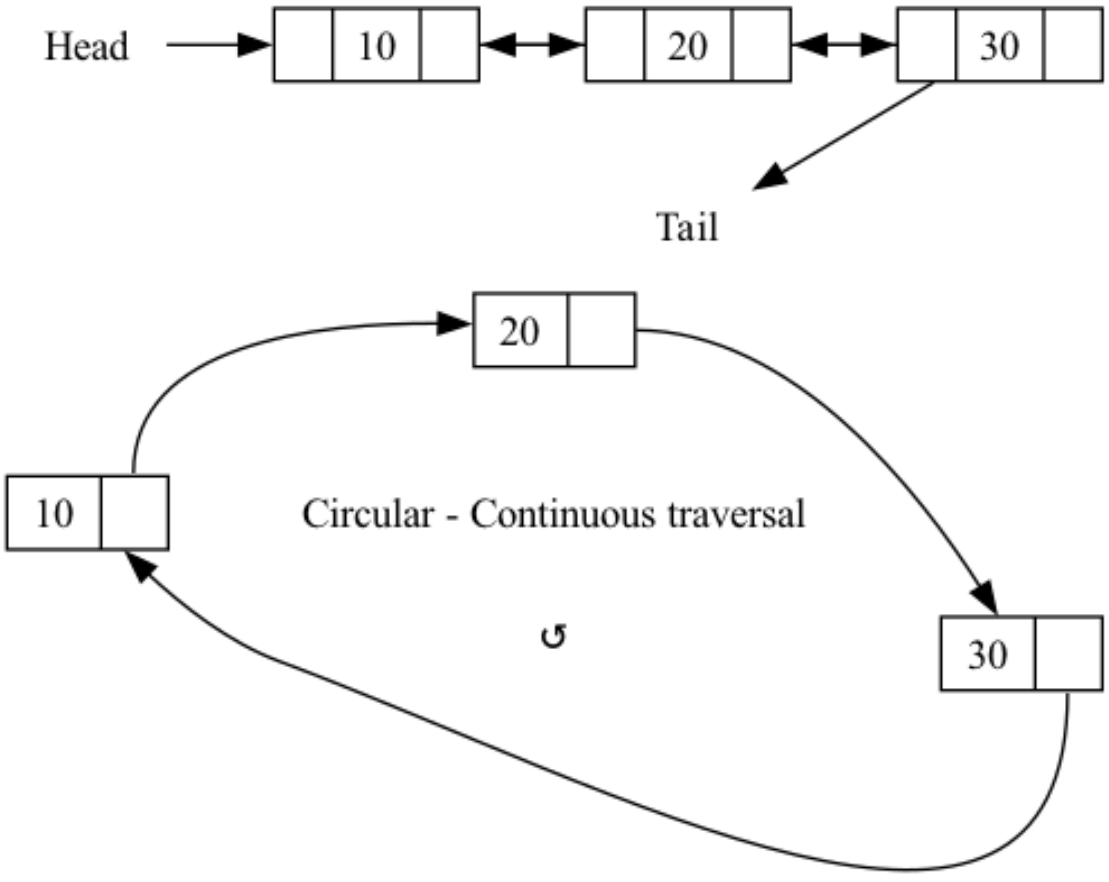
2. Advantages and Disadvantages

Advantages: - Dynamic size (can grow or shrink easily). - Efficient insertions/deletions (no need to shift elements).

Disadvantages: - No random access (must traverse sequentially). - Extra memory for pointers.

3. Types of Linked Lists

1. **Singly Linked List** — Each node points to the next node.
2. **Doubly Linked List** — Each node points to both next and previous nodes.
3. **Circular Linked List** — The last node points back to the head.



4. Node Structure in Python

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

5. Singly Linked List Implementation

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

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

```
        current = self.head
        while current.next:
            current = current.next
        current.next = new_node

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

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

        prev = None
        while current and current.data != key:
            prev = current
            current = current.next

        if current:
            prev.next = current.next

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

# Example usage
ll = LinkedList()
ll.insert_at_end(10)
ll.insert_at_end(20)
ll.insert_at_end(30)
ll.display()
ll.delete(20)
ll.display()
```

🧩 In-Class Exercises

Exercise 1 — Create and Display

Create a singly linked list of 5 numbers and display them using traversal.

Exercise 2 — Insert at Beginning

Modify the LinkedList class to support insertion at the beginning.

Exercise 3 — Delete by Position

Add a function to delete a node at a specific position.

🏠 Take-Home Assignments

Assignment 1 — Count Nodes

Write a method `count_nodes()` that returns the total number of nodes in the linked list.

Assignment 2 — Search Element

Write a method `search(key)` that returns `True` if key exists, otherwise `False`.

Assignment 3 — Reverse Linked List

Write a method `reverse()` to reverse the linked list in-place.

📚 References

- Goodrich, Tamassia & Goldwasser — *Data Structures and Algorithms in Python*
- GeeksforGeeks — Linked List in Python
- TutorialsPoint — *Data Structures: Linked Lists*

🔮 What's Next?

Next week we'll explore **Stacks and Queues**, including:

- Stack implementation using lists and linked lists
- Queue operations and applications
- Expression evaluation with stacks