TREE DATA STRUCTURE

1. Definition

A **Tree** is a **non-linear data structure** used to represent data in a **hierarchical form**.
It consists of **nodes** connected by **edges**.

One node is called the **root**, and every other node is connected to it directly or indirectly.

2. Basic Terminologies

Term	**Meaning**
Root	Topmost node of the tree
Node	An element in the tree
Edge	Connection between two nodes
Parent	A node that has child nodes
Child	A node that comes from a parent
Leaf	A node with no children
Level	Position of a node in the tree
Height	Longest path from root to leaf
Subtree	A tree within a tree

3. Characteristics of Trees

- Has **one root**
- No cycles (no circular paths)
- Every child has **one parent**
- Can have **many children**
- Used for **hierarchical data**

4. Types of Trees

4. **Binary Tree** – Each node has at most 2 children
5. **Binary Search Tree (BST)** – Left < Root < Right
6. **General Tree** – Any number of children
7. **AVL Tree** – Self-balancing tree
8. **Heap Tree** – Used in priority queues

5. Applications of Trees

Trees are used in:

File systems (Folders & files)

Databases (Indexing)

Organization structures

Family trees

Artificial Intelligence

Decision making systems

PYTHON EXAMPLE: Tree Implementation

This example shows a **Student Management System** using a Tree.

Code

```
# Tree Node Class
class TreeNode:
    def __init__(self, name):
        self.name = name
        self.children = []
    # Add child
    def add_child(self, child):
        self.children.append(child)
    # Display tree
    def display(self, level=0):
        print("  " * level + "- " + self.name)
        for child in self.children:
            child.display(level + 1)
# Create root
school = TreeNode("School")
# Departments
ict = TreeNode("ICT Department")
business = TreeNode("Business Department")
school.add_child(ict)
school.add_child(business)
# Courses
web = TreeNode("Web Development")
network = TreeNode("Networking")
accounting = TreeNode("Accounting")
ict.add_child(web)
ict.add_child(network)
business.add_child(accounting)
# Students
web.add_child(TreeNode("Alice"))
web.add_child(TreeNode("Brian"))
accounting.add_child(TreeNode("Diana"))
# Display Tree
print("Student Records Tree:\n")
school.display()
```

OUTPUT

When you run the program, you get:

```
Student Records Tree:

- School

  - ICT Department

    - Web Development

      - Alice

      - Brian

    - Networking

  - Business Department

    - Accounting

      - Diana
```

WHAT THIS PROGRAM ACHIEVES

This program:

1. Creates a Tree Structure

It builds this hierarchy:

School → Departments → Courses → Students

Which is real-life hierarchical data.

2. Stores Data Efficiently

Instead of using many lists, it organizes data in **parent-child form**.

Example:

- School → Parent
- ICT → Child
- Web → Child
- Alice → Leaf

3. Displays Data Using Traversal

The display() method uses **recursion** to visit every node.

This is called **Tree Traversal**.

It prints:

- Root first
- Then children
- Then sub-children

4. Makes Data Easy to Manage

You can easily:

✓ Add new students
✓ Add new departments
✓ Remove nodes
✓ Search nodes

Without changing the whole structure.

ADVANTAGES OF TREES

Fast searching (in BST)
Organizes complex data
Easy hierarchy representation
Flexible structure

DISADVANTAGES OF TREES

Uses more memory
More complex than arrays
Needs careful implementation