**📘 Tree – Data Structures & Algorithms (DSA) in Python**

**🔹 What is a Tree?**

A **Tree** is a non-linear, hierarchical data structure made of **nodes** connected by **edges**.

It has a **root node**, and every node (except the root) has **one parent** and **zero or more children**.

**🔹 Real-Life Examples:**

* File system directories (Folders)
* Family tree
* Organization chart
* Decision trees (in AI/ML)

**🔹 Terminology:**

| **Term** | **Meaning** |
| --- | --- |
| Root | Top node of the tree |
| Node | Each element of the tree |
| Parent | A node that has child nodes |
| Child | A node that has a parent |
| Leaf | A node with no children |
| Edge | Connection between parent and child nodes |
| Depth | Distance from root to the node |
| Height | Distance from node to its deepest leaf |
| Subtree | A smaller tree within a tree |
| Binary Tree | A tree where each node has at most 2 children |

**🔹 Basic Tree Structure in Python**

class TreeNode:

def \_\_init\_\_(self, data):

self.data = data

self.children = []

def add\_child(self, child):

self.children.append(child)

def print\_tree(self, level=0):

print(" " \* level \* 2 + str(self.data))

for child in self.children:

child.print\_tree(level + 1)

**🔹 Example:**

root = TreeNode("Electronics")

laptop = TreeNode("Laptop")

laptop.add\_child(TreeNode("MacBook"))

laptop.add\_child(TreeNode("Dell"))

mobile = TreeNode("Mobile")

mobile.add\_child(TreeNode("Samsung"))

mobile.add\_child(TreeNode("iPhone"))

root.add\_child(laptop)

root.add\_child(mobile)

root.print\_tree()

**🔹 Output:**

Electronics

Laptop

MacBook

Dell

Mobile

Samsung

iPhone

**🔹 Binary Tree in Python**

A **Binary Tree** is a tree in which each node has **at most 2 children**.

class BinaryTreeNode:

def \_\_init\_\_(self, data):

self.data = data

self.left = None

self.right = None

**🔹 Binary Tree Traversals (Recursive)**

**1. Inorder (L → Root → R)**

def inorder(root):

if root:

inorder(root.left)

print(root.data, end=" ")

inorder(root.right)

**2. Preorder (Root → L → R)**

def preorder(root):

if root:

print(root.data, end=" ")

preorder(root.left)

preorder(root.right)

**3. Postorder (L → R → Root)**

def postorder(root):

if root:

postorder(root.left)

postorder(root.right)

print(root.data, end=" ")

**🔹 Example Tree:**

10

/ \

5 15

python

CopyEdit

root = BinaryTreeNode(10)

root.left = BinaryTreeNode(5)

root.right = BinaryTreeNode(15)

inorder(root) # Output: 5 10 15

**🔹 Types of Trees**

| **Tree Type** | **Description** |
| --- | --- |
| Binary Tree | Each node has at most 2 children |
| Binary Search Tree | Left < Root < Right |
| AVL Tree | Self-balancing BST |
| B-Tree | Used in databases & file systems |
| Trie | Used in word dictionaries, prefix search |
| Heap (Min/Max) | Complete binary tree used for priority queues |

**🔹 Applications of Trees**

* Expression parsing (syntax tree)
* File systems & folder hierarchies
* Databases indexing (B-Trees)
* Routing algorithms (Tries)
* AI games (Decision Trees)
* Huffman Encoding Tree

Let me know if you want notes or code for:

* **Binary Search Tree (BST)**
* **Balanced Trees (AVL)**
* **Trie (Prefix Tree)**
* **Heap Trees (Min/Max)**