🌳 1. Binary Tree Node Definition

A **binary tree** is a tree where each node has at most two children: left and right.
We define a node class as follows:

```
class TreeNode:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right
```

- `val`: The value stored in the node.
- `left`: Reference to the left child node.
- `right`: Reference to the right child node.

🔍 2. Binary Search Tree (BST) Properties

A **Binary Search Tree** is a special kind of binary tree where:

- All values in the left subtree are **less than** the parent node.
- All values in the right subtree are **greater than** the parent node.

This property allows for efficient searching, insertion, and deletion.

🔄 3. Tree Traversals

a. Depth-First Search (DFS) Traversals

DFS explores as far as possible along each branch before backtracking.
There are three common types:

Preorder Traversal (Root, Left, Right)

- Visit the root node first, then recursively traverse the left subtree, then the right subtree.

```
def preorder(root):
    if root:
        print(root.val, end=" ")
        preorder(root.left)
        preorder(root.right)
```

Inorder Traversal (Left, Root, Right)

- Recursively traverse the left subtree, visit the root node, then traverse the right subtree.
- In BSTs, inorder traversal visits nodes in **sorted order**.

```
def inorder(root):
    if root:
        inorder(root.left)
        print(root.val, end=" ")
        inorder(root.right)
```

Postorder Traversal (Left, Right, Root)

- Recursively traverse the left subtree, then the right subtree, then visit the root node.

```
def postorder(root):
    if root:
        postorder(root.left)
        postorder(root.right)
        print(root.val, end=" ")
```

---

b. Breadth-First Search (BFS) / Level Order Traversal

- BFS visits nodes level by level from top to bottom, left to right.
- It uses a **queue** to keep track of nodes at the current level.

```
from collections import deque

def level_order(root):
    if not root:
        return
    queue = deque([root])
    while queue:
        node = queue.popleft()
        print(node.val, end=" ")
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
```

🧪 4. Example Usage

Let’s build a simple tree and demonstrate all traversals:

```
# Construct the tree:
#     1
#    / \
#   2   3
root = TreeNode(1)
root.left = TreeNode(2)
root.right = TreeNode(3)

print("Preorder:")
preorder(root)        # Output: 1 2 3

print("\nInorder:")
inorder(root)         # Output: 2 1 3

print("\nPostorder:")
postorder(root)       # Output: 2 3 1

print("\nLevel Order:")
level_order(root)     # Output: 1 2 3
```

Let’s build a more complex tree and demonstrate all traversals:

```
# Build the tree:
#       1
#      / \
#     2   3
#    / \
#   4   5


root = TreeNode(1)
root.left = TreeNode(2, TreeNode(4), TreeNode(5))
root.right = TreeNode(3)

print("DFS - Preorder:")
preorder(root)        # Output: 1 2 4 5 3

print("\nDFS - Inorder:")
inorder(root)         # Output: 4 2 5 1 3

print("\nDFS - Postorder:")
postorder(root)       # Output: 4 5 2 3 1

print("\nBFS - Level Order:")
level_order(root)     # Output: 1 2 3 4 5
```

📌 Summary

- **Binary trees** organize data in a hierarchical structure.
- **BSTs** allow fast search, insert, and delete due to their ordering property.
- **DFS traversals** (preorder, inorder, postorder) visit nodes in different orders for various applications.
- **BFS traversal** (level order) visits nodes level by level, useful for shortest path and serialization.