A tree data structure in Python is a non-linear, hierarchical data structure composed of interconnected nodes. Unlike linear structures like linked lists, trees allow for branching, where a single node can have multiple "next" elements.

**Key Components of a Tree:**

* **Node:** The fundamental unit of a tree, containing data and references (edges) to other nodes.
* **Root Node:** The topmost node of the tree; it has no parent.
* **Parent Node:** A node that has one or more child nodes connected to it.
* **Child Node:** A node that is a descendant of another node (its parent).
* **Edge:** The connection or reference linking a parent node to a child node.
* **Leaf Node:** A node that has no children.
* **Internal Node:** A node that has at least one child.

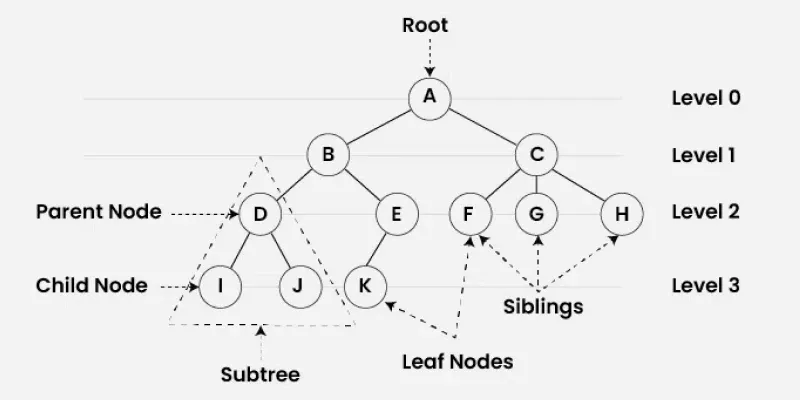
**Common Tree Types:**

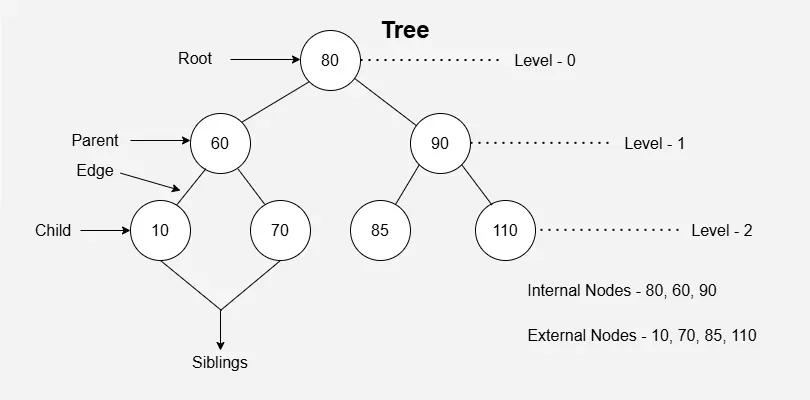
* **Binary Tree:** Each node has at most two children (left and right).
* **Ternary Tree** **:**Every node has at most three children
* **N-ary Tree :**Every node has at most n children.
* **Binary Search Tree (BST):** A binary tree where the left child's value is less than the parent's, and the right child's value is greater.
* **Heap:** A specialized tree-based data structure that satisfies the heap property (e.g., in a min-heap, the parent's value is always less than or equal to its children's values).

**Applications of Trees:**

Trees are widely used in computer science for:

* Representing hierarchical data (e.g., file systems, organizational charts).
* Efficient data storage and retrieval in databases.
* Implementing routing algorithms in networks.
* Sorting and searching algorithms.
* Building priority queues.





**Implementation of a Tree:**

class Node:

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

self.data = data

self.left = None # Reference to the left child node

self.right = None # Reference to the right child node

# Creating the root node

root = Node(10)

# Adding child nodes

root.left = Node(5)

root.right = Node(15)

# Adding more levels

root.left.left = Node(2)

root.left.right = Node(7)

root.right.left = Node(12)

root.right.right = Node(18)

# You can then perform operations like traversal to verify the tree structure

def inorder\_traversal(node):

if node:

inorder\_traversal(node.left)

print(node.data, end=" ")

inorder\_traversal(node.right)

print("Inorder Traversal:")

inorder\_traversal(root)

print()

**Binary Tree:**

A binary tree can be implemented in Python using a class to represent each node.

class Node:

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

self.data = data

self.left = None

self.right = None

# Example of creating a simple binary tree

root = Node(1)

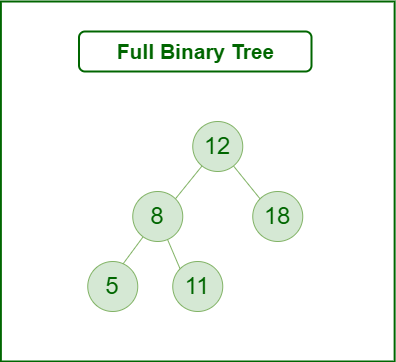
root.left = Node(2)

root.right = Node(3)

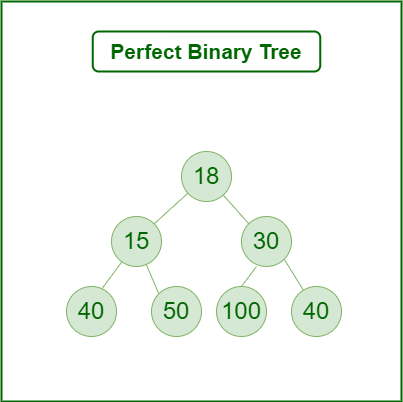
root.left.left = Node(4)

**Types of Binary Tree:**

Full Binary Tree: A Binary Tree is a full binary tree if every node has 0 or 2 children. A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children. It is also known as a proper binary tree.



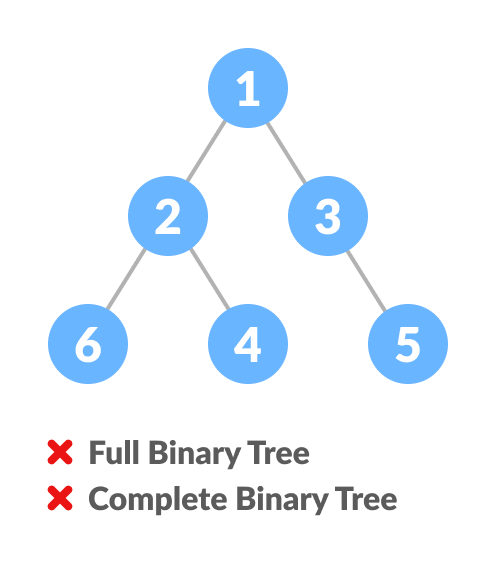
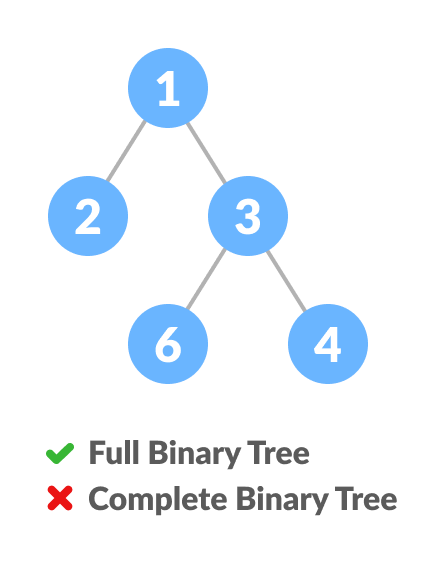
Perfect Binary Tree: A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.

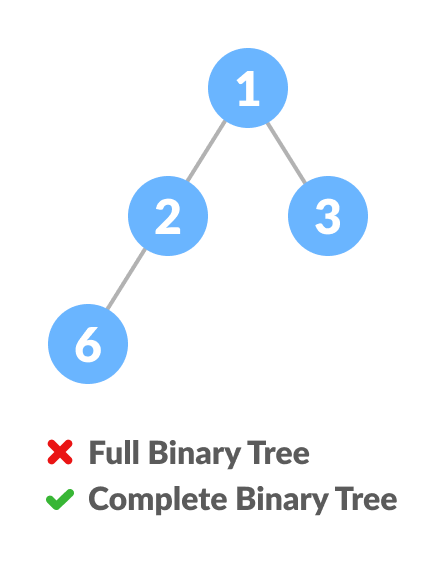
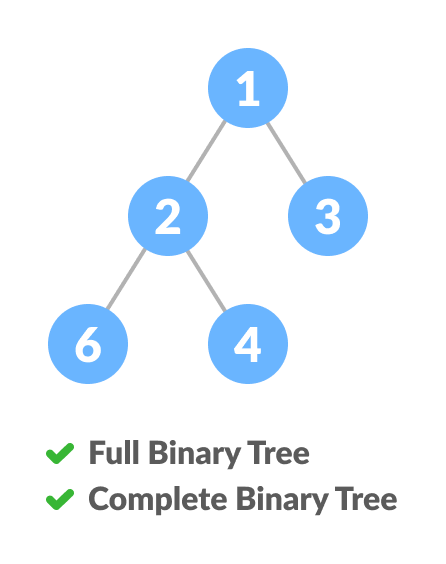


Complete Binary Tree: A complete binary tree is a binary tree in which all the levels are completely filled except possibly the lowest one, which is filled from the left.

A complete binary tree is just like a full binary tree, but with two major differences

1. All the leaf elements must lean towards the left.
2. The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.

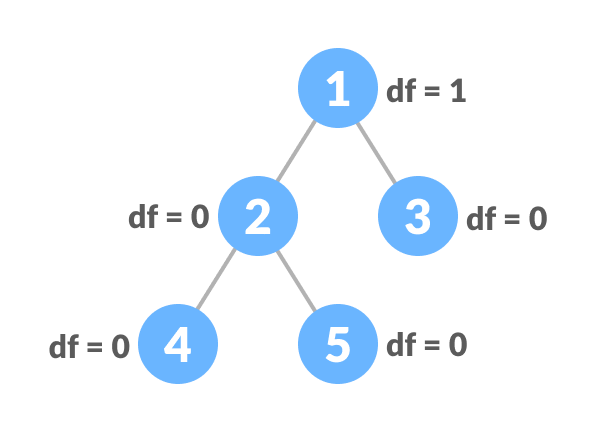
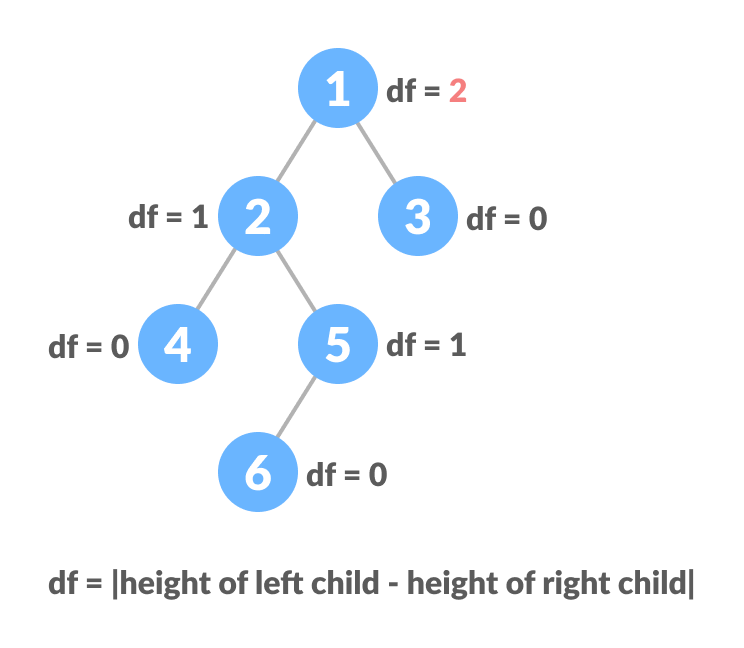
 

Balanced Binary Tree: A balanced binary tree, also referred to as a height-balanced binary tree, is defined as a binary tree in which the height of the left and right subtree of any node differ by not more than 1.

Following are the conditions for a height-balanced binary tree:

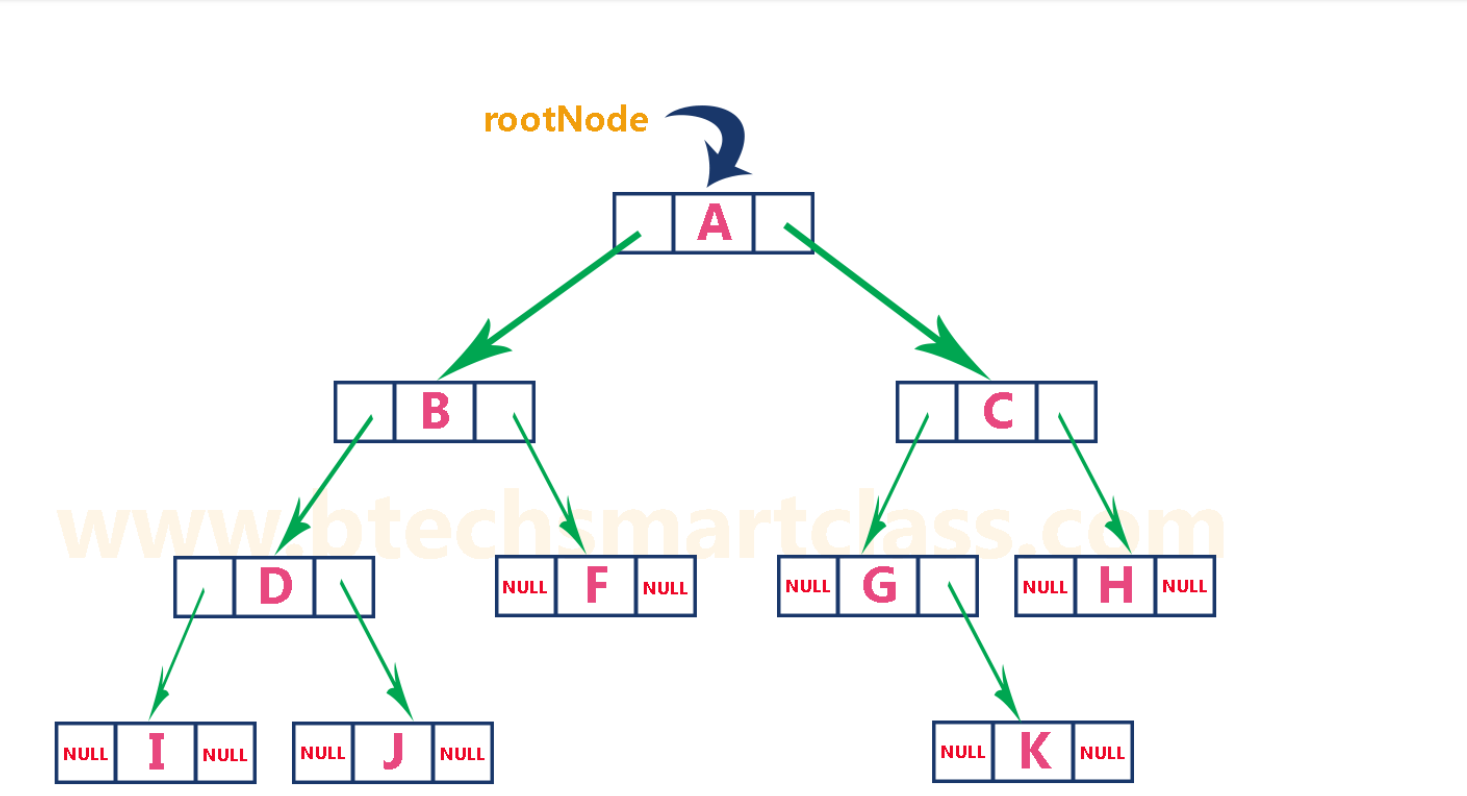
1. difference between the left and the right subtree for any node is not more than one
2. the left subtree is balanced
3. the right subtree is balanced

Unbalanced Binary Tree

Balanced Binary Tree

**REPRESENTATION OF A BINARY TREE USING LINKED LIST AND LIST:**



Binary tree represented by a linked list

When we use list to represent a binary tree, we don’t use the first index (0). This is because it makes the mathematical calculation easy.

To store the left and right child of a tree in a list, we use the following formula:

Left-child = cell[2x]

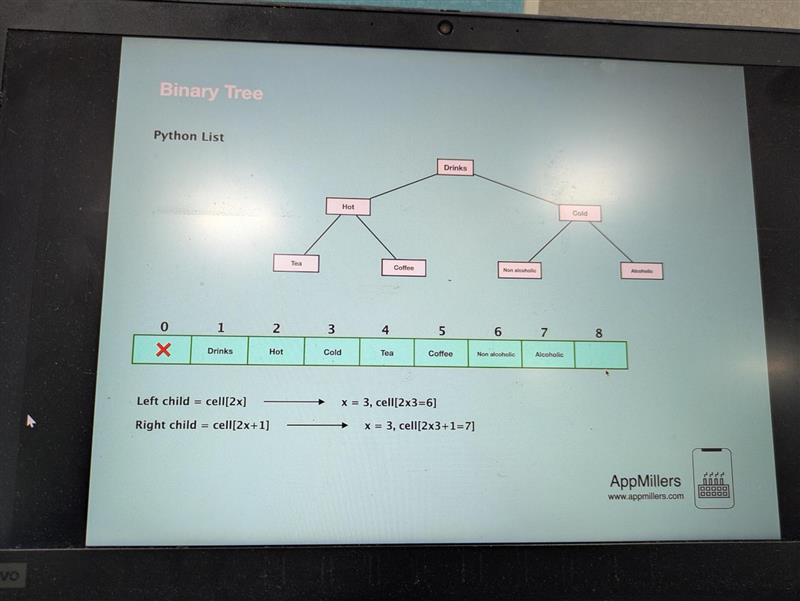
Right-child = cell[2x+1]

x – root node’s index in the python list (which is 1)

We always place the root node at index 1 and based on this node, we calculate left and right child.

Left child = 2\*1 = index 2

Right child = (2\*1)+1 = index 3



To prevent leaving an empty index (Eg: index 8 in the above figure), we need to know the number of nodes in a tree upfront then we can initialize the fixed size of the python list.