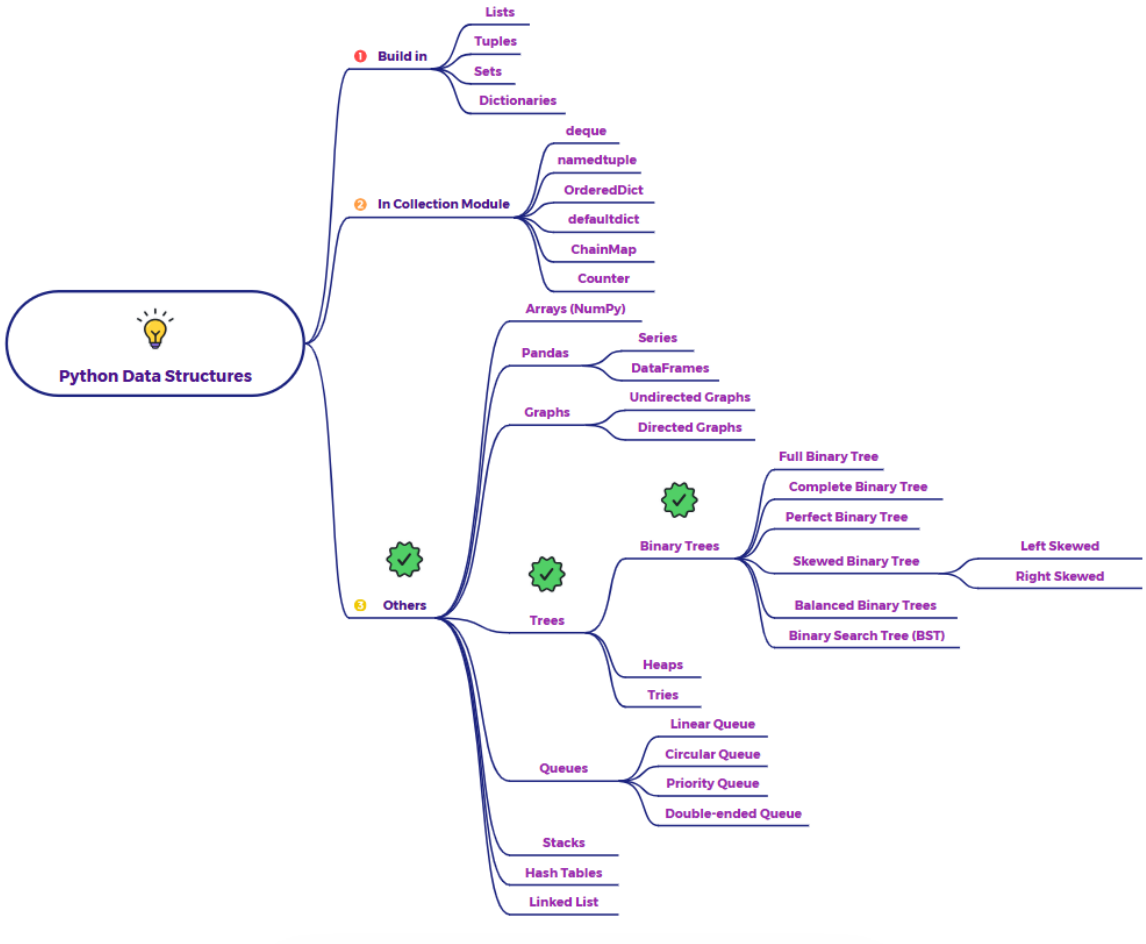
## Different types of Binary trees in python




the common types of binary trees you might encounter and implement in Python:

### 1. Full Binary Tree:

- **Definition:** A binary tree in which every node other than the leaves has exactly two children.
- **Characteristics:** All internal nodes are "full" with two children.

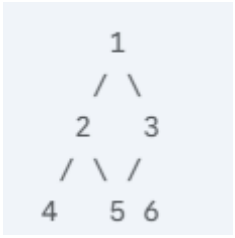
**Example:**

```
    4
   / \
  2   7
 / \
1   3
```

## 2. Complete Binary Tree:

- **Definition:** A binary tree in which all levels are completely filled except possibly the last level, which is filled from left to right.
- **Characteristics:**
    - All levels except the last are full.
    - In the last level, nodes are as far left as possible.
- **Importance:** Often used to implement Heaps because of its compact array representation.

**Example:**




## 3. Perfect Binary Tree:

- **Definition:** A binary tree in which all internal nodes have exactly two children, and all leaf nodes are at the same level.
- **Characteristics:** It's both a full and a complete binary tree. All levels are completely filled.
- **Relationship to nodes and height (h):** A perfect binary tree of height h has 2^(h+1) - 1 nodes.

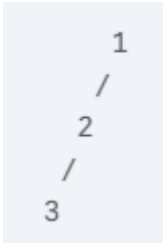
**Example:**


## 4. Degenerate (or Skewed) Binary Tree:

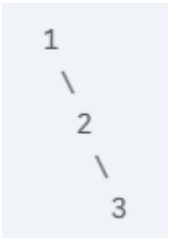
- **Definition:** A binary tree where each internal node has only one child (either left or right).

- **Characteristics:** Resembles a linked list.
- **Worst-case scenario:** Can lead to O(n) time complexity for search and insertion operations in what is intended to be a tree structure (like a Binary Search Tree).
- **Examples:**

  Left-skewed:




  Right-skewed:




## 5. Balanced Binary Trees:

- **Definition:** Binary trees that try to maintain a certain balance in their structure (height difference between left and right subtrees) to ensure efficient operations (typically logarithmic time complexity).
- **Importance:** Crucial for implementing efficient search structures like Binary Search Trees.
- **Types:**
  - **AVL Trees:** A self-balancing Binary Search Tree where the height difference between the left and right subtrees of any node is at most 1.
  - **Red-Black Trees:** Another self-balancing Binary Search Tree that uses color attributes for nodes to maintain balance. They offer

good performance and are used in many standard library implementations (e.g., sets and maps in some languages).

- **B-Trees and B+ Trees:** While technically n-ary trees (more than two children), they are important balanced tree structures often discussed in the context of binary trees as they solve similar balancing problems, especially for disk-based storage. However, they don't strictly adhere to the "at most two children" rule.

### 6. Binary Search Tree (BST):

- **Definition:** A binary tree with a specific ordering property:
  - The value of the left child (and all nodes in its left subtree) is less than the value of the parent node.
  - The value of the right child (and all nodes in its right subtree) is greater than the value of the parent node.
  - This property holds for all nodes in the tree.
- **Characteristics:** Allows for efficient searching, insertion, and deletion (average time complexity O(log n) if the tree is reasonably balanced).

**Example:**