Binärbaum: Im Wesentlichen die Wurzel.

```
class BinaryTree:
    """Basically, a BinaryTree is defined by its root
    (and all the subtrees pending from it)."""
    def __init__(self, key, data = None, parent = None, node_type = "root"):
        """Initialize: Either with existing BinaryTreeNode,
        or with key."""
        if isinstance(key, BinaryTreeNode):
            self.root = key
        else:
            self.root = BinaryTreeNode(key, data=data)

    def __str__(self):
        return f'BinaryTree (root: {self.root})'

    # Search depth-first for key:
    def df_search_subtree(self, key):
        return self.root.df_search_subtree(key)

    # Traverse the tree depth-first and apply a function which
    # takes (node, level_of_node) as arguments
    def df_traverse(self, the_func, level, account):
        self.root.df_traverse(the_func, level, account)

    # Traverse the tree (starting at root) breadth-first and apply
    # a function which takes (node, level_of_node, number_of_node)
    # and an optional "account" (in most cases: a dictionary to be
    # updated by the nodes) as arguments:
    def bf_traverse(self, the_func, account):
        self.root.bf_traverse(self, the_func, account=account)

    # Der Baum muss verändert werden können, durch Anhängen oder
    # Löschen von Blättern:
    def append_leaf(self, child, parent_key, side="left"):
        # Find the node parent_key:
        if (parent_node := self.df_search_subtree(parent_key)) is None:
            print(f"Key {parent_key} not found!")
            return
        # Implicit else:

        # Fügen Sie hier bitte geeignete Codezeilen ein!
```

```
    def delete_leaf(self, key):
        """Find and delete leaf with key."""
        # Find the node key:
        if (node := self.df_search_subtree(key)) is None:
            print(f"No node {key} in binary tree!")
            return
        # Implicit else:

        # Fügen Sie hier bitte geeignete Codezeilen ein!
```