AVL–Baum, als Sub-Klasse von BinaryTree.

```
class AVLTree(BinaryTree):
    """Basically, a AVLTree is defined by its root, an AVLTreeNode
    (and all the subtrees pending from it)."""
    def __init__(self, key, data = None):
        """Initialize"""
        self.root = AVLTreeNode(key, data=data)

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

    # Beim Anfügen und Entfernen von Blättern müssen wir "rebalancen",
    # wenn nötig: Ausserdem kann es "in der Praxis" vorkommen, dass wir
    # mehrere data-Einträge unter demselben key speichern wollen (darum
    # ist contents.data in AVLTreeNode ja auch als Liste definiert), das
    # implementieren wir hier auch.
    def append(self, key, data=None):
        """Append new data in leaf key."""

        # Find leaf node where key should be appended as new leaf:
        node, side = self.root.binary_search(key)

        # If this key is _equal_ to the existing node.key, simply _append_
        # the data to node.contents:
        if key == node.key:
            node.contents["data"].append(data)
            # Nichts weiter zu tun!
            return


        # Implicit else: key != node.key

        # Append a new leaf, using function replace_subtree:
        node.replace_subtree(AVLTreeNode(key, data), side)

        other_side = OTHER_SIDE[side]
        # Former leaf node "slides down" to other side:
        node.replace_subtree(
            AVLTreeNode(node.key, node.contents),
            other_side
        )
        # Former leaf node now is an inner node (containing no data):
        node.contents["data"] = []
        # Update key of node:
        if side == 'left':
            node.key = key

        # Rebalancing might change the root of the rebalanced tree:
        self.root = node.rebalance()

    def delete(self, key, data=None):
        """Find the leaf with key: If there is such leaf, remove the
        data; _if_ remaining data is empty, then also remove the leaf."""

        node, side = self.root.binary_search(key)
        if side == 'left' and node.key == key:
            # Ok: There is a leaf node with this key!
            if data is not None:
                try:
                    node.contents["data"].remove(data)
                    # Sind noch andere Daten in diesem node gespeichert?
                    remove_this_leaf = (node.contents["data"] == [])
                except ValueError:
                    print(f"Node {key} does not contain {data}!")
                    # Hmm: Diese Daten gibt´s gar nicht! Der Baum wird
                    # also nicht verändert, und wir "springen aus der
                    # Funktion heraus".
                    return
            else:
                remove_this_leaf = True

            # Implicit else:
            if remove_this_leaf:
                # Ok, this leaf node actually should be deleted:

                parent = node.parent
                side = node.node_type
                other_side = OTHER_SIDE[side]

                # Fügen Sie hier den passenden Code ein: Wenn das
                # Blatt gleich der Wurzel ist, setzen Sie self.root = None,
```

```
# andernfalls entfernen Sie das Blatt und rebalancieren
# Sie den Baum.
```