

Knoten in einem Binärbaum.

```
# Kleiner Trick: "Übersetzung andere Seite" (also "links" <-> "rechts"):
OTHER_SIDE = {
    "left" : "right",
    "right" : "left"
}

# Eine Klasse für einen Knoten - enthält (fast) alles, was für einen
# binary tree nötig ist:
class BinaryTreeNode:
    """A binary tree _basically_ is a _root_ (together with
    all its subtrees).

    A node in a binary tree may be viewed as the
    root of the subtrees pending from it and thus as
    a binary tree itself.

    In this sense there is little difference
    between binary trees and nodes of binary trees."""
    def __init__(self, key, data = None, parent = None, node_type = "root"):
        """Initialize:"""
        # "Pointer" (sort of...) to parent node
        self.parent = parent
        # Is this node the "global" root, or a left or right child
        # of its parent?
        self.node_type = node_type
        # Informations on left/right children:
        self.child = {"left" : None, "right" : None}
        # And, of course: Store the key ...
        self.key = key
        # ... and the data _as a dict_: Maybe we want to add other data
        # later for the _same_ key!
        self.contents = dict() if data is None else deepcopy(data)

    def __str__(self):
        """Return string representation:"""
        return f'{self.node_type} ({self.key}) <{self.contents}>.'

# Auxiliary functions:
def is_leaf(self):
    """Return True if self is, in fact, a leaf of the tree
    it belongs to:"""
    return (self.child["left"] is None) and (self.child["right"] is None)

# Append child:
def append(self, side, child):
    """Append BinaryTreeNode as a (left or right) child:"""
    self.child[side] = child
    if child is not None:
        child.node_type=side
        child.parent = self

# Search depth-first for key in subtree rooted at self:
def df_search_subtree(self, key):
    """Depth-First-Suchen nach Knoten key."""
    if self.key == key:
        # Found node "key"!
        return self

    # Implicit else:
    # Search left & right subtree by (somewhat recursive) function calls
    # for the child-nodes:

    # Fügen Sie hier bitte geeignete Codezeilen ein!

    # Implicit else:
    # Key not found yet, and no more subtree to search!
    return None

# Traverse the tree rooted at self "depth-first" and apply a function
# which takes (node, level_of_node, account) as arguments:
def df_traverse(self, the_func, level=0, account=None):
    """Durchlaufen des gesamten Baumes, depth-first, wobei
    für jeden Knoten eine Funktion "the_func" aufgerufen wird,
    die auch das aktuelle level (Niveau) und ein (optionales)
    Dictionary "account" (in dem eventuell Informationen
    gespeichert werden; z.B. eine Liste der Blätter) als
    Argumente übernimmt."""
    level = level
    the_func(self, level, account)
    # Traverse left & right subtree by recursive function call:
```

Fügen Sie hier bitte geeignete Codezeilen ein!

*# Traverse the tree rooted at self "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=**None**):

*""""Durchlaufen des gesamten Baumes, breadth-first, wobei
für jeden Knoten eine Funktion "the_func" aufgerufen wird,
die auch das aktuelle level (Niveau), die Nummer des Knotens
"in seinem Niveau" und ein (optionales) Dictionary
"account" (in dem eventuell Informationen gespeichert
werden; z.B. eine Liste der Blätter) als Argumente
übernimmt."""""*

*# Use a queue for storing the nodes (in their order:
"level by level, from left to right":*

queue = **deque**()

*# Also store the level (distance from root) and the
number (in its level, from left to right) of the node.*

*# Initially, append (root,level=0) to the right of the queue,
which contains the list of nodes yet to be processed:*

queue.append((self,0))

Traverse the tree:

count = 0

old_level = -1 *# If new level is reached, then reset count!*

while **queue**:

Pop first node (from the left) ...

node,level = **queue.popleft**()

Update count of node:

Fügen Sie hier bitte geeignete Codezeilen ein!

Don't forget to apply the_func:

count += 1

the_func(**node**,**level**,**count**,**account**=**account**)