### Recursive description (wooden trees):
- A tree has a root label and a list of branches.
- Each branch is a tree.
- A tree with zero branches is called a leaf.
- A tree starts at the root.

### Relative description (family trees):
- Each location in a tree is called a node.
- Each node has a label that can be any value.
- One node can be the parent/child of another.
- The top node is the root node.

---

People often refer to labels by their locations: "each parent is the sum of its children."
Implementing the Tree Abstraction

```python
def tree(label, branches=[]):
    return [label] + branches

def label(tree):
    return tree[0]

def branches(tree):
    return tree[1:]

>>> tree(3, [tree(1), ...
    tree(2, [tree(1), ...
        [3, [1], [2, [1], [1]]])])
[3, [1], [2, [1], [1]]]
```

```
Tree Processing

Processing a leaf is often the base case of a tree processing function.
The recursive case typically makes a recursive call on each branch, then aggregates.

```python
def count_leaves(t):
    """Count the leaves of a tree."""
    if is_leaf(t):
        return 1
    else:
        branch_counts = [count_leaves(b) for b in branches(t)]
        return sum(branch_counts)
```
```
Discussion Question

Implement `leaves`, which returns a list of the leaf labels of a tree

*Hint: If you `sum` a list of lists, you get a list containing the elements of those lists*

```python
>>> sum([[1], [2, 3], [4]], [])
[1, 2, 3, 4]

>>> sum([[1], []], [1])
[1]

>>> sum([[1], [2]], [])
[[1], 2]
```

Creating Trees

A function that creates a tree from another tree is typically also recursive

```python
def increment_leaves(t):
    """Return a tree like t but with leaf labels incremented.""
    if is_leaf(t):
        return tree(label(t) + 1)
    else:
        bs = [increment_leaves(b) for b in branches(t)]
        return tree(label(t), bs)
```

```python
def increment(t):
    """Return a tree like t but with all labels incremented.""
    return tree(label(t) + 1, [increment(b) for b in branches(t)])
```
Count Paths that have a Total Label Sum

def count_paths(t, total):
    """Return the number of paths from the root to any node in tree t
    for which the labels along the path sum to total."

    >>> t = tree(3, [tree(-1), tree(1, [tree(2, [tree(1)]), tree(3)]), tree(1, [tree(-1)])])
    >>> count_paths(t, 3)
    2
    >>> count_paths(t, 4)
    2
    >>> count_paths(t, 5)
    0
    >>> count_paths(t, 6)
    1
    >>> count_paths(t, 7)
    2
    >>>
    if label(t) == total:
        found = 1
    else:
        found = 0
    return found + sum([count_paths(b, total - label(t)) for b in branches(t)])