Announcements
Tree Class
**Recursive description** (wooden trees):

A tree has a label value and a list of branches.
Each branch is a tree.
A tree with zero branches is called a leaf.

<table>
<thead>
<tr>
<th>Label Values</th>
<th>Nodes</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0 1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

**Relative description** (family trees):

Each location in a tree is called a node.
Each node has a value.
One node can be the parent/child of another.
Top node of tree is its root.
Tree Class

A Tree has a label value and a list of branches; each branch is a Tree

class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = left.label + right.label
        return Tree(fib_n, [left, right])

def fib_tree(n):
    if n == 0 or n == 1:
        return tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = label(left) + label(right)
        return tree(fib_n, [left, right])
Side Excursion: Equality

If \( x \) and \( y \) are two objects, the equality test, \( x == y \), does not automatically mean what you want it to mean.

For example, \( \text{Tree}(4) \neq \text{Tree}(4) \) but after performing \( x = \text{Tree}(4) \), we do have \( x == x \).

The reason for this is that in Python,

- All values (conceptually, at least) are in fact pointers to objects, and

- By default, == on pointers compares the pointers themselves (“are these pointing at exactly the same object?”).

- That is, by default == and != are the same as the is and is not operators.

- That can be changed on a class-by-class basis. For example, == on numbers, lists, tuples, strings, sets, and dictionaries means what we expect: the contents are the same.
Tree Mutation
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

def prune(t, n):
    """Prune sub-trees whose label value is n."""
    t.branches = [________________ for b in t.branches if ____________ b.label != n ________]
    for b in t.branches:
        prune_____________________________, ____________________________

(Demo)


**Example: Pruning Trees**

Removing subtrees from a tree is called *pruning*.

Prune branches before recursive processing.

E.g., want to prune cached (previously memorized) values.

**Memoization:**
- Returned by `fib`
- Found in cache
- Skipped

(Demo)
Hailstone Trees
Hailstone Trees

Pick a positive integer \( n \) as the start

If \( n \) is even, divide it by 2

If \( n \) is odd, multiply it by 3 and add 1

Continue this process until \( n \) is 1

(Demo)

def hailstone_tree(k, n=1):
    """Return a Tree in which the paths from the leaves to the root are all possible hailstone sequences of length \( k \) ending in \( n \)."""

All possible \( n \) that start a length-8 hailstone sequence

(Demo)