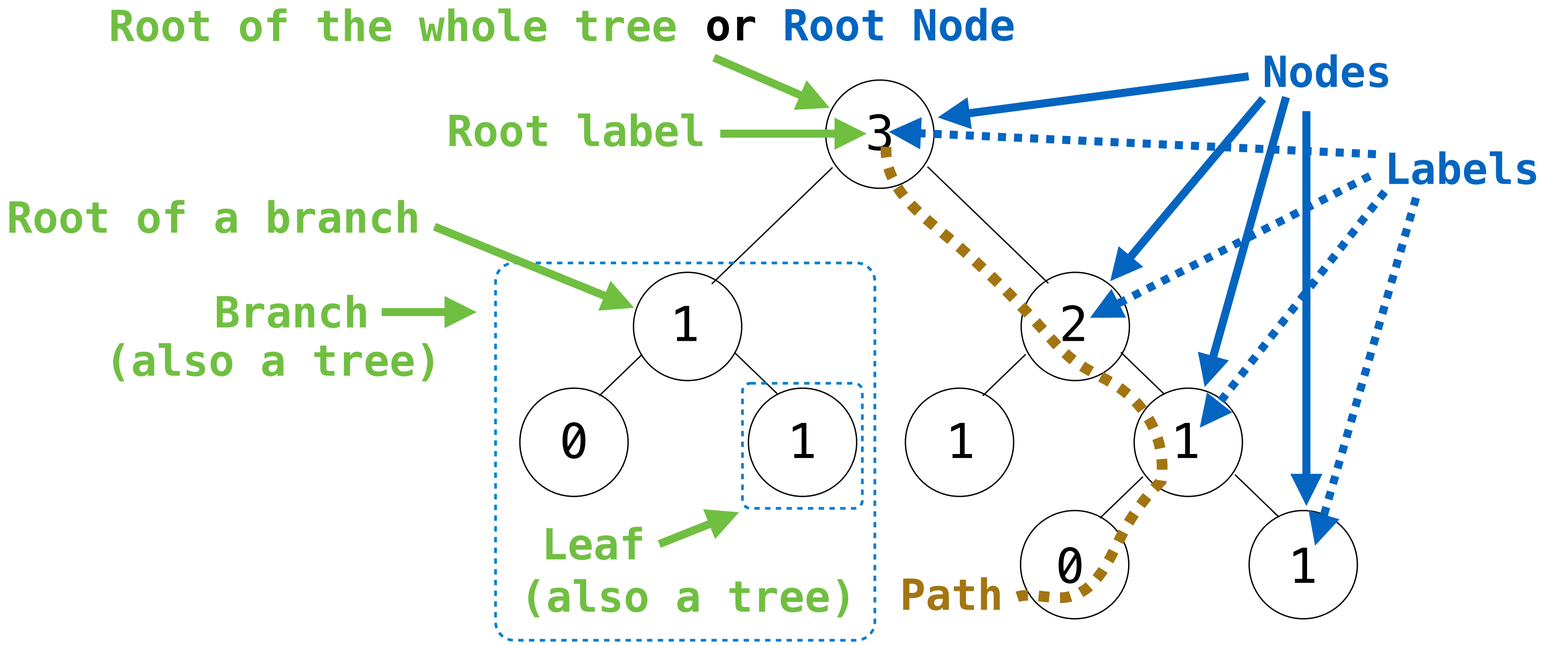
Trees

Announcements

Trees

Tree Abstraction



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**

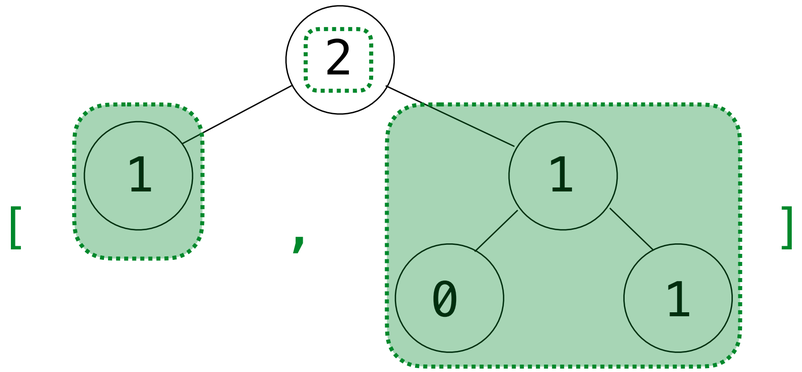
People often refer to labels by their locations: "*each parent is the sum of its children*"

Using the Tree Abstraction

For a tree t, you can **only**:

- Get the label for the root of the tree: `label(t)`
- Get the list of branches for the tree: `branches(t)`
- Get the branch at index i, which is a tree: `branches(t)[0]`
- Determine whether the tree is a leaf: `is_leaf(t)`
- Treat t as a value: `return t`, `f(t)`, `[t]`, `s = t`, etc.

An example tree `t`:



(Demo)

Tree Processing

Writing Recursive Functions

Make sure you can answer the following before you start writing code:

- What **small initial choice** can I make?
  - For trees, often: which branch to explore?
- What **recursive call for each option**?
- How can you **combine the results** of those recursive calls?
  - What type of values do they return?
  - What do the possible return values mean?
  - How can you use those return values to complete your implementation? E.g.,
    - Look to see if any option evaluated to true
    - Add up the results from each option

Tree Processing Uses Recursion

Small, initial choice: which branch's leaves to count?

Recursive call for each option: for each branch b, `count_leaves(b)`

Number of leaves on branch b

Combine results: add up all of the counts

What type of values do they return?
What do the possible return values mean?
How can you use those return values to complete your implementation?

```
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)
```

Processing a leaf is often the base case

Example: Largest Label

Small, initial choice: which branch to look for the largest label on?

Recursive call for each option: for each branch b, largest_label(b)

A label that's the largest one from branch b

Combine results: Return the largest of these, and the root label

What type of values do they return?
What do the possible return values mean?
How can you use those return values to complete your implementation?

```
def largest_label(t):
    """Return the largest label in tree t."""
    if is_leaf(t):
        return label(t)
    else:
        return max([largest_label(b) for b in branches(t)] + [label(t)])
```

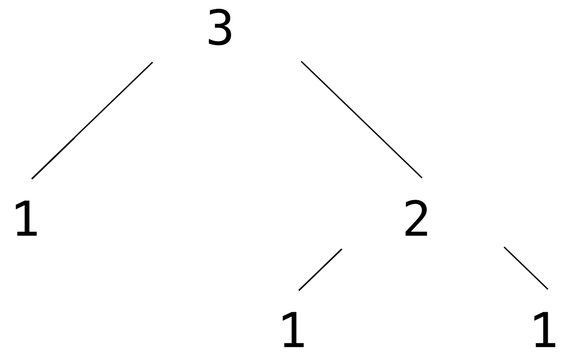
What would happen if we got rid of this?

Tree Implementation

Implementing the Tree Abstraction

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

def label(tree):
    return tree[0]

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

- A **tree** has a root **label** and a list of **branches**
- Each branch is a tree




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

Implementing the Tree Abstraction

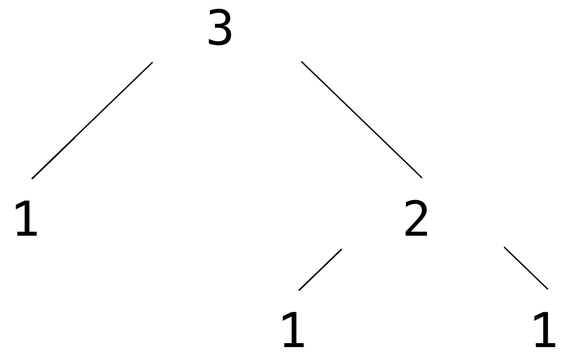
```
def tree(label, branches=[]):
    for branch in branches:
        assert is_tree(branch)
    return [label] + list(branches)

def label(tree):
    return tree[0]

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

def is_tree(tree):
    if type(tree) != list or len(tree) < 1:
        return False
    for branch in branches(tree):
        if not is_tree(branch):
            return False
    return True
```

Verifies the tree definition

Creates a list from a sequence of branches

Verifies that tree is bound to a list

- A **tree** has a root **label** and a list of **branches**
- Each branch is a tree




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

```
def is_leaf(tree):
    return not branches(tree)
```

Example: Above Root

Small, initial choice: Which branch to look at for labels to print?

Recursive call for each option: For each branch b, process(b)

Combine results: Don't need to combine the recursive return call!
Do need to print this label, if it's larger than the root

```
def above_root(t):
    """Print all the labels of t that are larger than the root label."""
    def process(u):
        if label(u) > label(t):
            print(label(u))
        for b in branches(u):
            process(b)
    process(t)
```

Min Practice

Example: Minimum x

Given these two related lists of the same length:

```
xs = list(range(-10, 11))
```

Write an expression that evaluates to the x in xs for which x*x - 2*x + 1 is smallest:

```
>>> xs
[-10,  -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4,  5,  6,  7,  8,  9, 10]
>>> [x*x - 2*x + 1 for x in xs]
[121, 100, 81, 64, 49, 36, 25, 16,  9,  4, 1, 0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>> ... some expression involving min ...
1
```