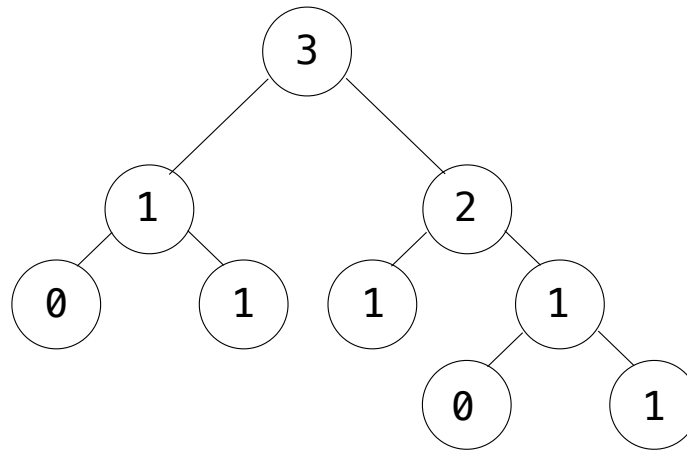


Trees

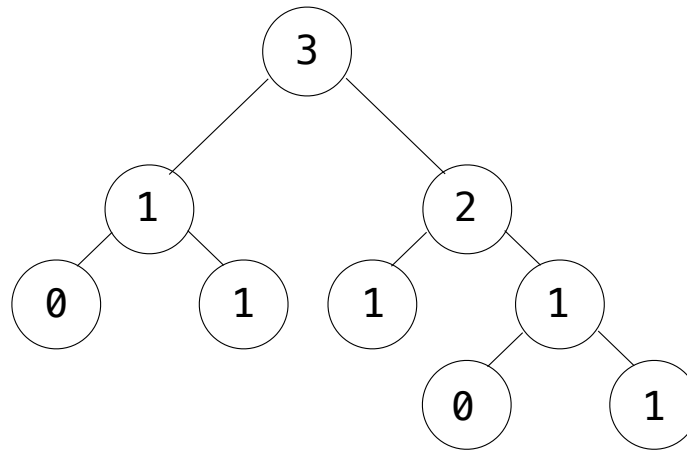
Announcements

Trees

Tree Abstraction



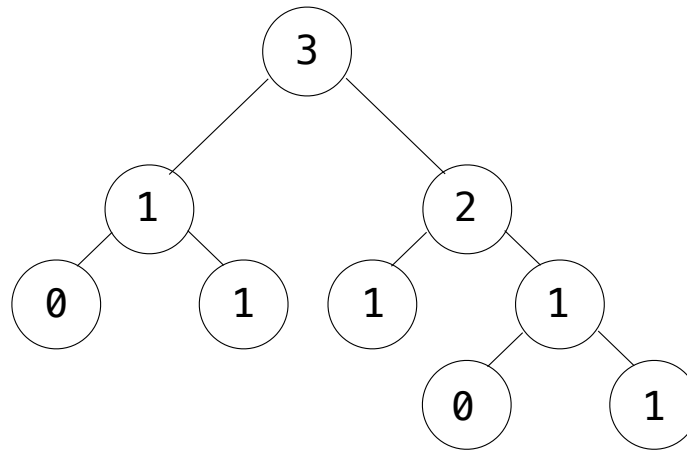
Tree Abstraction



Recursive description (wooden trees):

Relative description (family trees):

Tree Abstraction

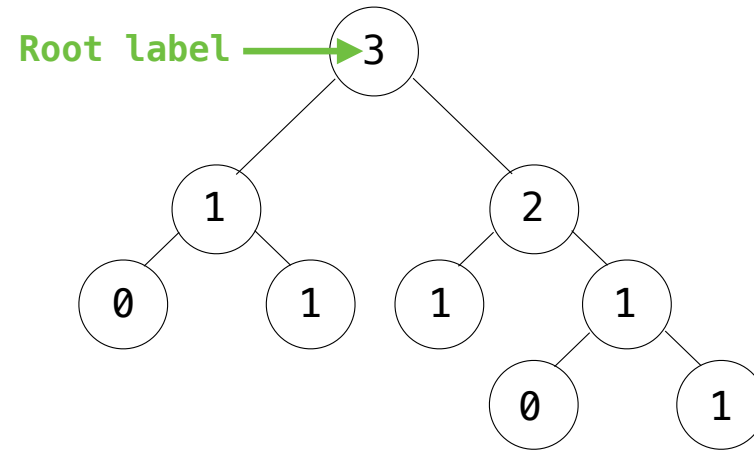


Recursive description (wooden trees):

A **tree** has a **root label** and a list of **branches**

Relative description (family trees):

Tree Abstraction

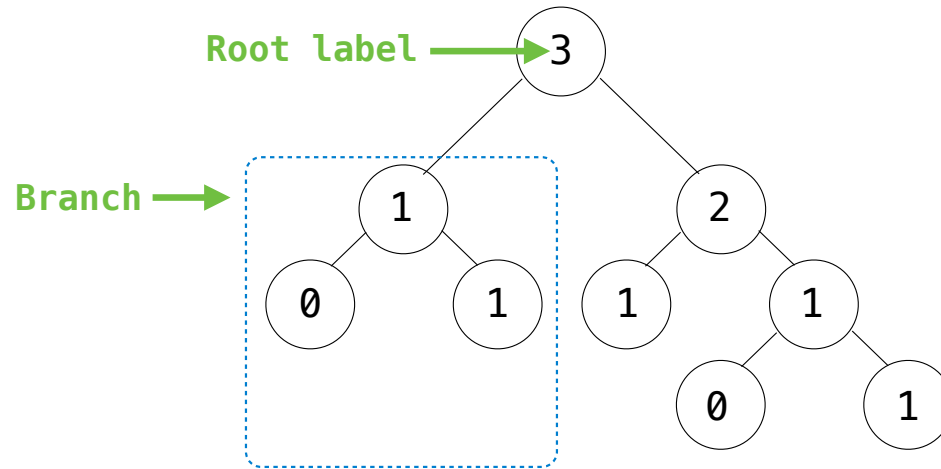


Recursive description (wooden trees):

A **tree** has a **root label** and a list of **branches**

Relative description (family trees):

Tree Abstraction

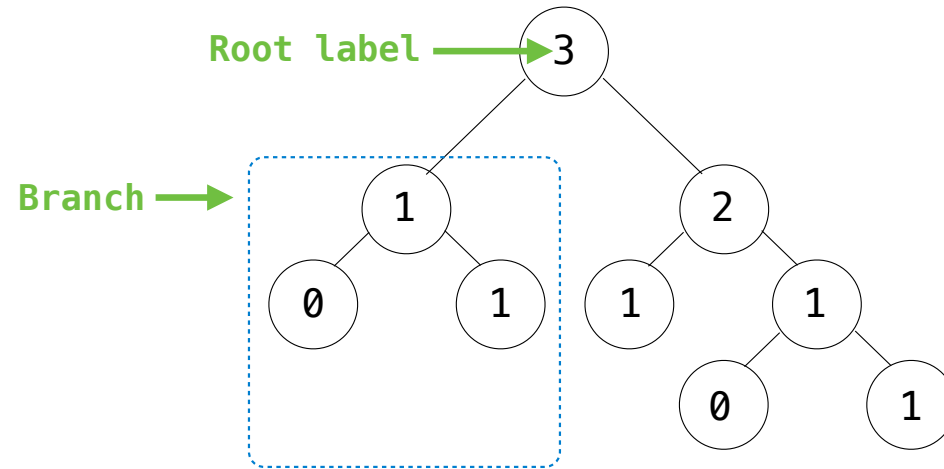


Recursive description (wooden trees):

A **tree** has a **root label** and a list of **branches**

Relative description (family trees):

Tree Abstraction



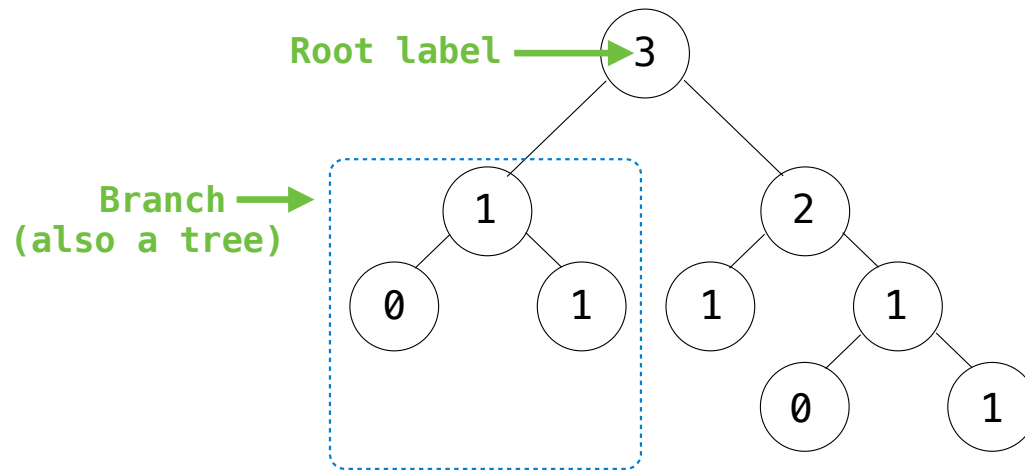
Recursive description (wooden trees):

A **tree** has a **root label** and a list of **branches**

Each **branch** is a **tree**

Relative description (family trees):

Tree Abstraction



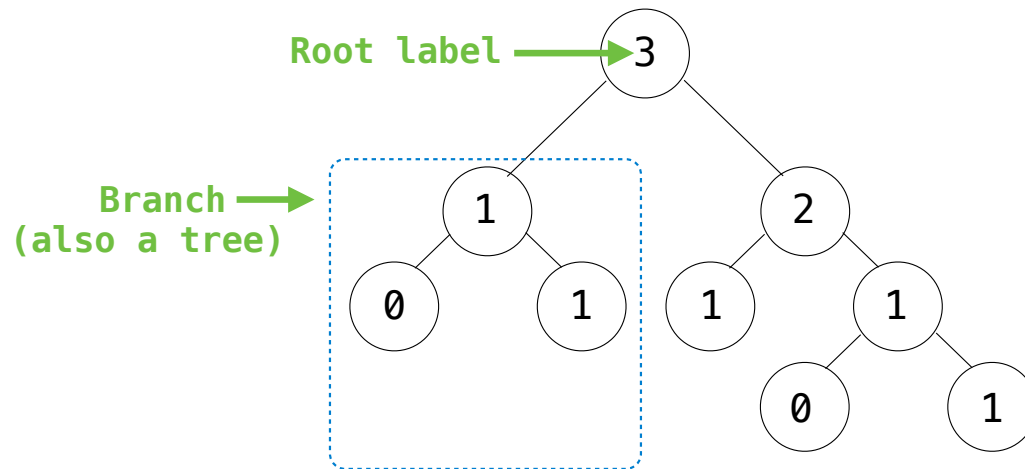
Recursive description (wooden trees):

A **tree** has a **root label** and a list of **branches**

Each **branch** is a **tree**

Relative description (family trees):

Tree Abstraction



Recursive description (wooden trees):

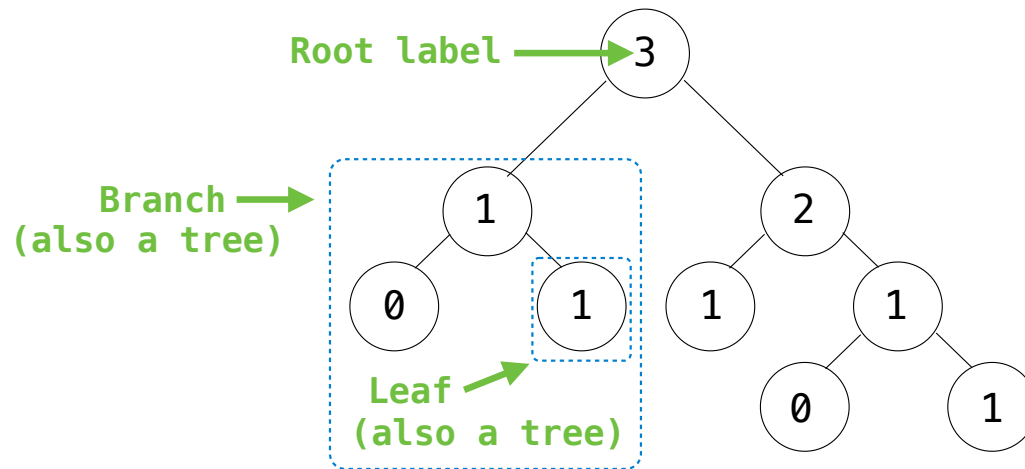
A **tree** has a **root label** and a list of **branches**

Each **branch** is a **tree**

A **tree** with zero **branches** is called a **leaf**

Relative description (family trees):

Tree Abstraction



Recursive description (wooden trees):

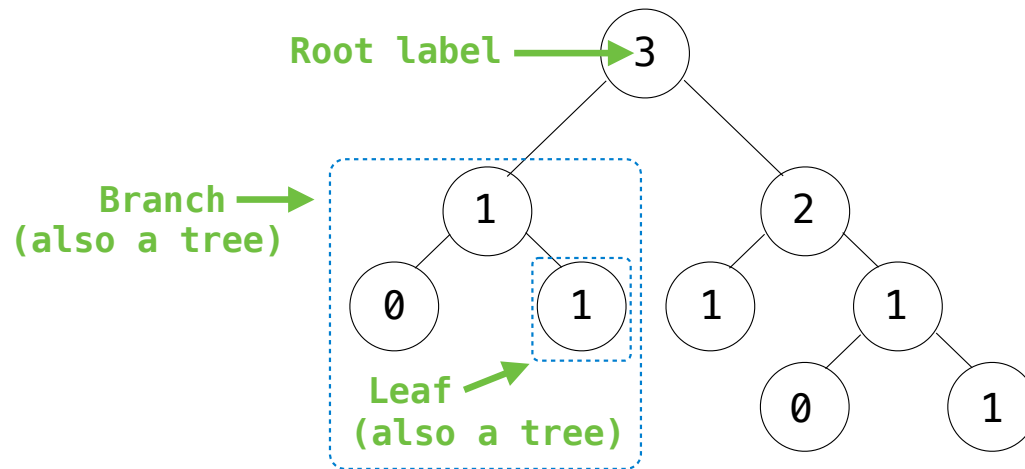
A **tree** has a **root label** and a list of **branches**

Each **branch** is a **tree**

A **tree** with zero **branches** is called a **leaf**

Relative description (family trees):

Tree Abstraction



Recursive description (wooden trees):

A **tree** has a **root label** and a list of **branches**

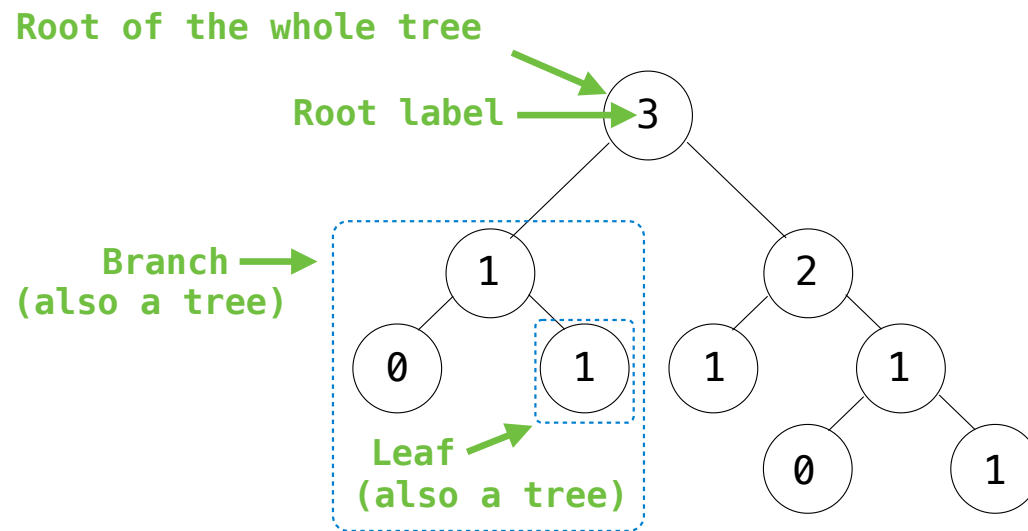
Each **branch** is a **tree**

A **tree** with zero **branches** is called a **leaf**

A **tree** starts at the **root**

Relative description (family trees):

Tree Abstraction



Recursive description (wooden trees):

A **tree** has a **root label** and a list of **branches**

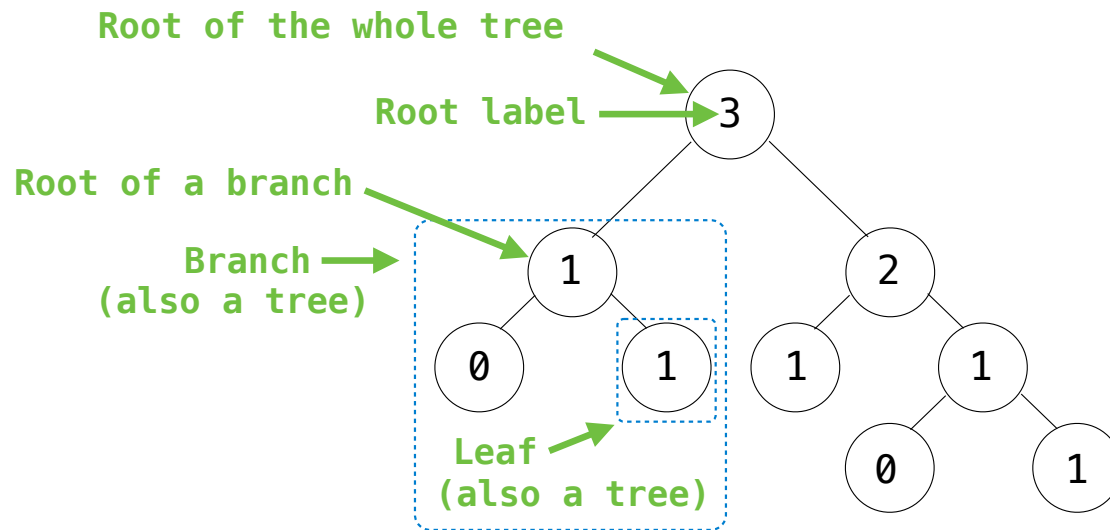
Each **branch** is a **tree**

A **tree** with zero **branches** is called a **leaf**

A **tree** starts at the **root**

Relative description (family trees):

Tree Abstraction



Recursive description (wooden trees):

A **tree** has a **root label** and a list of **branches**

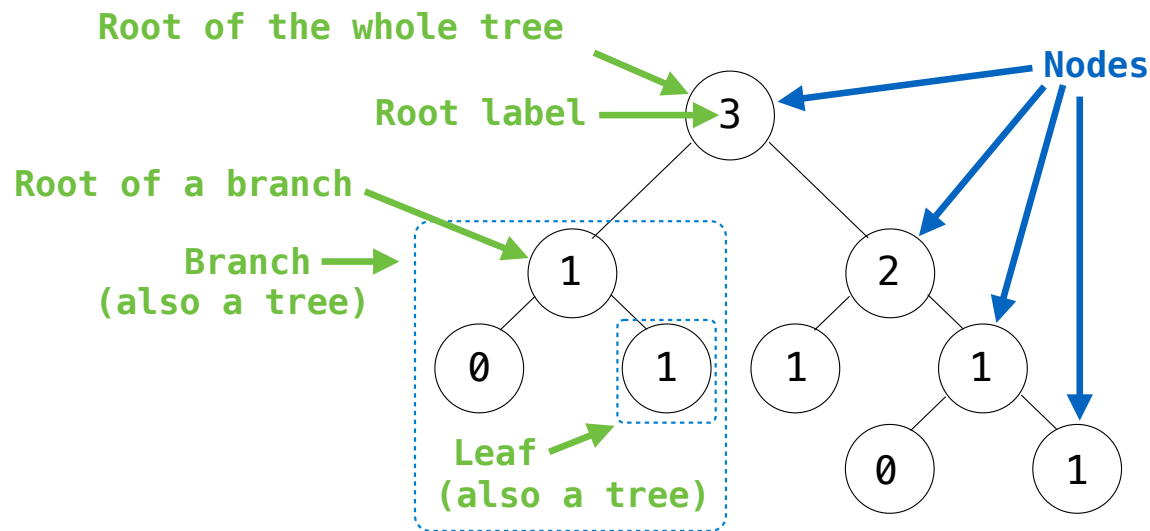
Each **branch** is a **tree**

A **tree** with zero **branches** is called a **leaf**

A **tree** starts at the **root**

Relative description (family trees):

Tree Abstraction



Recursive description (wooden trees):

A **tree** has a **root label** and a list of **branches**

Each **branch** is a **tree**

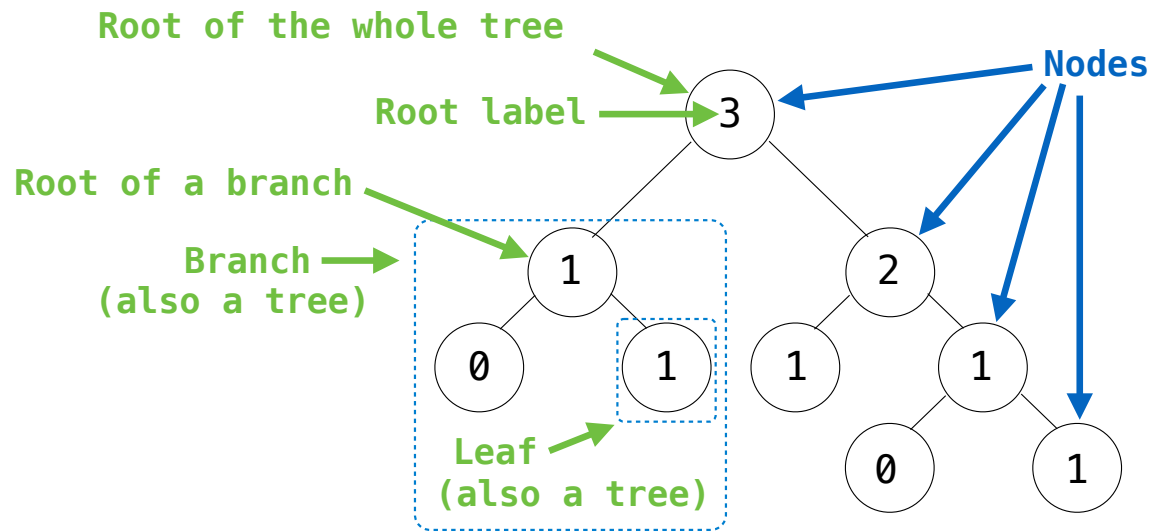
A **tree** with zero **branches** is called a **leaf**

A **tree** starts at the **root**

Relative description (family trees):

Each location in a tree is called a **node**

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**

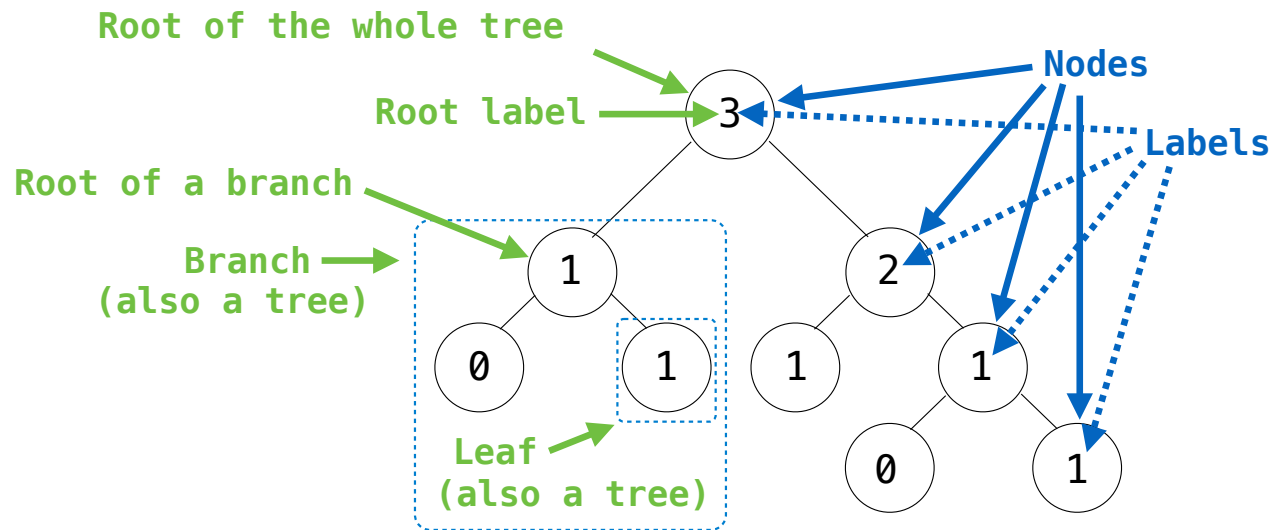
A **tree** starts at the **root**

Relative description (family trees):

Each location in a tree is called a **node**

Each **node** has a **label** that can be any value

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**

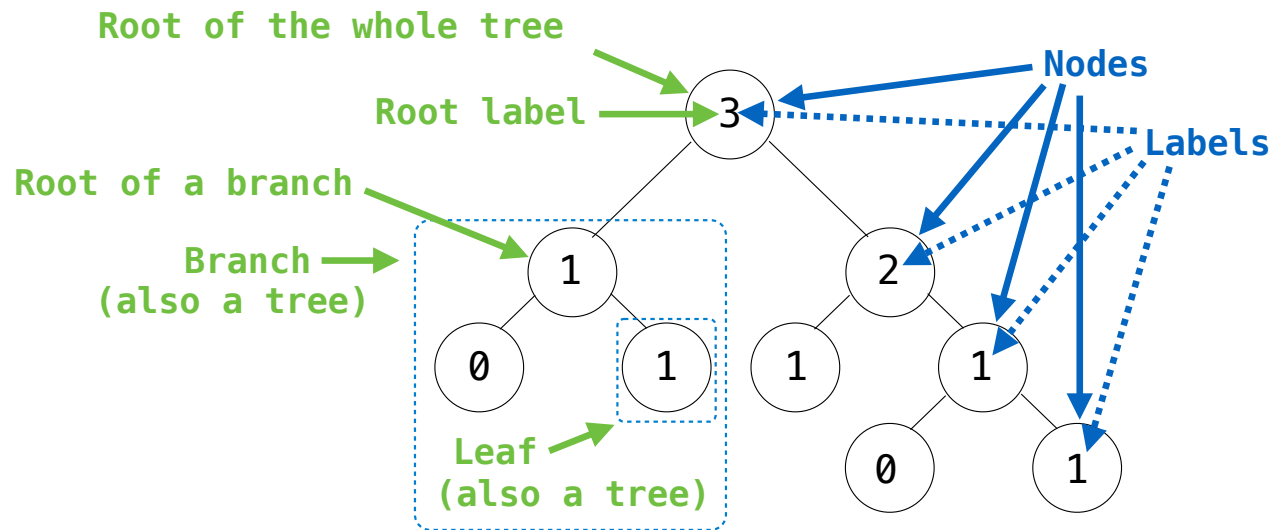
A **tree** starts at the **root**

Relative description (family trees):

Each location in a tree is called a **node**

Each **node** has a **label** that can be any value

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**

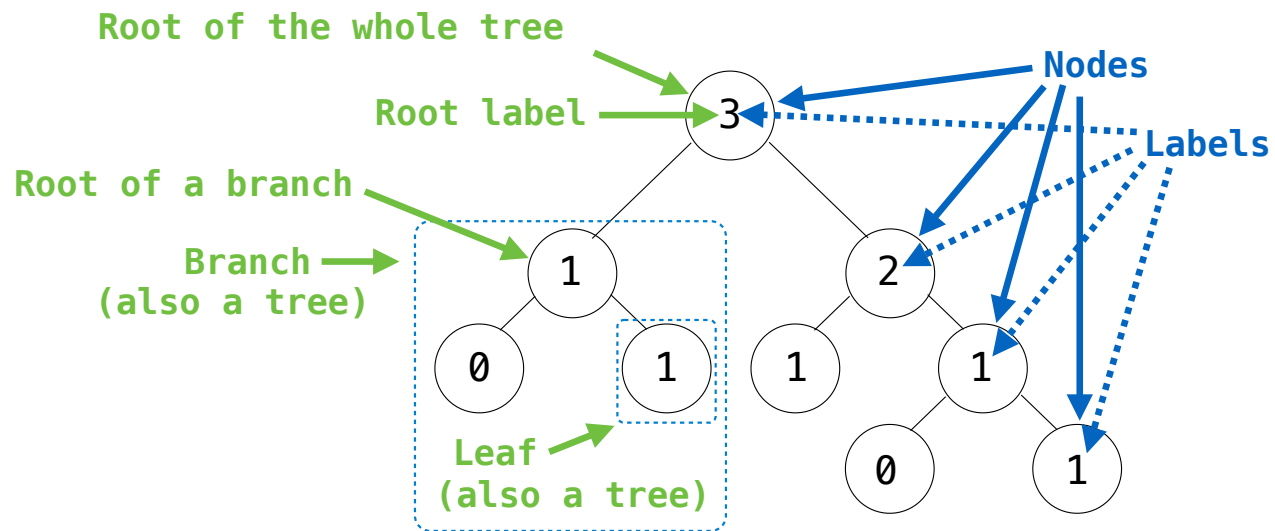
Relative description (family trees):

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

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):

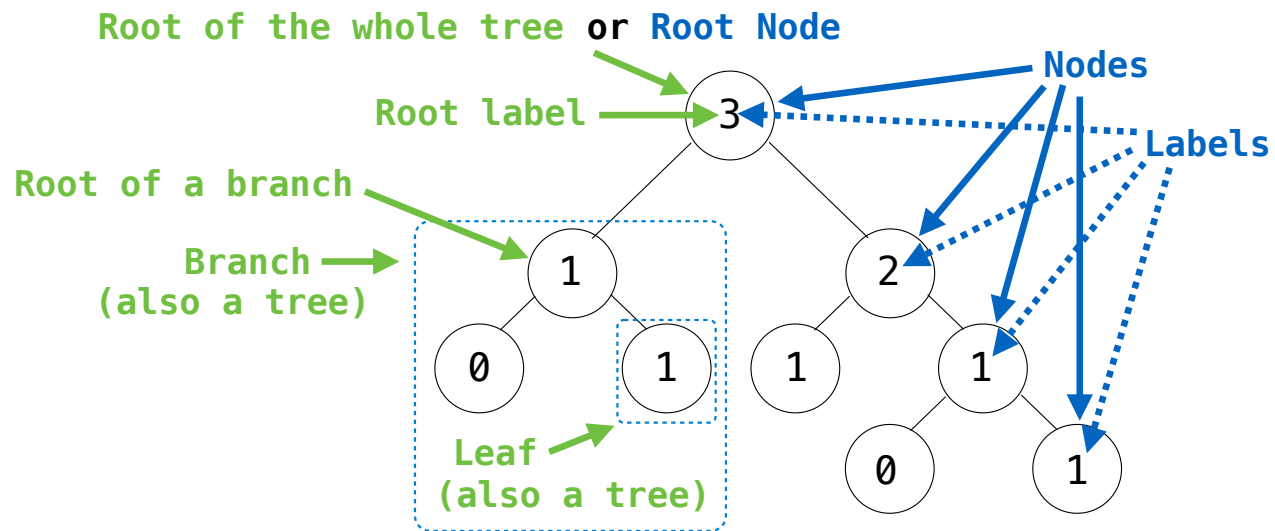
Each location in a tree is called a **node**

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**

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):

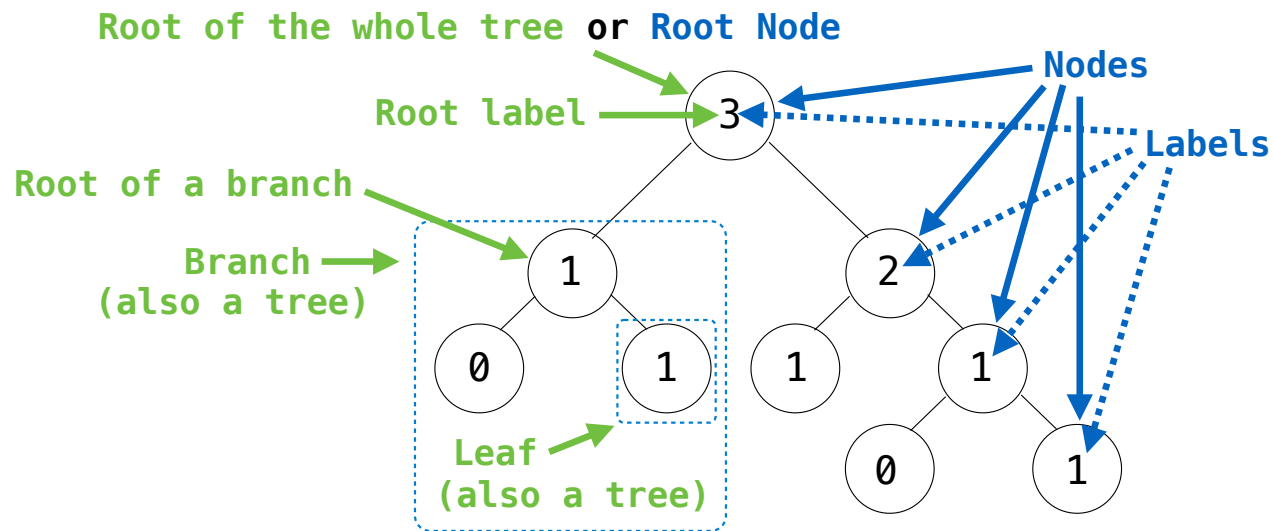
Each location in a tree is called a **node**

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**

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**

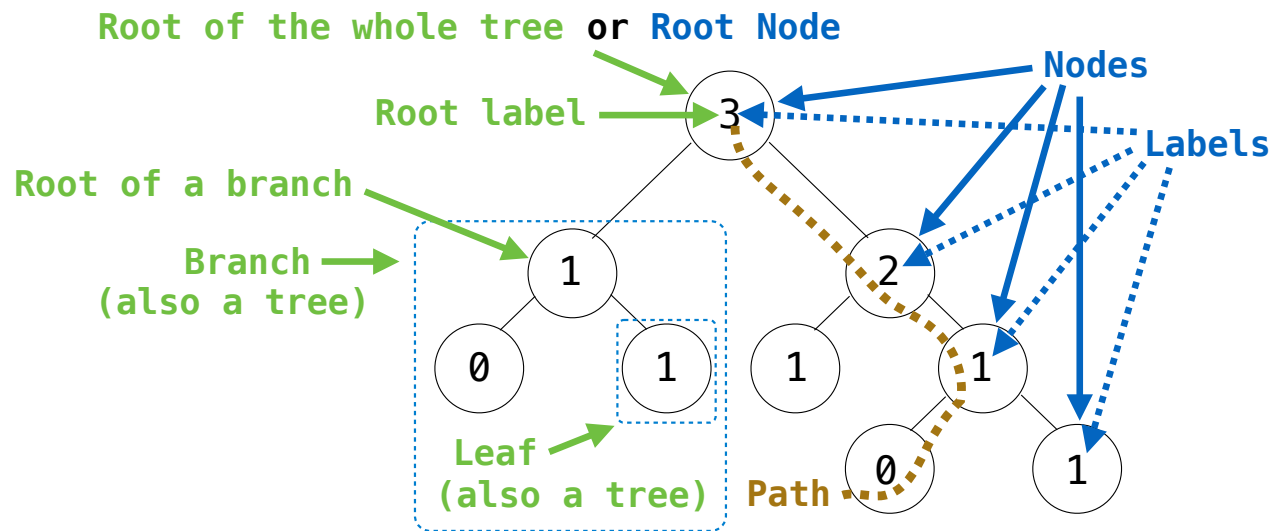
Each **node** has a **label** that can be any value

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"

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"

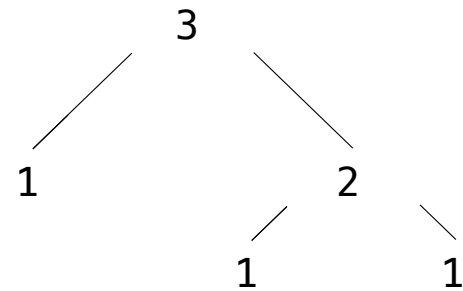
Implementing the Tree Abstraction

Implementing the Tree Abstraction

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

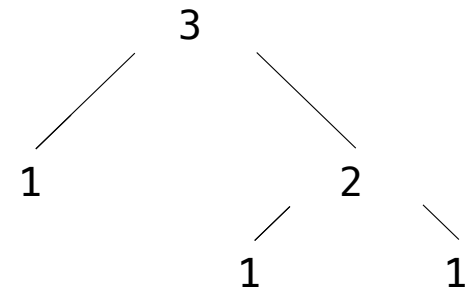
Implementing the Tree Abstraction

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



Implementing the Tree Abstraction

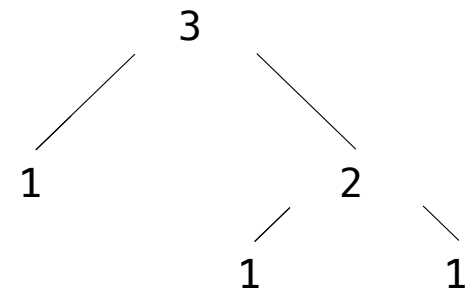
- A **tree** has a root **label** and a list of **branches**
- Each branch is a tree



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

Implementing the Tree Abstraction

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

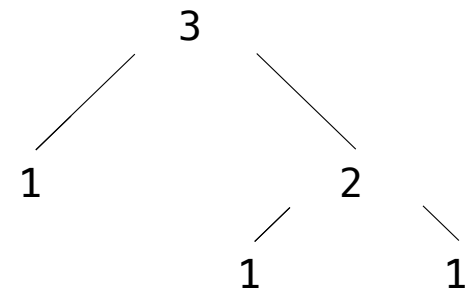


```
>>> tree(3, [tree(1),  
...         tree(2, [tree(1),  
...                 tree(1)])])  
[3, [1], [2, [1], [1]]]
```

Implementing the Tree Abstraction

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

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

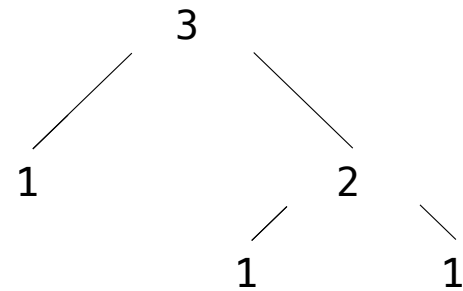


```
>>> tree(3, [tree(1),  
...         tree(2, [tree(1),  
...                 tree(1)])])  
[3, [1], [2, [1], [1]]]
```

Implementing the Tree Abstraction

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

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



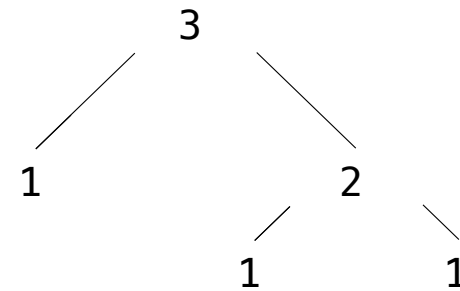
```
>>> tree(3, [tree(1),  
...         tree(2, [tree(1),  
...                 tree(1)])])  
[3, [1], [2, [1], [1]]]
```

Implementing the Tree Abstraction

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

```
def label(tree):
```

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



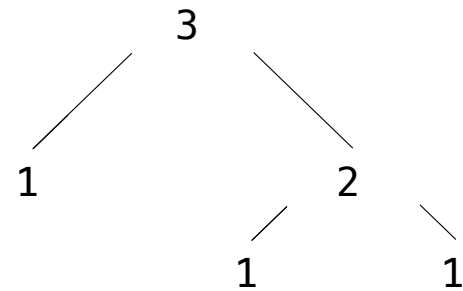
```
>>> tree(3, [tree(1),  
...         tree(2, [tree(1),  
...                 tree(1)])])  
[3, [1], [2, [1], [1]]]
```

Implementing the Tree Abstraction

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

```
def label(tree):  
    return tree[0]
```

- 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]]]
```

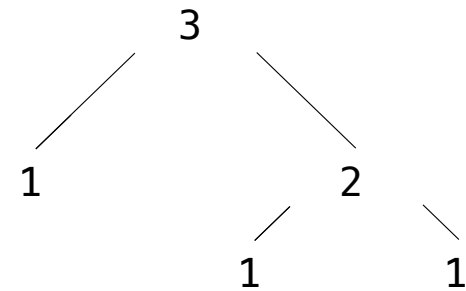

Implementing the Tree Abstraction

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

```
def label(tree):  
    return tree[0]
```

```
def branches(tree):
```

- 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]]]
```

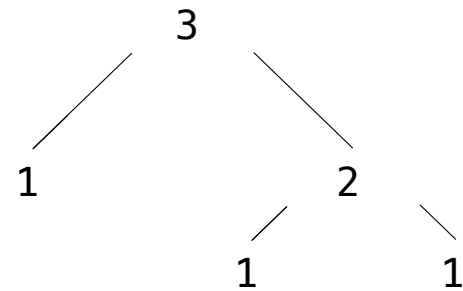
Implementing the Tree Abstraction

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

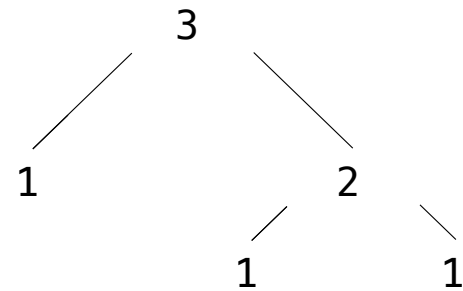
Implementing the Tree Abstraction

```
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:]
```

- 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

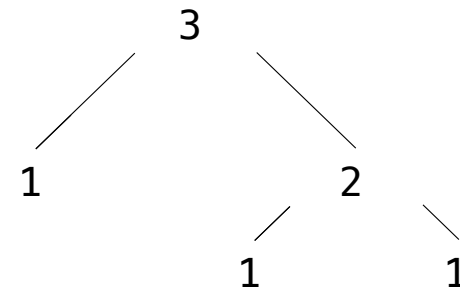
```
def tree(label, branches=[]):  
    for branch in branches:  
        assert is_tree(branch)  
    return [label] + list(branches)
```

```
def label(tree):  
    return tree[0]
```

Creates a list
from a sequence
of branches

```
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)
```

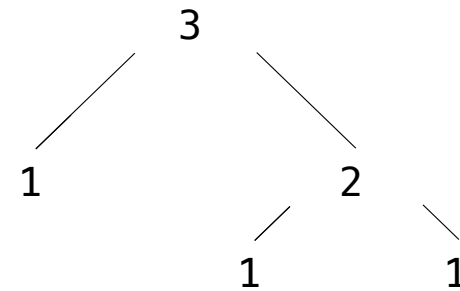
Verifies the tree definition

```
def label(tree):  
    return tree[0]
```

Creates a list from a sequence of branches

```
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)
```

Verifies the tree definition

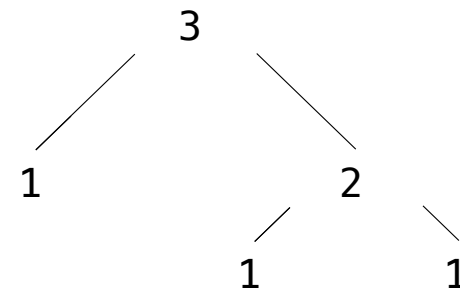
```
def label(tree):  
    return tree[0]
```

Creates a list from a sequence of branches

```
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
```

- 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)
```

Verifies the tree definition

```
def label(tree):  
    return tree[0]
```

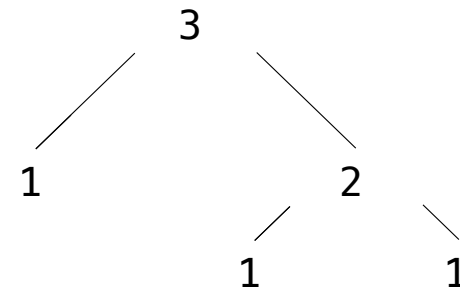
Creates a list from a sequence of branches

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

Verifies that tree is bound to a list

```
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
```

- 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)
```

Verifies the tree definition

```
def label(tree):  
    return tree[0]
```

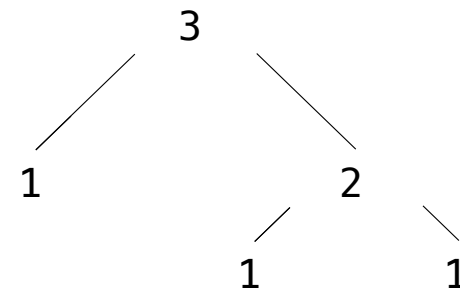
Creates a list from a sequence of branches

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

Verifies that tree is bound to a list

```
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
```

- 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)
```


Implementing the Tree Abstraction

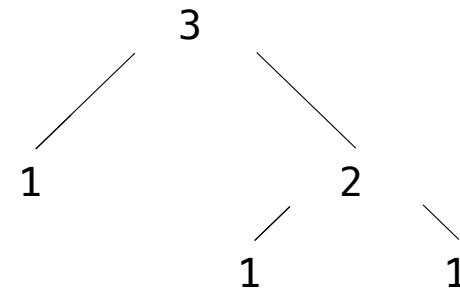
```
def tree(label, branches=[]):  
    for branch in branches:  
        assert is_tree(branch)  
    return [label] + list(branches)  
  
def label(tree):  
    return tree[0]  
  
def branches(tree):  
    return tree[1:]  
  
def is_tree(tree):  
    if type(tree) != list or len(tree) < 1:  
        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)      (Demo)
```

Tree Processing

Tree Processing

(Demo)

Tree Processing Uses Recursion

Tree Processing Uses Recursion

```
def count_leaves(t):  
    """Count the leaves of a tree."""
```

Tree Processing Uses Recursion

Processing a leaf is often the base case of a tree processing function

```
def count_leaves(t):  
    """Count the leaves of a tree."""
```

Tree Processing Uses Recursion

Processing a leaf is often the base case of a tree processing function

```
def count_leaves(t):  
    """Count the leaves of a tree."""  
    if is_leaf(t):  
        return 1
```

Tree Processing Uses Recursion

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

```
def count_leaves(t):  
    """Count the leaves of a tree."""  
    if is_leaf(t):  
        return 1
```


Tree Processing Uses Recursion

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

```
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)]
```

Tree Processing Uses Recursion

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

```
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)
```

Tree Processing Uses Recursion

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

```
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)
```

(Demo)

Discussion Question

Discussion Question

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

Discussion Question

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

```
def leaves(tree):  
    """Return a list containing the leaf labels of tree.  
  
    >>> leaves(fib_tree(5))  
    [1, 0, 1, 0, 1, 1, 0, 1]  
    """
```

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

```
def leaves(tree):  
    """Return a list containing the leaf labels of tree.  
  
    >>> leaves(fib_tree(5))  
    [1, 0, 1, 0, 1, 1, 0, 1]  
    """
```

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

```
>>> sum([ [1], [2, 3], [4] ], [])  
def leaves(tree):  
    """Return a list containing the leaf labels of tree.  
  
    >>> leaves(fib_tree(5))  
    [1, 0, 1, 0, 1, 1, 0, 1]  
    """
```


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

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

def leaves(tree):
    """Return a list containing the leaf labels of tree.

    >>> leaves(fib_tree(5))
    [1, 0, 1, 0, 1, 1, 0, 1]
    """
```

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

```
>>> sum([ [1], [2, 3], [4] ], [])    def leaves(tree):
[1, 2, 3, 4]                          """Return a list containing the leaf labels of tree.
>>> sum([ [1] ], [])                >>> leaves(fib_tree(5))
                                     [1, 0, 1, 0, 1, 1, 0, 1]
                                     """
```

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

```
>>> sum([ [1], [2, 3], [4] ], [])  def leaves(tree):
[1, 2, 3, 4]                        """Return a list containing the leaf labels of tree.
>>> sum([ [1] ], [])              >>> leaves(fib_tree(5))
[1]                                 [1, 0, 1, 0, 1, 1, 0, 1]
                                   """
```

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

```
>>> sum([ [1], [2, 3], [4] ], [])    def leaves(tree):
[1, 2, 3, 4]                          """Return a list containing the leaf labels of tree.
>>> sum([ [1] ], [])
[1]
>>> sum([ [[1]], [2] ], [])          >>> leaves(fib_tree(5))
                                     [1, 0, 1, 0, 1, 1, 0, 1]
                                     """
```

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

```
>>> sum([ [1], [2, 3], [4] ], [])  def leaves(tree):
[1, 2, 3, 4]                        """Return a list containing the leaf labels of tree.
>>> sum([ [1] ], [])
[1]
>>> sum([ [[1]], [2] ], [])        >>> leaves(fib_tree(5))
[[1], 2]                          [1, 0, 1, 0, 1, 1, 0, 1]
                                     """
```

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

```
>>> sum([ [1], [2, 3], [4] ], [])  def leaves(tree):
[1, 2, 3, 4]                        """Return a list containing the leaf labels of tree.
>>> sum([ [1] ], [])
[1]
>>> sum([ [[1]], [2] ], [])        >>> leaves(fib_tree(5))
[[1], 2]                          [1, 0, 1, 0, 1, 1, 0, 1]
                                     """
```

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

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

```
def leaves(tree):
    """Return a list containing the leaf labels of tree.

    >>> leaves(fib_tree(5))
    [1, 0, 1, 0, 1, 1, 0, 1]
    """
    if is_leaf(tree):
        return [label(tree)]
    else:
        return sum(_____, [])
```

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

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

```
def leaves(tree):
    """Return a list containing the leaf labels of tree.

    >>> leaves(fib_tree(5))
    [1, 0, 1, 0, 1, 1, 0, 1]
    """
    if is_leaf(tree):
        return [label(tree)]
    else:
        return sum(_____, [])
```

<code>branches(tree)</code>	<code>[b for b in branches(tree)]</code>
<code>leaves(tree)</code>	<code>[s for s in leaves(tree)]</code>
<code>[branches(b) for b in branches(tree)]</code>	<code>[branches(s) for s in leaves(tree)]</code>
<code>[leaves(b) for b in branches(tree)]</code>	<code>[leaves(s) for s in leaves(tree)]</code>

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

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

```
def leaves(tree):
    """Return a list containing the leaf labels of tree.

    >>> leaves(fib_tree(5))
    [1, 0, 1, 0, 1, 1, 0, 1]
    """
    if is_leaf(tree):
        return [label(tree)]
    else:
        return sum(List of leaf labels for each branch, [])
```

<code>branches(tree)</code>	<code>[b for b in branches(tree)]</code>
<code>leaves(tree)</code>	<code>[s for s in leaves(tree)]</code>
<code>[branches(b) for b in branches(tree)]</code>	<code>[branches(s) for s in leaves(tree)]</code>
<code>[leaves(b) for b in branches(tree)]</code>	<code>[leaves(s) for s in leaves(tree)]</code>

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

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

```
def leaves(tree):
    """Return a list containing the leaf labels of tree.

    >>> leaves(fib_tree(5))
    [1, 0, 1, 0, 1, 1, 0, 1]
    """
    if is_leaf(tree):
        return [label(tree)]
    else:
        return sum(List of leaf labels for each branch, [])
```

`branches(tree)`

`leaves(tree)`

`[branches(b) for b in branches(tree)]`

`[leaves(b) for b in branches(tree)]`

`[b for b in branches(tree)]`

`[s for s in leaves(tree)]`

`[branches(s) for s in leaves(tree)]`

`[leaves(s) for s in leaves(tree)]`

Creating Trees

Creating Trees

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

Creating Trees

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

```
def increment_leaves(t):  
    """Return a tree like t but with leaf labels incremented."""
```

Creating Trees

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

```
def increment_leaves(t):  
    """Return a tree like t but with leaf labels incremented."""  
    if is_leaf(t):  
        return tree(label(t) + 1)
```

Creating Trees

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

```
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)
```

Creating Trees

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

```
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)

def increment(t):
    """Return a tree like t but with all labels incremented."""
```


Creating Trees

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

```
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)

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)])
```

Example: Printing Trees

(Demo)

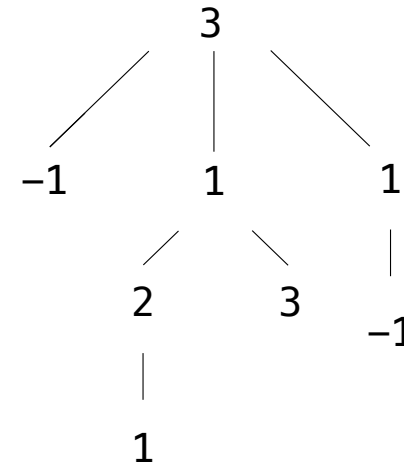
Example: Summing Paths

(Demo)

Example: Counting Paths

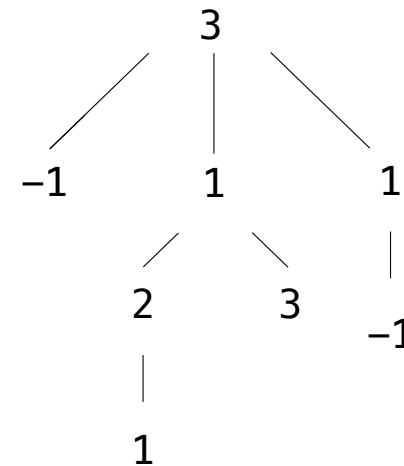
Count Paths that have a Total Label Sum

```
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 _____:  
        found = _____  
    else:  
        _____  
    return found + _____ ([_____ for b in branches(t)])
```



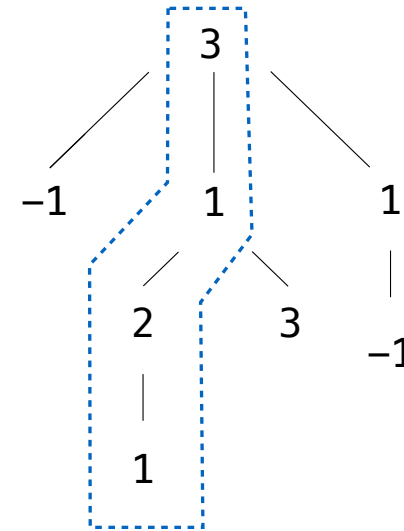
Count Paths that have a Total Label Sum

```
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 _____:  
        found = _____  
    else:  
        _____  
  
    return found + _____ ([_____ for b in branches(t)])
```



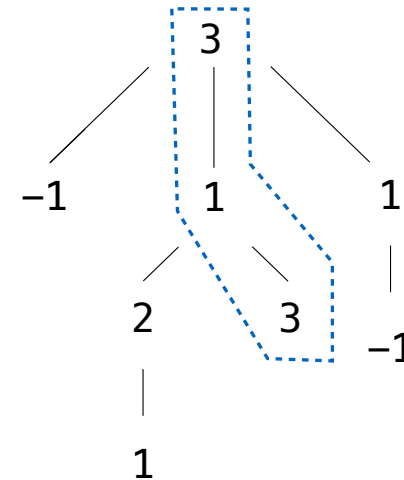
Count Paths that have a Total Label Sum

```
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 _____:  
        found = _____  
    else:  
        _____  
  
    return found + _____ ([_____ for b in branches(t)])
```



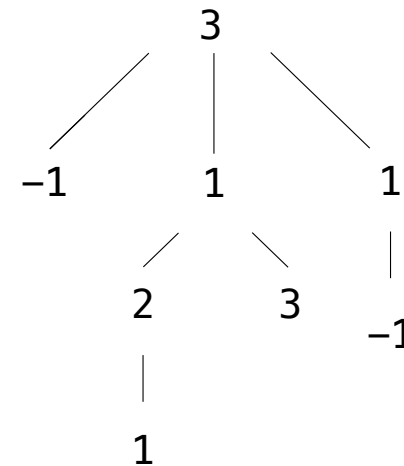
Count Paths that have a Total Label Sum

```
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 _____:  
        found = _____  
    else:  
        _____  
    return found + _____ ([_____ for b in branches(t)])
```



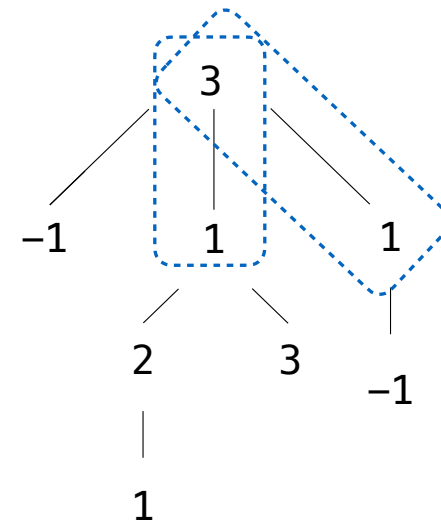
Count Paths that have a Total Label Sum

```
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 _____:  
        found = _____  
    else:  
        _____  
  
    return found + _____ ([_____ 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 _____:  
        found = _____  
    else:  
        _____  
    return found + _____ ([_____ for b in branches(t)])
```

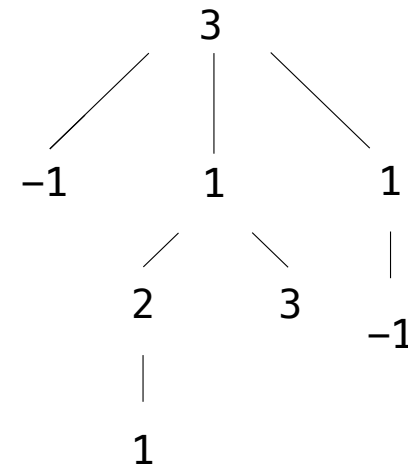


Count Paths that have a Total Label Sum

```
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.

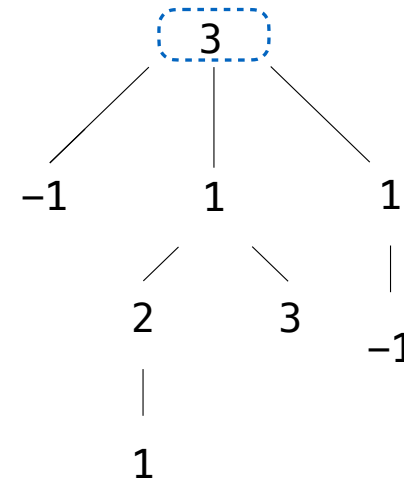
    >>> t = tree(3, [tree(-1), tree(1, [tree(2, [tree(1)]), tree(3)]), tree(1, [tree(-1)])])
    >>> count_paths(t, 3) ◀
    2
    >>> count_paths(t, 4)
    2
    >>> count_paths(t, 5)
    0
    >>> count_paths(t, 6)
    1
    >>> count_paths(t, 7)
    2
    """
    if _____:
        found = _____
    else:
        _____

    return found + _____ ([_____ for b in branches(t)])
```



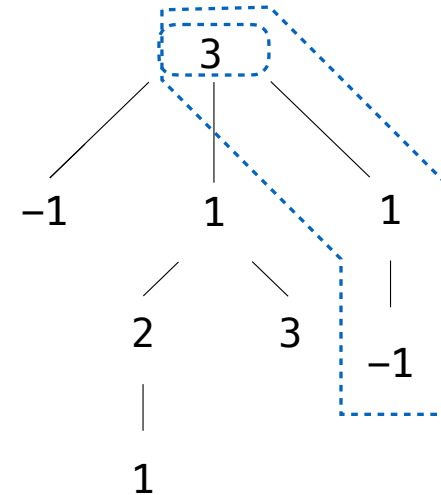
Count Paths that have a Total Label Sum

```
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 _____:  
        found = _____  
    else:  
        _____  
  
    return found + _____ ([_____ for b in branches(t)])
```



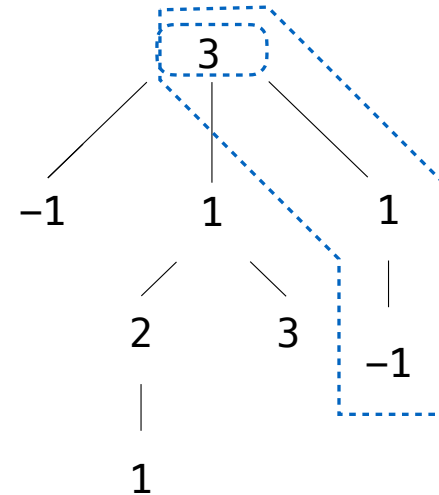
Count Paths that have a Total Label Sum

```
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 _____:  
        found = _____  
    else:  
        _____  
    return found + _____ ([_____ for b in branches(t)])
```



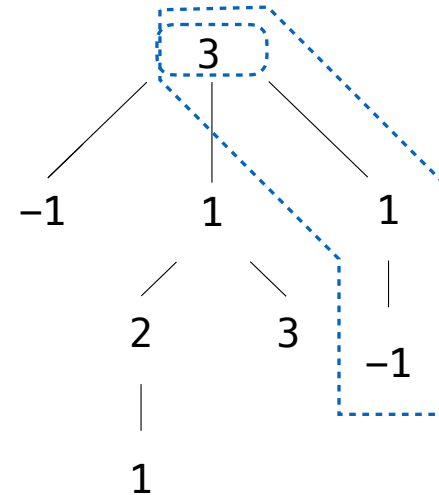
Count Paths that have a Total Label Sum

```
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 = _____  
    else:  
        _____  
    return found + _____ ([_____ for b in branches(t)])
```



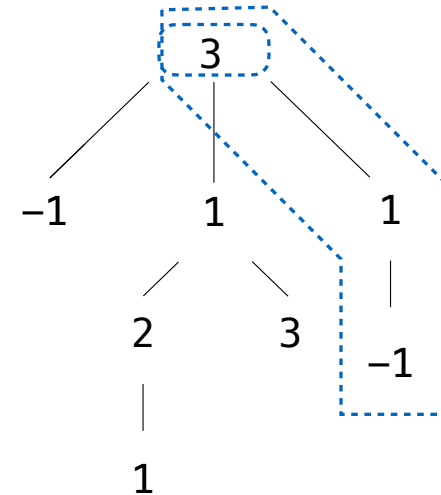
Count Paths that have a Total Label Sum

```
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 = _____  
    else:  
        _____  
  
    return found + _____ ([_____ for b in branches(t)])
```



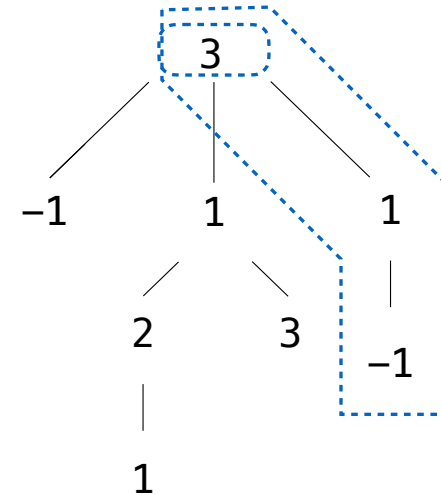
Count Paths that have a Total Label Sum

```
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 + count_paths(t, total - label(t)) 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) 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)])
```

