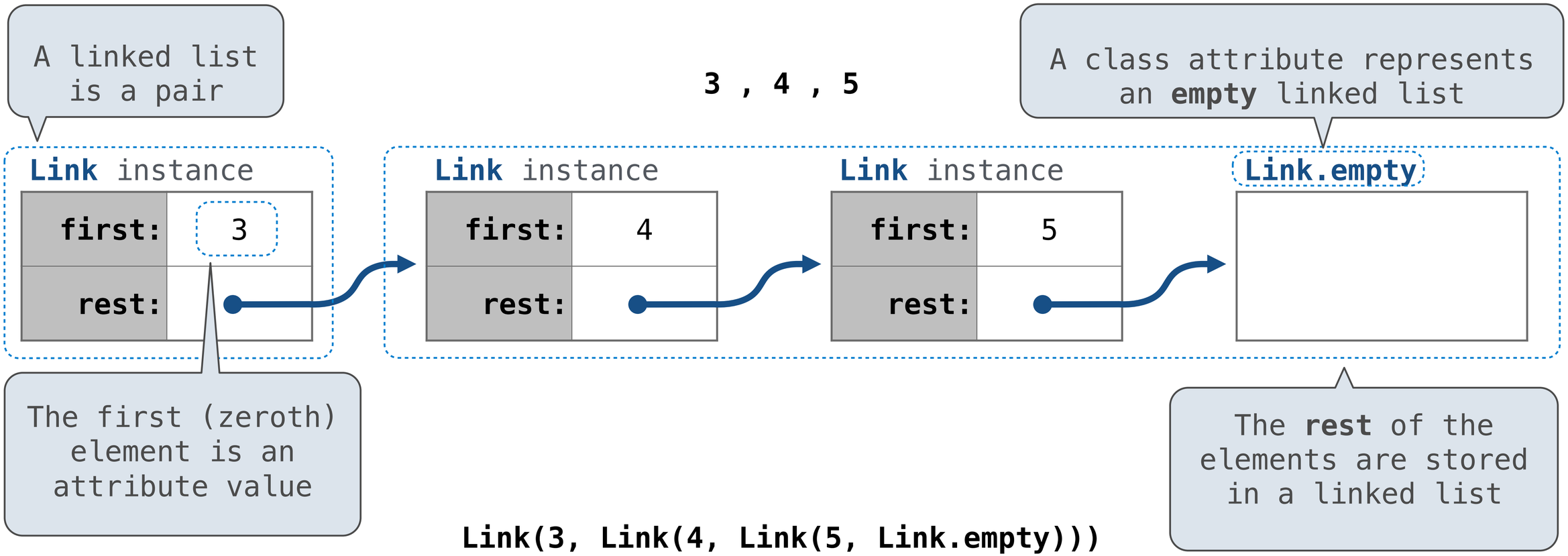
Linked Lists

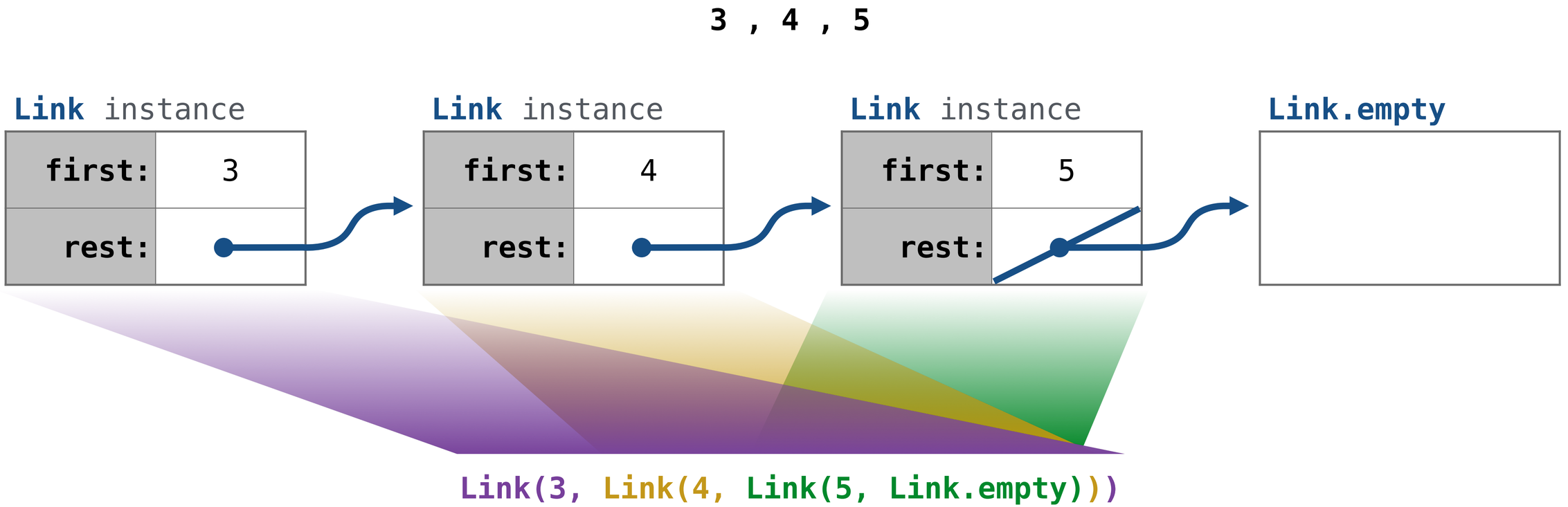
Linked List Structure

A linked list is either empty **or** a first value and the rest of the linked list



Linked List Structure

A linked list is either empty **or** a first value and the rest of the linked list



Linked List Class

Linked list class: attributes are passed to `__init__`

```
class Link:

    empty = ()

    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest
```

Some zero-length sequence

Returns whether rest is a Link

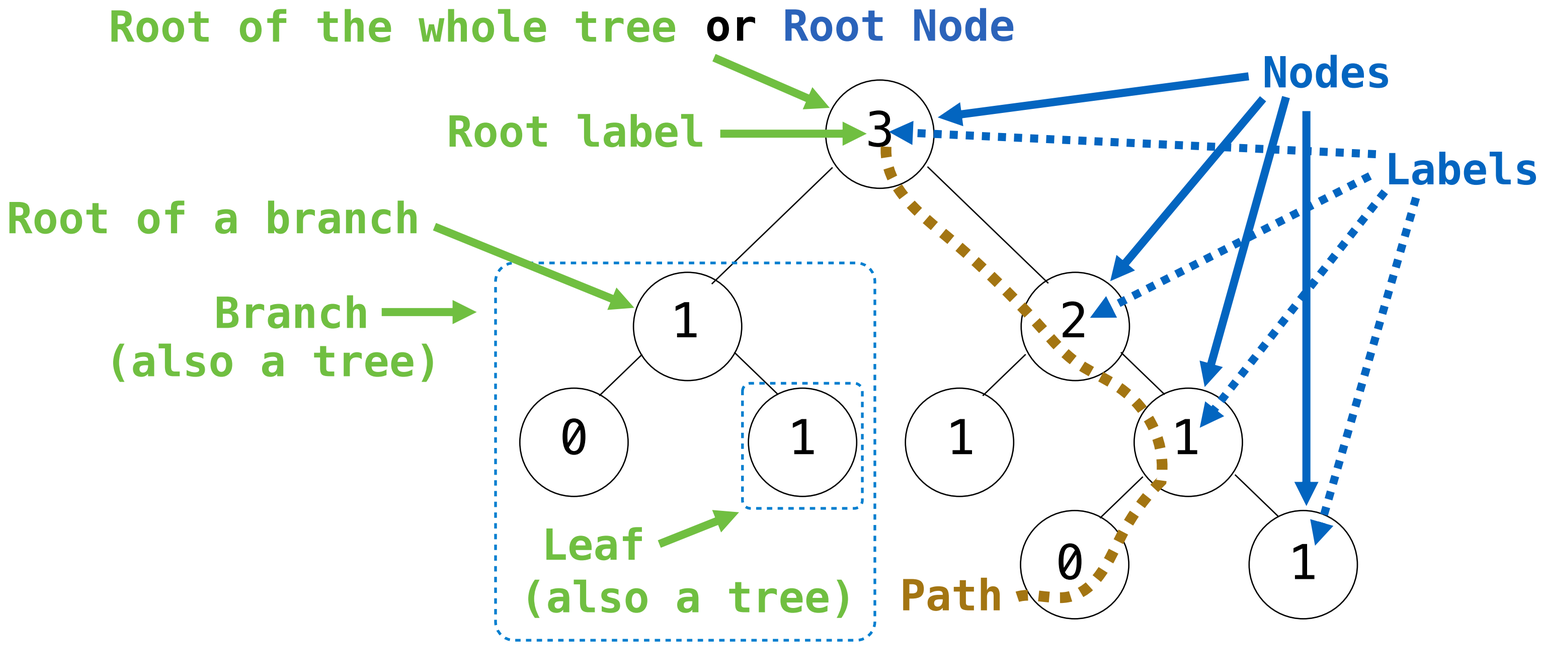
`help(isinstance)`: Return whether an object is an instance of a class or of a subclass thereof.

```
Link(3, Link(4, Link(5              )))
```

(Demo)

Tree Class

Tree Abstraction (Review)



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 Class

A Tree has a label 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])
```

(Demo)

```
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:]
```

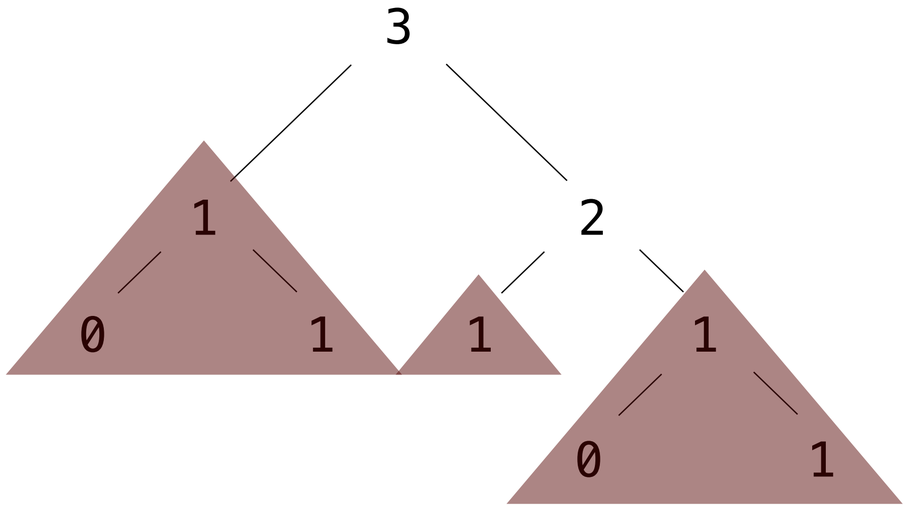
```
def fib_tree(n):
    if n == 0 or n == 1:
        return tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        fib_n = label(left) + label(right)
        return tree(fib_n, [left, right])
```

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 = [b for b in t.branches if b.label != n]
    for b in t.branches:
        prune(b, n)
```

(Demo)

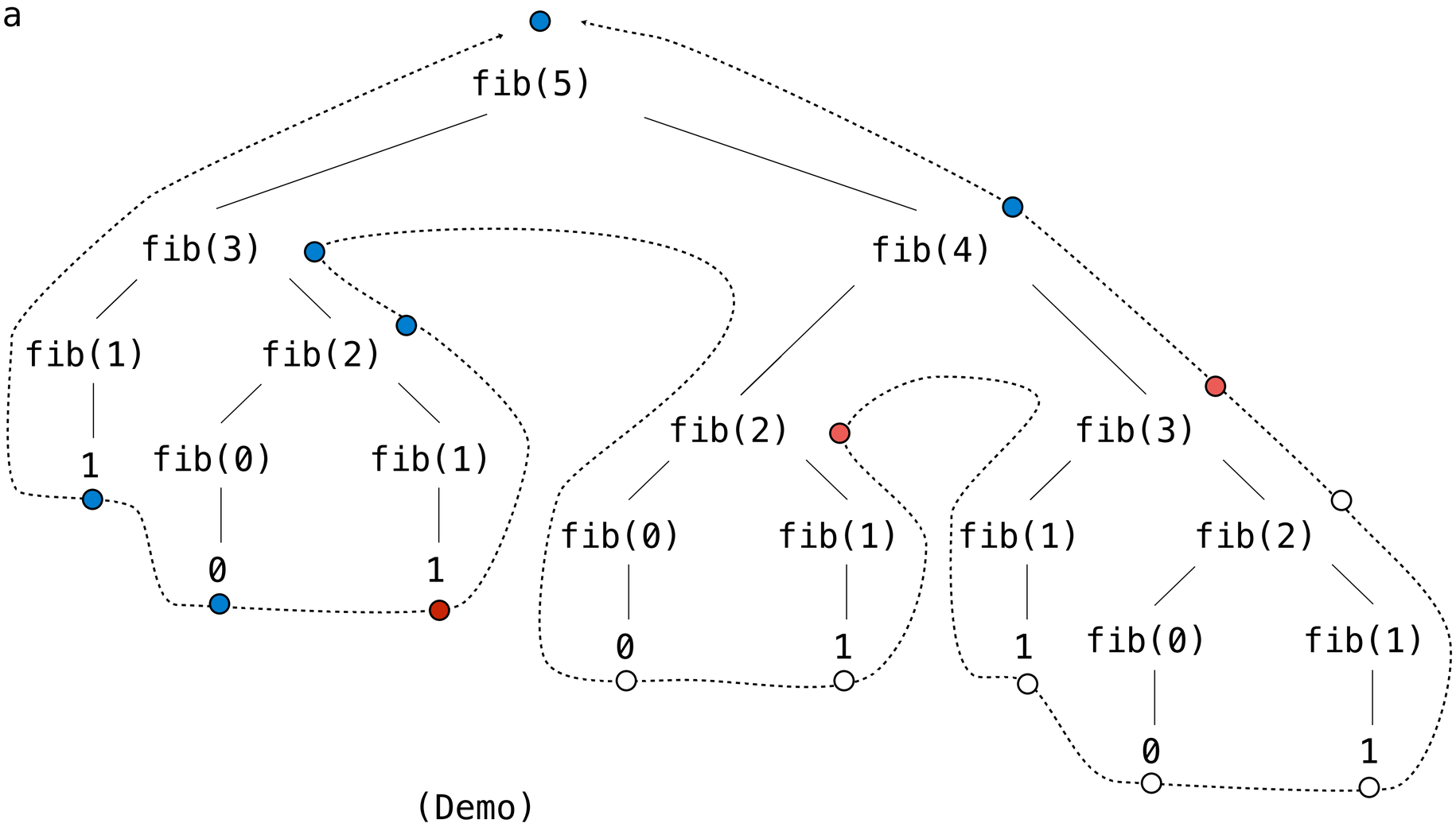
Example: Pruning Trees

Removing subtrees from a tree is called *pruning*

Prune branches before recursive processing

Memoization:

- Returned by fib
- Found in cache
- Skipped



(Demo)