Linked Lists & Trees

By Chris Allsman from Berkeley cs61a

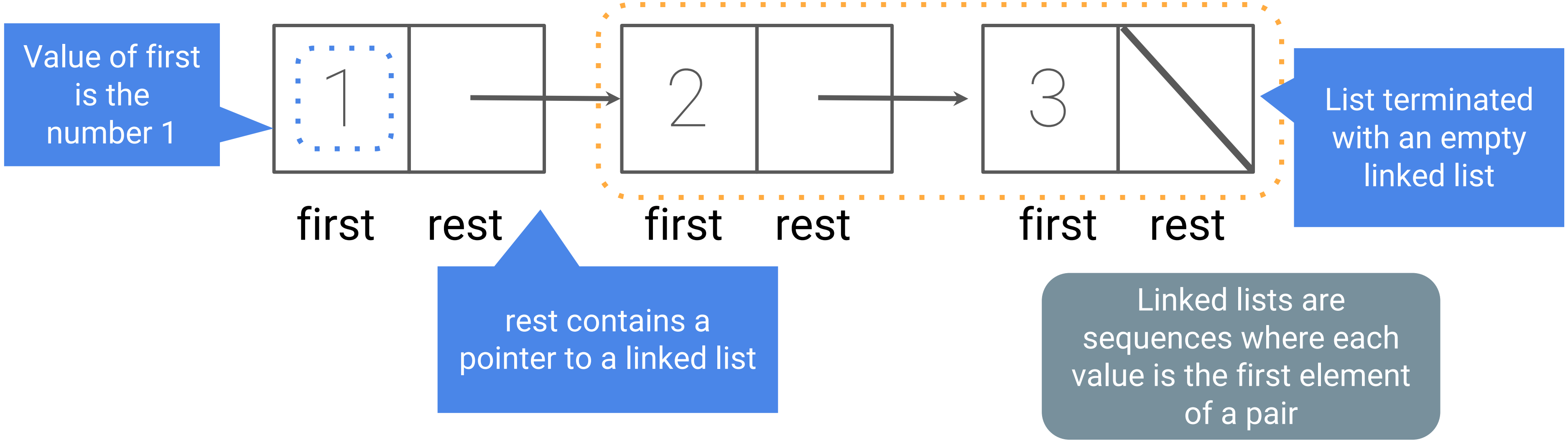
Linked Lists

- A simple but powerful data structure
- Can be used to implement other data structures, e.g., stack, queues
- Fast insertions/deletions, etc.

Linked List Definition

A Linked List is either:

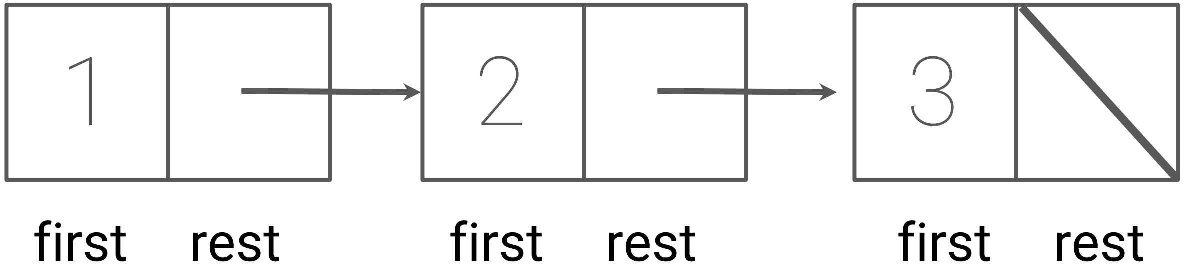
- Empty
- Composed of a first element and the rest of the linked list


Value of first is the number 1
1
2
3
first
rest
first
rest
first
rest
List terminated with an empty linked list
rest contains a pointer to a linked list
Linked lists are sequences where each value is the first element of a pair

Creating Linked Lists

Demo_1

We'll define a linked list recursively by making a constructor that takes in a first and rest value



```
Link(1 , ___________________________)
Link(1 , Link(2, ___________________))
Link(1 , Link(2, Link(3, empty linked list)))
```

The Link Class

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest
>>> lnk = Link(5, Link(6, Link(7)))
>>> lnk.rest.rest.first
7
>>> lnk.rest.rest.rest is Link.empty
True
```

You should not assume the represen-tation here. It could be 'I'm empty"'

Rest defaults to the empty list

.first -> lst[0]
.rest -> lst[1:]
lnk is Link.empty -> not lst

.first gives elements in the list, .rest traverses

Compare to empty list

You Try:

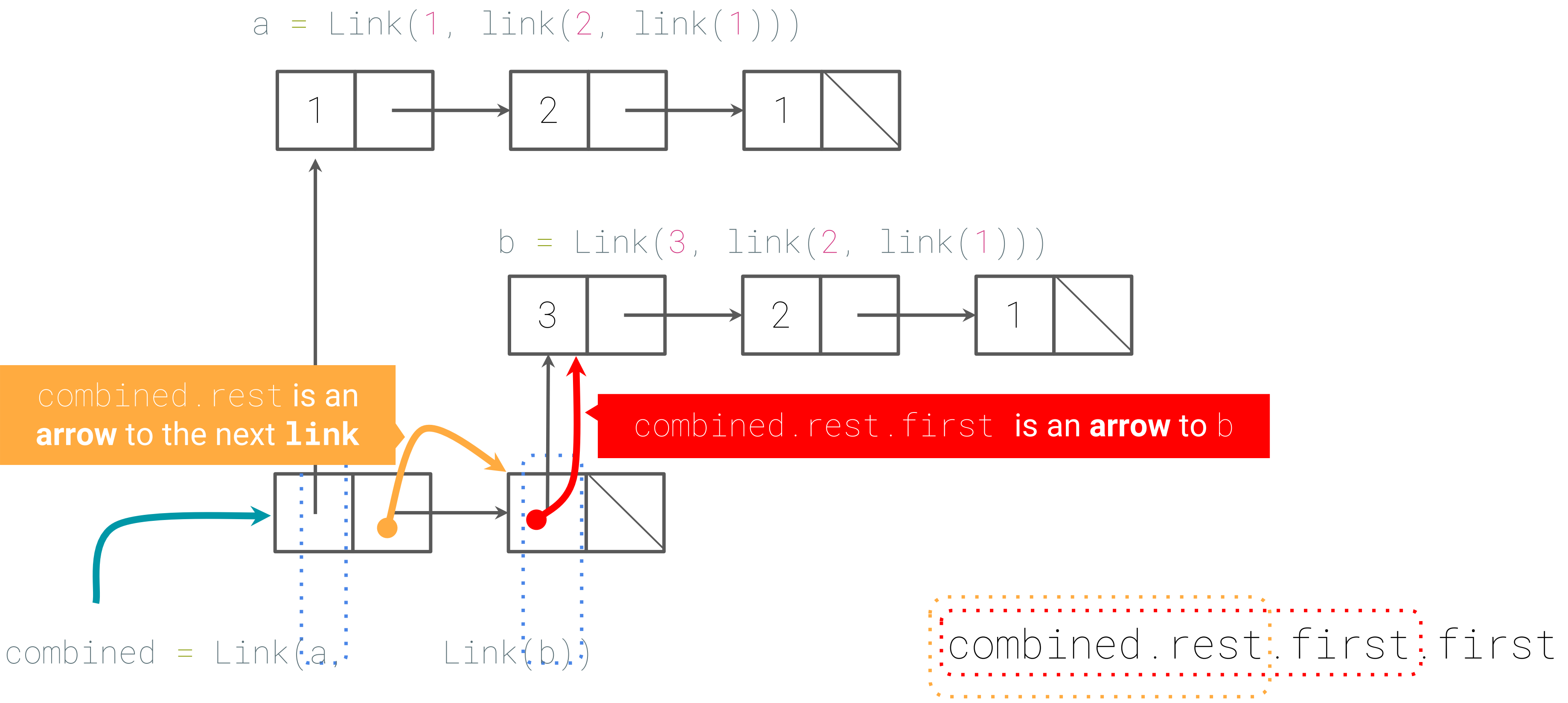
```
class Link:
    empty = ()
    def __init__(self, first,
                       rest=empty):
        self.first = first
        self.rest = rest
```

```
>>> a = Link(1, Link(2, Link(1)))
>>> b = Link(3, Link(2, Link(1)))
>>> combined = Link(a, Link(b))
```

How would you retrieve the element 3?

1. combined.rest.first.rest
2. combined.rest.rest.first
3. combined.rest.first.first
4. combined.first.rest.rest
5. combined.first.rest.first

You Try:



```
a = Link(1, link(2, link(1)))
```

```
b = Link(3, link(2, link(1)))
```

```
combined = Link(a, Link(b))
```

`combined.rest` is an **arrow** to the next **`link`**

`combined.rest.first` is an **arrow** to `b`

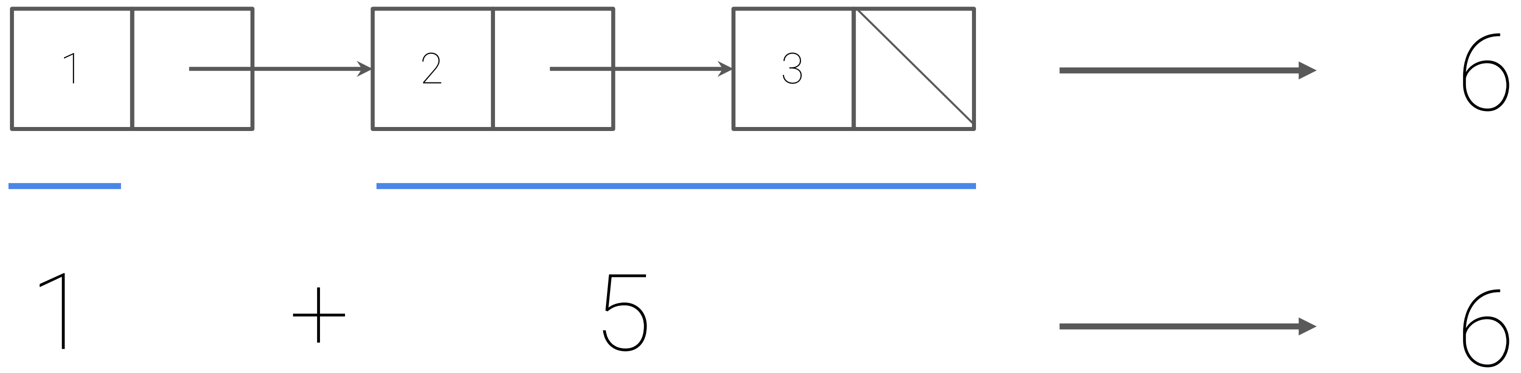
```
combined.rest.first.first
```

Processing Linked Lists

Sum

Demo_2

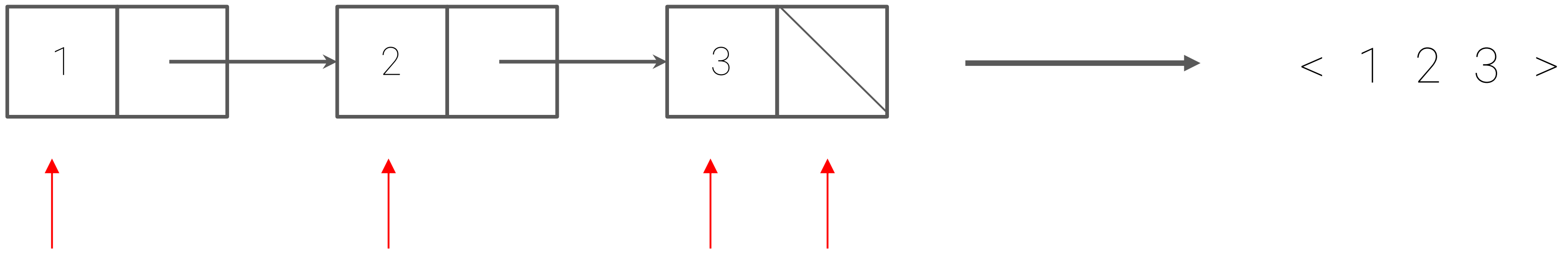
Goal: Given a linked list, lnk, return the sum of all elements in the linked list



display_link

Demo_3

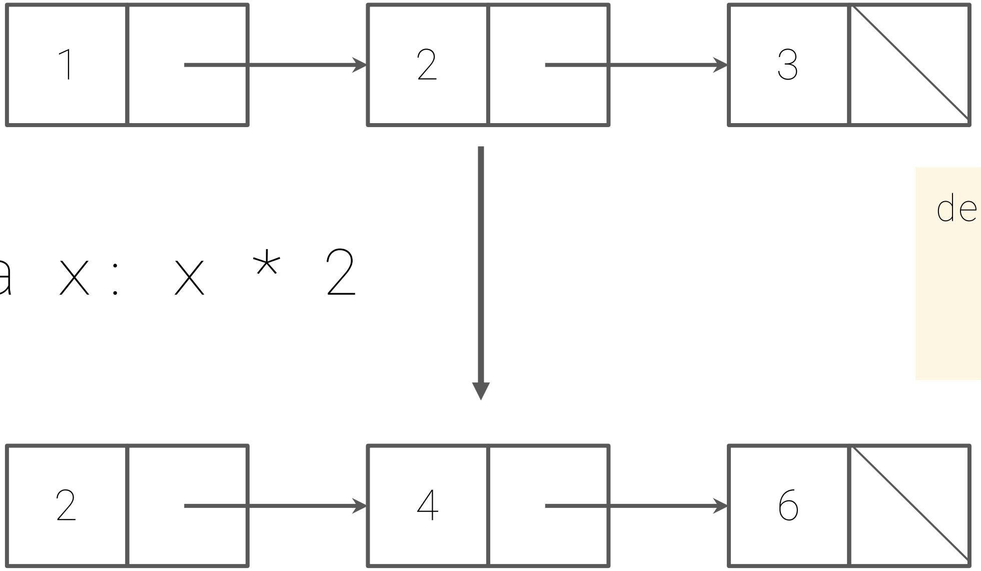
Goal: Given a linked list, lnk, return a string representing the elements in the linked list



Map

Demo_4

Goal: Given a linked list, lnk, and a one argument function, f, return a new linked list obtained from applying f to each element of lnk



lambda x: x * 2

```
def map(f, lnk):

        """Your Code Here"""
```

Demo_5

Mutating Linked Lists

Map, V2

Goal: Given a linked list, lnk, and a one argument function, f, mutate the linked list by applying f to each element.

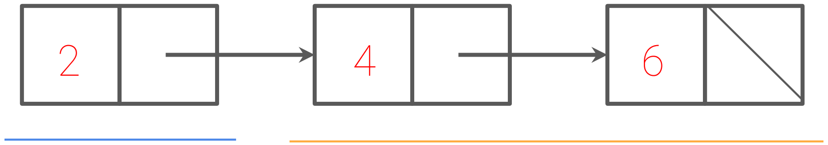
```
def map(lnk, f):

        """
    >>> lnk = Link(1, Link(2, Link(3)))
    >>> map(lnk, lambda x: x * 2)
    >>> print(display_link(lnk))
    <2, 4, 6>
    """
```

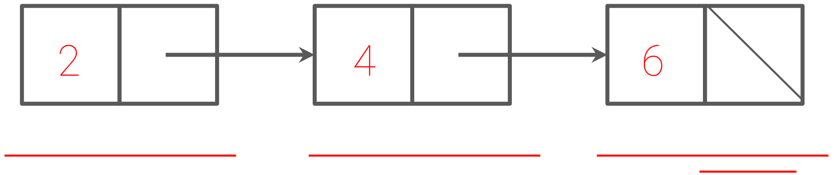
Map, V2

Goal: Given a linked list, lnk, and a one argument function, f, mutate the linked list by applying f to each element.

```
def map(lnk, f):
    if lnk is Link.empty:
        return
    lnk.first = f(lnk.first)
    map(lnk.rest, f)
```

```
def map(lnk, f):

    while lnk is not Link.empty:
        lnk.first = f(lnk.first)
        lnk = lnk.rest
```





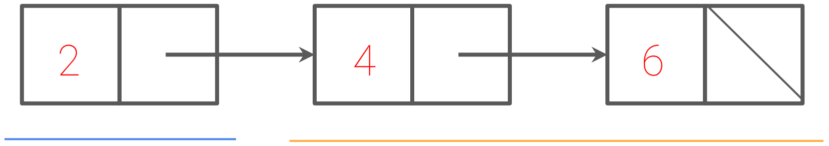
Map, V2

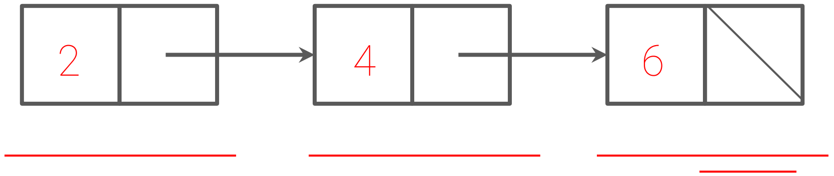
Goal: Given a linked list, lnk, and a one argument function, f, mutate the linked list by applying f to each element.

```
def map(lnk, f):
    if lnk is Link.empty:
        return
    lnk.first = f(lnk.first)
    map(lnk.rest, f)
```

```
def map(lnk, f):

    while lnk is not Link.empty:
        lnk.first = f(lnk.first)
        lnk = lnk.rest
```

Note that the original lnk will not shrink

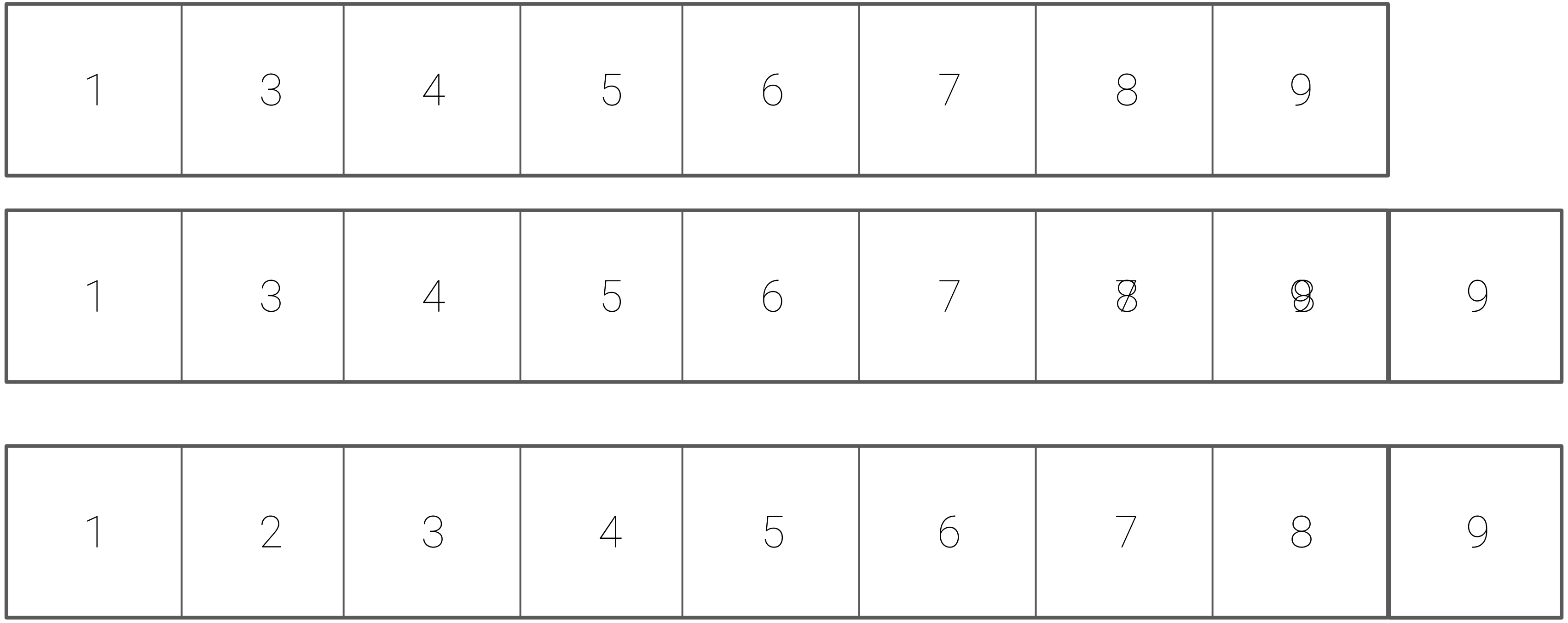




Why Linked Lists?

Insert element at index 1

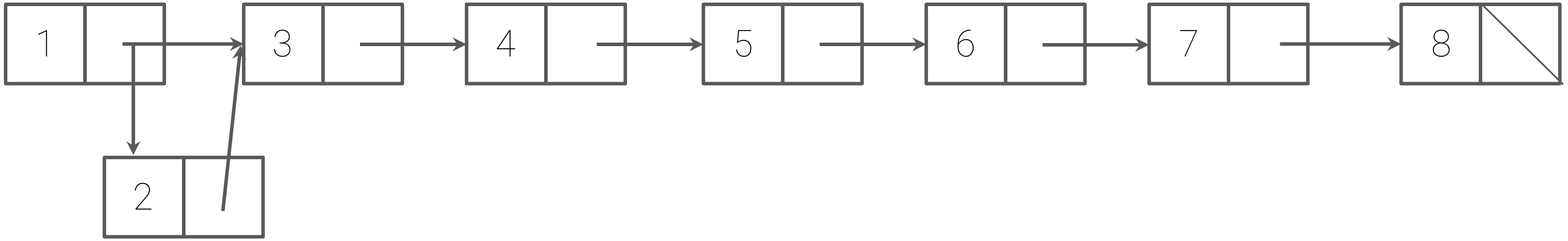
Total number of operations = the length of the list minus 1



Why Linked Lists?

Insert element at index 1

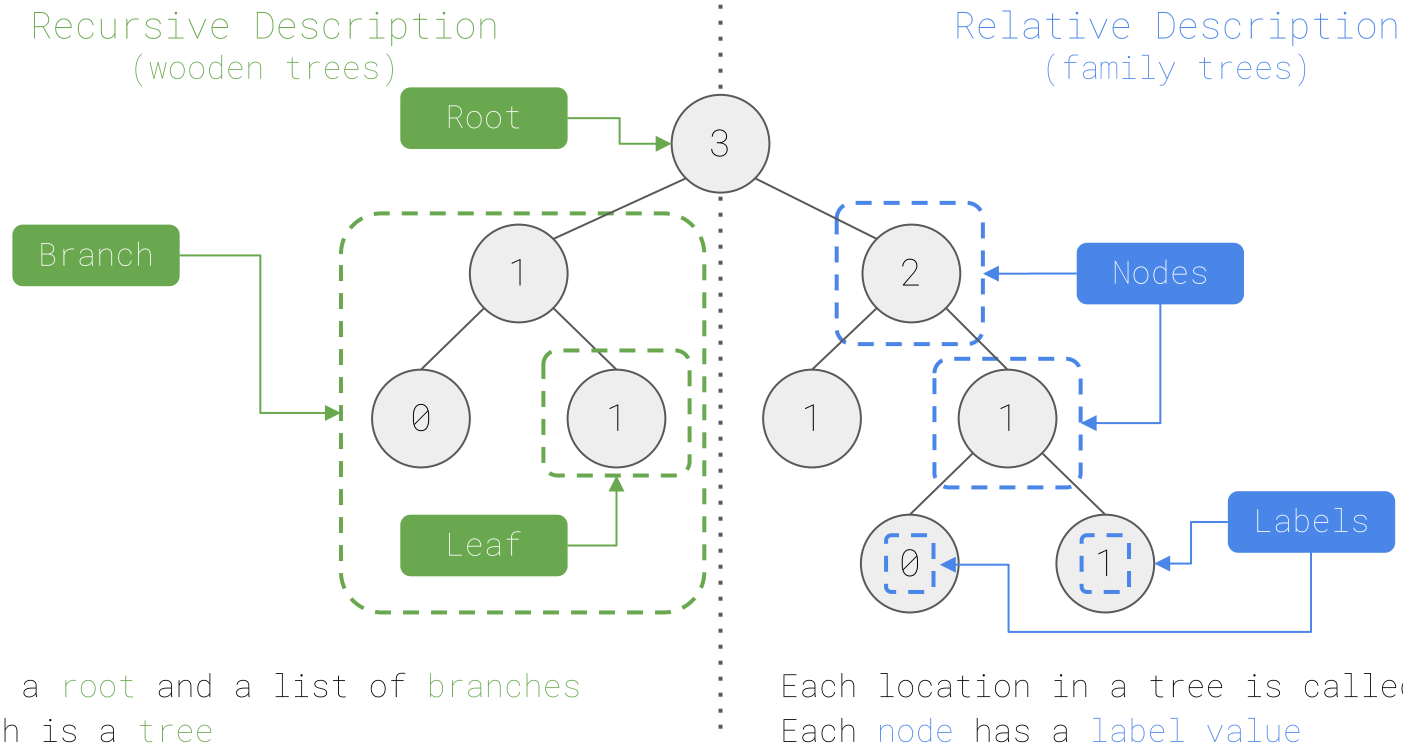
Total number of operations = 2
(Regardless of length of list)



```
inserted_elem = Link(2)
inserted_elem.rest = lnk.rest
lnk.rest = inserted_elem
```

Tree Class

Tree Abstraction



A tree has a root and a list of branches
Each branch is a tree
A tree with zero branches is called a leaf

Each location in a tree is called a node
Each node has a label value
One node can be the parent/child of another

Tree Class

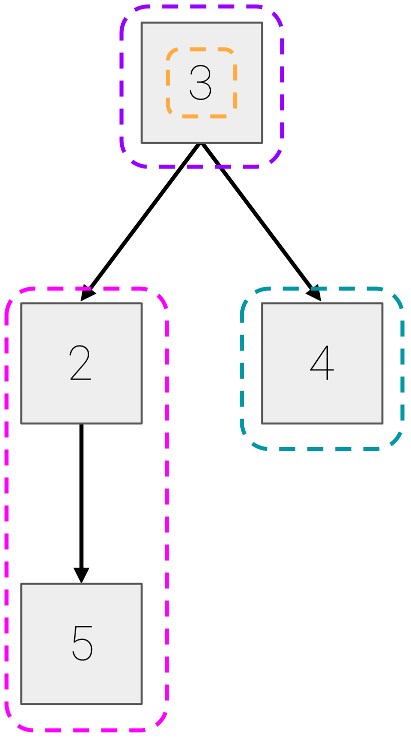
A Tree has a label and a list of branches; each branch is a Tree

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

```
def 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:]
```

```python
class Tree:
    def __init__(self, label, branches=[]):
        for b in branches:
                assert isinstance(b, Tree)
        self.label = label
        self.branches = branches

    def is_leaf(self):
        return not self.branches
```

```
>>> t = Tree(3, [Tree(2, [Tree(5)]), Tree(4)])
>>> t.label
3
>>> t.branches[0].label
2
>>> t.branches[1].is_leaf()
True
```



Demo_1

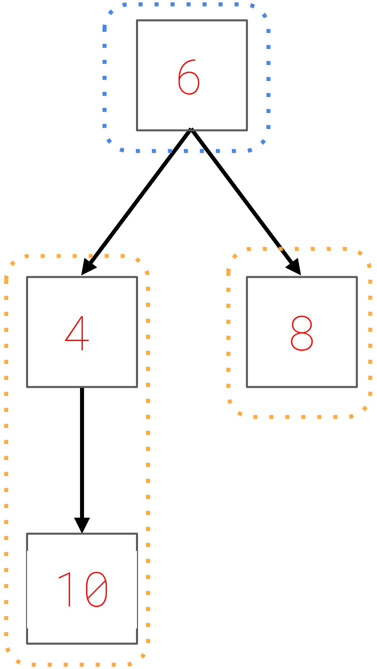
Map, V3

Demo_2

Goal: Given a Tree, t, and a one argument function, f, mutate the tree by applying f to each label.

```
def map(f, t):

    t.label = f(t.label)
    for b in t.branches:
        map(f, b)
```

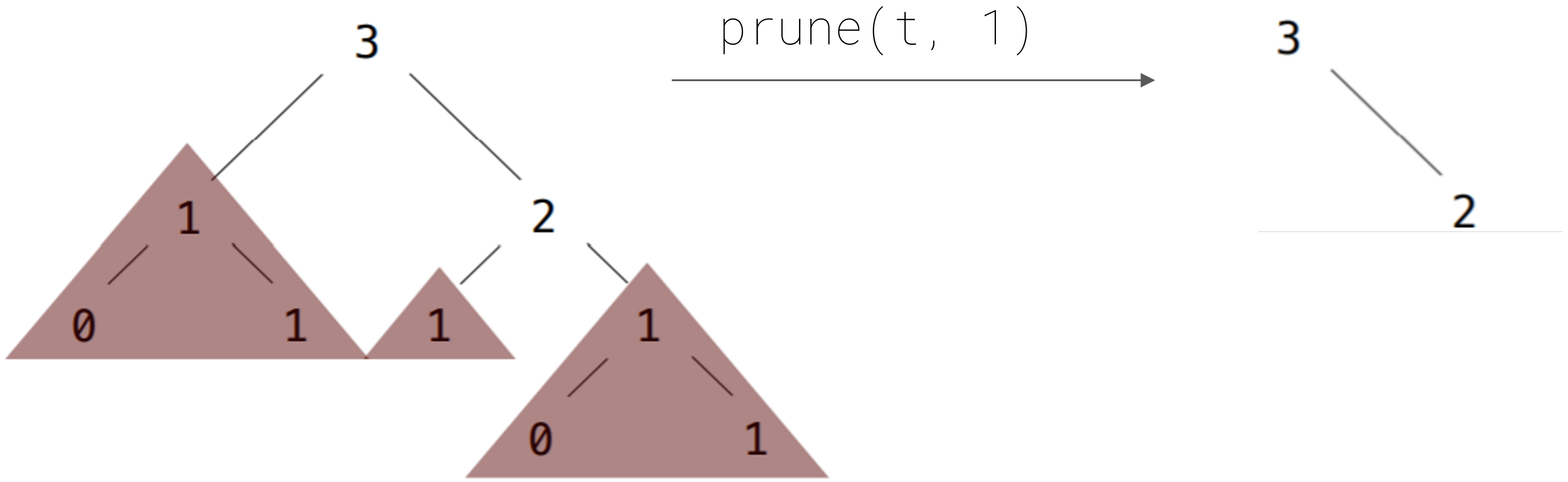
```
t = Tree(3, [Tree(2, [Tree(5)]),
             Tree(4)])
map(lambda x: x * 2, t)
```



Pruning

Demo_3

Goal: Given a Tree, t, and a value n, remove all branches (sub-trees) with label equal to n


3
1
0
1
2
1
1
0
1
prune(t, 1)
3
2