### Composition

- Linked Lists

### Announcements

- Linked List Structure

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

- Linked List Class

A class attribute represents an empty linked list

- Property Methods

In some cases, we want the value of instance attributes to be computed on demand. For example, if we want to access the second element of a linked list:

```python
>>> x = Link(3, Link(4, Link(5)))
>>> x.second
6
*
```

The @property decorator on a method designates that it will be called whenever it is looked up on an instance. A @attribute.setter decorator on a method designates that it will be called whenever that attribute is assigned. `<attribute>` must be an existing property method.
**Tree Recursion Efficiency**

**Recursive Computation of the Fibonacci Sequence**

Our first example of tree recursion:

\[
\begin{align*}
\text{fib}(5) &= \text{fib}(4) + \text{fib}(3) \\
\text{fib}(4) &= \text{fib}(3) + \text{fib}(2) \\
\text{fib}(3) &= \text{fib}(2) + \text{fib}(1) \\
\text{fib}(2) &= \text{fib}(1) + \text{fib}(0) \\
\text{fib}(1) &= 1 \\
\text{fib}(0) &= 0
\end{align*}
\]

---

**Memoization**

Idea: Remember the results that have been computed before

```python
def memo(f):
    cache = {}
    def memoized(n):
        if n not in cache:
            cache[n] = f(n)
        return cache[n]
    return memoized
```

---

**Memoized Tree Recursion**

---

**Tree Class**

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

```python
class Tree:
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)\n
def fib_tree(n):
    if n == 0 or n == 1:
        return Tree(n)
    else:
        left = fib_tree(n-2)
        right = fib_tree(n-1)
        return Tree(left.label + right.label)
```

---

**Tree Abstraction (Review)**

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

```python
def _list_left(label, branches=[]):
    self.label = label
    for branch in branches:
        assert isinstance(branch, Tree)
    self.branches = list(branches)

def tree(label, branches=[]):
    if label == 0 or label == 1:
        return Tree(label)
    return Tree(label, _list_left(label, branches=branches))
```

---

**Tree Class**

---

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