Announcements

Box-and-Pointer Notation

The Closure Property of Data Types

- A method for combining data values satisfies the closure property if:
  - The result of combination can itself be combined using the same method
- Closure is powerful because it permits us to create hierarchical structures
- Hierarchical structures are made up of parts, which themselves are made up of parts, and so on

Lists can contain lists as elements (in addition to anything else)

Box-and-Pointer Notation in Environment Diagrams

Lists are represented as a row of index-labeled adjacent boxes, one per element
Each box either contains a primitive value or points to a compound value

```
pair = [1, 2]
```

Interactive Diagram

Interactive Diagram

Sequence Operations

Membership & Slicing

Python sequences have operators for membership and slicing

Membership.

```
digits = [1, 8, 2, 8]
>>> 2 in digits
True
```

```
digits[start:middle]
```

Slicing.

```
digits[0:2]
```
Slicing creates a new object
**Trees**

### Implementing the Tree Abstraction

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

def root(tree):
    return tree[0]

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

### Implementing the Tree Abstraction

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

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
```

```python
def tree(root, branches=[]):
    ...  # Return a tree with root and branches
```

### Demo

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

```python
def root(tree):
    return tree[0]
```

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

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

```python
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
```

```python
def tree(root, branches=[]):
    ...  # Return a tree with root and branches
```

### Discussion Question

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

**Hint:** If you `sum` a list of lists, you get a list containing the elements of those lists.

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

```python
>>> sum([[1], [2]])
[1, 2]
```

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

```python
def leaves(tree):
    # Return a list containing the leaves of tree.
    return leaves(fib_tree(5))
```

```python
def leaves(fib_tree):
    # Return a list containing the leaves of tree.
    def leaves(s):
        if is_leaf(s):
            return s
        else:
            branch_counts = [count_leaves(b) for b in branches(s)]
            return sum(branch_counts)
    return leaves
```

```python
>>> leaves(fib_tree(5))
[1, 2, 3, 4]
```

```python
>>> sum([[1], [2]])
[1, 2]
```

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

### Example: Partition Trees

Interactive Diagram