CS 61A Lecture 11
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]
```
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.

```
1  pair = [1, 2]
2
3  nested_list = [[1, 2], [],
4      [[3, False, None],
4      [4, lambda: 5]]]
```
Sequence Operations
### Membership & Slicing

Python sequences have operators for membership and slicing.

**Membership.**

```python
>>> digits = [1, 8, 2, 8]
>>> 2 in digits
True
>>> 1828 not in digits
True
```

**Slicing.**

```python
>>> digits[0:2]  # Slicing creates a new object
[1, 8]
>>> digits[1:]
[8, 2, 8]
```
Trees
A tree has a root value and a sequence of branches; each branch is a tree.

A tree with zero branches is called a leaf.

The root values of sub-trees within a tree are often called node values or nodes.
Implementing the Tree Abstraction

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

def root(tree):
    return tree[0]

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

A tree has a root value and a sequence of branches; each branch is a tree.

```python
>>> tree(3, [tree(1), ... tree(2, [tree(1), ... tree(1)]))])
[3, [1], [2, [1], [1]]]
```
Implementing the Tree Abstraction

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

def root(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

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

A tree has a root value and a sequence of branches; each branch is a tree.
Tree Processing
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

```python
def count_leaves(tree):
    """Count the leaves of a tree."""
    if is_leaf(tree):
        return 1
    else:
        branch_counts = [count_leaves(b) for b in branches(tree)]
        return sum(branch_counts)
```

(Demo)
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
def leaves(tree):
    """Return a list containing the leaves of tree."
    if is_leaf(tree):
        return [root(tree)]
    else:
        return [leaves(b) for b in branches(tree)]
```

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

```python
branches(tree)
leaves(tree)
[branches(b) for b in branches(tree)]
[leaves(b) for b in branches(tree)]
```
Example: Partition Trees

(Demo)

Interactive Diagram