Congratulations to the Winners of the Hog Strategy Contest

1st Place with 146 wins:

A five-way tie for first place!

https://hog-contest.cs61a.org

Congratulations to Timothy Gu, Shomini Sen, Samuel Berkun, Mitchell Zhen, Lucas Clark, Dominik de Bettencourt, Allen Gu, Alec Li, Aaron Jaffe

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

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

Box-and-Pointer Notation in Environment Diagrams

Slicing (Demo)
Slicing Creates New Values

```python
digits = [1, 8, 2, 8]
start = digits[1]
middle = digits[1:13]
end = digits[2:]
full = digits[
```

Processing Container Values

Sequence Aggregation

Several built-in functions take iterable arguments and aggregate them into a value:

- `sum(iterable[, start]) -> value`
  - Return the sum of a `start` value (default: 0) plus an iterable of numbers.

- `max(iterable[, key=func]) -> value`
  - With a single iterable argument, return its largest item.
  - With two or more arguments, return the largest argument.

- `all(iterable) -> bool`
  - Return True if bool(x) is True for all values x in the iterable.
    - If the iterable is empty, return True.

Implementing the Tree Abstraction

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

def label(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)
```

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

Discussion Question

Implement `leaves`, which returns a list of the leaf labels 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 leaf labels of tree.
    >>> leaves(fib_tree(5))
    [1, 0, 1, 0, 1, 1, 0, 1]
    """
    if is_leaf(tree):
        return [label(tree)]
    else:
        return sum((leaves(b) for b in branches(tree)), [])
```

Creating Trees

A function that creates a tree from another tree is typically also recursive.

```python
def increment(t):
    """Return a tree like t but with all labels incremented."""
    return tree(label(t) + 1, [increment(b) for b in branches(t)])
```

Example: Printing Trees

```python
def increment_leaves(t):
    """Return a tree like t but with leaf labels incremented.""
    if is_leaf(t):
        return tree(label(t) + 1)
    else:
        bs = [increment_leaves(b) for b in branches(t)]
        return tree(label(t), bs)
```

Example: Summing Paths

```python
def increment(t):
    """Return a tree like t but with all labels incremented.""
    return tree(label(t) + 1, [increment(b) for b in branches(t)])
```