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 the key to power in any means of combination 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

Box-and-Pointer Notation in Environment Diagrams

Trees

Trees are Nested Sequences

A tree is either a single value called a leaf or a sequence of trees

Typically, some type restriction is placed on the leaves. E.g., a tree of numbers:

```
  tree = [[1, 2], [3, 4], [5, 6, 7]]
  => 2
```

Tree Processing Uses Recursion

(Demo)

Processing a leaf is often the base case of a tree processing function

The recursive case often makes a recursive call on each branch and then aggregates

```
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 tree]
        return sum(branch_counts)
```
Discussion Question

Complete the definition of flatten, which takes a tree and returns a list of its leaves.

Hint: If you sum a sequence of lists, you get 1 list containing the elements of those lists.

```python
def sum(lst):
    return sum(lst, [])

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

```python
def flatten(tree):
    if is_leaf(tree):
        return [tree]
    else:
        return [flatten(b) for b in tree]

def is_leaf(tree):
    return type(tree) != list

flatten([[1], [2, 3], [4], [], []])
```

**Sequence Operations**

**Membership & Slicing**

Python sequences have operators for membership and slicing.

- Membership: `2 in digits` returns `True`.
- Slicing: `digits[0:2]` creates a new object.

**Binary Trees**

Trees may also have restrictions on their structure.

A binary tree is either a leaf or a sequence containing at most two binary trees.

The process of transforming a tree into a binary tree is called **binarization**.

```python
def right_binarize(tree):
    if is_leaf(tree):
        return tree
    else:
        return [right_binarize(b) for b in tree]

def is_leaf(tree):
    return type(tree) != list
```

**Strings**

- **Strings are Sequences**
  - Length and element selection are similar to all sequences.
    ```python
    city = 'Berkeley'
    city[1]
    city[1] = 'B'
    city
    ```
  - However, the "in" and "not in" operators match substrings.
    ```python
    'here' in 'Where's Waldo?'
    '324' in [1, 2, 3, 4, 5]
    ```

- **Strings are an Abstraction**
  - Representing data:
    ```python
    '200'  '1.2e-5'  'False'  '1, 2'
    ```
  - Representing language:
    ```python
    'And, as imagination bodies forth
    The forms of things to unknown, and the poet's pen
    Turns them to shapes, and gives to airy nothing
    A local habitation and a name.'
    ```
  - Representing programs:
    ```python
    'curry = lambda f: lambda x: lambda y: f(x, y)'
    ```

- **Strings Literals Have Three Forms**
  - Single-quoted and double-quoted strings are equivalent:
    ```python
    'I am string!'  'I am string!'  'I’ve got an apostrophe'
    ```
  - "The Zen of Python\n  claims, Readability counts. Read more: import this."
Dictionaries

{'Dm': 0}

**Limitations on Dictionaries**

Dictionaries are unordered collections of key-value pairs.

Dictionary keys do have two restrictions:

- A key of a dictionary cannot be a list or a dictionary (or any mutable type).
- Two keys cannot be equal; There can be at most one value for a given key.

This first restriction is tied to Python's underlying implementation of dictionaries.

The second restriction is part of the dictionary abstraction.

If you want to associate multiple values with a key, store them all in a sequence value.