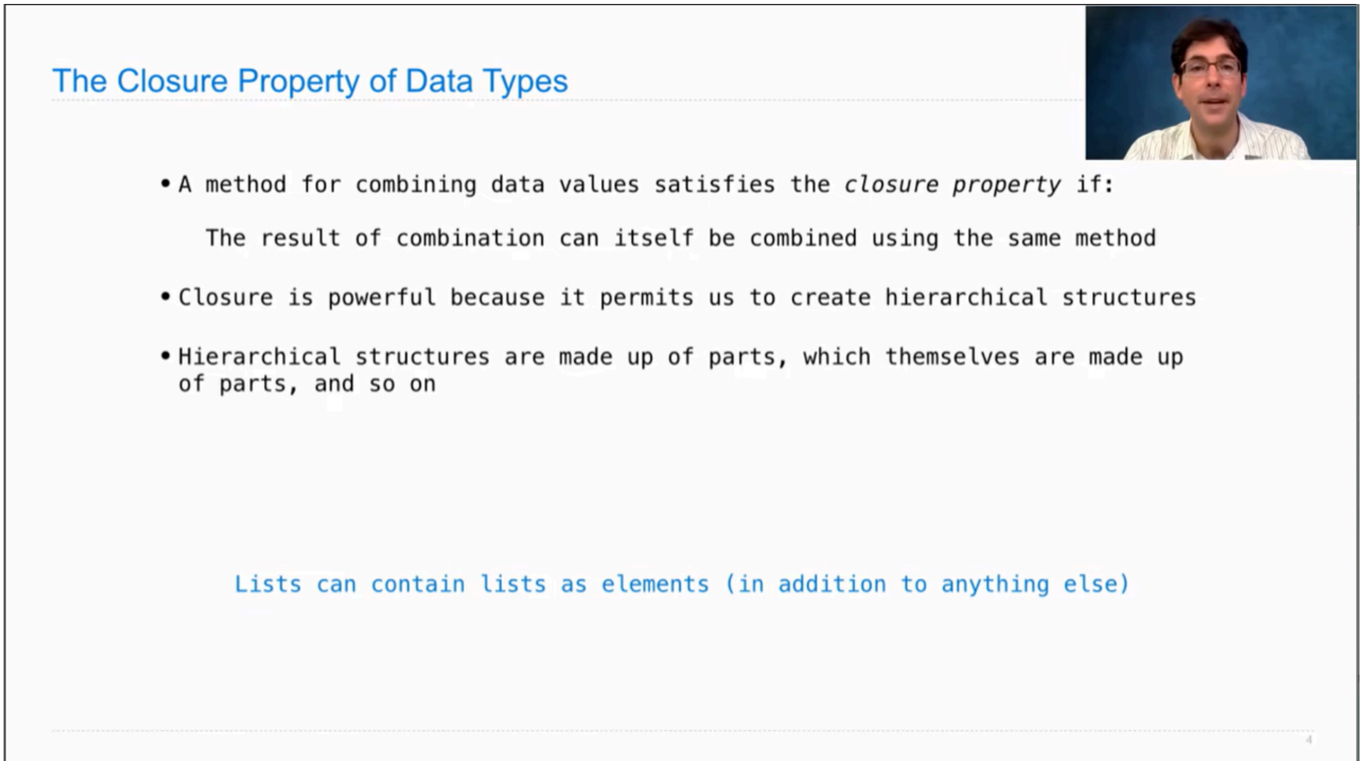
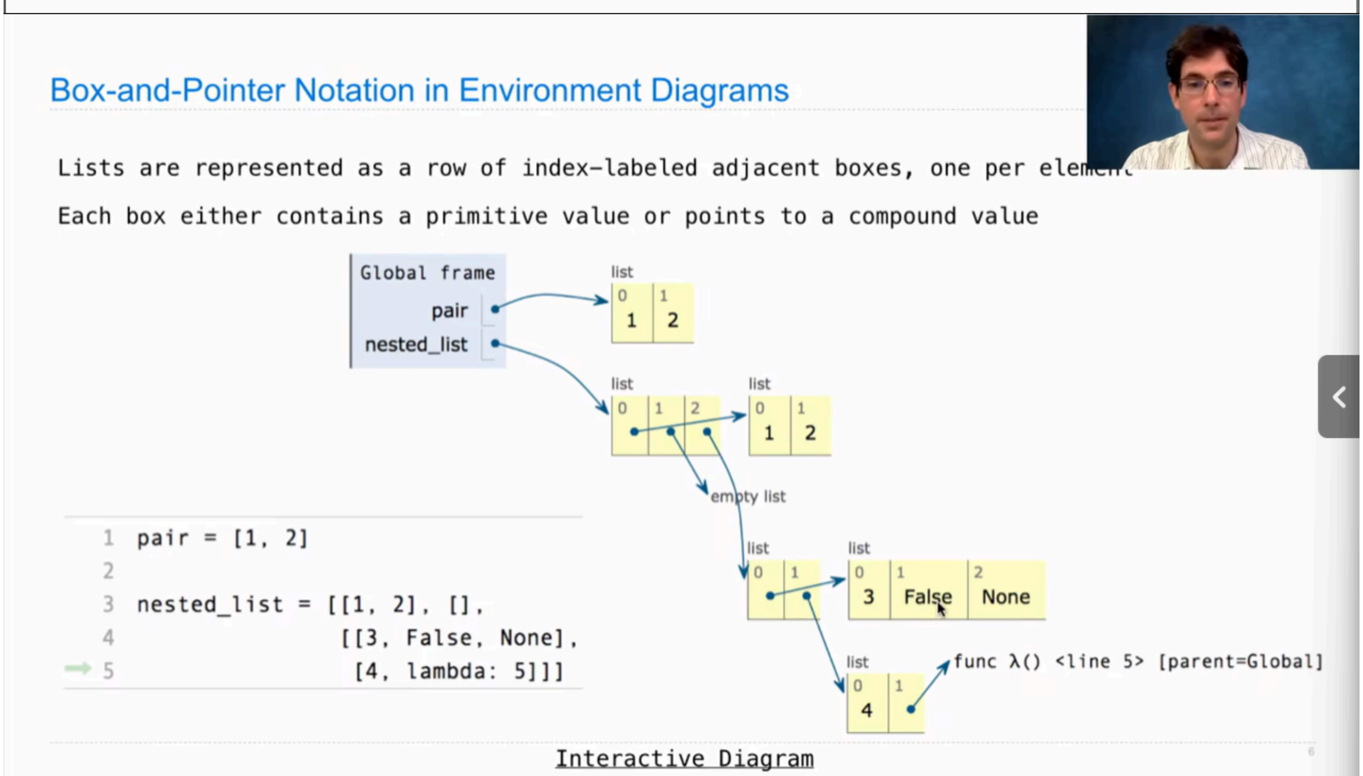
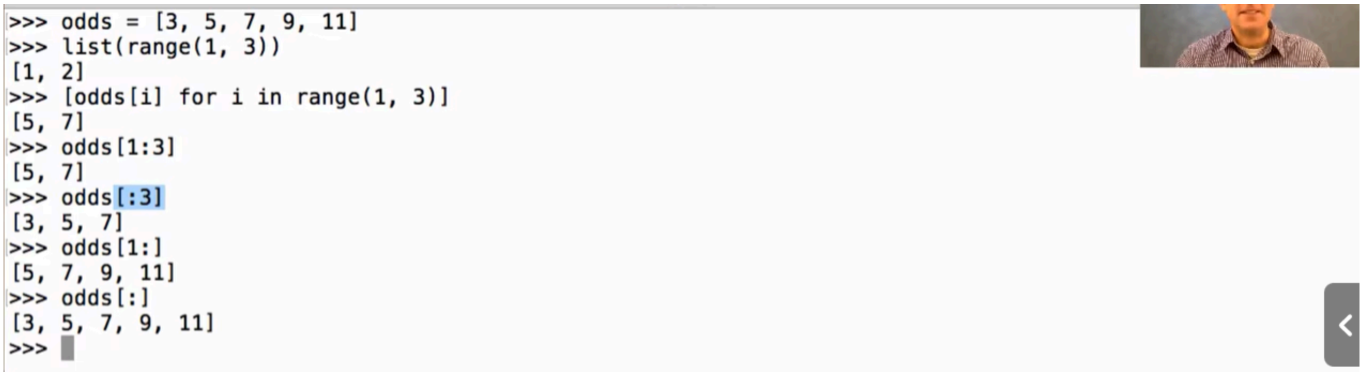
Lecture12 Trees

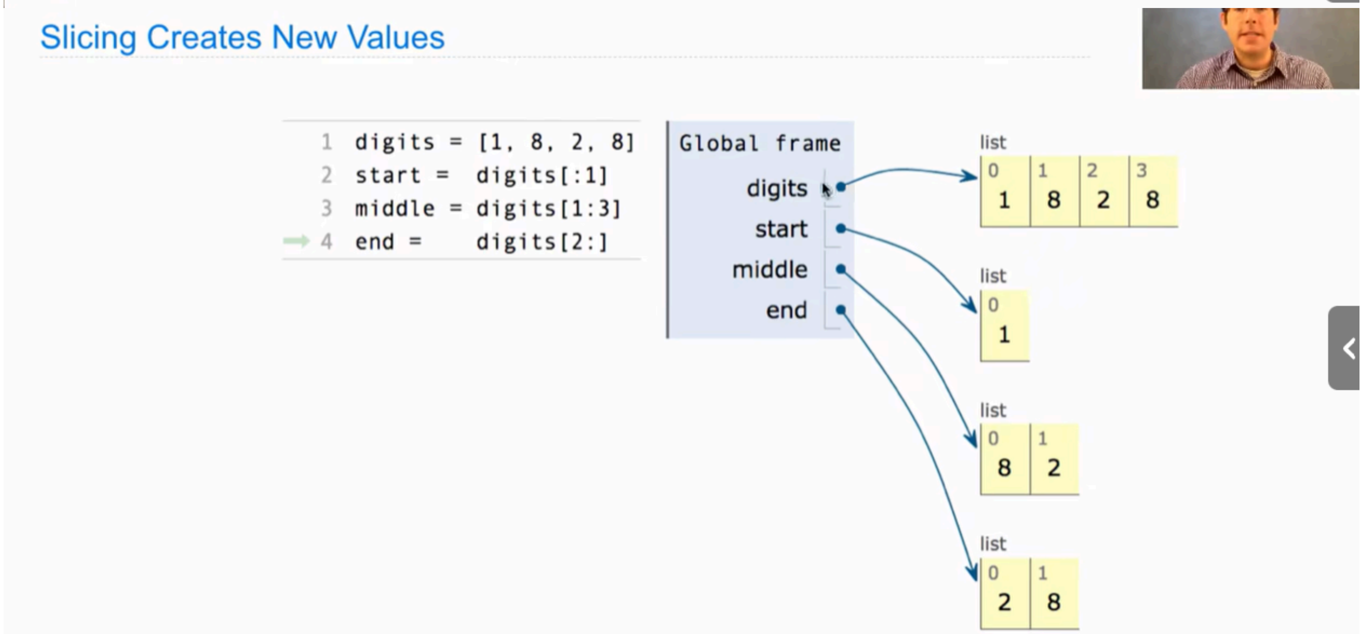
Box-and-Pointer Notation





Slices





Processing Container Elements

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

```
sum(iterable[,start]) -> value

>>> sum([2, 3, 4])
9

>>> sum([2, 3, 4], 5)
14

# the start value is kinda like telling the com what is the expected value of this expression
```

```
max(iterable[, key = func]) -> value
max(a, b, c, ...[, key = func]) -> value

>>> max(range(4))
4
```

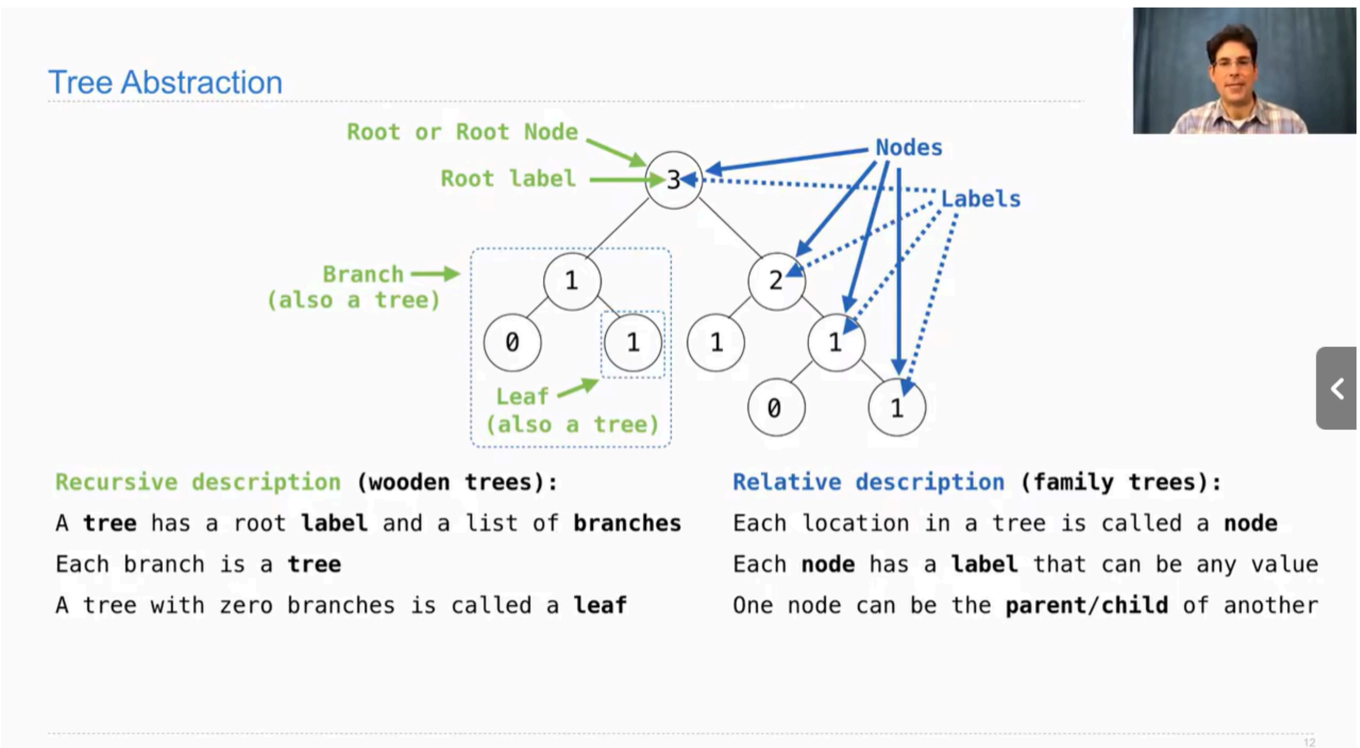
```
>>> max(0, 1, 2, 3, 4)
4

>>> max(range(10), lambda x: 7 - (x - 4) * (x - 2))
3
# the largest output from the result of the func taking the former list as input
```

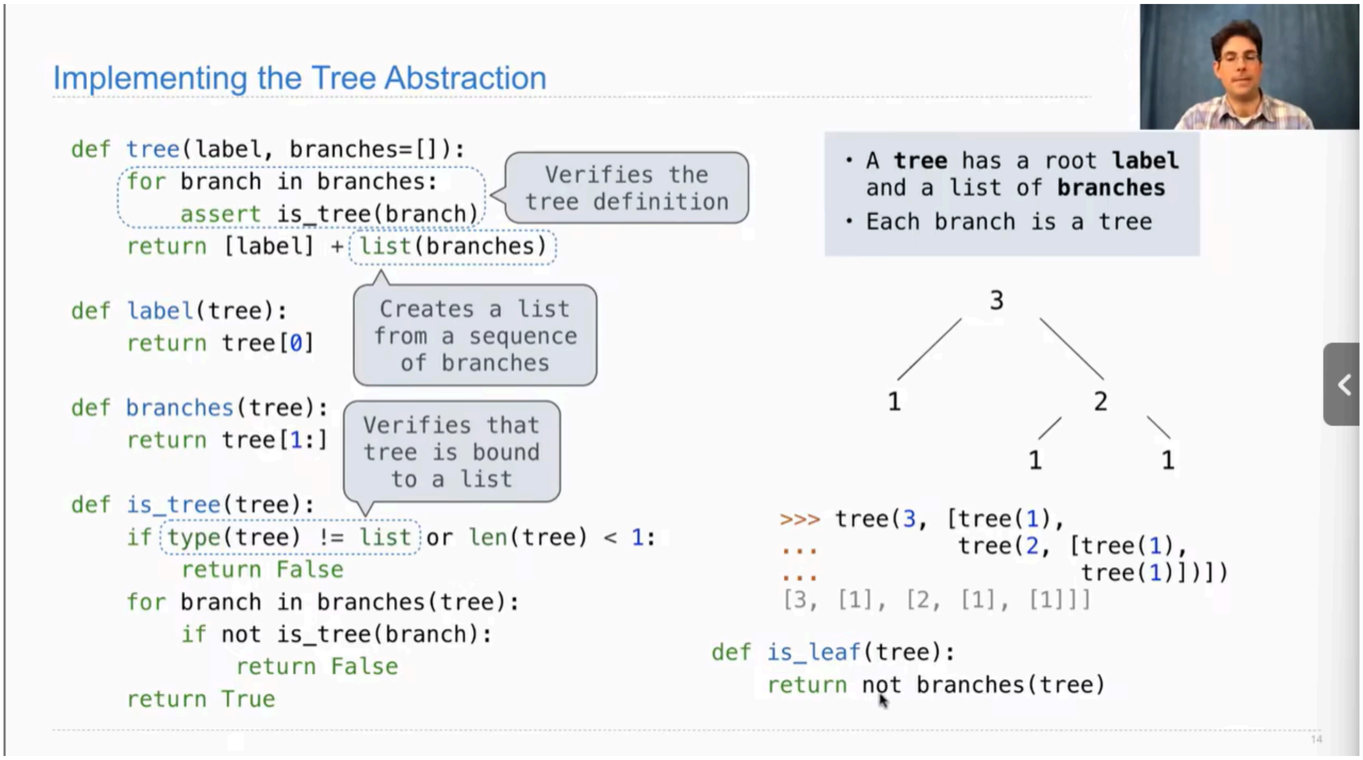
```
• all(iterable) -> bool

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

Trees



implementing the tree abstraction



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

```
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

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

```
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(List of leaf labels for each branch, [])
```

```
branches(tree)                              [b for b in branches(tree)]
leaves(tree)                                [s for s in leaves(tree)]
[branches(b) for b in branches(tree)]       [branches(s) for s in leaves(tree)]
[leaves(b) for b in branches(tree)]         [leaves(s) for s in leaves(tree)]
```


Creating Trees

Creating Trees

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

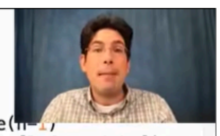
```python
def increment_leaves(t):
    """Return a tree like t but with leaf labels incremented."""
    if is_leaf(t):
        return tree(label(t) + 1)
    else:
        bs = [increment_leaves(b) for b in branches(t)]
        return tree(label(t), bs)


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)])
```

18

Example

```
~/lec$ python3 -i ex.py
>>> print_tree(fib_tree(4))
3
  1
    0
    1
  2
    1
    1
      0
      1
>>> print_tree(fib_tree(5))
5
  2
    1
    1
      0
      1
  3
    1
      0
      1
    2
      1
      1
        0
        1
>>>
```



```python
def fib_tree(n):
    if n <= 1:
        return tree(n)
    else:
        left, right = fib_tree(n-2), fib_tree(n-1)
        return tree(label(left)+label(right), [left, right])

def count_leaves(t):
    """Count the leaves of tree T."""
    if is_leaf(t):
        return 1
    else:
        return sum([count_leaves(b) for b in branches(t)])

def print_tree(t, indent=0):
    print('  ' * indent + str(label(t)))
    for b in branches(t):
        print_tree(b, indent+1)
```