

Mutation

Announcements

List Mutation

(Demo)

List Mutation

Create a new list:

List literal: `s = [1, 2, 3]`

List constructor: `t = list(s)`

List comprehension: `u = [x for x in s]`

$$+ : v = s + t$$

Slicing: `w = t[1:]`

Modify a list:

```
s.append(4) # Adds 4 to the end
```

```
s[1] = 'hi' # Changes index 1 to 'hi'
```

```
s.extend(t) # Modifies s but not t
```

```
s += t    # Same as s.extend(t)!
```

```
s.remove(2) # Removes the first 2
```

```
s.pop()      # Removes the last item
```

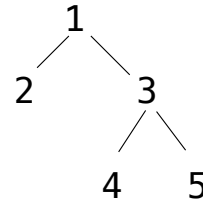
```
s.pop(1)    # Removes the item at index 1
```

(Demo)

List Mutation and Recursion (from the Previous Lecture)

```
def print_big(t, n):  
    """Print the paths in t that have a sum larger or equal to n.
```

```
>>> t = Tree(1, [Tree(2), Tree(3, [Tree(4), Tree(5)])])  
>>> print_big(t, 3)  
[1, 2]  
[1, 3, 4]  
[1, 3, 5]  
>>> print_big(t, 6)  
[1, 3, 4]  
[1, 3, 5]  
>>> print_big(t, 9)  
[1, 3, 5]  
"""
```



```
def extend_and_maybe_print(t, p):
```

```
    p = p + [t.label]
```

```
    if is_leaf(t):
```

```
        if sum(p) >= n:  
            print(p)
```

```
    else:
```

```
        for b in t.branches:  
            helper(b, p)
```

```
extend_and_maybe_print(t, [])
```

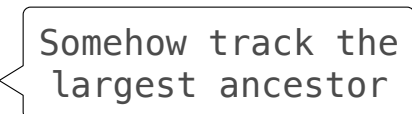
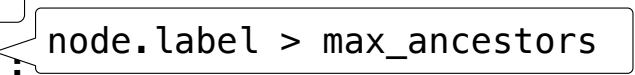
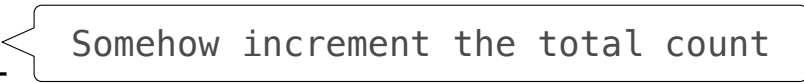
Why didn't we write `p.append(t.label)` or `p += [t.label]` ?

(Demo)

Recursive Accumulation

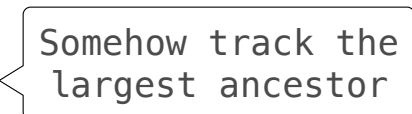
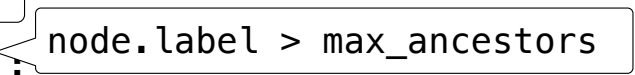
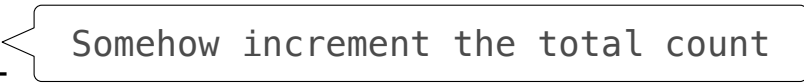
Solving Tree Problems

Implement **big**s, which takes a Tree instance *t* containing integer labels. It returns the number of nodes in *t* whose labels are larger than any labels of their ancestor nodes.

```
def bigs(t):  
    """Return the number of nodes in t that are larger than all their ancestors."""  
    n = [0]  
  
    def f(a, x):  
         Somehow track the largest ancestor  
        if  node.label > max_ancestors:  
             Somehow increment the total count  
        :  
        f(  
             Root label is always larger than its ancestors  
        )  
    :  
    return n[0]
```

Solving Tree Problems

Implement **big**s, which takes a Tree instance *t* containing integer labels. It returns the number of nodes in *t* whose labels are larger than any labels of their ancestor nodes.

```
def bigs(t):  
    """Return the number of nodes in t that are larger than all their ancestors."""  
    n = [0]  
  
    def f(a, x):  
         Somehow track the largest ancestor  
        if a.label > x:  node.label > max_ancestors  
             Somehow increment the total count  
            n[0] += 1  
        for b in a.branches:  
            f(b, max(a.label, x))  
    f(t, t.label - 1)  Root label is always larger than its ancestors  
  
    return n[0]
```

Sum more Fun (Building Lists with Append)

```
def sums(n, m):
```

"""Return lists that sum to n containing positive numbers up to m that have no adjacent repeats, for $n > 0$ and $m > 0$.

```
>>> sums(5, 1)
[]
```

What are all of the ways to build the rest of the list, now that we've used a 1?
Figure out how to sum to 4

```
>>> sums(5, 2)
[[2, 1, 2]]
```

Start with a 1

```
>>> sums(5, 3)
[[1, 3, 1], [2, 1, 2], [2, 3], [3, 2]]
```

```
>>> sums(5, 5)
[[1, 3, 1], [1, 4], [2, 1, 2], [2, 3], [3, 2], [4, 1], [5]]
```

```
>>> sums(6, 3)
[[1, 2, 1, 2], [1, 2, 3], [1, 3, 2], [2, 1, 2, 1], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
'''''
```

```
result = []
for k in range(1, min(m + 1, n)): # k is the first number of a list
    for rest in sums(n-k, m):
        if rest[0] != k:
            result.append([k] + rest) # build a list out of k and rest
if n <= m:
    result.append([n])
return result
```

Environment Diagram Practice

Spring 2023 Midterm 2 Question 1

```
def chain(s):
    return [s[0], s[1:]]
silver = [2, chain([3, 4, 5])]
gold = [silver[0], silver[1].pop()]
silver[0] = 1
platinum = chain(chain([6, 7, 8]))
```

Reminder: `s.pop()` removes and returns the last item in list `s`.

```
>>> silver
```

[1, [3]]

>>> gold

[2, [4, 5]]

```
>>> platinum
```

```
[6, [[7, 8]]]
```

