Mutation

Announcements

List Mutation

(Demo)

https://pythontutor.com/cp/composingprograms.html#code=s%20%3D%20%5B3,%205,%207,%209,%2011%5D%0Afor%20i%20in%20range%28len%28s%29%29%3A%0A%20%20%20%20s%5Bi%5D%20%3D%20s%5B-%28i%2B1%29%5D&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

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)

https://pythontutor.com/cp/composingprograms.html#code=s%20%3D%20%5B%5B1,%202%5D,%203%5D%0At%20%3D%20s%5B0%5D%0As%5B0%5D%5B1%5D%20%3D%204%0As.append%28t%29%0At.append%285%29%0Aprint%28s%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Accumulation Using Mutation

Lists of Fibonacci Numbers

List mutations can be used to build up a list incrementally

```
def fibs(n):
    """Return a list of the first n fibonacci numbers.

    >>> fibs(10)
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
    """
    s = []
    for i in range(n):
        if i < 2:
            s.append(i)
        else:
            s.append(s[-2] + s[-1])
    return s
```

Lists of Fibonacci Numbers Recursively

List mutations can be used to build up a list incrementally, even using recursion

```
def fibs_rec(n):
    """Return a list of the first n fibonacci numbers.

    >>> fibs_rec(10)
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
    """
    s = []
    def f(i):
        _________:
            if i < 2:
                s.append(i)
            else:
                s.append(s[-2] + s[-1])
            __________
    _____
    return s
```

Lists of Fibonacci Numbers Recursively

List mutations can be used to build up a list incrementally, even using recursion

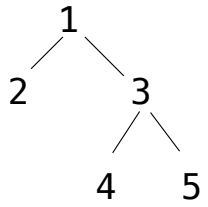
```
def fibs_rec(n):
    """Return a list of the first n fibonacci numbers.

    >>> fibs_rec(10)
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
    """
    s = []
    def f(i):
        if i < n:
            if i < 2:
                s.append(i)
            else:
                s.append(s[-2] + s[-1])
            f(i + 1)
    f(0)
    return s
```

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:
                extend_and_maybe_print(b, p)
    extend_and_maybe_print(t, [])
```



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

(Demo)

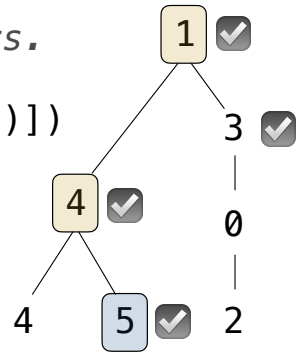
Recursive Accumulation (Flashback to Last Lecture!)

Solving Tree Problems

Implement **bigs**, which takes a Tree instance t containing integer labels. It returns the number of nodes in t whose labels are larger than all labels of their ancestor nodes. (Assume the root label is always larger than all its ancestors, since it has no ancestors.)

```
def bigs(t: Tree[int]):
    """Return the number of nodes in t that are larger than all their ancestors.

    >>> a = Tree(1, [Tree(4, [Tree(4), Tree(5)]), Tree(3, [Tree(0, [Tree(2)])])])
    >>> bigs(a)
    4
```



Somehow increment the total count

Somehow track the largest ancestor

Solving Tree Problems

Implement **bigs**, 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. (Assume the root label is always larger than all its ancestors, since it has no ancestors.)

```
def bigs(t):
    """Return the number of nodes in t that are larger than all their ancestors."""
    n = [0]

    def f(a, x):
        if ____________________________:

            ____________________________

        ____________________________:

            f(_________________________)
    ____________________________________

    return n[0]
```

Somehow track the largest ancestor

node.label > max_ancestors

Somehow increment the total count

Root label is always larger than its ancestors

Solving Tree Problems

Implement **bigs**, 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. (Assume the root label is always larger than all its ancestors, since it has no ancestors.)

```
def bigs(t):
    """Return the number of nodes in t that are larger than all their ancestors."""
    n = [0]

    def f(a, x):
        if a.label > x:
            n[0] += 1
        for b in a.branches:
            f(b, max(a.label, x))
    f(t, t.label - 1)

    return n[0]
```

Somehow track the largest ancestor

node.label > max_ancestors

Somehow increment the total count

Root label is always larger than its ancestors

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

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

Start with a 1

https://pythontutor.com/cp/composingprograms.html#code=result%20%3D%20%5B%5D%0Afor%20k%20in%20range%281,%204%29%3A%0A%20%20%20%20for%20s%20in%20result%3A%0A%20%20%20%20%20%20%20%20if%20s%5B-1%5D%20!%3D%20k%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20s.append%28k%29%0A%20%20%20%20%20%20%20%20%20%20%20%20result.append%28s%29%0Aprint%28result%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Environment Diagram Practice

Spring 2023 Midterm 2 Question 1

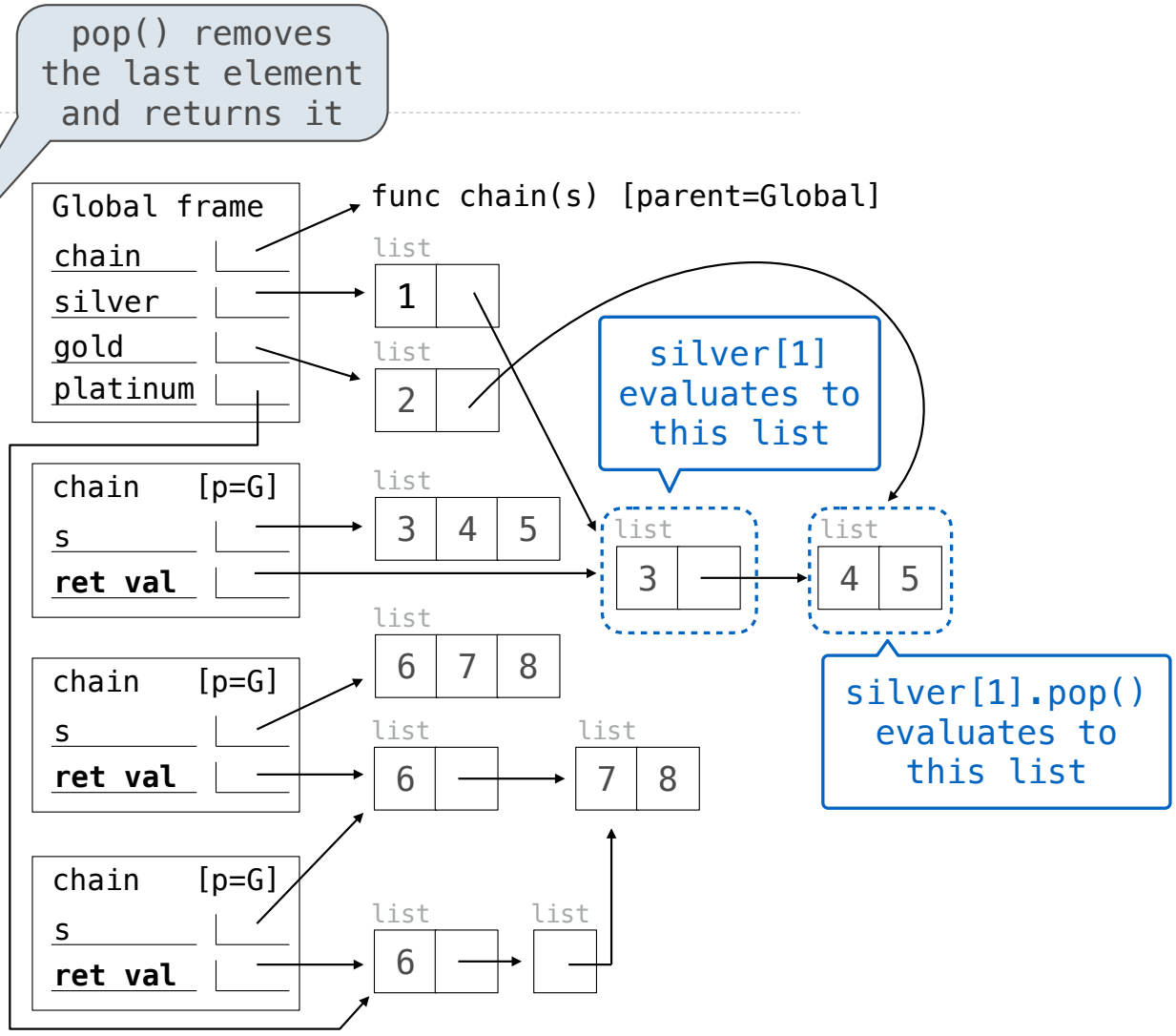
```
def chain(s):
    return [s[0], s[1:]]
silver = [2, chain([3, 4, 5])]
gold = [silver[0], silver[1].pop()]
silver[0] = 1
platinum = chain(chain([6, 7, 8]))
```

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

```
>>> silver
[1, [3]]

>>> gold
[2, [4, 5]]

>>> platinum
[6, [[7, 8]]]
```



https://pythontutor.com/cp/composingprograms.html#code=def%20chain%28s%29%3A%0A%20%20%20%20return%20%5Bs%5B0%5D,%20s%5B1%3A%5D%5D%0Asilver%20%3D%20%5B2,%20chain%28%5B3,%204,%205%5D%29%5D%0Agold%20%3D%20%5Bsilver%5B0%5D,%20silver%5B1%5D.pop%28%29%5D%0Asilver%5B0%5D%20%3D%201%0Aplatinum%20%3D%20chain%28chain%28%5B6,%207,%208%5D%29%29%0A&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D