Iterators and Generators

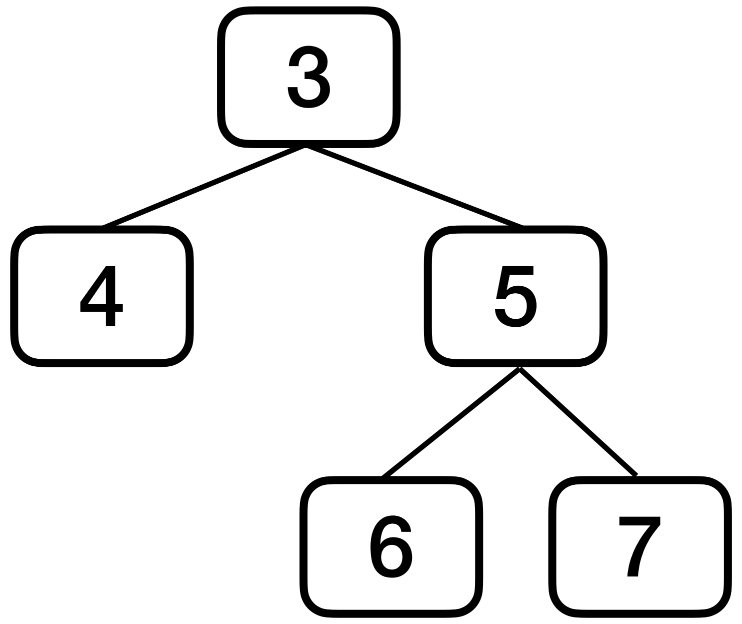
Announcements

Building Lists of Branches

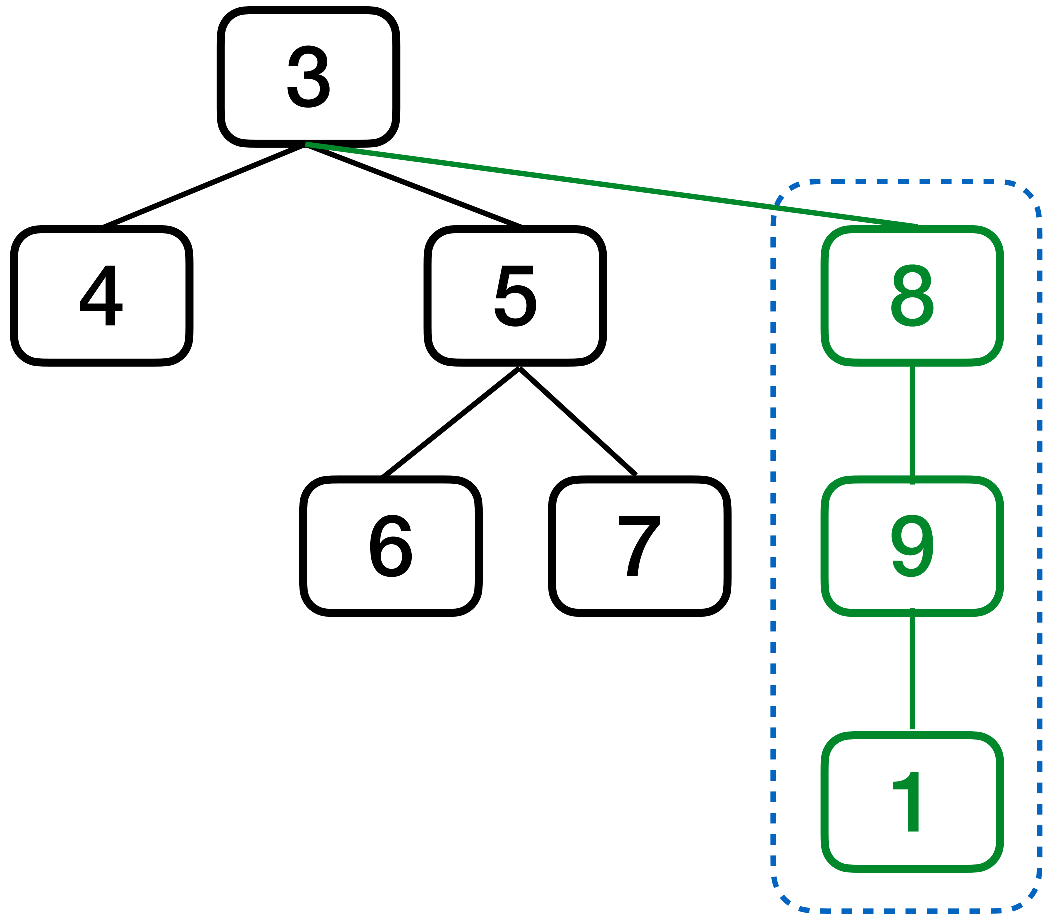
Example: Make Path

A list describes a path if it contains labels along a path from the root of a tree. Implement `make_path`, which takes a tree `t` with unique labels and a list `p` that starts with the root label of `t`. It returns the tree `u` with the fewest nodes that contains all the paths in `t` as well as a (possibly new) path `p`.

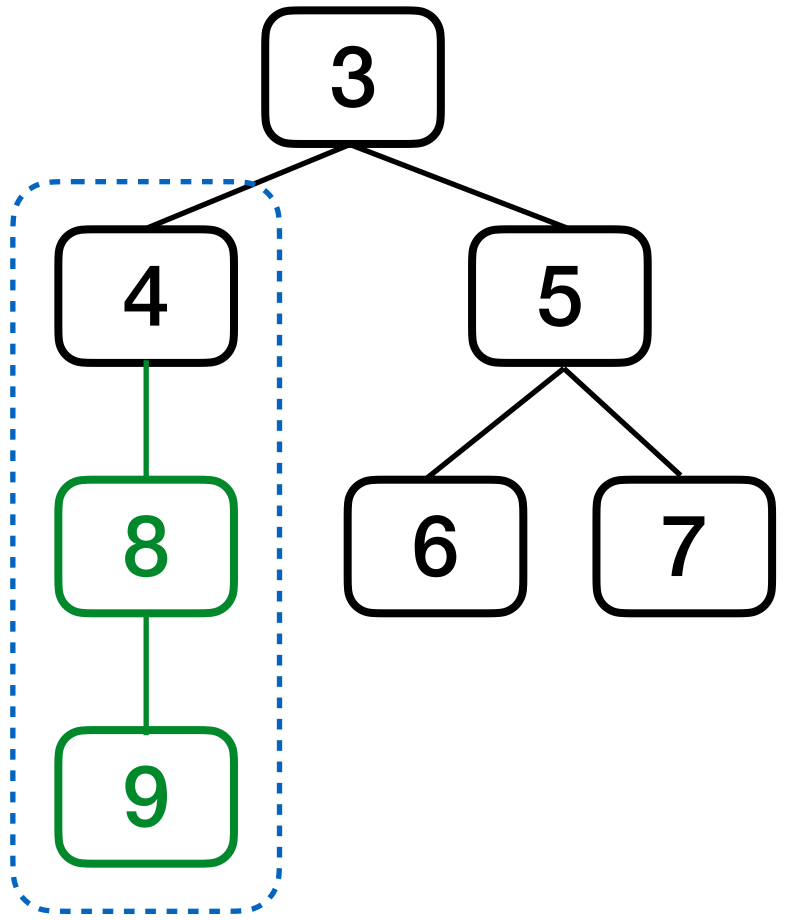
t1



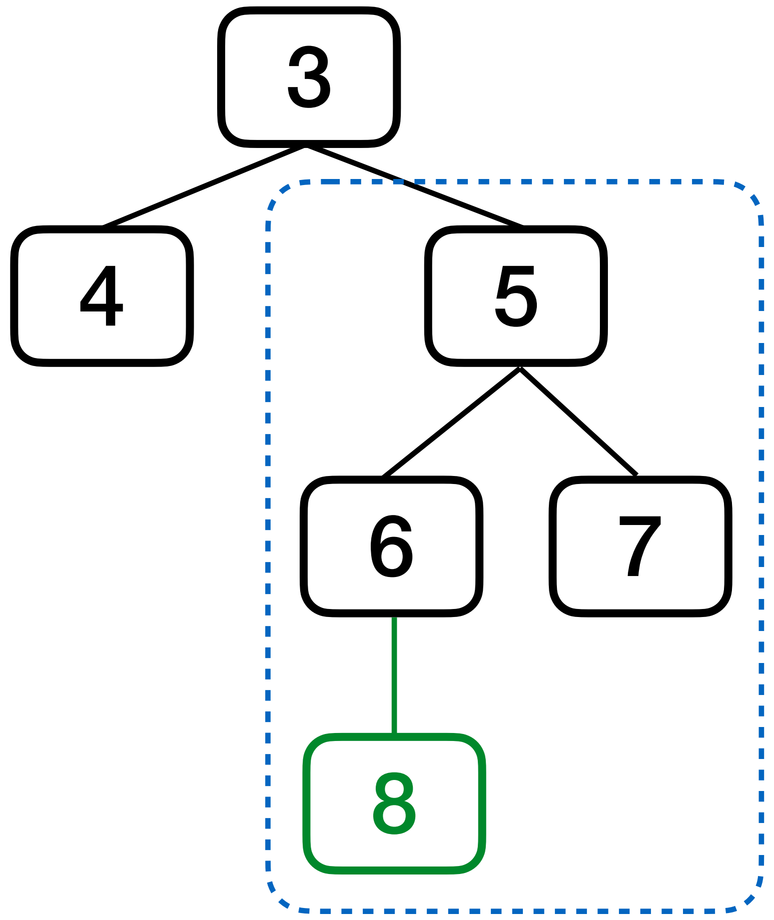
make_path(t1, [3,8,9,1])



make_path(t1, [3,4,8,9])



make_path(t1, [3,5,6,8])



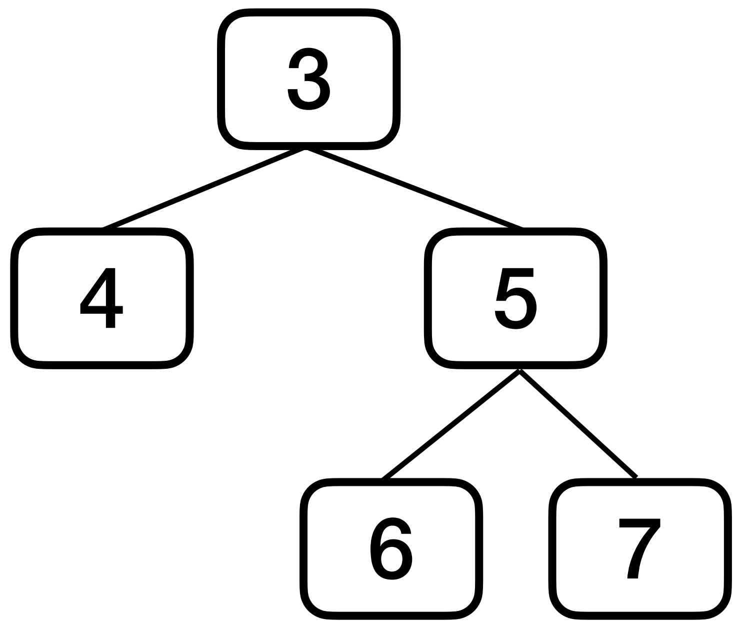
Recursive idea: `make_path(b, p[1:])` is a branch of the tree returned by `make_path(t, p)`

Special case: if no branch starts with p[1], then a leaf labeled p[1] needs to be added

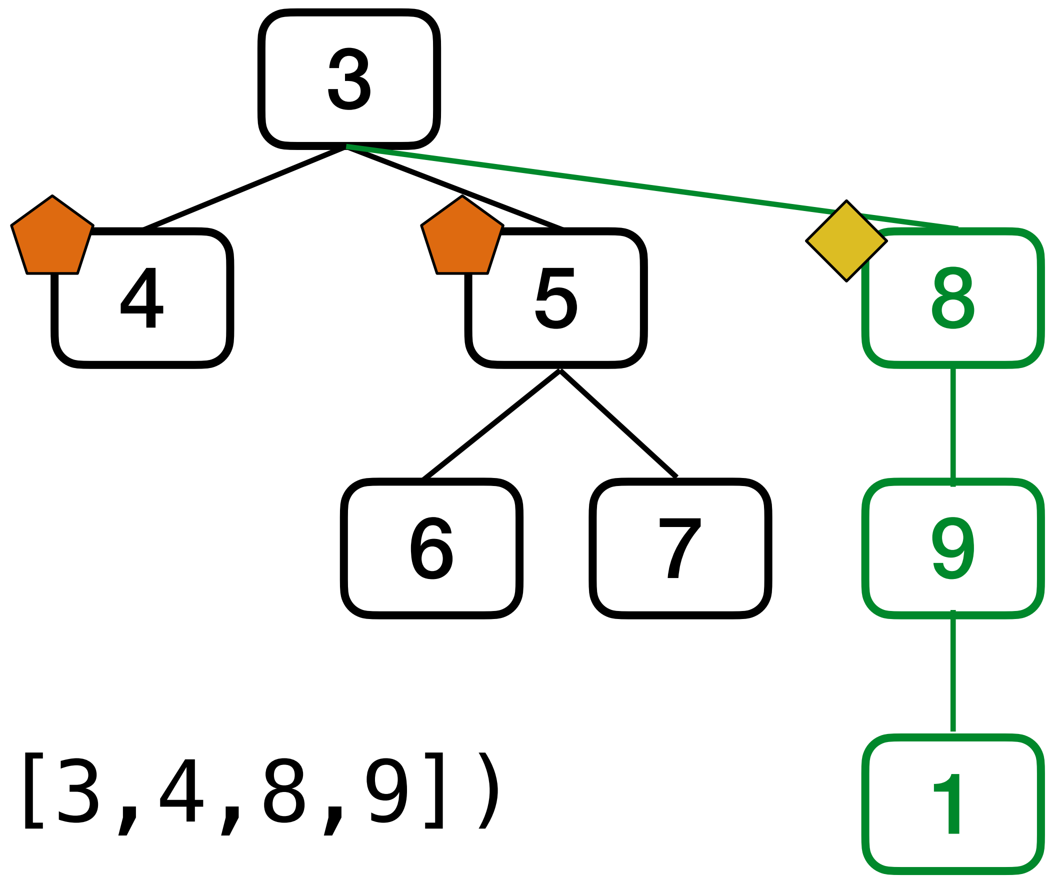
Example: Make Path

A list describes a path if it contains labels along a path from the root of a tree. Implement `make_path`, which takes a tree `t` with unique labels and a list `p` that starts with the root label of `t`. It returns the tree `u` with the fewest nodes that contains all the paths in `t` as well as a (possibly new) path `p`.

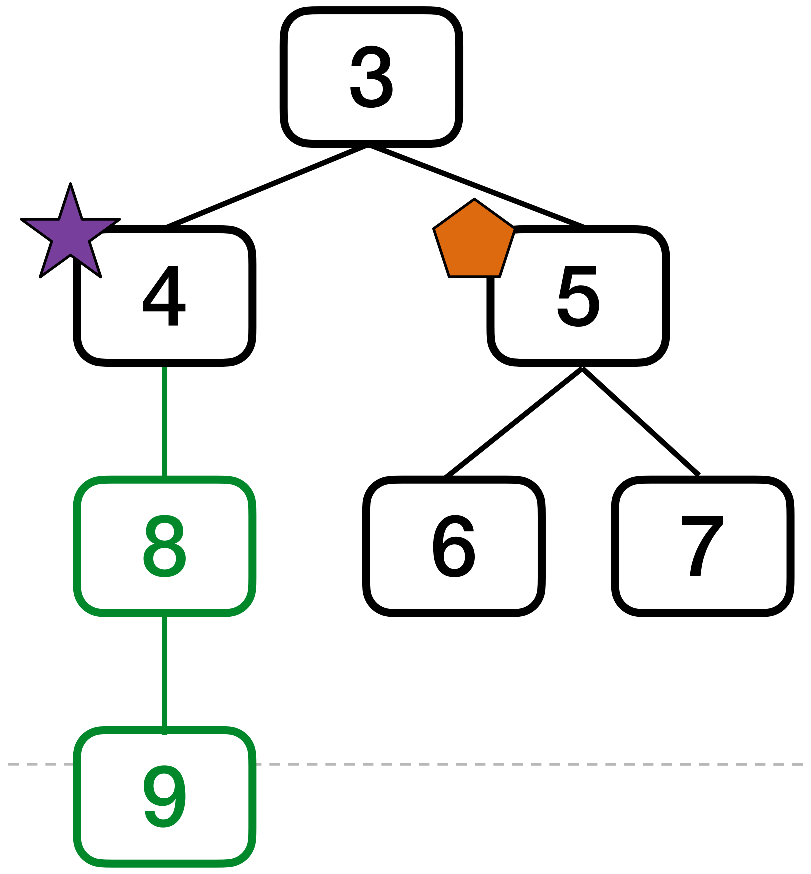
t1



make_path(t1, [3,8,9,1])



make_path(t1, [3,4,8,9])



```
def make_path(t, p):
    "Return a tree like t also containing path p."
    assert p[0] == label(t), 'Impossible'
    if len(p) == 1:
        return t
    new_branches = []
    found_p1 = False
    for b in branches(t):
        if label(b) == p[1]:
          ★ new_branches.append(make_path(b, p[1:]))
            found_p1 = True
        else:
          ⬟ new_branches.append(b)
    if not found_p1:
      ◆ new_branches.append(make_path(tree(p[1]), p[1:]))
    return tree(label(t), new_branches)
```

List Practice

https://pythontutor.com/cp/composingprograms.html#code=s%20%3D%20%5B2,%20%5B3,%204%5D,%205%5D%0As.append%28%5B6,%207%5D%29%0As.extend%28%5B8,%209%5D%29%0At%20%3D%20%5B%5D%0At.extend%28s%29%0As%5B3%5D.append%28s%5B1%5D.pop%28%29%29%0Aprint%28t%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

Spring 2023 Midterm 2 Question 1

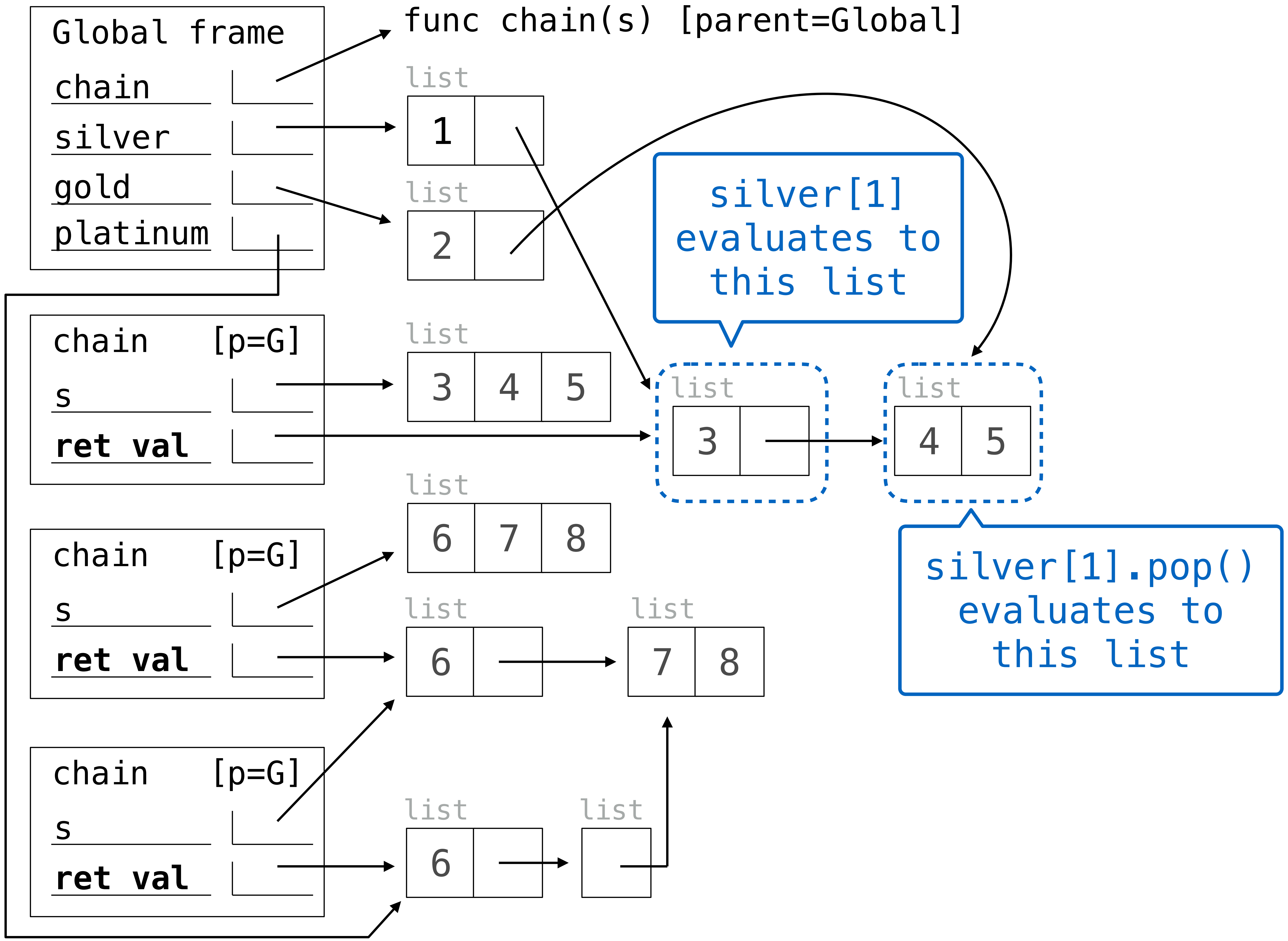
```
def chain(s):
    return [s[0], s[1:]]
silver = [2, chain([3, 4, 5])]
gold = [silver[0], silver[1].pop()]
silver[0] = 1
platinum = chain(chain([6, 7, 8]))
```

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

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

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

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



silver[1] evaluates to this list

silver[1].pop() evaluates to this list

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

Tuples

(Demo)

Iterators

Iterators

A container can provide an iterator that provides access to its elements in order

```
iter(iterable): Return an iterator over the elements
                of an iterable value

next(iterator): Return the next element in an iterator
```

```
>>> s = [3, 4, 5]
>>> t = iter(s)
>>> next(t)
3
>>> next(t)
4
>>> u = iter(s)
>>> next(u)
3
>>> next(t)
5
>>> next(u)
4
```

(Demo)

Break: 5 minutes

Map Function

Map

map(func, iterable): Make an iterator over the return values of calling func on each element of the iterable.

(Demo)

Generators

Generators and Generator Functions

```
>>> def plus_minus(x):
...     yield x
...     yield -x

>>> t = plus_minus(3)
>>> next(t)
3
>>> next(t)
-3
>>> t
<generator object plus_minus ...>
```

A *generator function* is a function that **yield**s values instead of **return**ing them

A normal function **return**s once; a *generator function* can **yield** multiple times

A *generator* is an iterator created automatically by calling a *generator function*

When a *generator function* is called, it returns a *generator* that iterates over its yields

(Demo)

Spring 2023 Midterm 2 Question 5(b) Revisited

Definition. When parking vehicles in a row, a motorcycle takes up 1 parking spot and a car takes up 2 adjacent parking spots. A string of length n can represent n adjacent parking spots using % for a motorcycle, <> for a car, and . for an empty spot.

For example: '.%%.<><>' (Thanks to the Berkeley Math Circle for introducing this question.)

Implement **park**, a **generator function** that yields all the ways, represented as strings, that vehicles can be parked in n adjacent parking spots for positive integer n.

```
def park(n):
    """Yield the ways to park cars and motorcycles in n adjacent spots.

    >>> sorted(park(1))
    ['%', '.']
    >>> sorted(park(2))
    ['%%', '%.', '.%', '..', '<>']
    >>> len(list(park(4)))  # some examples: '<><>', '.%%.', '%<>%', '%.<>'
    29
    """
```