Midterm Review

Announcements

Generator Problem

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
    """
```

List Practice

Lists in Environment Diagrams

Assume that before each example below we execute:

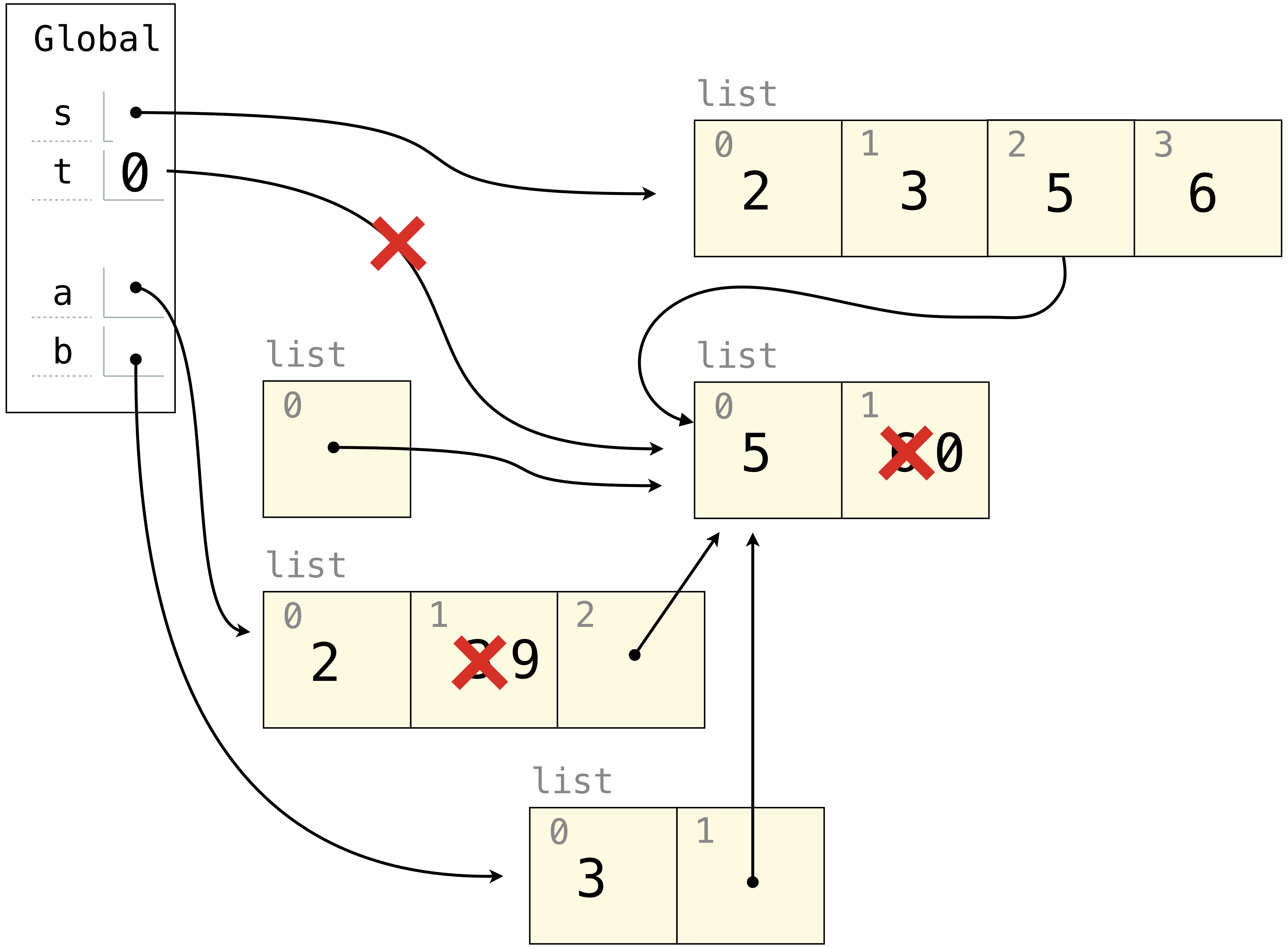
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	s.extend(t)
	t[1] = 0
		s → [2, 3, 5, 6]
		t → [5, 0]

addition & **slicing** create new lists containing existing elements	a = s + [t]
	b = a[1:]
	a[1] = 9
	b[1][1] = 0
		s → [2, 3]
		t → [5, 0]
		a → [2, 9, [5, 0]]
		b → [3, [5, 0]]



Lists in Environment Diagrams

Assume that before each example below we execute:

```
s = [2, 3]
t = [5, 6]
```

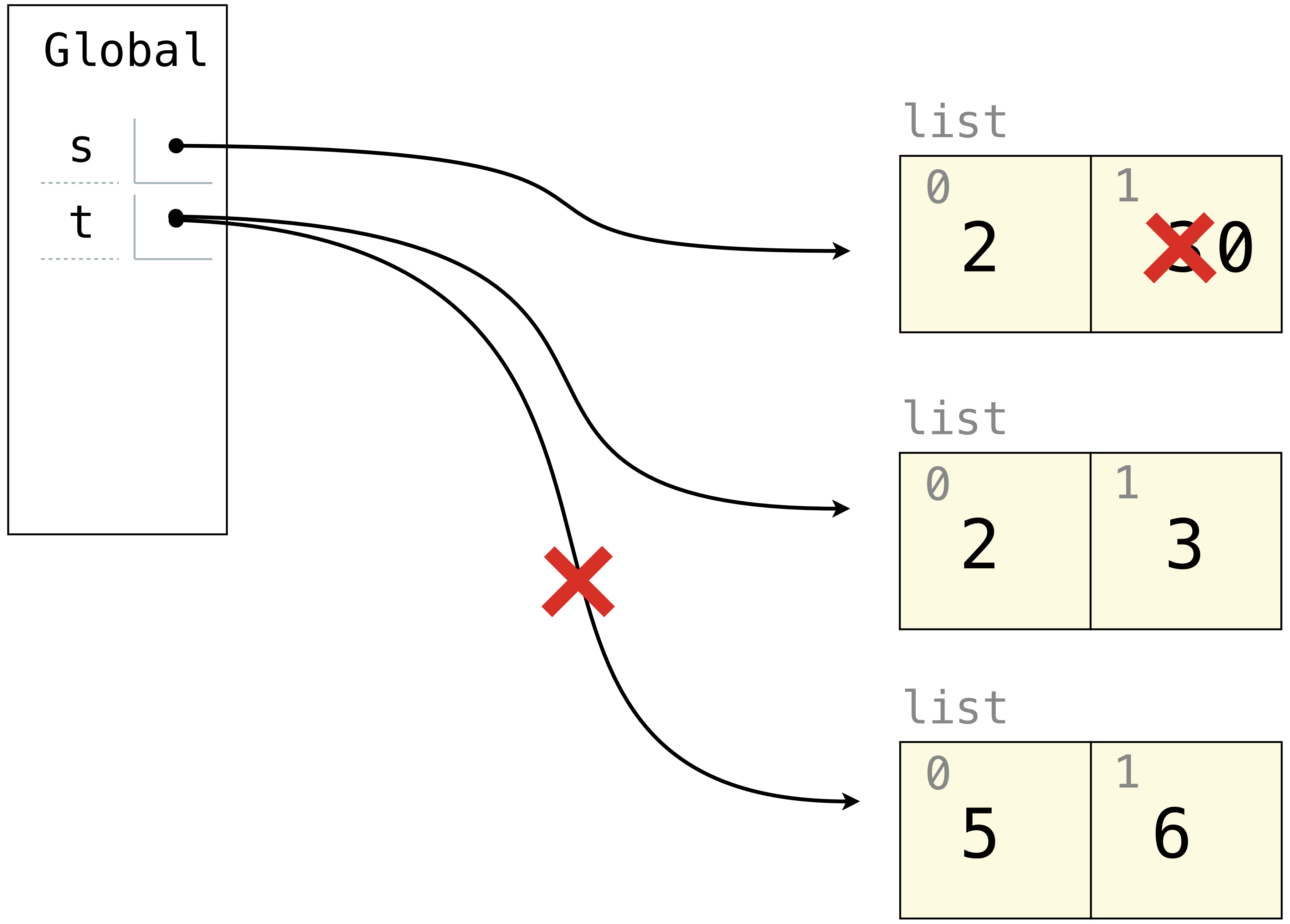
Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		s → [2, 3, 5, 6]
		t → [5, 0]

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`
		s → [2, 3]
		t → [5, 0]
		a → [2, 9, [5, 0]]
		b → [3, [5, 0]]

The **list** function also creates a new list containing existing elements	`t = list(s)`
	`s[1] = 0`
		s → [2, 0]
		t → [2, 3]



Lists in Environment Diagrams

Assume that before each example below we execute:

```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

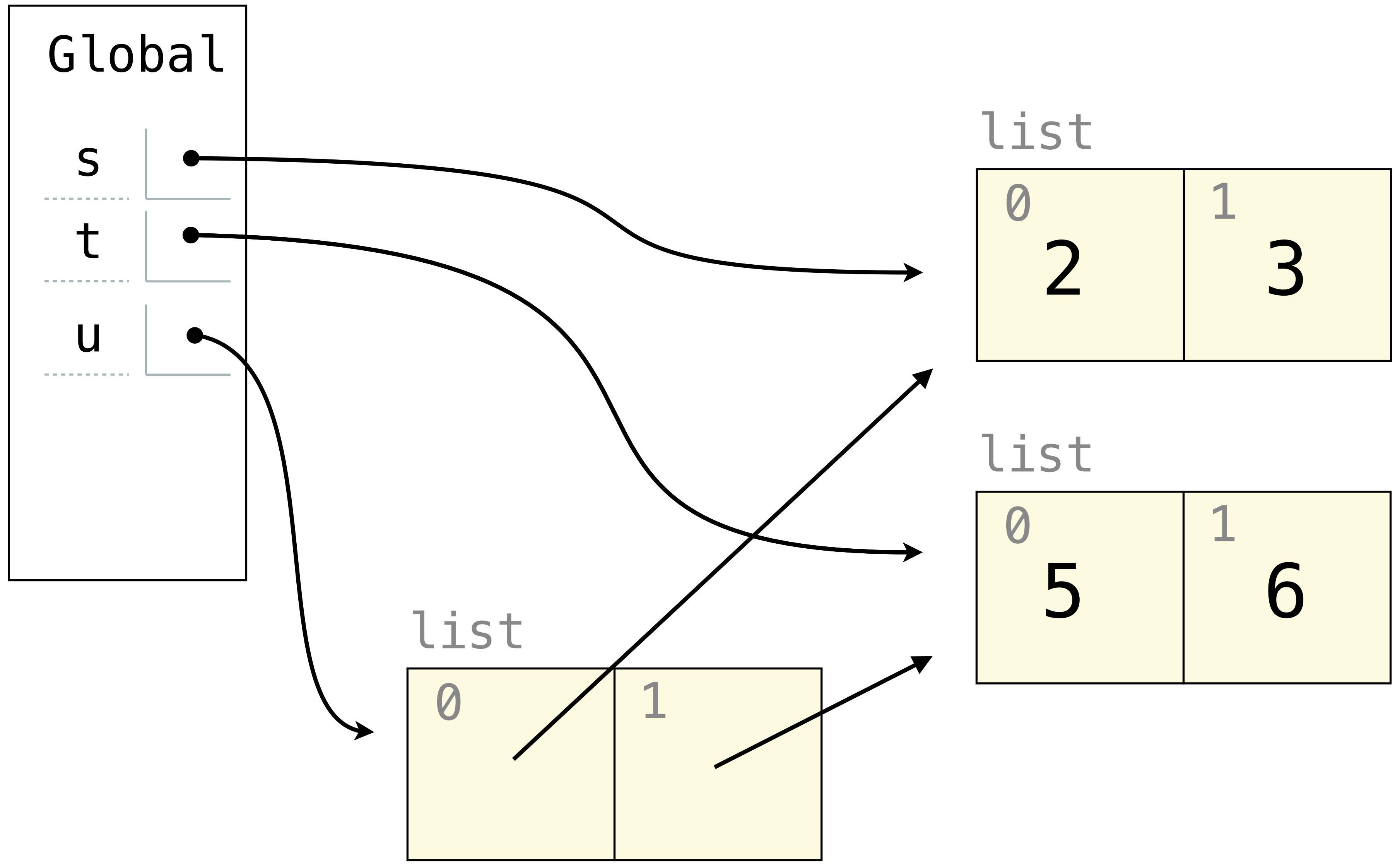
append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	s.extend(t)
	t[1] = 0
		s → [2, 3, 5, 6]
		t → [5, 0]

addition & **slicing** create new lists containing existing elements	a = s + [t]
	b = a[1:]
	a[1] = 9
	b[1][1] = 0
		s → [2, 3]
		t → [5, 0]
		a → [2, 9, [5, 0]]
		b → [3, [5, 0]]

The **list** function also creates a new list containing existing elements	t = list(s)
	s[1] = 0
		s → [2, 0]
		t → [2, 3]

[...] creates a new list	u = [s, t]	s → [2, 3] t → [5, 6] u → [[2, 3], [5, 6]]



Lists in Environment Diagrams

Assume that before each example below we execute:

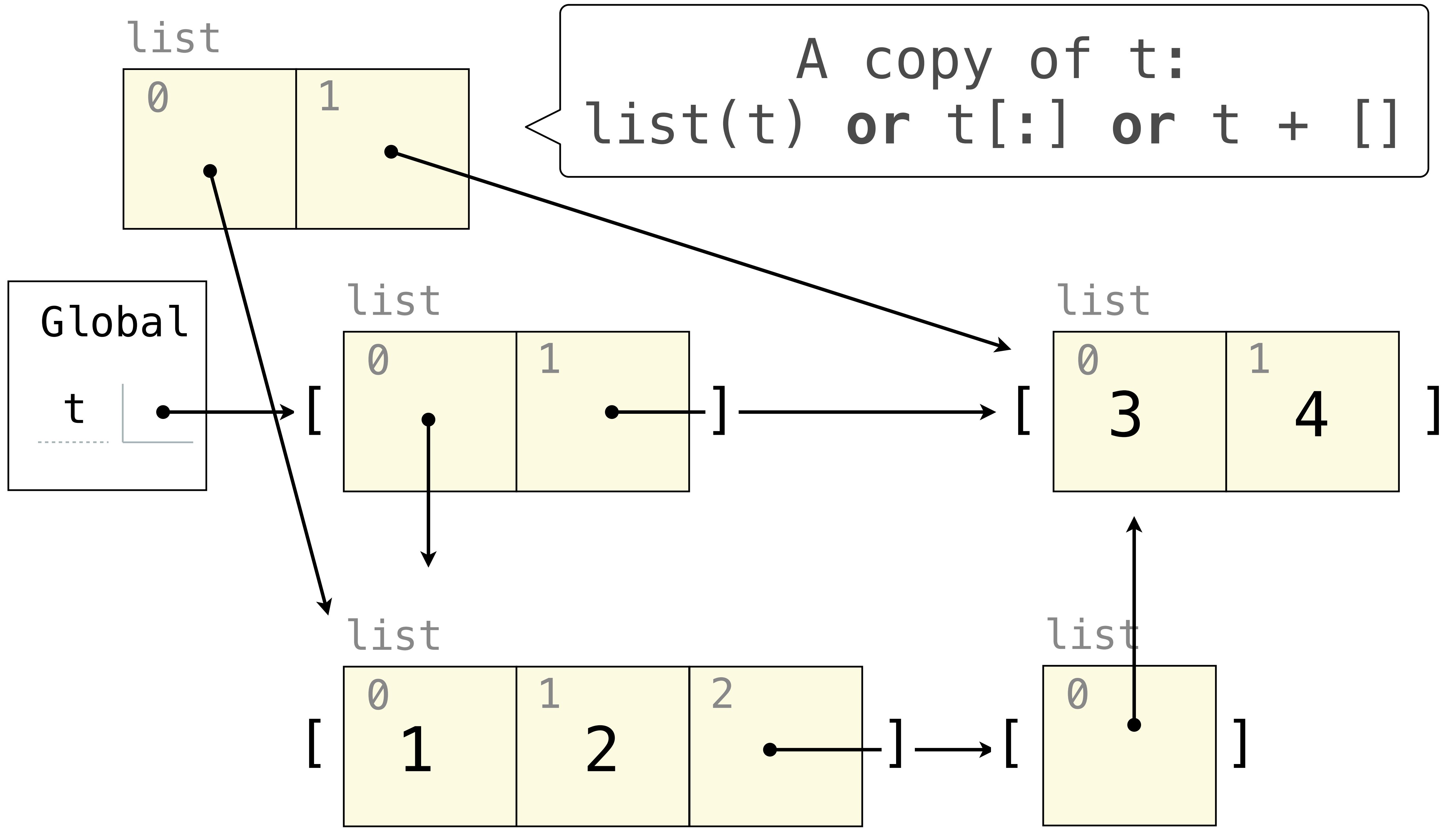
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result
pop removes & returns the last element	t = s.pop()	s → [2] t → 3

remove removes the first element equal to the argument	t.extend(t)
	t.remove(5)
		s → [2, 3]
		t → [6, 5, 6]

Lists in Lists in Lists in Environment Diagrams

```
t = [[1, 2], [3, 4]]
list(t)
t[0].append(t[1:2])
print(t)
```

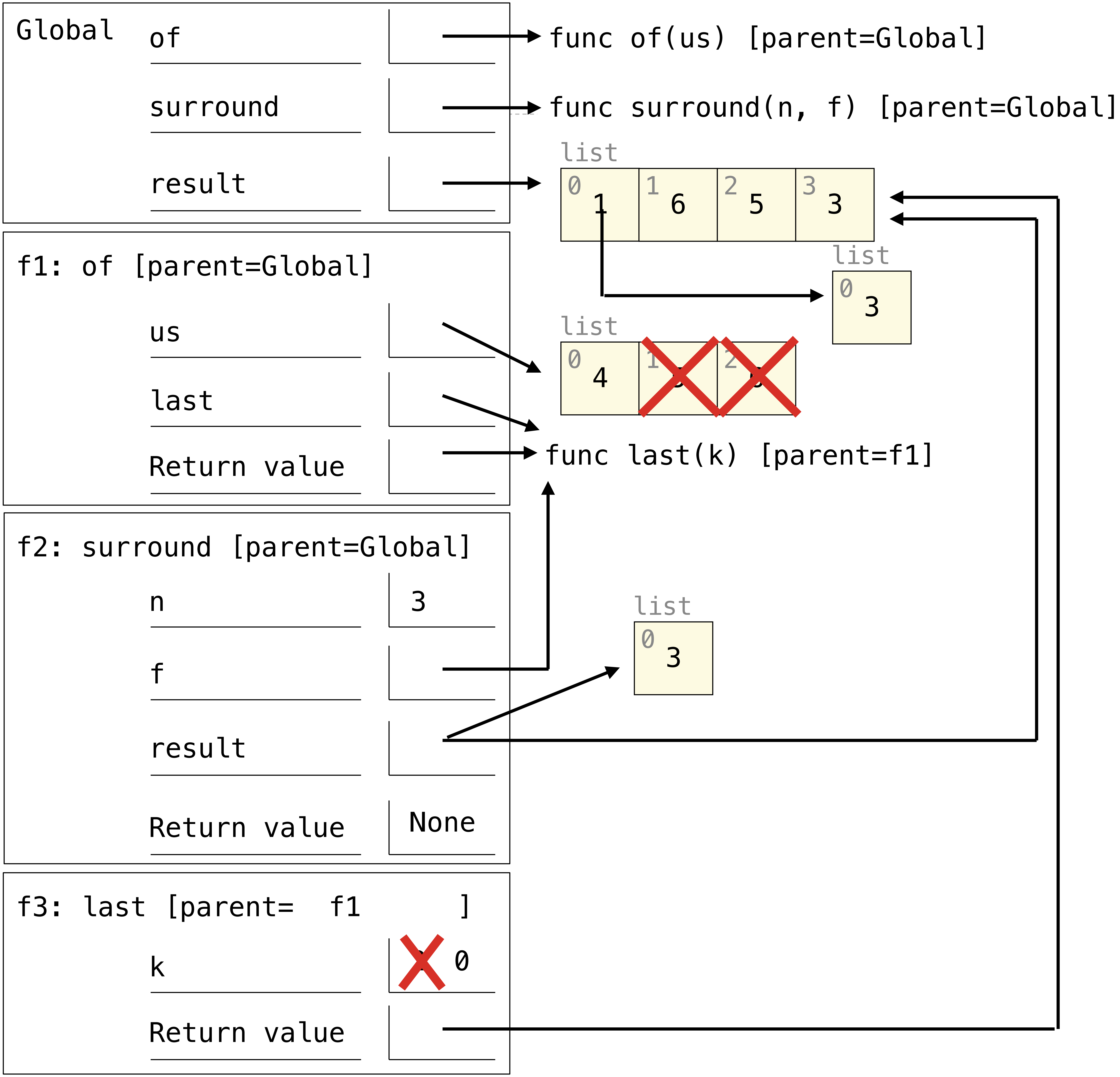


```
[[1, 2, [[3, 4]]], [3, 4]]
```

Fall 2022 Midterm 2 Question 2

```
def of(us):
    def last(k):
        "The last k items of us"
        while k > 0:
            result.append(us.pop())
            k = k - 1
        return result
    return last

def surround(n, f):
    "n is the first and last item of f(2)"
    result = [n]
    result = f(2)
    result[0] = [n]
    return result.append(n)

result = [1]
surround(3, of([4, 5, 6]))
print(result)
```

```
[[3], 6, 5, 3]
```



Global
of → func of(us) [parent=Global]
surround → func surround(n, f) [parent=Global]
result → list: 0: 1, 1: 6, 2: 5, 3: 3

list: 0: 3

list: 0: 4, 1: ~~5~~, 2: ~~6~~

f1: of [parent=Global]
us
last → func last(k) [parent=f1]
Return value

f2: surround [parent=Global]
n 3
f
result
Return value None

list: 0: 3

f3: last [parent= f1]
k ~~2~~ 0
Return value

Tree Practice

Spring 2023 Midterm 2 Question 4(a)

Implement exclude, which takes a tree t and a value x. It returns a tree containing the root node of t as well as each non-root node of t with a label not equal to x. The parent of a node in the result is its nearest ancestor node that is not excluded.

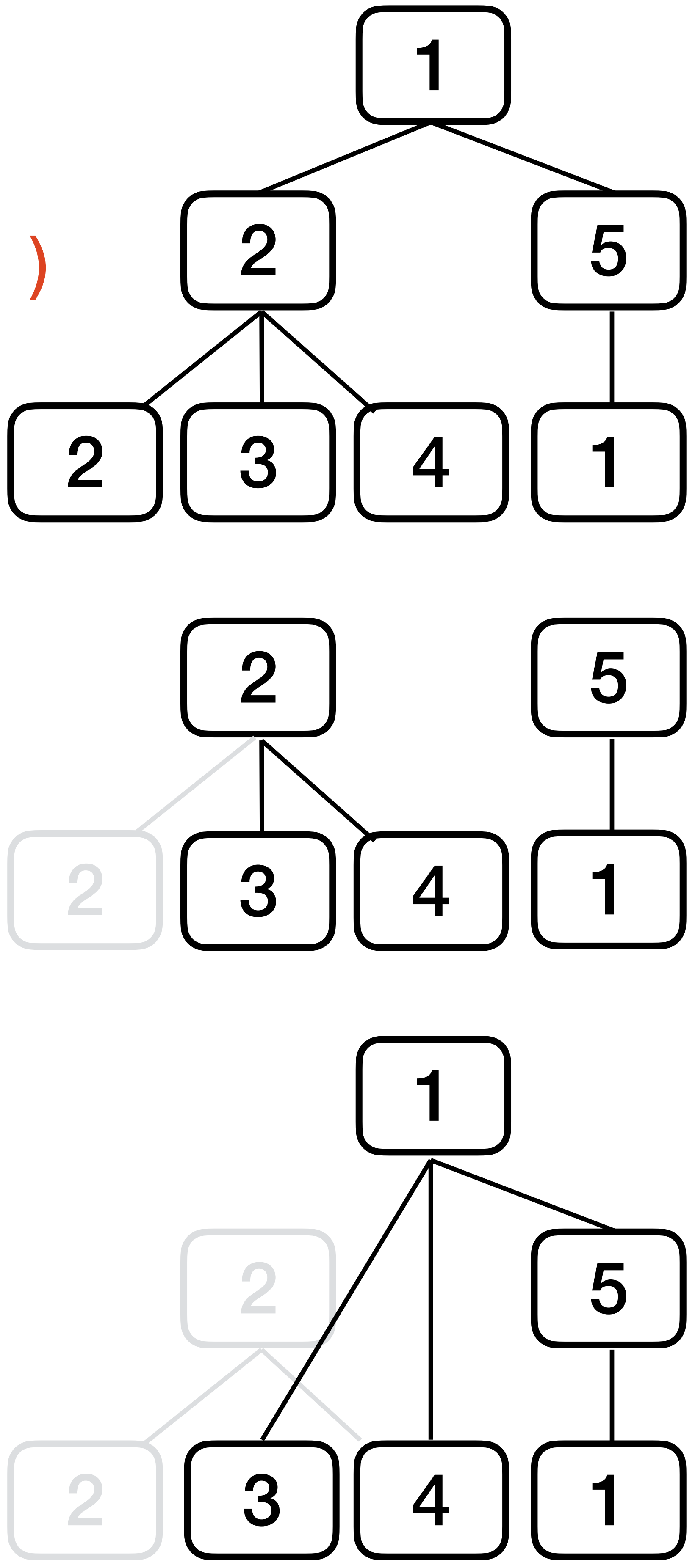
```
def exclude(t, x):
    """Return a tree with the non-root nodes of tree t labeled anything but x.

    >>> t = tree(1, [tree(2, [tree(2), tree(3), tree(4)]), tree(5, [tree(1)])])
    >>> exclude(t, 2)
    [1, [3], [4], [5, [1]]]
    >>> exclude(t, 1)  # The root node cannot be excluded
    [1, [2, [2], [3], [4]], [5]]
    """
    filtered_branches = map(lambda y: exclude(y, x), branches(t))
    bs = []
    for b in filtered_branches:
        if label(b) == x:
            bs.extend(branches(b))
        else:
            bs.append(b)
    return tree(label(t), bs)
```

In Spring 2023, 20% of students got this right

37% of students got this right

30% got it right; 1 of 4 options

24% got it right



Break: 5 minutes

Lists & Recursion

Recursion Example: Large Sums

Definition: A sublist of a list s is a list with some (or none or all) of the elements of s.

Implement **large,** which takes a list of positive numbers **s** and a non-negative number **n.**

It returns the sublist of **s** with the largest sum that is less than or equal to **n.**

You may call **sum_list,** which takes a list and returns the sum of its elements.

```
def large(s, n):
    """Return the sublist of positive numbers s with the
    largest sum that is less than or equal to n.

    >>> large([4, 2, 5, 6, 7], 3)
    [2]
    >>> large([4, 2, 5, 6, 7], 8)
    [2, 6]
    >>> large([4, 2, 5, 6, 7], 19)
    [4, 2, 6, 7]
    >>> large([4, 2, 5, 6, 7], 20)
    [2, 5, 6, 7]
    """
    if s == []:
        return []
    elif s[0] > n:
        return large(s[1:], n)
    else:
        first = s[0]
        with_s0 = [first] + large(s[1:], n - first)
        without_s0 = large(s[1:], n)
        if sum_list(with_s0) > sum_list(without_s0):
            return with_s0
        else:
            return without_s0
```

Add Consecutive

https://cs61a.org/exam/su24/midterm/61a-su24-midterm.pdf#page=11

Tree Recursion Exam Problem

`https://cs61a.org/exam/su22/midterm/61a-su22-midterm.pdf#page=10`