Data Examples

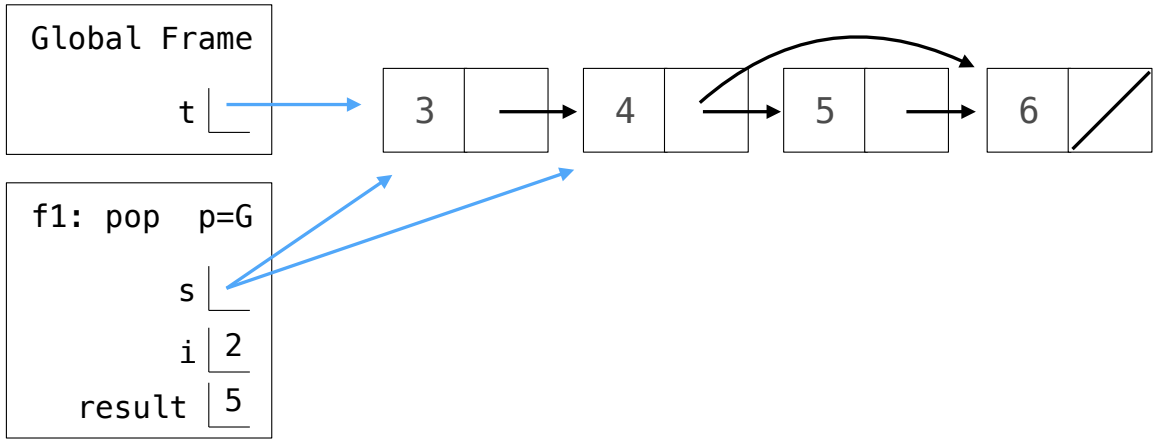
Announcements

More Linked List Practice

Pop

Implement pop, which takes a linked list s and positive integer i. It removes and returns the element at index i of s (assuming s.first has index 0).

```
def pop(s, i):
    """Remove and return element i from linked list s for positive i.
    >>> t = Link(3, Link(4, Link(5, Link(6))))
    >>> pop(t, 2)
    5
    >>> pop(t, 2)
    6
    >>> pop(t, 1)
    4
    >>> t
    Link(3)
    """
    assert i > 0 and i < length(s)
    for x in range(  i - 1  ):
        s = s.rest
    result = s.rest.first
    s.rest = s.rest.rest
    return   result
```

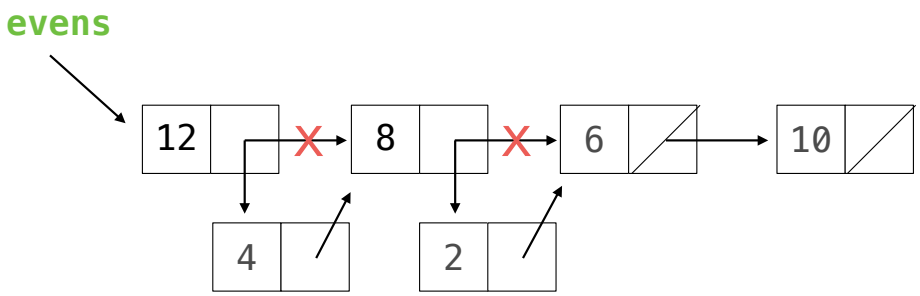


pollev.com/cs61a

Inserting into a Linked List

```
def insert_link(s, x, i):
    """Insert x into linked list s at index i.

    >>> evens = Link(4, Link(2, Link(6)))
    >>> insert_link(evens, 8, 1)
    >>> insert_link(evens, 10, 4)
    >>> insert_link(evens, 12, 0)
    >>> insert_link(evens, 14, 10)
    Index out of range
    >>> print(evens)
    (12 4 8 2 6 10)
    """
    if s is Link.empty:
        print('Index out of range')
    elif i == 0:
        second = Link(s.first, s.rest)
        s.first = x
        s.rest = second
    elif i == 1 and s.rest is Link.empty:
        s.rest = Link(x)
    else:
        insert_link(s.rest, x, i-1)
```



pollev.com/cs61a

Lists

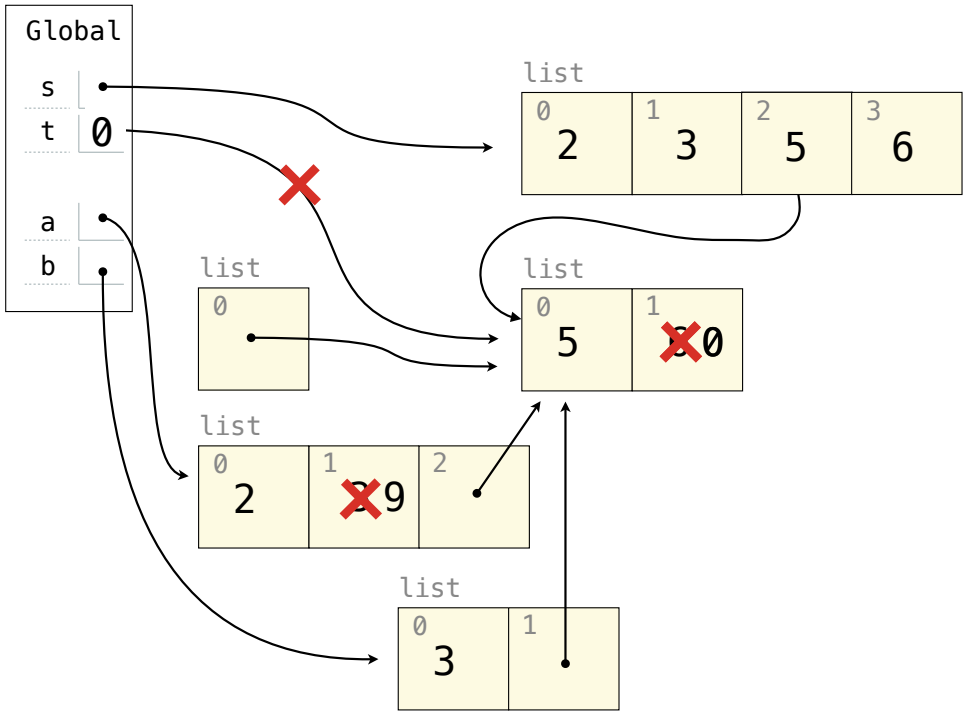
Lists in Environment Diagrams

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

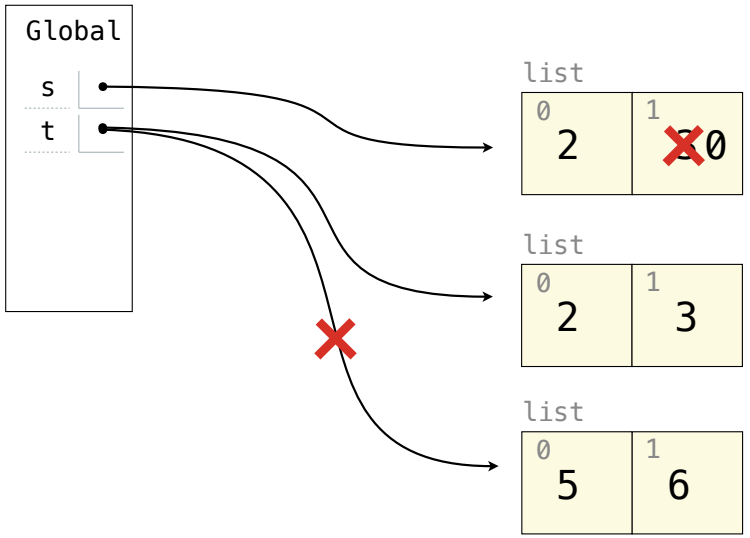
Assume that before each example below we execute:

```
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:

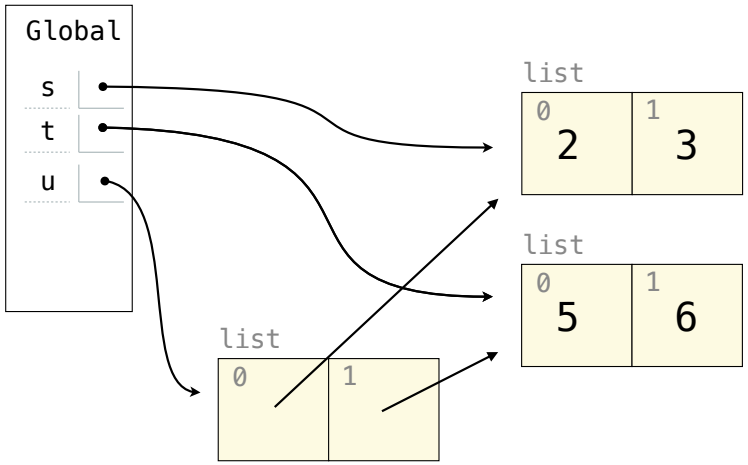
```
s = [2, 3]
t = [5, 6]
```

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

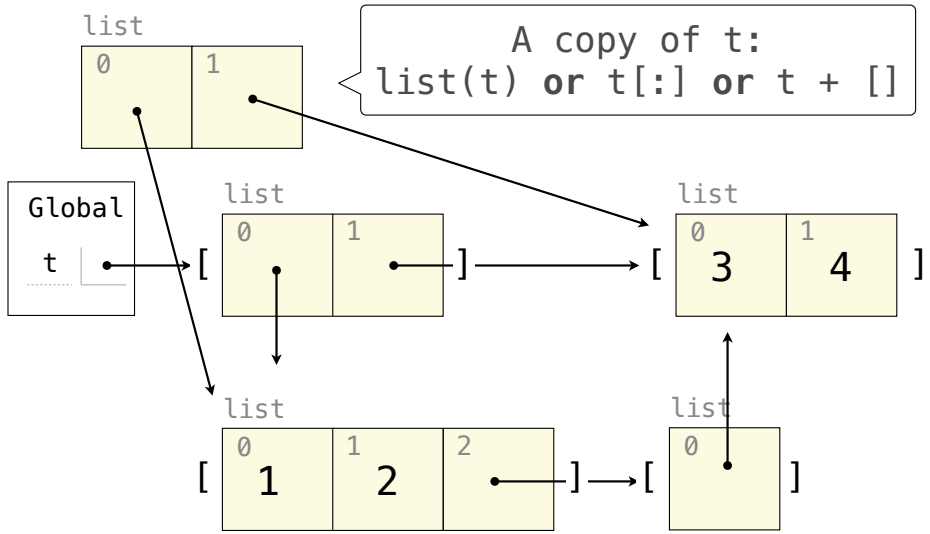
Assume that before each example below we execute:

```
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)
```



The call to print(t) displays `[[1, 2, [[3, 4]]], [3, 4]]`

pollev.com/cs61a

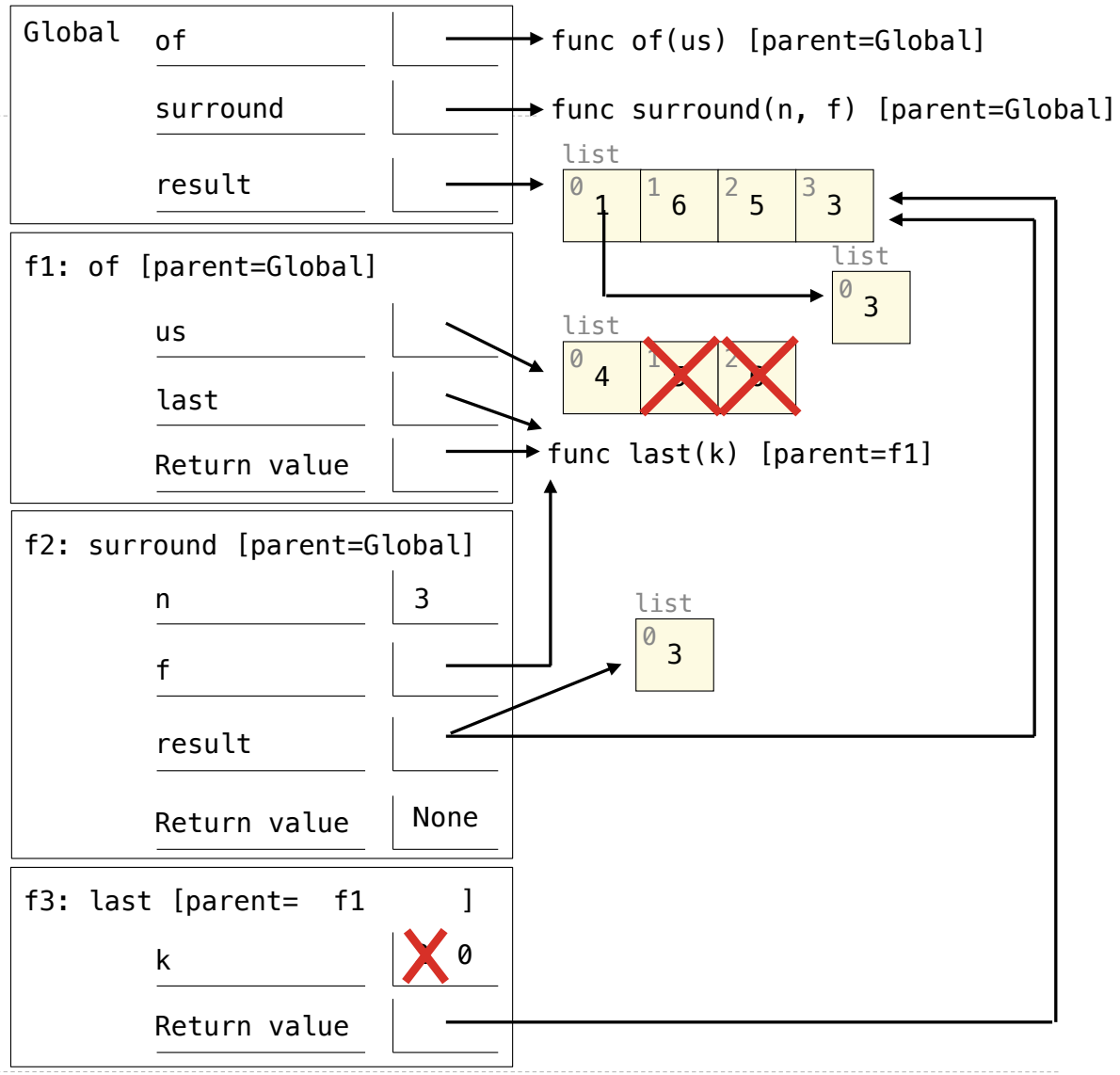
Fall 2022 Midterm 2 Question 2

```
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]
```



```
class Tree:
    """A tree is a label and a list of branches."""
    def __init__(self, label, branches=[]):
        self.label = label
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)

    def is_leaf(self):
        return not self.branches
```

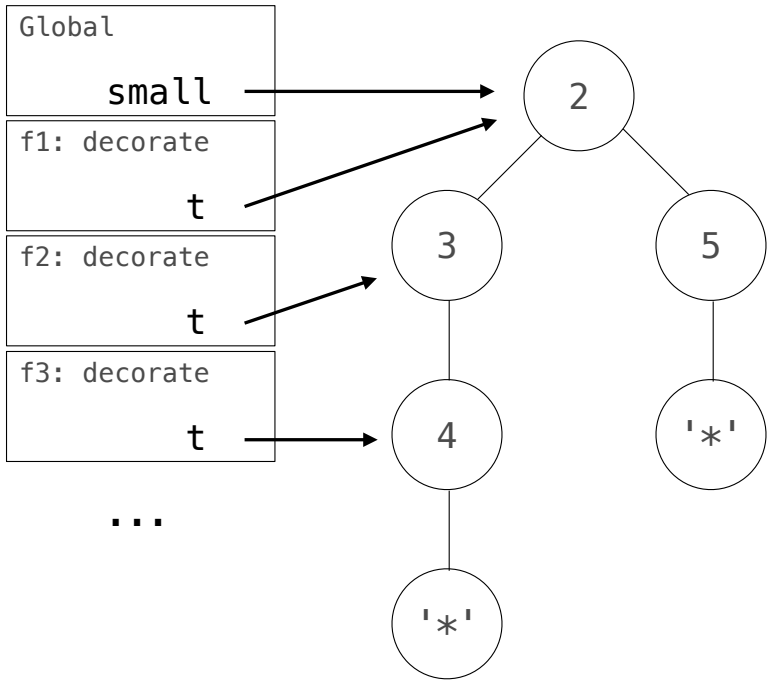
Trees

Heracles, Iolaus and the Hydra, Paestan black-figure hydra C6th B.C., The J. Paul Getty Museum

Warm-Up Question

Implement decorate, which takes a Tree instance t and mutates the tree by adding a new child labeled '*' to each of its leaves.

```
def decorate(t):
    """Add a * child to each leaf of Tree t.
    >>> small = Tree(2, [Tree(3, [Tree(4)]), Tree(5)])
    >>> print(small)
    2
      3
        4
      5
    >>> decorate(small)
    >>> print(small)
    2
      3
        4
          *
      5
        *
    >>> decorate(small)
    >>> print(small)
    2
      3
        4
          *
            *
      5
        *
          *
    """
    if t.is_leaf():
        t.branches.append(Tree('*'))
    else:
        for b in t.branches:
            decorate(b)
```



pollev.com/cs61a

Fall 2022 Midterm 2 Question 4(b)

A *hydra* is a Tree with a special structure. Each node has 0 or 2 children. All leaves are heads labeled 1. Each non-leaf body node is labeled with the number of leaves among its descendants.

Implement **chop_head(hydra, n)**, which takes a hydra and a positive integer n. It mutates hydra by chopping off the nth head from the left, which adds two new adjacent heads in its place. Update all ancestor labels.

```
def chop_head(hydra,  n):
    assert n > 0 and n <= hydra.label
    if hydra.is_leaf():
        hydra.label = 2
        hydra.branches = [Tree(1),  Tree(1)]
    else:
        hydra.label += 1
        left, right = hydra.branches
        if n > left.label :
            chop_head(right, n - left.label )
        else:
            chop_head(left, n)
```