Data Examples

Announcements
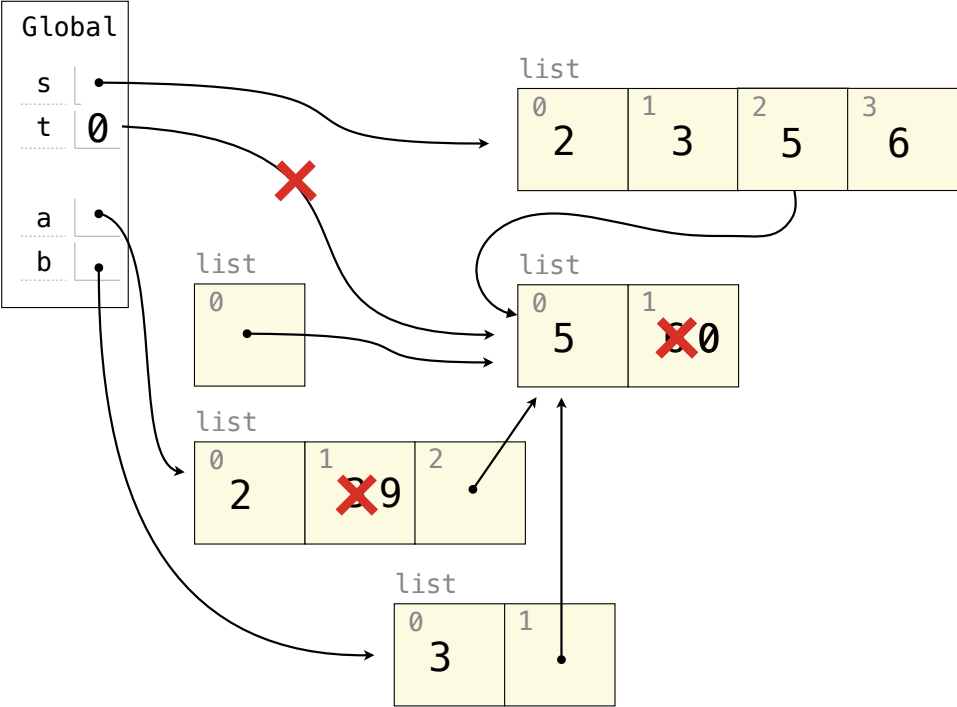
Lists

Lists in Environment Diagrams

Assume that before each example below we execute:
```
s = [2, 3]
t = [5, 6]
```

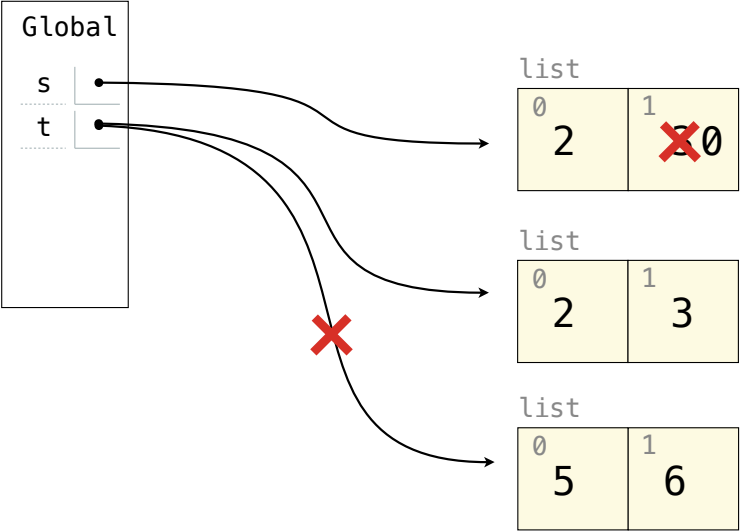| Operation	Example	Result
append adds one element to a list	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
extend adds all elements in one list to another list	s.extend(t) t[1] = 0	s → [2, 3, 5, 6] t → [5, 0]
addition & **slicing** create new lists containing existing elements	a = s + [t] b = a[1:] a[1] = 9 b[1][1] = 0	s → [2, 3] t → [5, 0] a → [2, 9, [5, 0]] b → [3, [5, 0]]

4

Lists in Environment Diagrams

Assume that before each example below we execute:
```
s = [2, 3]
t = [5, 6]
```

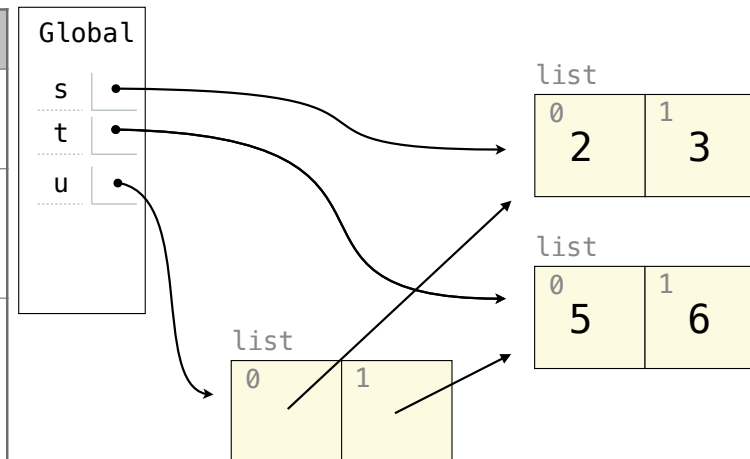| Operation	Example	Result
append adds one element to a list	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
extend adds all elements in one list to another list	s.extend(t) t[1] = 0	s → [2, 3, 5, 6] t → [5, 0]
addition & **slicing** create new lists containing existing elements	a = s + [t] b = a[1:] a[1] = 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]

Global

s
t

list
0	1
2	✖0

list
0	1
2	3

list
0	1
5	6

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
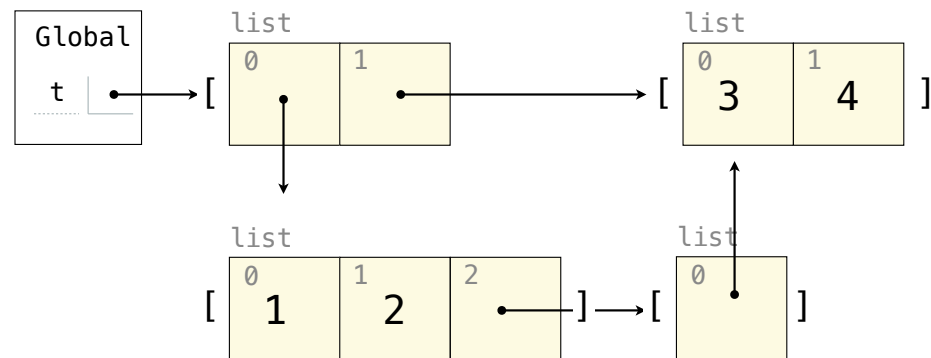
Assume that before each example below we execute:
```
s = [2, 3]
t = [5, 6]
```

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]]
t[0].append(t[1:2])
```
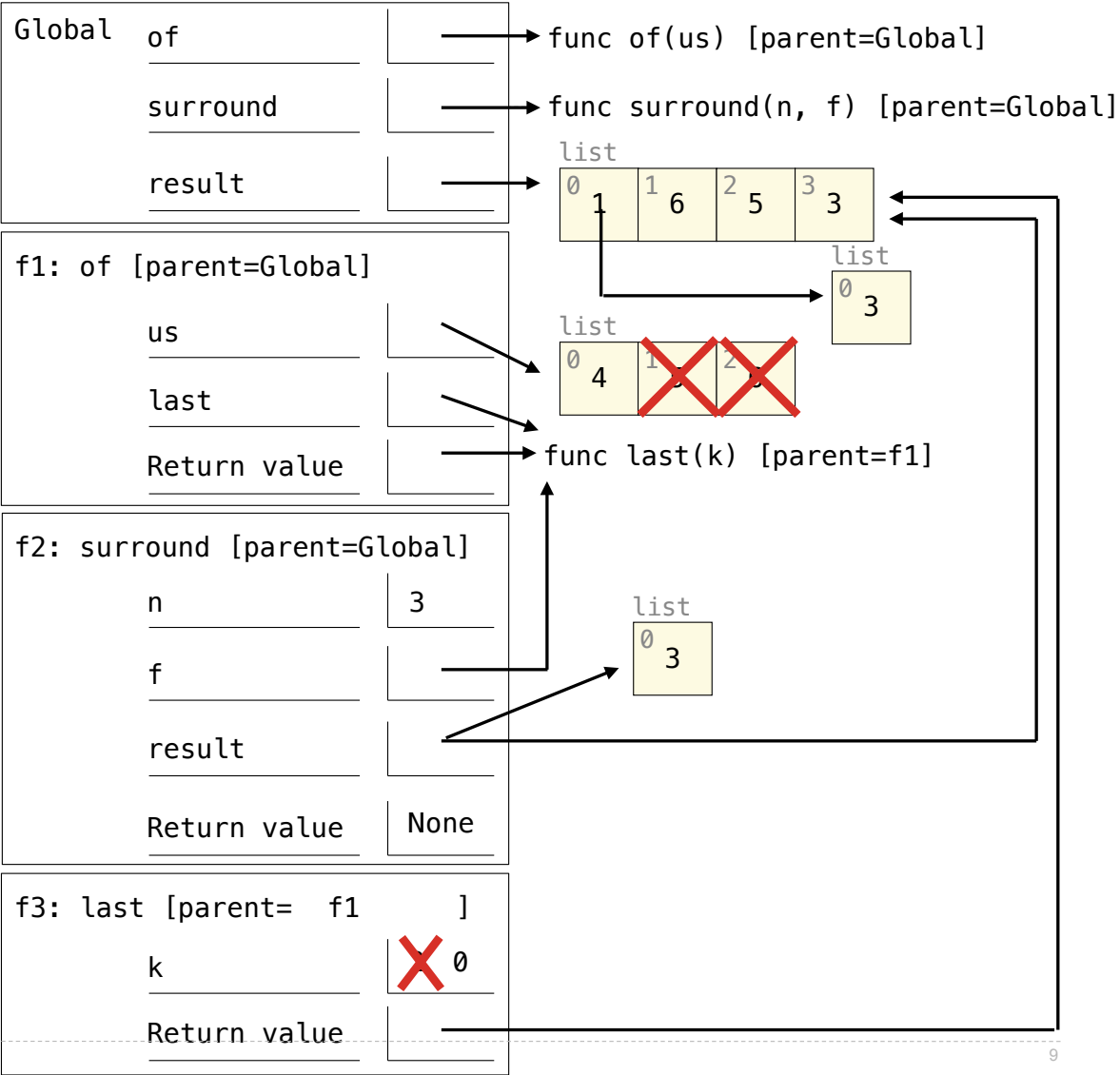

[[1, 2, [[3, 4]]], [3, 4]]

Fall 2022 Midterm 2 Question 2

```python
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)
```
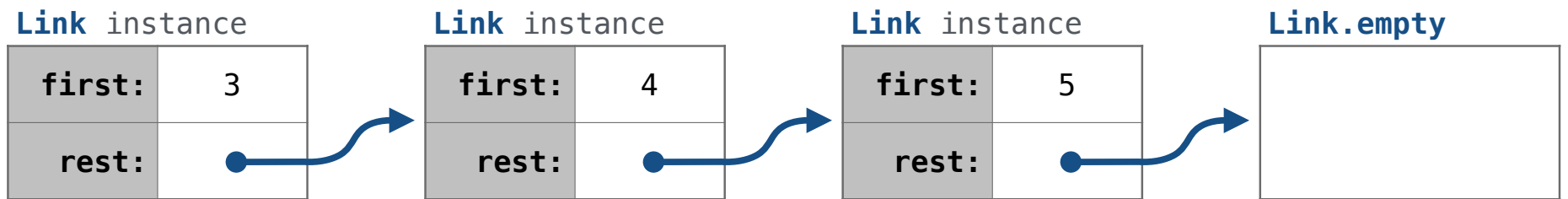
                [[3], 6, 5, 3]

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~~ | ~~2~~ |

f1: of [parent=Global]

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

f2: surround [parent=Global]

n	3
f	
result	
Return value	None

list
| 0 3 |

f3: last [parent= f1]

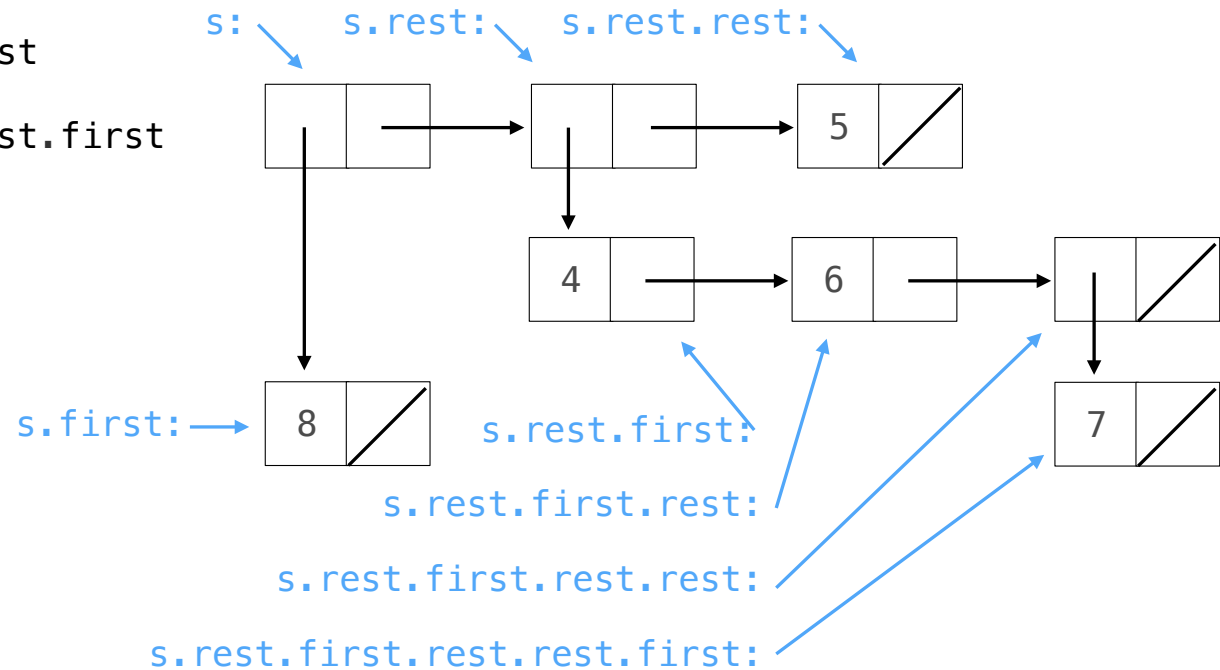| k | ✗ 0 |
| Return value | |

9

Linked Lists Practice

Linked List Notation

`s = Link(3, Link(4, Link(5)))`

Nested Linked Lists

```
>>> s = Link(Link(8), Link(Link(4, Link(6, Link(Link(7)))), Link(5)))
>>> print(s)
<<8> <4 6 <7>> 5>
>>> s.first.first
8
>>> s.rest.first.rest.rest.first
Link(7)
>>> s.rest.first.rest.rest.first.first
7
```

Recursion and Iteration

Many linked list processing functions can be written both iteratively and recursively

Recursive approach:

- What recursive call do you make?
- What does this recursive call do/return?
- How is this result useful in solving the problem?

```
def length(s):
    """The number of elements in s.

    >>> length(Link(3, Link(4, Link(5))))
    3
    """
    if s is Link.empty:
        return 0
    else:
        return _____1 + length(s.rest)_____
```

Iterative approach:

- Describe a process that solves the problem.
- Figure out what additional names you need to carry out this process.
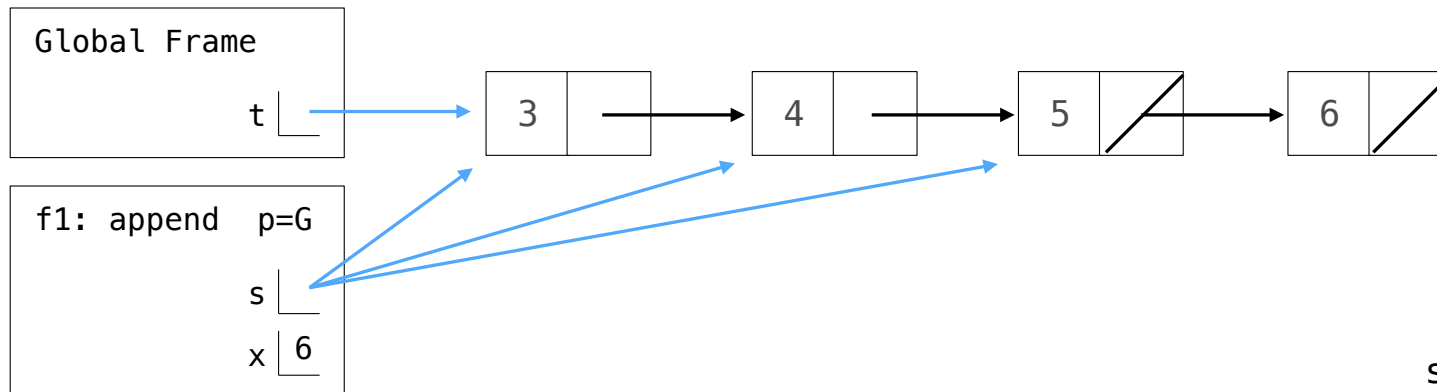- Implement the process using those names.

```
def length(s):
    """The number of elements in s.

    >>> length(Link(3, Link(4, Link(5))))
    3
    """
    k = __0__
    while __s is not Link.empty__:
        s, k = s.rest, __k + 1__
    return k
```

Linked List Mutation

To change the contents of a linked list, assign to first and rest attributes

Example: Append x to the end of non-empty s

```
>>> t = Link(3, Link(4, Link(5)))
>>> append(t, 6)
>>> t
Link(3, Link(4, Link(5, Link(6))))
```


s = s.rest

s.rest = Link(x)

Recursion and Iteration

Many linked list processing functions can be written both iteratively and recursively

Recursive approach:

- What recursive call do you make?
- What does this recursive call do/return?
- How is this result useful in solving the problem?

```python
def append(s, x):
    """Append x to the end of non-empty s.
    >>> append(s, 6)  # returns None!
    >>> print(s)
    <3 4 5 6>
    """
    if ___s.rest is not Link.empty___:
        append(_s.rest_, _x_)
    else:
        _s.rest = Link(x)_____
```

Iterative approach:

- Describe a process that solves the problem.
- Figure out what additional names you need to carry out this process.
- Implement the process using those names.

```python
def append(s, x):
    """Append x to the end of non-empty s.
    >>> append(s, 6)  # returns None!
    >>> print(s)
    <3 4 5 6>
    """
    while ___s.rest is not Link.empty___:
        _s = s.rest_____
    _s.rest = Link(x)_____
```

Constructing a Linked List

Build the rest of the linked list, then combine it with the first element.

```
s = Link.empty
s = Link(5, s)
s = Link(4, s)
s = Link(3, s)
```

```
+---+---+      +---+---+      +---+--/+
| 3 | ------->| 4 | ------->| 5 |  / |
+---+---+      +---+---+      +---+--/+
```

```python
def range_link(start, end):
    """Return a Link containing consecutive
    integers from start up to end.

    >>> range_link(3, 6)
    Link(3, Link(4, Link(5)))
    """

    if start >= end:

        return Link.empty

    else:

        return  Link(start, range_link(start + 1, end))
```

```python
def range_link(start, end):
    """Return a Link containing consecutive
    integers from start to end.

    >>> range_link(3, 6)
    Link(3, Link(4, Link(5)))
    """

    s = Link.empty

    k =  end - 1

    while  k >= start :

        s = Link(k, s)

        k -= 1

    return s
```

16

Example: 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____
```