Data Examples
Announcements
Linked List Construction
Constructing a Linked List

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

```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 = _________
    while _________:
        s = Link(k, s)
        k -= 1
    return s
```

Lists
Lists in Environment Diagrams

Assume that before each example below we execute:

\( s = [2, 3] \)
\( t = [5, 6] \)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| **append** adds one element to a list | \( s.append(t) \)  
\( t = 0 \) | \( s \rightarrow [2, 3, [5, 6]] \)  
\( t \rightarrow 0 \) |
| **extend** adds all elements in one list to another list | \( s.extend(t) \)  
\( t[1] = 0 \) | \( s \rightarrow [2, 3, 5, 6] \)  
\( t \rightarrow [5, 0] \) |
| **addition & slicing** create new lists containing existing elements | \( a = s + [t] \)  
\( b = a[1:] \)  
\( a[1] = 9 \)  
\( b[1][1] = 0 \) | \( s \rightarrow [2, 3] \)  
\( t \rightarrow [5, 0] \)  
\( a \rightarrow [2, 9, [5, 0]] \)  
\( b \rightarrow [3, [5, 0]] \) |
Lists in Environment Diagrams

Assume that before each example below we execute:

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one element to a list</td>
<td>s.append(t) t = 0</td>
<td>s → [2, 3, [5, 6]] t → 0</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>s.extend(t) t[1] = 0</td>
<td>s → [2, 3, 5, 6] t → [5, 0]</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td>a = s + [t] b = a[1:] a[1] = 9 b[1][1] = 0</td>
<td>s → [2, 3] t → [5, 0] a → [2, 9, [5, 0]] b → [3, [5, 0]]</td>
</tr>
<tr>
<td>The <strong>list</strong> function also creates a new list containing existing elements</td>
<td>t = list(s) s[1] = 0</td>
<td>s → [2, 0] t → [2, 3]</td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>append</strong> adds one</td>
<td>s.append(t) t = 0</td>
<td>s → [2, 3, [5, 6]] t → 0</td>
</tr>
<tr>
<td>element to a list</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>extend</strong> adds all</td>
<td>s.extend(t) t[1] = 0</td>
<td>s → [2, 3, 5, 6] t → [5, 0]</td>
</tr>
<tr>
<td>elements in one list to</td>
<td></td>
<td></td>
</tr>
<tr>
<td>another list</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong></td>
<td>a = s + [t] b = a[1:]</td>
<td>s → [2, 3] t → [5, 0]</td>
</tr>
<tr>
<td>create new lists</td>
<td>a[1] = 9 b[1][1] = 0</td>
<td>a → [2, 9, [5, 0]] b → [3, [5, 0]]</td>
</tr>
<tr>
<td>containing existing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The <strong>list</strong> function</td>
<td>t = list(s) s[1] = 0</td>
<td>s → [2, 0] t → [2, 3]</td>
</tr>
<tr>
<td>also creates a new list</td>
<td></td>
<td></td>
</tr>
<tr>
<td>containing existing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>elements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[...] creates a new list</td>
<td>u = [s, t]</td>
<td>s → [2, 3] t → [5, 6] u → [[2, 3], [5, 6]]</td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pop</strong></td>
<td>t = s.pop()</td>
<td>s → [2] t → 3</td>
</tr>
<tr>
<td><strong>remove</strong></td>
<td>t.extend(t)</td>
<td>s → [2, 3] t → [6, 5, 6]</td>
</tr>
</tbody>
</table>
t = [[1, 2], [3, 4]]
t[0].append(t[1:2])

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

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

```python
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)
```

```