Data Examples
Announcements
Examples: Lists
Lists in Environment Diagrams
Lists in Environment Diagrams

Assume that before each example below we execute:
s = \[2, 3\]
t = \[5, 6\]
Lists in Environment Diagrams

Assume that before each example below we execute:

\[
\begin{align*}
  s &= [2, 3] \\
  t &= [5, 6]
\end{align*}
\]
Lists in Environment Diagrams

Assume that before each example below we execute:
\[
\begin{align*}
    s &= [2, 3] \\
    t &= [5, 6]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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>

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>append</td>
<td>adds one</td>
<td></td>
</tr>
<tr>
<td></td>
<td>element to a list</td>
<td></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)</td>
<td>t = 0</td>
</tr>
<tr>
<td>element to a list</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>t = 0</td>
<td></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>append adds one</td>
<td>s.append(t)</td>
<td></td>
</tr>
<tr>
<td>element to a list</td>
<td>t = 0</td>
<td></td>
</tr>
</tbody>
</table>

Global
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>append adds one</td>
<td>s.append(t)</td>
<td></td>
</tr>
<tr>
<td>element to a list</td>
<td>t = 0</td>
<td></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>append adds one</td>
<td>s.append(t)</td>
<td>t = 0</td>
</tr>
<tr>
<td>element to a list</td>
<td></td>
<td></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>append adds one</td>
<td>s.append(t)</td>
<td>t = 0</td>
</tr>
<tr>
<td>element to a list</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
Global
s

    t

0

list
0 2 1 3 2

list
0 5 1 6
```
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>append adds one</td>
<td>s.append(t)</td>
<td>s → [2, 3, [5, 6]]</td>
</tr>
<tr>
<td>element to a list</td>
<td>t = 0</td>
<td>t → 0</td>
</tr>
</tbody>
</table>

![Diagram showing the environment and list operations]
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>
</tbody>
</table>

![Diagram showing the effect of the append operation on lists in an environment diagram.](image-url)
Lists in Environment Diagrams

Assume that before each example below we execute:

\[
\begin{align*}
  s &= [2, 3] \\
  t &= [5, 6]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append adds one</td>
<td>s.append(t)</td>
<td>(s \rightarrow [2, 3, [5, 6]])</td>
</tr>
<tr>
<td>element to a list</td>
<td>(t = 0)</td>
<td>(t \rightarrow 0)</td>
</tr>
<tr>
<td>extend adds all</td>
<td></td>
<td></td>
</tr>
<tr>
<td>elements in one</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list to another</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```python
s = [2, 3]
t = [5, 6]
s.append(t)  # \(s \rightarrow [2, 3, [5, 6]]\)
t = 0        # \(t \rightarrow 0\)
```
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>append 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>extend adds all elements in one list to another list</td>
<td>s.extend(t) t[1] = 0</td>
<td></td>
</tr>
</tbody>
</table>

Global

```
  s
  t
```

list

```
  0   1
  2   3
```

list

```
  0   1
  5   6
```
Lists in Environment Diagrams

Assume that before each example below we execute:

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>append adds one</td>
<td>( s.\text{append}(t) )</td>
<td>( s \rightarrow [2, 3, [5, 6]] )</td>
</tr>
<tr>
<td>element to a list</td>
<td>( t = 0 )</td>
<td>( t \rightarrow 0 )</td>
</tr>
<tr>
<td>extend adds all</td>
<td>( s.\text{extend}(t) )</td>
<td>( t[1] = 0 )</td>
</tr>
<tr>
<td>elements in one</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list to another</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list</td>
<td></td>
<td></td>
</tr>
<tr>
<td>list</td>
<td></td>
<td></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 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></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>append adds one element to a list</td>
<td>[ s.append(t) ]  [ t = 0 ]</td>
<td>[ s \rightarrow [2, 3, [5, 6]] ] [ t \rightarrow 0 ]</td>
</tr>
<tr>
<td>extend adds all elements in one list to another list</td>
<td>[ s.extend(t) ]  [ t[1] = 0 ]</td>
<td>[ s \rightarrow [2, 3, 5, 6] ] [ t \rightarrow [5, 0] ]</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>
</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 | \( s.extend(t) \)
\( t[1] = 0 \) | \( s \rightarrow [2, 3, 5, 6] \)
\( t \rightarrow [5, 0] \) | |
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) \n t = 0</td>
<td>s → [2, 3, [5, 6]] \n t → 0</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>s.extend(t) \n t[1] = 0</td>
<td>s → [2, 3, 5, 6] \n t → [5, 0]</td>
</tr>
</tbody>
</table>

addition & slicing create new lists containing existing elements

![Environment Diagram](image-url)
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 → [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 |                      |
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></td>
</tr>
</tbody>
</table>

![Environment Diagram]
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 | \( s = [2, 3] \)  \( t = [5, 6] \)  
\[ s.append(t) \]  
\[ t = 0 \] | \( s \rightarrow [2, 3, [5, 6]] \)  
\( t \rightarrow 0 \) | |
| extend adds all elements in one list to another list | \( s = [2, 3] \)  \( t = [5, 6] \)  
\[ s.extend(t) \]  
\[ t[1] = 0 \] | \( s \rightarrow [2, 3, 5, 6] \)  
\( t \rightarrow [5, 0] \) | |
| addition & slicing create new lists containing existing elements | \( a = [2, 3, [5, 6]] \)  \( b = [5, 0] \)  
\[ a[1] = 9 \]  
\[ b[1][1] = 0 \] | |

Global `s` list

Global `t` list

`list`

```python
s = [2, 3]
t = [5, 6]
s.append(t)  
t = 0
s = [2, 3, [5, 6]]
t = 0
s.extend(t)  
t[1] = 0
s = [2, 3, 5, 6]
t = [5, 0]
a = s + [t]
b = a[1:]  
a[1] = 9
b[1][1] = 0
```
### Lists in Environment Diagrams

Assume that before each example below we execute:

\[
\begin{align*}
s &= [2, 3] \\
t &= [5, 6]
\end{align*}
\]

<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><code>s.append(t)</code></td>
<td><code>s → [2, 3, [5, 6]]</code></td>
</tr>
<tr>
<td></td>
<td><code>t = 0</code></td>
<td><code>t → 0</code></td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td><code>s.extend(t)</code></td>
<td><code>s → [2, 3, 5, 6]</code></td>
</tr>
<tr>
<td></td>
<td><code>t[1] = 0</code></td>
<td><code>t → [5, 0]</code></td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td><code>a = s + [t]</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>b = a[1:]</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>a[1] = 9</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>b[1][1] = 0</code></td>
<td></td>
</tr>
</tbody>
</table>

![Environment Diagram](image)

- **Global**: `s` and `t` list
- **List**: `0 2 3`, `5 6`
## 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 \rightarrow [2, 3, [5, 6]] ) ( t \rightarrow 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 \rightarrow [2, 3, 5, 6] ) ( t \rightarrow [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></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 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></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>
</table>
| 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] \[ \]
a[1] = 9 \[ \]
b[1] [1] = 0 | a | b |

Diagram:

- Global
- Lists:
  - 0 1 2 3
  - 0 5 1 6
  - 0 3 1
  - a
  - b

Diagram arrows:
- From Global to Lists:
  - s → 2
  - t → 0
- Between Lists:
  - 0 → 1
  - 1 → 2
  - 2 → 3
  - 3 → 1
  - a → b
  - b → a
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>append 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>extend 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>addition &amp; slicing create new lists containing existing elements</td>
<td>a = s + [t] b = a[1:] a[1] = 9 b[1][1] = 0</td>
<td></td>
</tr>
</tbody>
</table>

Global list

```
0 2 3

```

```
0 5 6

```

```
0 3 1

```
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>append adds one element to a list</td>
<td>s.append(t)</td>
<td>s → [2, 3, [5, 6]]</td>
</tr>
<tr>
<td></td>
<td>t = 0</td>
<td>t → 0</td>
</tr>
<tr>
<td>extend adds all elements in one list to another list</td>
<td>s.extend(t)</td>
<td>s → [2, 3, 5, 6]</td>
</tr>
<tr>
<td></td>
<td>t[1] = 0</td>
<td>t → [5, 0]</td>
</tr>
<tr>
<td>addition &amp; slicing create new lists containing existing elements</td>
<td>a = s + [t]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b = a[1:]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a[1] = 9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b[1][1] = 0</td>
<td></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>append 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>extend 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>addition &amp; slicing 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>
</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 element to a list</td>
<td>( s.\text{append}(t) )  ( t = 0 )</td>
<td>( s \to [2, 3, [5, 6]] )  ( t \to 0 )</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>( s.\text{extend}(t) )  ( t[1] = 0 )</td>
<td>( s \to [2, 3, 5, 6] )  ( t \to [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 \to [2, 3] )  ( t \to [5, 0] )  ( a \to [2, 9, [5, 0]] )  ( b \to [3, [5, 0]] )</td>
</tr>
<tr>
<td>The <strong>list</strong> function also creates a new list containing existing elements</td>
<td>( t = \text{list}(s) )  ( s[1] = 0 )</td>
<td></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 element to a list</td>
<td><code>s.append(t)</code>&lt;br&gt;<code>t = 0</code></td>
<td><code>s → [2, 3, [5, 6]]</code>&lt;br&gt;<code>t → 0</code></td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td><code>s.extend(t)</code>&lt;br&gt;<code>t[1] = 0</code></td>
<td><code>s → [2, 3, 5, 6]</code>&lt;br&gt;<code>t → [5, 0]</code></td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td><code>a = s + [t]</code>&lt;br&gt;<code>b = a[1:]</code>&lt;br&gt;<code>a[1] = 9</code>&lt;br&gt;<code>b[1][1] = 0</code></td>
<td><code>s → [2, 3]</code>&lt;br&gt;<code>t → [5, 0]</code>&lt;br&gt;<code>a → [2, 9, [5, 0]]</code>&lt;br&gt;<code>b → [3, [5, 0]]</code></td>
</tr>
<tr>
<td>The list function also creates a new list containing existing elements</td>
<td><code>t = list(s)</code>&lt;br&gt;<code>s[1] = 0</code></td>
<td></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 element to a list</td>
<td>( s.append(t) ) \hspace{1cm} ( t = 0 )</td>
<td>( s \to [2, 3, [5, 6]] ) \hspace{1cm} ( t \to 0 )</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td>( s.extend(t) ) \hspace{1cm} ( t[1] = 0 )</td>
<td>( s \to [2, 3, 5, 6] ) \hspace{1cm} ( t \to [5, 0] )</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td>( a = s + [t] ) \hspace{1cm} ( b = a[1:] ) \hspace{1cm} ( a[1] = 9 ) \hspace{1cm} ( b[1][1] = 0 )</td>
<td>( s \to [2, 3] ) \hspace{1cm} ( t \to [5, 0] ) \hspace{1cm} ( a \to [2, 9, [5, 0]] ) \hspace{1cm} ( b \to [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) ) \hspace{1cm} ( s[1] = 0 )</td>
<td></td>
</tr>
</tbody>
</table>

Global

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```
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 \rightarrow [2, 3, [5, 6]] ) ( t \rightarrow 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 \rightarrow [2, 3, 5, 6] ) ( t \rightarrow [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 \rightarrow [2, 3] ) ( t \rightarrow [5, 0] ) ( a \rightarrow [2, 9, [5, 0]] ) ( b \rightarrow [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></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 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 list 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>

The list function also creates a new list containing existing elements.
Lists in Environment Diagrams

Assume that before each example below we execute:
\[
\begin{align*}
  s &= [2, 3] \\
  t &= [5, 6]
\end{align*}
\]

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

Assume that before each example below we execute:
\[
\begin{align*}
  s &= [2, 3] \\
  t &= [5, 6]
\end{align*}
\]

<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 → [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]                                    |
| **slice assignment** replaces a slice with new values | s[0:0] = t  
  s[3:] = t  
  t[1] = 0   |                                      |
## Lists in Environment Diagrams

Assume that before each example below we execute:

```python
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><code>s.append(t)</code> t = 0</td>
<td><code>s → [2, 3, [5, 6]]</code> t → 0</td>
</tr>
<tr>
<td><strong>extend</strong> adds all elements in one list to another list</td>
<td><code>s.extend(t)</code> t[1] = 0</td>
<td><code>s → [2, 3, 5, 6]</code> t → [5, 0]</td>
</tr>
<tr>
<td><strong>addition &amp; slicing</strong> create new lists containing existing elements</td>
<td><code>a = s + [t]</code> <code>b = a[1:]</code> a[1] = 9 b[1][1] = 0</td>
<td><code>s → [2, 3]</code> t → [5, 0] a → [2, 9, [5, 0]] b → [3, [5, 0]]</td>
</tr>
<tr>
<td>The list function also creates a new list containing existing elements</td>
<td><code>t = list(s)</code> s[1] = 0</td>
<td><code>s → [2, 0]</code> t → [2, 3]</td>
</tr>
<tr>
<td><strong>slice assignment</strong> replaces a slice with new values</td>
<td><code>s[0:0] = t</code> s[3:] = t t[1] = 0</td>
<td></td>
</tr>
</tbody>
</table>
### Lists in Environment Diagrams

Assume that before each example below we execute:

\[
\begin{align*}
  s &= [2, 3] \\
  t &= [5, 6]
\end{align*}
\]

<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 \rightarrow [2, 3, [5, 6]]) t \rightarrow 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 \rightarrow [2, 3, 5, 6]) t \rightarrow [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 \rightarrow [2, 3]) t \rightarrow [5, 0] a \rightarrow [2, 9, [5, 0]] b \rightarrow [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 \rightarrow [2, 0]) t \rightarrow [2, 3]</td>
</tr>
<tr>
<td><strong>slice assignment</strong> replaces a slice with new values</td>
<td>(s[0:0] = t) s[3:] = t t[1] = 0</td>
<td></td>
</tr>
</tbody>
</table>

---

[Diagram showing environment variables and operations]
## 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 | \(a = s + [t]\)
  \(b = a[1:]\)
  \(a[1] = 9\)
  \(b[1][1] = 0\) | \(s \rightarrow [2, 3]\)
  \(t \rightarrow [5, 0]\)
  \(a \rightarrow [2, 9, [5, 0]]\)
  \(b \rightarrow [3, [5, 0]]\) |
| **The list function also creates a new list containing existing elements** | \(t = list(s)\)
  \(s[1] = 0\) | \(s \rightarrow [2, 0]\)
  \(t \rightarrow [2, 3]\) |
| **slice assignment** replaces a slice with new values | \(s[0:0] = t\)
  \(s[3:] = t\)
  \(t[1] = 0\) | \(s \rightarrow [5, 6, 2, 5, 6]\)
  \(t \rightarrow [5, 0]\) |
Lists in Environment Diagrams

Assume that before each example below we execute:
s = [2, 3]
t = [5, 6]
**Lists in Environment Diagrams**

Assume that before each example below we execute:

\[
\begin{align*}
    \text{s} &= [2, 3] \\
    \text{t} &= [5, 6]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</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>removes &amp; returns the last element</td>
<td></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>pop removes &amp; returns the last element</td>
<td>t = s.pop()</td>
<td></td>
</tr>
</tbody>
</table>
**Lists in Environment Diagrams**

Assume that before each example below we execute:

\[ \text{s} = [2, 3] \]
\[ \text{t} = [5, 6] \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pop</strong> removes &amp; returns the last element</td>
<td>( t = \text{s}.pop() )</td>
<td>( s \rightarrow [2] ) ( t \rightarrow 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><code>pop</code> removes &amp; returns the last element</td>
<td>t = s.pop()</td>
<td>s → [2] t → 3</td>
</tr>
<tr>
<td><code>remove</code> removes the first element equal to the argument</td>
<td></td>
<td></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> removes &amp; returns the last element</td>
<td>t = s.pop()</td>
<td>s → [2]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>t → 3</td>
</tr>
<tr>
<td><strong>remove</strong> removes the first element equal to the argument</td>
<td>t.extend(t)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>t.remove(5)</td>
<td></td>
</tr>
</tbody>
</table>
Lists in Environment Diagrams

Assume that before each example below we execute:

\[
\begin{align*}
\text{s} &= [2, 3] \\
\text{t} &= [5, 6]
\end{align*}
\]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
</table>
| **pop** removes & returns the last element | \text{t} = \text{s}.pop() | \text{s} \rightarrow [2] \\
|                   |                          | \text{t} \rightarrow 3 |
| **remove** removes the first element equal to the argument | \text{t}.extend(t) \text{t}.remove(5) | \text{s} \rightarrow [2, 3] \\
|                   |                          | \text{t} \rightarrow [6, 5, 6] |
### 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> removes &amp; returns the last element</td>
<td>[ t = s.pop() ]</td>
<td>[ s \rightarrow [2] ] [ t \rightarrow 3 ]</td>
</tr>
<tr>
<td><strong>remove</strong> removes the first element equal to the argument</td>
<td>[ t.extend(t) ] [ t.remove(5) ]</td>
<td>[ s \rightarrow [2, 3] ] [ t \rightarrow [6, 5, 6] ]</td>
</tr>
</tbody>
</table>

**slice assignment** can remove elements from a list by assigning \([\] \) to a slice.
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> removes &amp; returns the last element</td>
<td>( t = s.pop() )</td>
<td>( s \to [2] ) ( t \to 3 )</td>
</tr>
<tr>
<td><strong>remove</strong> removes the first element equal to the argument</td>
<td>( t.extend(t) ) ( t.remove(5) )</td>
<td>( s \to [2, 3] ) ( t \to [6, 5, 6] )</td>
</tr>
<tr>
<td><strong>slice assignment</strong> can remove elements from a list by assigning ([,]) to a slice.</td>
<td>( s[:1] = [,] ) ( t[0:2] = [,] )</td>
<td></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> removes &amp; returns the last element</td>
<td>( t = s.pop() )</td>
<td>( s \rightarrow [2] ) ( t \rightarrow 3 )</td>
</tr>
<tr>
<td><strong>remove</strong> removes the first element equal to the argument</td>
<td>( t.extend(t) ) ( t.remove(5) )</td>
<td>( s \rightarrow [2, 3] ) ( t \rightarrow [6, 5, 6] )</td>
</tr>
<tr>
<td><strong>slice assignment</strong> can remove elements from a list by assigning ([]) to a slice.</td>
<td>( s[:1] = [] ) ( t[0:2] = [] )</td>
<td>( s \rightarrow [3] ) ( t \rightarrow [] )</td>
</tr>
</tbody>
</table>
Lists in Lists in Lists in Environment Diagrams

t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)

t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
Lists in Lists in Lists in Environment Diagrams

t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)

Global list

t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
Lists in Lists in Lists in Environment Diagrams

\[
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
\]

\[
t = \begin{bmatrix} 1, 2 \end{bmatrix}, \begin{bmatrix} 3, 4 \end{bmatrix}
t[0].append(t[1:2])
\]
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)

[t] evaluates to:

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

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

```
[[1, 2, 3, 4], [3, 4]]
```
Lists in Lists in Lists in Environment Diagrams

t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)

[t] evaluates to:


t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
Lists in Lists in Lists in Environment Diagrams

t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)

t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
Lists in Lists in Lists in Environment Diagrams

```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```

```
t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
```
Lists in Lists in Lists in Environment Diagrams

\[ t = [1, 2, 3] \]
\[ t[1:3] = [t] \]
\[ t.extend(t) \]

\[ t = \{1, 2\}, \{3, 4\} \]
\[ t[0].append(t[1:2]) \]
Lists in Lists in Lists in Environment Diagrams

\[
t = [1, 2, 3]
\]

\[
t[1:3] = [t]
\]

\[
t.extend(t)
\]

\[
t = [[1, 2], [3, 4]]
\]

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

\[
[1, [...], 1, [...]]
\]
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)

[1, [...], 1, [...]]

t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
Lists in Lists in Lists in Environment Diagrams

\[
t = [1, 2, 3] \\
t[1:3] = [t] \\
t.extend(t)
\]

\[
t = [[1, 2], [3, 4]] \\
t[0].append(t[1:2])
\]
Lists in Lists in Lists in Environment Diagrams

t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)

[1, [...], 1, [...]]

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

[[1, 2, [[3, 4]]], [3, 4]]
Examples: Objects
Land Owners

Instance attributes are found before class attributes; class attributes are inherited
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```python
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
```
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```python
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
```
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
### Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```python
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()

>>> jack

>>> jack.work()

>>> john.work()

>>> john.elf.work(john)
```
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work() <class Worker>
greeting: 'Sir'

>>> jack

>>> jack.work()

>>> john.work()

>>> john.elf.work(john)
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```python
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()  # <class Worker>  greeting: 'Sir'
>>> jack
# <class Bourgeoisie>  greeting: 'Peon'
>>> jack.work()
>>> john.work()
>>> john.elf.work(john)
```
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
<class Worker>
greeting: 'Sir'

>>> jack
<class Bourgeoisie>
greeting: 'Peon'

>>> jack.work()
jack <Worker>

>>> john.work()
john <Worker>

>>> john.elf.work(john)
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()  
<class Worker>
    greeting: 'Sir'

>>> jack
<class Bourgeoisie>
    greeting: 'Peon'

>>> jack.work()  
jack <Worker>
    elf:  
    greeting: 'Maam'

>>> john.work()  
>>> john.elf.work(john)  
john <Bourgeoisie>
    elf:  

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```python
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
<class Worker>
greeting: 'Sir'

>>> jack
<class Bourgeoisie>
greeting: 'Peon'

>>> jack.work()
jack <Worker>
ell: 
greeting: 'Maam'

>>> john.work()

>>> john.elf.work(john)
john <Bourgeoisie>
ell: 

>>> john.elf.work(john)
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```python
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
<class Worker>
greeting: 'Sir'

>>> jack
<class Bourgeoisie>
greeting: 'Peon'

>>> jack.work()
jack <Worker>
ell: 
greeting: 'Maam'

>>> john.work()

>>> john.elf.work(john)
john <Bourgeoisie>
ell: 
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```python
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
<class Worker>
greeting: 'Sir'

>>> jack
<class Bourgeoisie>
greeting: 'Peon'

>>> jack.work()
jack <Worker>
ell: 
greeting: 'Maam'

>>> john.work()

>>> john.elf.work(john)
john <Bourgeoisie>
ell: 
```

Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```python
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
<class Worker>
greeting: 'Sir'

>>> jack
<class Bourgeoisie>
greeting: 'Peon'

>>> jack.work()
jack <Worker>
ell: 
greeting: 'Maam'

>>> john.work()

>>> john.elf.work(john)
john <Bourgeoisie>
ell: 
```
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ' , I work'
    def __repr__(self):
        return 'Bourgeoisie' + greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
'Sir, I work'

>>> jack
<class Worker>
greeting: 'Sir'

>>> jack.work()
'class Bourgeoisie>
greeting: 'Peon'

>>> john.work()

>>> john.elf.work(john)

jack <Worker>
elf: 

greeting: 'Maam'

john <Bourgeoisie>
elf: 
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```python
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
'Sir, I work'

>>> jack
<class Worker>
greeting: 'Sir'

>>> jack.work()
'class Bourgeoisie'>
greeting: 'Peon'

>>> john.work()

>>> john.elf.work(john)
''
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
'Sir, I work'

jack
Peon

>>> jack.work()

>>> john.work()

>>> john.elf.work(john)
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return 'Bourgeoisie.greeting'

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
'Sir, I work'

>>> jack
Peon

>>> jack.work()

>>> john.work()

>>> john.elf.work(john)
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return 'Bourgeoisie.greeting'

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
'Sir, I work'

>>> jack
Peon

>>> jack.work()
'Maam, I work'

>>> john.work()

>>> john.elf.work(john)

jack <Worker>
    greeting: 'Sir'
    elf: Worker

<class Worker>
    greeting: 'Sir'

jack <Worker>
    greeting: 'Maam'

john <Bourgeoisie>
    elf: Bourgeoisie
    greeting: 'Peon'

<class Bourgeoisie>
    greeting: 'Peon'

john <Bourgeoisie>
    elf: Bourgeoisie
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

```python
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
'Sir, I work'

>>> jack
Peon

>>> jack.work()
'Maam, I work'

>>> john.work()

>>> john.elf.work(john)
```

---

```
<class Worker>
greeting: 'Sir'

<class Bourgeoisie>
greeting: 'Peon'

type(jack) <Worker>

type(john) <Bourgeoisie>
elf:

greeting: 'Maam'
```

---
**Land Owners**

Instance attributes are found before class attributes; class attributes are inherited

```python
class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return 'Bourgeoisie.greeting'

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
'Sir, I work'

>>> jack
Peon

>>> jack.work()
'Maam, I work'

>>> john.work()
'Peon, I work
'I gather wealth'

>>> john.elf.work(john)

<class Worker>
greeting: 'Sir'

<class Bourgeoisie>
greeting: 'Peon'

jack <Worker>
greeting: 'Peon'

jack <Worker>
elf: john <Bourgeoisie>
greeting: 'Maam'

john <Bourgeoisie>
elf: john <Bourgeoisie>
```
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
'Sir, I work'

>>> jack
Peon

>>> jack.work()
'Maam, I work'

>>> john.work()
'Peon, I work
'I gather wealth'

>>> john.elf.work(john)

"""
Land Owners

Instance attributes are found before class attributes; class attributes are inherited

class Worker:
    greeting = 'Sir'
    def __init__(self):
        self.elf = Worker
    def work(self):
        return self.greeting + ', I work'
    def __repr__(self):
        return Bourgeoisie.greeting

class Bourgeoisie(Worker):
    greeting = 'Peon'
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'

jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'

>>> Worker().work()
'Sir, I work'

>>> jack
Peon

>>> jack.work()
'Maam, I work'

>>> john.work()
'Peon, I work'

>>> john.elf.work(john)
'Peon, I work'
Examples: Iterables & Iterators
Using Built-In Functions & Comprehensions
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list $s$ that have the smallest absolute value?
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list $s$ that have the smallest absolute value?

$[-4, -3, -2, 3, 2, 4]$
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list $s$ that have the smallest absolute value?

$$s = [-4, -3, -2, 3, 2, 4]$$

0 1 2 3 4 5
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list \( s \) that have the smallest absolute value?

\[ [-4, -3, -2, 3, 2, 4] \]

\[ [0, 1, 2, 3, 4, 5] \] \[ \rightarrow \] \[ [2, 4] \]
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list \( s \) that have the smallest absolute value?

\([-4, -3, -2, 3, 2, 4]\)

\(\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
\end{array}\)  \(\begin{array}{cccc}
[2, 4] & [1, 2, 3, 4, 5] \\
\end{array}\)
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list $s$ that have the smallest absolute value?

$[-4, -3, -2, 3, 2, 4]$  

$0 1 2 3 4 5$  

$[2, 4] [1, 2, 3, 4, 5] [0]$
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list s that have the smallest absolute value?

\([-4, -3, -2, 3, 2, 4]\)

0 1 2 3 4 5

\([2, 4]\)  \([1, 2, 3, 4, 5]\)  \([0]\)

What's the largest sum of two adjacent elements in a list s? (Assume len(s) > 1)
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list \( s \) that have the smallest absolute value?

\[
[-4, -3, -2, 3, 2, 4] \\
0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5
\]

\[ [2, 4] \]

\[ [1, 2, 3, 4, 5] \]

\[ [0] \]

What's the largest sum of two adjacent elements in a list \( s \)? (Assume \( \text{len}(s) > 1 \))

\[
[-4, -3, -2, 3, 2, 4]
\]
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list \( s \) that have the smallest absolute value?

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
[-4, -3, -2, 3, 2, 4] & [2, 4] & [1, 2, 3, 4, 5] & 0
\end{array}
\]

What's the largest sum of two adjacent elements in a list \( s \)? (Assume \( \text{len}(s) > 1 \))

\[
\begin{array}{cccccc}
0 & 1 & 2 & 3 & 4 & 5 \\
[-4, -3, -2, 3, 2, 4] & 6
\end{array}
\]
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list \( s \) that have the smallest absolute value?

\[
\begin{align*}
\text{[-4, -3, -2, 3, 2, 4]} & \quad \text{[2, 4]} \\
0 & \quad 1 & \quad 2 & \quad 3 & \quad 4 & \quad 5
\end{align*}
\]

[-4, -3, -2, 3, 2, 4] \[2, 4\] [1, 2, 3, 4, 5] \[0\]

What's the largest sum of two adjacent elements in a list \( s \)? (Assume \( \text{len}(s) > 1 \))

\[
\begin{align*}
\text{[-4, -3, -2, 3, 2, 4]} & \quad 6 \\
\text{[-4, 3, -2, -3, 2, -4]} & \quad 6
\end{align*}
\]
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list $s$ that have the smallest absolute value?

$$[-4, -3, -2, 3, 2, 4] \quad [2, 4] \quad [1, 2, 3, 4, 5] \quad [0]$$

What's the largest sum of two adjacent elements in a list $s$? (Assume len(s) > 1)

$$[-4, -3, -2, 3, 2, 4] \quad 6 \quad [-4, 3, -2, -3, 2, -4] \quad 1$$
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?

```
[-4, -3, -2, 3, 2, 4] ▶ [2, 4]   [1, 2, 3, 4, 5] ▶ [0]
```

What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)

```
[-4, -3, -2, 3, 2, 4] ▶ 6         [-4, 3, -2, -3, 2, -4] ▶ 1
```

Create a dictionary mapping each digit `d` to the lists of elements in `s` that end with `d`. 
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list \( s \) that have the smallest absolute value?

\[
[-4, -3, -2, 3, 2, 4]
\]

0 1 2 3 4 5

\[ [2, 4] \]

\[ [1, 2, 3, 4, 5] \]

\[ [0] \]

What's the largest sum of two adjacent elements in a list \( s \)? (Assume \( \text{len}(s) > 1 \))

\[
[-4, -3, -2, 3, 2, 4]
\]

6

\[
[-4, 3, -2, -3, 2, -4]
\]

1

Create a dictionary mapping each digit \( d \) to the lists of elements in \( s \) that end with \( d \).

\[
[5, 8, 13, 21, 34, 55, 89]
\]
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list \( s \) that have the smallest absolute value?

\[
[-4, -3, -2, 3, 2, 4] \\
0 1 2 3 4 5 \\
\]

\[
[2, 4] \\
[1, 2, 3, 4, 5] \\
\]

\[
[0] \\
\]

What's the largest sum of two adjacent elements in a list \( s \)? (Assume \( \text{len}(s) > 1 \))

\[
[-4, -3, -2, 3, 2, 4] \\
6 \\
[-4, 3, -2, -3, 2, -4] \\
1 \\
\]

Create a dictionary mapping each digit \( d \) to the lists of elements in \( s \) that end with \( d \).

\[
[5, 8, 13, 21, 34, 55, 89] \\
\}

\[
1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89] \\
\]
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?

```
[-4, -3, -2, 3, 2, 4]  [2, 4]  [1, 2, 3, 4, 5]  [0]
```

What's the largest sum of two adjacent elements in a list `s`? (Assume len(s) > 1)

```
[-4, -3, -2, 3, 2, 4]  6  [-4, 3, -2, -3, 2, -4]  1
```

Create a dictionary mapping each digit `d` to the lists of elements in `s` that end with `d`.

```
[5, 8, 13, 21, 34, 55, 89]  {1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]}
```

Does every element equal some other element in `s`?
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list \( s \) that have the smallest absolute value?

\[
[-4, -3, -2, 3, 2, 4] \quad \Rightarrow \quad [2, 4] \quad [1, 2, 3, 4, 5] \quad \Rightarrow \quad [0]
\]

What's the largest sum of two adjacent elements in a list \( s \)? (Assume \( \text{len}(s) > 1 \))

\[
[-4, -3, -2, 3, 2, 4] \quad \Rightarrow \quad 6 \quad [-4, 3, -2, -3, 2, -4] \quad \Rightarrow \quad 1
\]

Create a dictionary mapping each digit \( d \) to the lists of elements in \( s \) that end with \( d \).

\[
[5, 8, 13, 21, 34, 55, 89] \quad \Rightarrow \quad \{1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]\}
\]

Does every element equal some other element in \( s \)?

\[
[-4, -3, -2, 3, 2, 4] \quad \Rightarrow \quad \text{False}
\]
Using Built-In Functions & Comprehensions

What are the indices of all elements in a list `s` that have the smallest absolute value?

```
[-4, -3, -2, 3, 2, 4] 0 1 2 3 4 5 ➔ [2, 4]  [1, 2, 3, 4, 5] ➔ [0]
```

What's the largest sum of two adjacent elements in a list `s`? (Assume `len(s) > 1`)

```
[-4, -3, -2, 3, 2, 4] ➔ 6  [-4, 3, -2, -3, 2, -4] ➔ 1
```

Create a dictionary mapping each digit `d` to the lists of elements in `s` that end with `d`.

```
[5, 8, 13, 21, 34, 55, 89] ➔ {1: [21], 3: [13], 4: [34], 5: [5, 55], 8: [8], 9: [89]}
```

Does every element equal some other element in `s`?

```
```
Examples: Linked Lists
Linked List Exercises
Linked List Exercises

Is a linked list s ordered from least to greatest?
Is a linked list $s$ ordered from least to greatest?
Linked List Exercises

Is a linked list s ordered from least to greatest?

![Linked List Diagrams]

1. 1 → 3 → 4
2. 1 → 4 → 3
Is a linked list $s$ ordered from least to greatest?

Is a linked list $s$ ordered from least to greatest by absolute value (or a key function)?
Linked List Exercises

Is a linked list $s$ ordered from least to greatest?

Is a linked list $s$ ordered from least to greatest by absolute value (or a key function)?
Linked List Exercises

Is a linked list $s$ ordered from least to greatest?

\[
1 \rightarrow 3 \rightarrow 4 \quad 1 \rightarrow 4 \rightarrow 3
\]

Is a linked list $s$ ordered from least to greatest by absolute value (or a key function)?

\[
1 \rightarrow -3 \rightarrow 4 \quad -4 \rightarrow -1 \rightarrow 3
\]
Linked List Exercises

Is a linked list $s$ ordered from least to greatest?

Is a linked list $s$ ordered from least to greatest by absolute value (or a key function)?

Create a sorted Link containing all the elements of both sorted Links $s$ & $t$. 
Linked List Exercises

Is a linked list s ordered from least to greatest?

```
1 -> 3 -> 4
1 -> 4 -> 3
```

Is a linked list s ordered from least to greatest by absolute value (or a key function)?

```
1 -> -3 -> 4
-4 -> -1 -> 3
```

Create a sorted Link containing all the elements of both sorted Links s & t.

```
1 -> 5
1 -> 4
```
**Linked List Exercises**

Is a linked list \( s \) ordered from least to greatest?

\[
\begin{align*}
1 & \rightarrow 3 & \rightarrow 4 \\
1 & \rightarrow 4 & \rightarrow 3
\end{align*}
\]

Is a linked list \( s \) ordered from least to greatest by absolute value (or a key function)?

\[
\begin{align*}
1 & \rightarrow -3 & \rightarrow 4 \\
-4 & \rightarrow -1 & \rightarrow 3
\end{align*}
\]

Create a sorted Link containing all the elements of both sorted Links \( s \) & \( t \).

\[
\begin{align*}
1 & \rightarrow 5 \\
1 & \rightarrow 4 \\
1 & \rightarrow 1 & \rightarrow 4 & \rightarrow 5
\end{align*}
\]
Linked List Exercises

Is a linked list $s$ ordered from least to greatest?

![Linked List Diagram 1](image1)

Is a linked list $s$ ordered from least to greatest by absolute value (or a key function)?

![Linked List Diagram 2](image2)

Create a sorted Link containing all the elements of both sorted Links $s$ & $t$.

![Linked List Diagram 3](image3)

Do the same thing, but never call Link.
Linked List Exercises

Is a linked list $s$ ordered from least to greatest?

```
1 -> 3 -> 4
1 -> 4 -> 3
```

Is a linked list $s$ ordered from least to greatest by absolute value (or a key function)?

```
1 -> -3 -> 4
-4 -> -1 -> 3
```

Create a sorted Link containing all the elements of both sorted Links $s$ & $t$.

```
1 -> 5
1 -> 4
1 -> 1 -> 4 -> 5
```

Do the same thing, but never call Link.

```
1 -> 5
1 -> 4
```
Is a linked list \( s \) ordered from least to greatest?

\[
\begin{array}{ccc}
1 & \rightarrow & 3 & \rightarrow & 4 \\
1 & \rightarrow & 4 & \rightarrow & 3
\end{array}
\]

Is a linked list \( s \) ordered from least to greatest by absolute value (or a key function)?

\[
\begin{array}{ccc}
1 & \rightarrow & -3 & \rightarrow & 4 \\
-4 & \rightarrow & -1 & \rightarrow & 3
\end{array}
\]

Create a sorted Link containing all the elements of both sorted Links \( s \) & \( t \).

\[
\begin{array}{ccc}
1 & \rightarrow & 5 \\
1 & \rightarrow & 4 \\
1 & \rightarrow & 1 & \rightarrow & 4 & \rightarrow & 5
\end{array}
\]

Do the same thing, but never call Link.

\[
\begin{array}{ccc}
1 & \rightarrow & 5 \\
1 & \rightarrow & 4
\end{array}
\]
Linked List Exercises

Is a linked list $s$ ordered from least to greatest?

$\begin{align*}
1 \rightarrow 3 \rightarrow 4 \\
1 \rightarrow 4 \rightarrow 3
\end{align*}$

Is a linked list $s$ ordered from least to greatest by absolute value (or a key function)?

$\begin{align*}
1 \rightarrow -3 \rightarrow 4 \\
-4 \rightarrow -1 \rightarrow 3
\end{align*}$

Create a sorted Link containing all the elements of both sorted Links $s$ & $t$.

$\begin{align*}
1 \rightarrow 5 \\
1 \rightarrow 4 \\
1 \rightarrow 1 \rightarrow 4 \rightarrow 5
\end{align*}$

Do the same thing, but never call Link.

$\begin{align*}
1 \rightarrow 5 \\
1 \rightarrow 4
\end{align*}$