



Examples: Lists

Assume that before each example below we execute:

Assume that before each example below we execute:

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

Operation

Assume that before each example below we execute:

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

Operation Example

Assume that before each example below we execute:

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

Operation Example Result

Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>		

Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	

Assume that before each example below we execute:

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

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	

Global

Assume that before each example below we execute:

Operation	Example	Result	Global		
<pre>append adds one element to a list</pre>	s.append(t) t = 0		s •	list 0 2	3
				list 0 5	16

Assume that before each example below we execute:

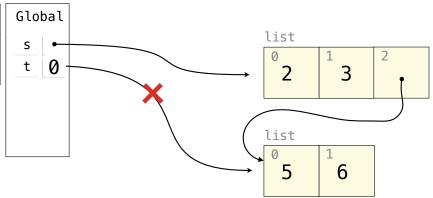
Operation	Example	Result	Global	
<pre>append adds one element to a list</pre>	<pre>s.append(t) t = 0</pre>		s t	list 0 2 1 3 2 list 0 5 1 6

Assume that before each example below we execute:

Operation	Example	Result	Global			
<pre>append adds one element to a list</pre>	<pre>s.append(t) t = 0</pre>		t 0	list 2 list 5	¹ 3	2

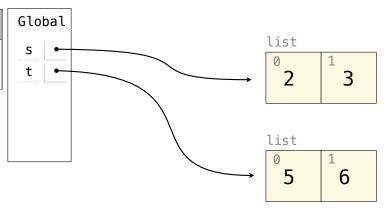
Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0



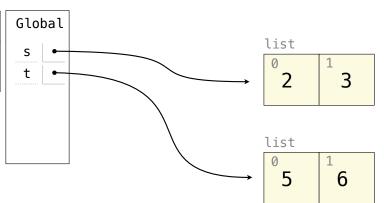
Assume that before each example below we execute:

Operation Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	$s \rightarrow [2, 3, [5, 6]]$ $t \rightarrow 0$



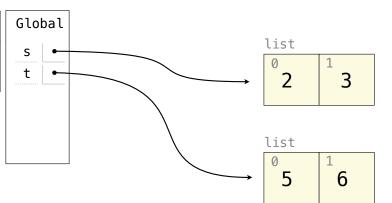
Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>		



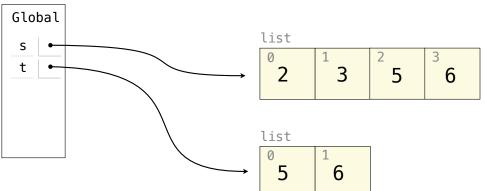
Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	$s \rightarrow [2, 3, [5, 6]]$ $t \rightarrow 0$
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	



Assume that before each example below we execute:

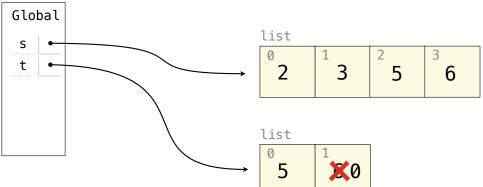
Operation Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	



Assume that before each example below we execute:

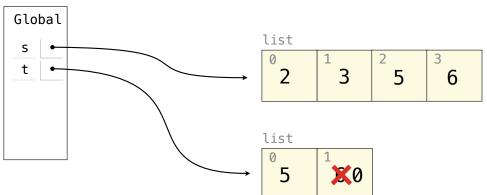
$$t = [5, 6]$$

Operation Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	



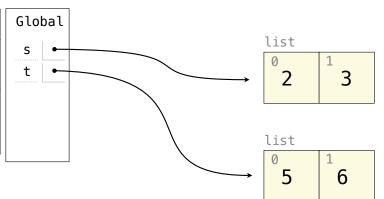
Assume that before each example below we execute:

Operation Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	$s \rightarrow [2, 3, 5, 6]$ $t \rightarrow [5, 0]$



Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	$s \rightarrow [2, 3, 5, 6]$ $t \rightarrow [5, 0]$



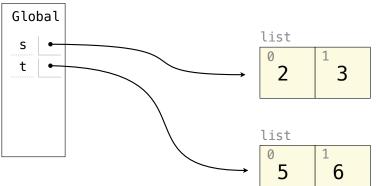
Assume that before each example below we execute:

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

create new lists containing existing

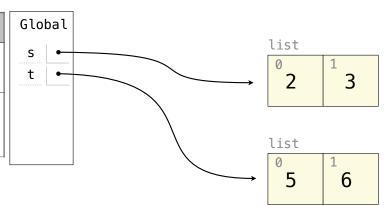
elements

Operation Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	$s \rightarrow [2, 3, 5, 6]$ t \rightarrow [5, 0]
addition & slicing		-



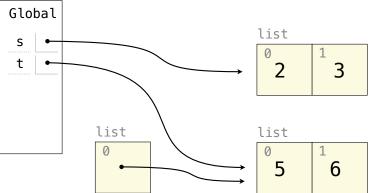
Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	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	



Assume that before each example below we execute:

Operation	Example	Result	Global
<pre>append adds one element to a list</pre>	<pre>s.append(t) t = 0</pre>	s → [2, 3, [5, 6]] t → 0	s • t
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	$s \rightarrow [2, 3, 5, 6]$ t \rightarrow [5, 0]	list
addition & slicing create new lists containing existing elements	a = s + [t] b = a[1:] a[1] = 9 b[1][1] = 0		0



Assume that before each example below we execute:

Operation	Example	Result	Global	
<pre>append adds one element to a list</pre>	<pre>s.append(t) t = 0</pre>	s → [2, 3, [5, 6]] t → 0	s •	$\begin{array}{c c} & \text{list} \\ \hline & 2 & 3 \end{array}$
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	$s \rightarrow [2, 3, 5, 6]$ t \rightarrow [5, 0]		list
addition & slicing create new lists containing existing	a = s + [t] b = a[1:] a[1] = 9			$ \begin{array}{c c} 0 & & \\ \hline & 5 & \\ \hline & 6 \end{array} $
elements	b[1][1] = 0			0 1 3 2 1 3 2 1 1 3 1 2 1 1 1 1 1 1 1 1 1

Assume that before each example below we execute:

Operation	Example	Result	Global	
<pre>append adds one element to a list</pre>	<pre>s.append(t) t = 0</pre>	s → [2, 3, [5, 6]] t → 0	s • • • • • • • • • • • • • • • • • • •	$ \begin{array}{c c} & \text{list} \\ \hline & 2 & 3 \end{array} $
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	$s \rightarrow [2, 3, 5, 6]$ t \rightarrow [5, 0]	a	list
addition & slicing create new lists containing existing elements	a = s + [t] b = a[1:] a[1] = 9 b[1][1] = 0			$\begin{array}{c c} 0 & & & \\ \hline 1 & & \\ 1 & & \\ \end{array}$
		-		2 3 2

Assume that before each example below we execute:

append adds one element to a list $t = 0$ $t \to 0$ $t \to 0$ extend adds all elements in one list to another list $t = 0$ $t \to 0$ addition & slicing create new lists containing existing elements $t = 0$ $t \to 0$ $t $	Operation	Example	Result	Global	
elements in one list to another list $ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	1	1			
create new lists containing existing elements $ \begin{vmatrix} a & b & b & b \\ b & a & b \\ a & b & b \\ b & b & b \\ a & b & b \\$	elements in one list		$s \rightarrow [2, 3, 5, 6]$ $t \rightarrow [5, 0]$	a	list
	create new lists containing existing	b = a[1:] a[1] = 9			5 6

Assume that before each example below we execute:

append adds one element to a list $t=0$	Operation	Example	Result	Global	
elements in one list to another list $ \begin{array}{c ccccccccccccccccccccccccccccccccccc$	1				0 1
create new lists containing existing elements $ \begin{vmatrix} a & s & t & t \\ b & s & t & t \\ b & s & t & t \\ b & s & s & t \\ b & s & t & t \\ b &$	elements in one list			·····	list
	create new lists containing existing	b = a[1:] a[1] = 9			5 6

Assume that before each example below we execute:

Operation	Example	Result	Global	
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0	s •	$\begin{array}{c c} & \text{list} \\ & 2 & 3 \end{array}$
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	$s \rightarrow [2, 3, 5, 6]$ t \rightarrow [5, 0]	a b	list
addition & slicing create new lists containing existing elements	a = s + [t] b = a[1:] a[1] = 9 b[1][1] = 0		lis	
		-	0	2 13 2 list
				3 1

Assume that before each example below we execute:

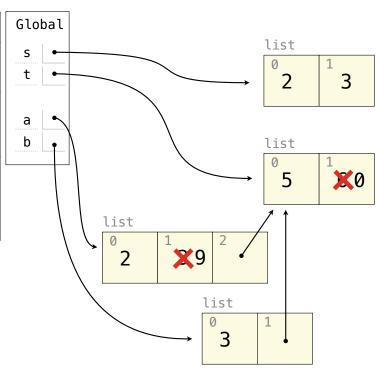
Operation	Example	Result	Global	
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0	s • t	$\begin{array}{c c} & \text{list} \\ \hline & 2 & 3 \end{array}$
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	$s \rightarrow [2, 3, 5, 6]$ t \rightarrow [5, 0]	a b	list
addition & slicing create new lists containing existing elements	a = s + [t] b = a[1:] a[1] = 9 b[1][1] = 0		list	$\begin{array}{c c} & 0 & 1 \\ \hline & 5 & 6 \end{array}$
			2	list 0 1
				3 1

Assume that before each example below we execute:

Operation	Example	Result	Global	
<pre>append adds one element to a list</pre>	<pre>s.append(t) t = 0</pre>	s → [2, 3, [5, 6]] t → 0	s •	$\begin{array}{c c} & \text{list} \\ \hline & 2 & 1 \\ \hline & 2 & 3 \end{array}$
<pre>extend adds all elements in one list to another list</pre>	s.extend(t) t[1] = 0	$s \rightarrow [2, 3, 5, 6]$ t \rightarrow [5, 0]	a b	list
addition & slicing create new lists containing existing elements	a = s + [t] b = a[1:] a[1] = 9 b[1][1] = 0			0 5 1 X0
				2 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
				3 1

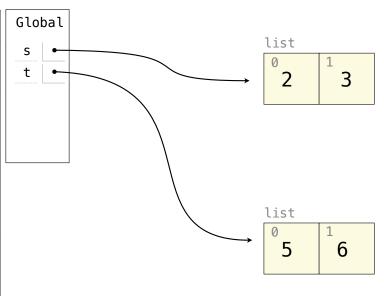
Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	$s \rightarrow [2, 3, [5, 6]]$ $t \rightarrow 0$
<pre>extend adds all elements in one list to another list</pre>	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	$s \rightarrow [2, 3]$ $t \rightarrow [5, 0]$ $a \rightarrow [2, 9, [5, 0]]$ $b \rightarrow [3, [5, 0]]$



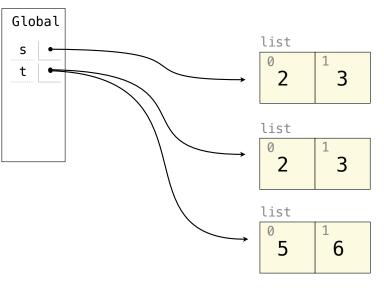
Assume that before each example below we execute:

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



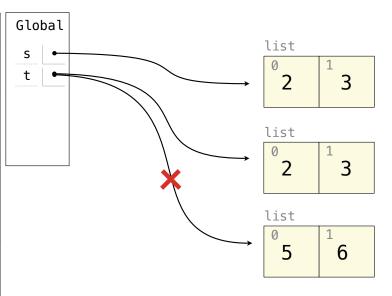
Assume that before each example below we execute:

Operation Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	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	



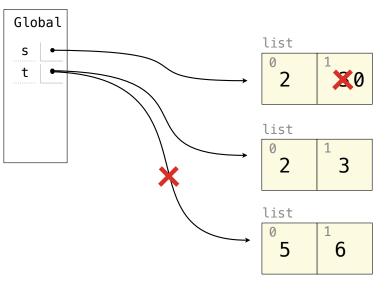
Assume that before each example below we execute:

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



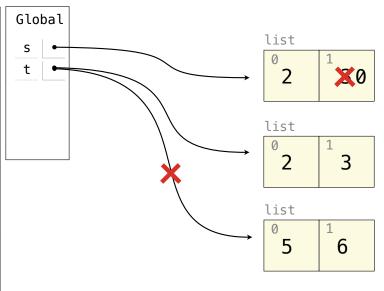
Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	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	



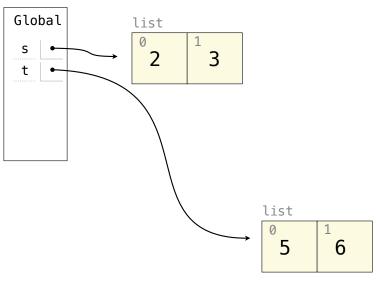
Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	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 → [2, 0] t → [2, 3]



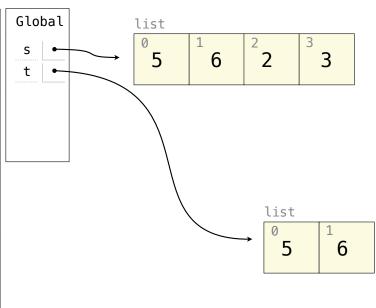
Assume that before each example below we execute:

Operation Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	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 → [2, 0] t → [2, 3]
<pre>slice assignment replaces a slice with new values</pre>	s[0:0] = t s[3:] = t t[1] = 0	



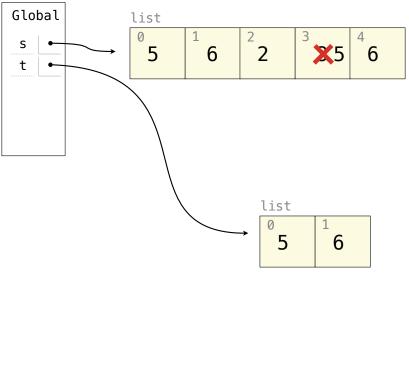
Assume that before each example below we execute:

Operation Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	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 → [2, 0] t → [2, 3]
<pre>slice assignment replaces a slice with new values</pre>	s[0:0] = t s[3:] = t t[1] = 0	



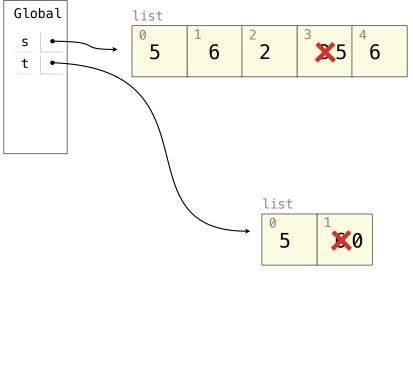
Assume that before each example below we execute:

Operation Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	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 → [2, 0] t → [2, 3]
<pre>slice assignment replaces a slice with new values</pre>	s[0:0] = t s[3:] = t t[1] = 0	



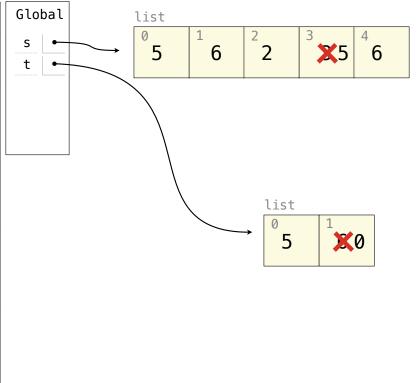
Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	$s \rightarrow [2, 3, [5, 6]]$ $t \rightarrow 0$
<pre>extend adds all elements in one list to another list</pre>	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 → [2, 0] t → [2, 3]
<pre>slice assignment replaces a slice with new values</pre>	s[0:0] = t s[3:] = t t[1] = 0	



Assume that before each example below we execute:

Operation	Example	Result
<pre>append adds one element to a list</pre>	s.append(t) t = 0	s → [2, 3, [5, 6]] t → 0
<pre>extend adds all elements in one list to another list</pre>	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 → [2, 0] t → [2, 3]
<pre>slice assignment replaces a slice with new values</pre>	s[0:0] = t s[3:] = t t[1] = 0	$s \rightarrow [5, 6, 2, 5, 6]$ $t \rightarrow [5, 0]$



Assume that before each example below we execute:

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

Assume that before each example below we execute:

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

C)peration	Example	Result	

Assume that before each example below we execute:

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

Operation	Example	Result
<pre>pop removes & returns the last element</pre>		

Assume that before each example below we execute:

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

Operation	Example	Result
<pre>pop removes & returns the last element</pre>	t = s.pop()	

Assume that before each example below we execute:

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

Assume that before each example below we execute:

Operation	Example	Result
<pre>pop removes & returns the last element</pre>	t = s.pop()	s → [2] t → 3
<pre>remove removes the first element equal to the argument</pre>		

Assume that before each example below we execute:

Operation	Example	Result
<pre>pop removes & returns the last element</pre>	t = s.pop()	s → [2] t → 3
remove removes the first element equal to the argument	t.extend(t) t.remove(5)	

Assume that before each example below we execute:

Operation	Example	Result
<pre>pop removes & returns the last element</pre>	t = s.pop()	s → [2] t → 3
remove removes the first element equal to the argument	t.extend(t) t.remove(5)	$s \rightarrow [2, 3]$ $t \rightarrow [6, 5, 6]$

Assume that before each example below we execute:

Operation Operation	Example	Result
<pre>pop removes & returns the last element</pre>	t = s.pop()	s → [2] t → 3
<pre>remove removes the first element equal to the argument</pre>	t.extend(t) t.remove(5)	$s \rightarrow [2, 3]$ $t \rightarrow [6, 5, 6]$
<pre>slice assignment can remove elements from a list by assigning [] to a slice.</pre>		

Assume that before each example below we execute:

Operation	Example	Result
<pre>pop removes & returns the last element</pre>	t = s.pop()	s → [2] t → 3
remove removes the first element equal to the argument	t.extend(t) t.remove(5)	$s \rightarrow [2, 3]$ $t \rightarrow [6, 5, 6]$
<pre>slice assignment can remove elements from a list by assigning [] to a slice.</pre>	s[:1] = [] t[0:2] = []	

Assume that before each example below we execute:

Operation	Example	Result
<pre>pop removes & returns the last element</pre>	t = s.pop()	s → [2] t → 3
remove removes the first element equal to the argument	t.extend(t) t.remove(5)	$s \rightarrow [2, 3]$ $t \rightarrow [6, 5, 6]$
<pre>slice assignment can remove elements from a list by assigning [] to a slice.</pre>	s[:1] = [] t[0:2] = []	s → [3] t → []

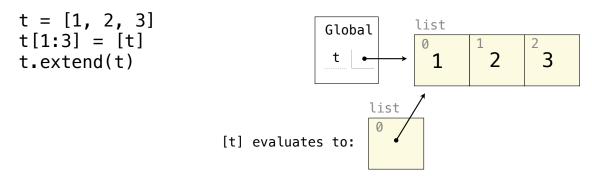
```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```

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

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



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



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

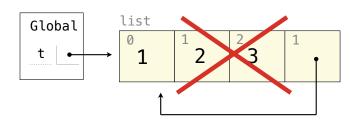
```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
[t] evaluates to:
[t] evaluates to:
[t] formula is the content of the conte
```

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

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

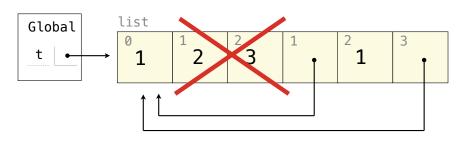
```
t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
```

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



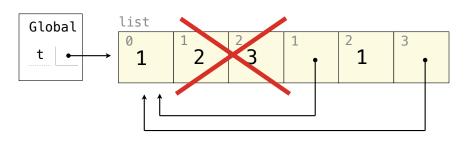
```
t = [[1, 2], [3, 4]]
t[0].append(t[1:2])
```

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



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

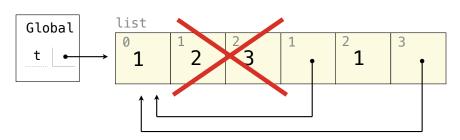
```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```



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

,

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



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

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

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

$$0$$

$$1$$

$$1$$

$$0$$

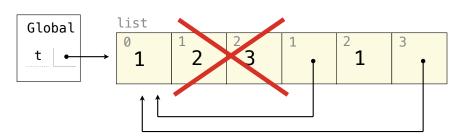
$$1$$

$$1$$

$$1$$

$$2$$

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



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

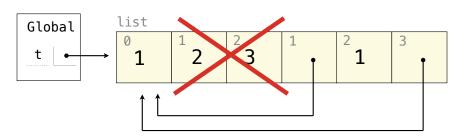
$$t = [[1, 2], [3, 4]]$$

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

$$0 \qquad 1$$

$$1 \qquad 2$$

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



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

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

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

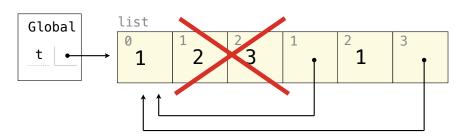
$$0 \qquad 1 \qquad 0 \qquad 1$$

$$0 \qquad 1 \qquad 0 \qquad 1$$

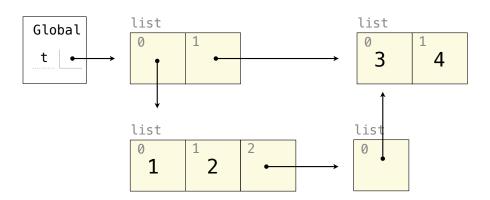
$$0 \qquad 1 \qquad 2 \qquad 0$$

$$1 \qquad 2 \qquad 0$$

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



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



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

Examples: Objects

Lar	A	Ou	V/D		-
Lai	IU	\cup	VII	U	2

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

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

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
```

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'
```

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
```

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):
```

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'
```

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'
```

```
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'
iack = Worker()
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                             >>> Worker() work()
   greeting = 'Sir'
    def init (self):
        self_elf = Worker
                                             >>> jack
   def work(self):
        return self greeting + ', I work'
   def repr (self):
        return Bourgeoisie greeting
                                             >>> jack work()
class Bourgeoisie(Worker):
   greeting = 'Peon'
                                             >>> john_work()
   def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
                                             >>> john_elf_work(john)
jack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

```
class Worker:
                                                                         <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
    def init (self):
                                                                          greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
   def work(self):
        return self greeting + ', I work'
   def repr (self):
        return Bourgeoisie greeting
                                             >>> jack work()
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john_work()
   def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
                                             >>> john.elf.work(john)
iack = Worker()
john = Bourgeoisie()
jack.greeting = 'Maam'
```

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

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
    def init (self):
                                                                          greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self greeting + ', I work'
   def repr (self):
                                                                          greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john_work()
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
                                             >>> john.elf.work(john)
iack = Worker()
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
                                             >>> john.elf.work(john)
iack = Worker()
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
    greeting = 'Sir'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self greeting + ', I work'
    def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self greeting + ', I work'
   def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self greeting + ', I work'
   def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self greeting + ', I work'
   def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
        return self greeting + ', I work'
   def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self greeting + ', I work'
   def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self greeting + ', I work'
   def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack.work()
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self greeting + ', I work'
   def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack.work()
                                             'Maam, I work'
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self greeting + ', I work'
   def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack.work()
                                             'Maam, I work'
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john work()
                                                                           elf: -
    def work(self):
                                                                           greeting: 'Maam'
        print(Worker.work(self))
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self greeting + ', I work'
   def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                             'Maam, I work'
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john_work()
                                                                           elf: -
    def work(self):
                                             Peon, I work
                                                                           greeting: 'Maam'
        print(Worker.work(self))
                                             'I gather wealth'
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self_elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self greeting + ', I work'
   def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack work()
                                             'Maam, I work'
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john_work()
                                                                           elf: -
    def work(self):
                                             Peon, I work
                                                                           greeting: 'Maam'
        print(Worker.work(self))
                                             'I gather wealth'
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john_elf_work(john)
iack = Worker()
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

```
class Worker:
                                                                          <class Worker>
                                             >>> Worker() work()
   greeting = 'Sir'
                                             'Sir, I work'
    def init (self):
                                                                           greeting: 'Sir'
        self.elf = Worker
                                             >>> jack
    def work(self):
                                                                          <class Bourgeoisie>
                                             Peon
        return self greeting + ', I work'
   def repr (self):
                                                                           greeting: 'Peon'
        return Bourgeoisie greeting
                                             >>> jack_work()
                                             'Maam, I work'
                                                                          jack <Worker>
class Bourgeoisie(Worker):
    greeting = 'Peon'
                                             >>> john_work()
                                                                           elf: -
    def work(self):
                                             Peon, I work
                                                                           greeting: 'Maam'
        print(Worker.work(self))
                                              'I gather wealth'
        return 'I gather wealth'
                                                                          john <Bourgeoisie>
                                             >>> john.elf.work(john)
iack = Worker()
                                              'Peon, I work'
                                                                           elf: -
john = Bourgeoisie()
jack greeting = 'Maam'
```

Examples: Iterables & Iterators

1.1	D 114 1	- C			The second second	
Lleina	Ruult In	Lunctions	X .	Compl	rahanciai	ne
USILIU	Dunt-III	Functions	CX	COLLID	CHCHOIOI	10
99			•			

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

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5]$$

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

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

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

$$[-4, -3, -2, 3, 2, 4]$$
 6

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

$$[-4, -3, -2, 3, 2, 4]$$
 6 $[-4, 3, -2, -3, 2, -4]$

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

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

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

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

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

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.

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

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.

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

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.

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

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.

Does every element equal some other element in s?

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

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.

Does every element equal some other element in s?

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

$$\begin{bmatrix}
-4, & -3, & -2, & 3, & 2, & 4 \\
0 & 1 & 2 & 3 & 4 & 5
\end{bmatrix} \qquad [2, 4] \qquad [1, 2, 3, 4, 5] \qquad [0]$$

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.

Does every element equal some other element in s?

Examples: Linked Lists

Is a linked list s ordered from least to greatest?

Is a linked list s ordered from least to greatest?



Is a linked list s ordered from least to greatest?





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)?

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)?

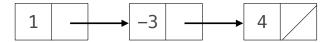


Is a linked list s ordered from least to greatest?





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





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)?





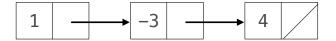
Create a sorted Link containing all the elements of both sorted Links s & t.

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.



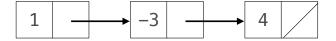


Is a linked list s ordered from least to greatest?



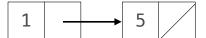


Is a linked list s ordered from least to greatest by absolute value (or a key function)?

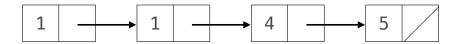




Create a sorted Link containing all the elements of both sorted Links s & t.







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.



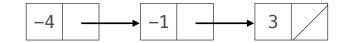
Is a linked list s ordered from least to greatest?





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

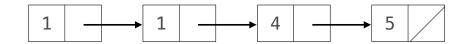


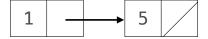


Create a sorted Link containing all the elements of both sorted Links s & t.









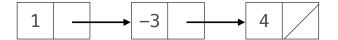


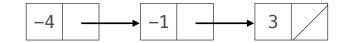
Is a linked list s ordered from least to greatest?



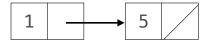


Is a linked list s ordered from least to greatest by absolute value (or a key function)?

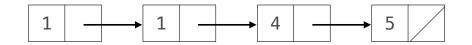


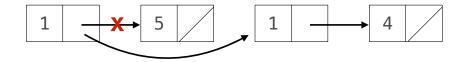


Create a sorted Link containing all the elements of both sorted Links s & t.







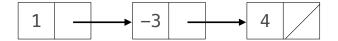


Is a linked list s ordered from least to greatest?





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.





