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:

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

Operation

Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example

Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result

Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result
append adds one element to a list		

Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`	

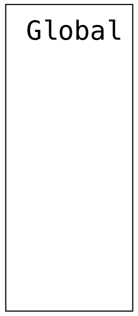
Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`	



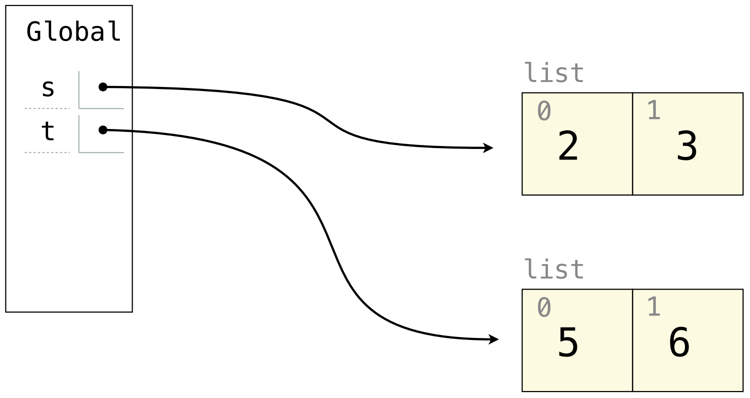
Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0	



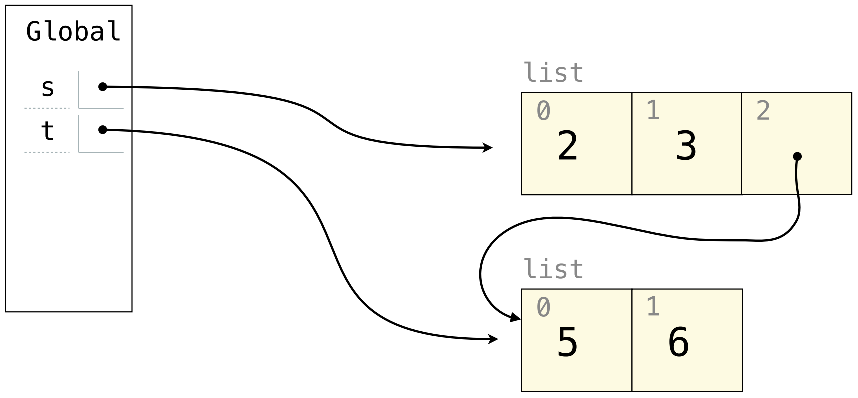
Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0	



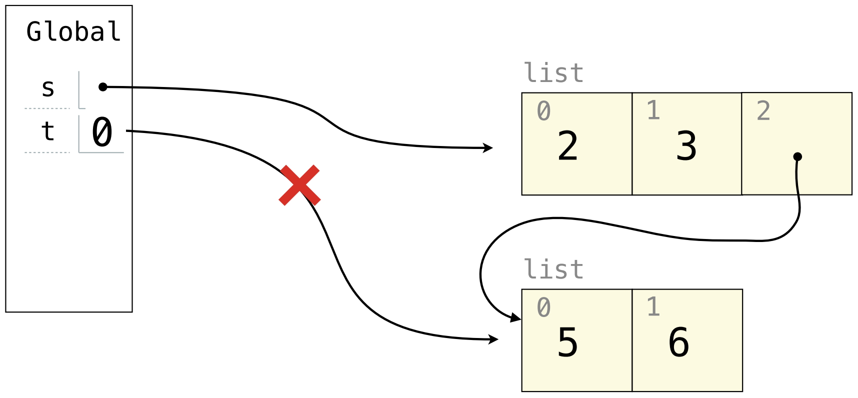
Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0	



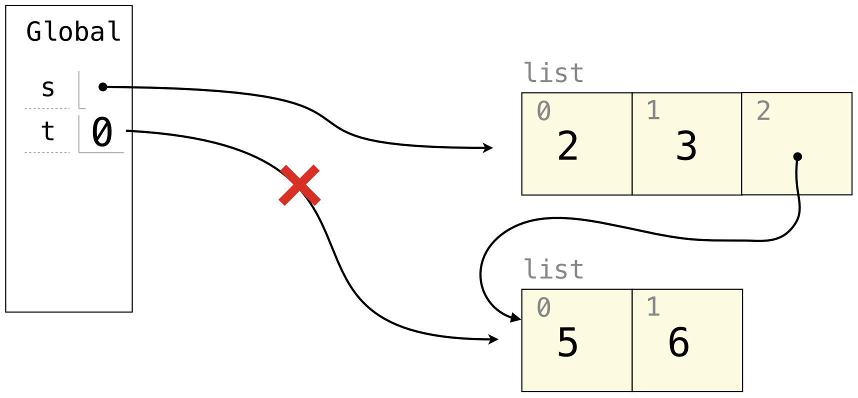
Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0



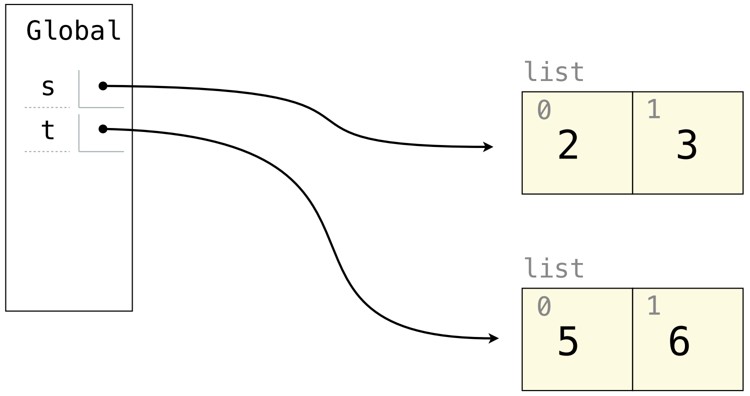
Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0



Lists in Environment Diagrams

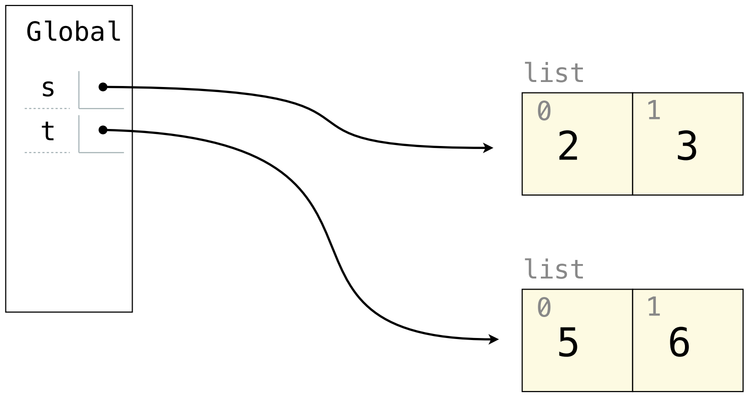
Assume that before each example below we execute:

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

Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list		

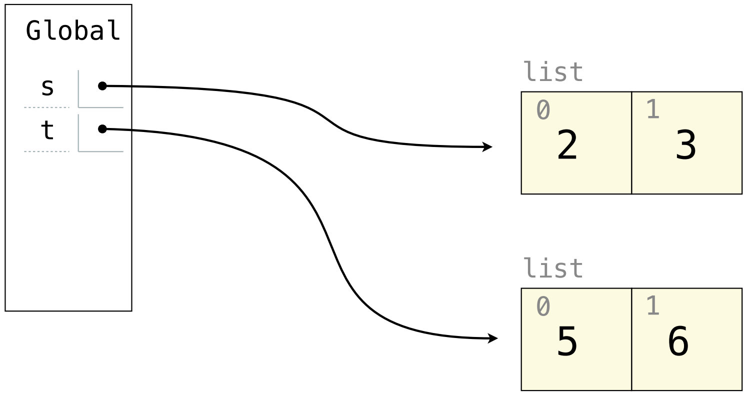


Lists in Environment Diagrams

Assume that before each example below we execute:

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

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



Lists in Environment Diagrams

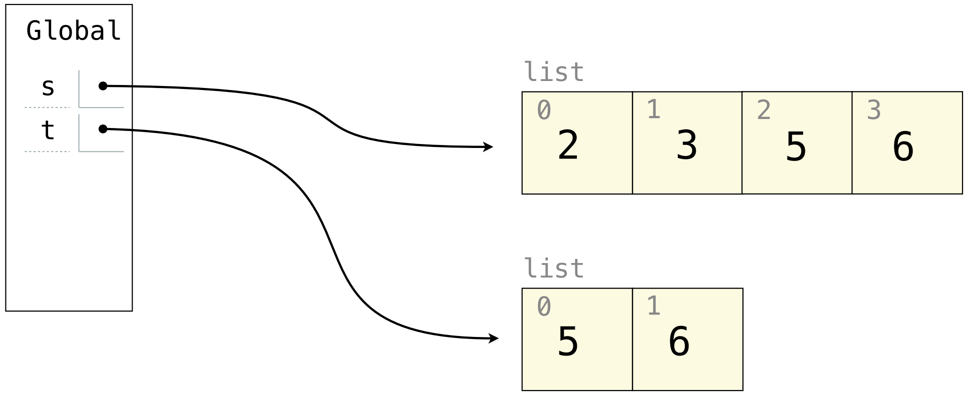
Assume that before each example below we execute:

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

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`	

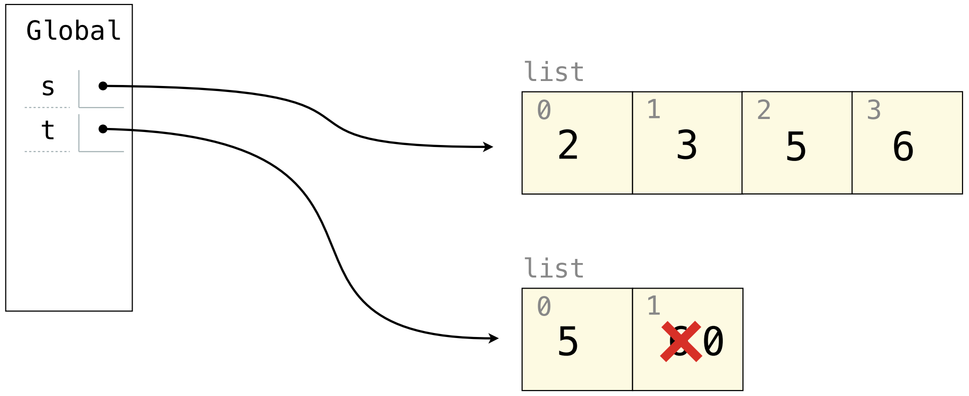


Lists in Environment Diagrams

Assume that before each example below we execute:

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

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



Lists in Environment Diagrams

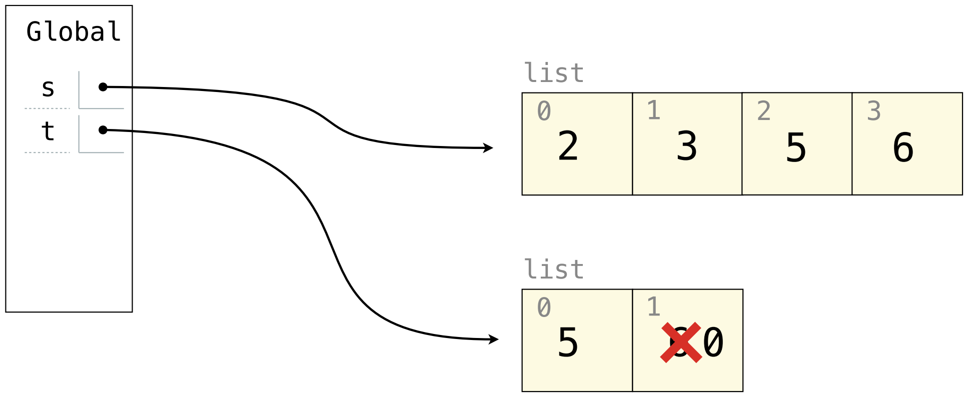
Assume that before each example below we execute:

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

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`



Lists in Environment Diagrams

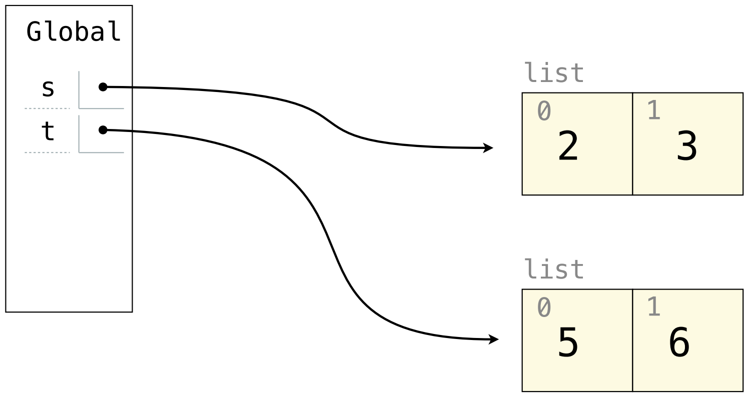
Assume that before each example below we execute:

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

Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	s.extend(t)
	t[1] = 0
		s → [2, 3, 5, 6]
		t → [5, 0]



Lists in Environment Diagrams

Assume that before each example below we execute:

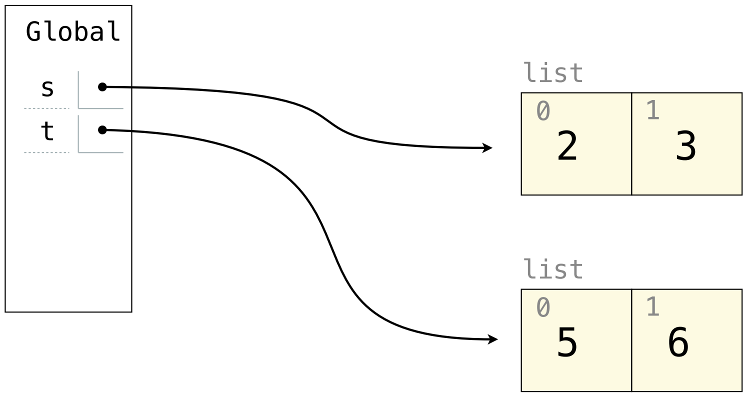
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements		



Lists in Environment Diagrams

Assume that before each example below we execute:

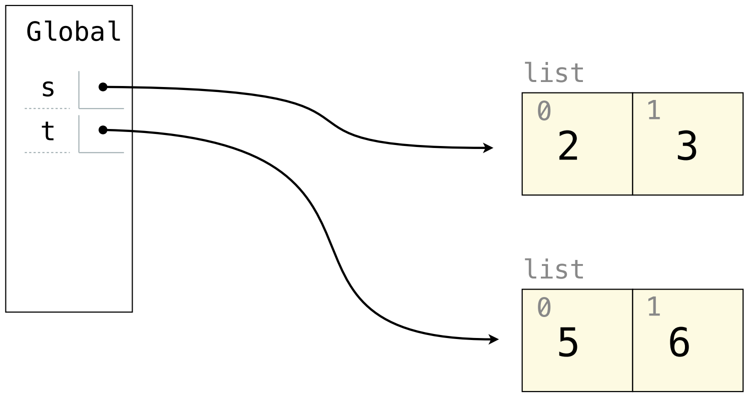
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`	



Lists in Environment Diagrams

Assume that before each example below we execute:

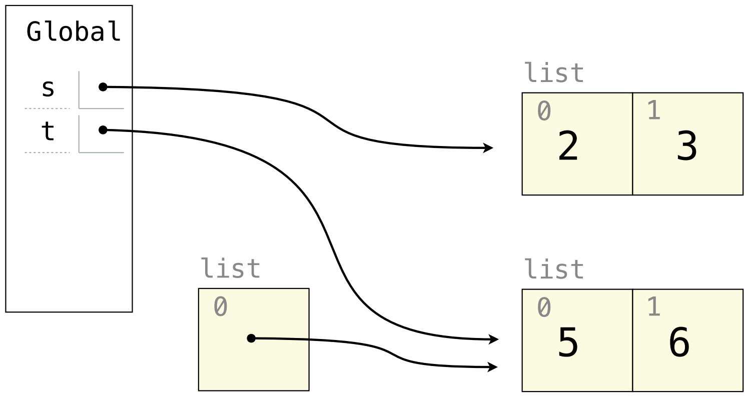
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`	



Lists in Environment Diagrams

Assume that before each example below we execute:

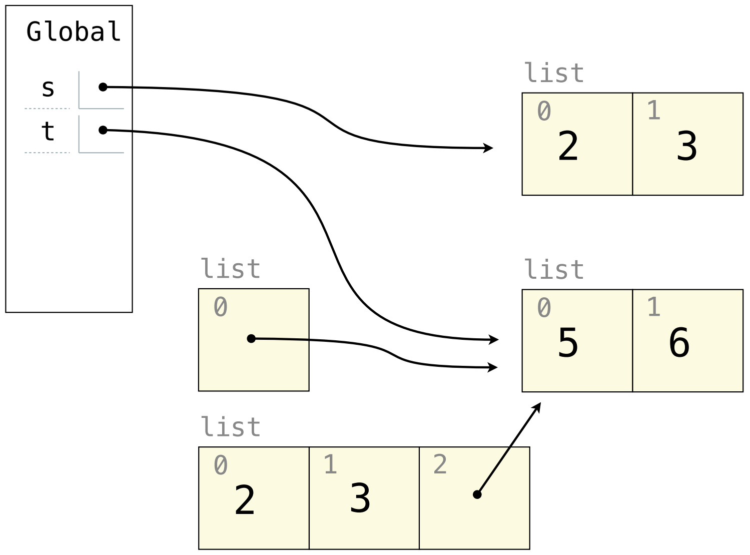
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	s.extend(t)
	t[1] = 0
		s → [2, 3, 5, 6]
		t → [5, 0]

addition & **slicing** create new lists containing existing elements	a = s + [t]
	b = a[1:]
	a[1] = 9
	b[1][1] = 0	



Lists in Environment Diagrams

Assume that before each example below we execute:

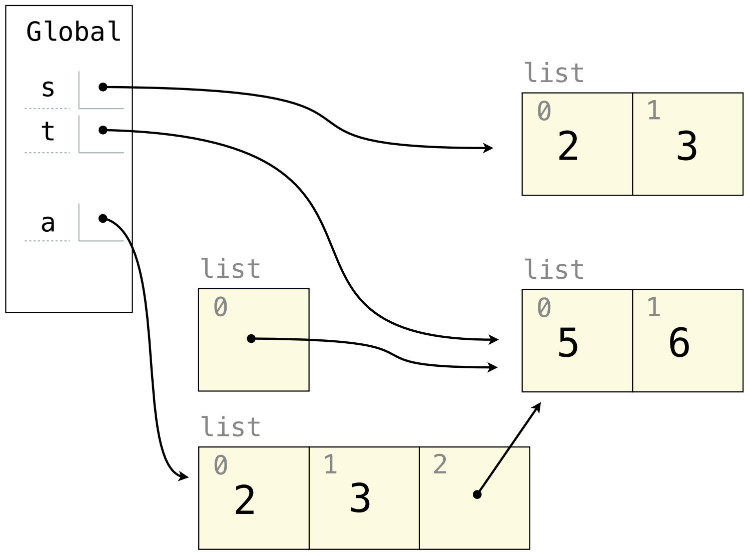
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`	



Lists in Environment Diagrams

Assume that before each example below we execute:

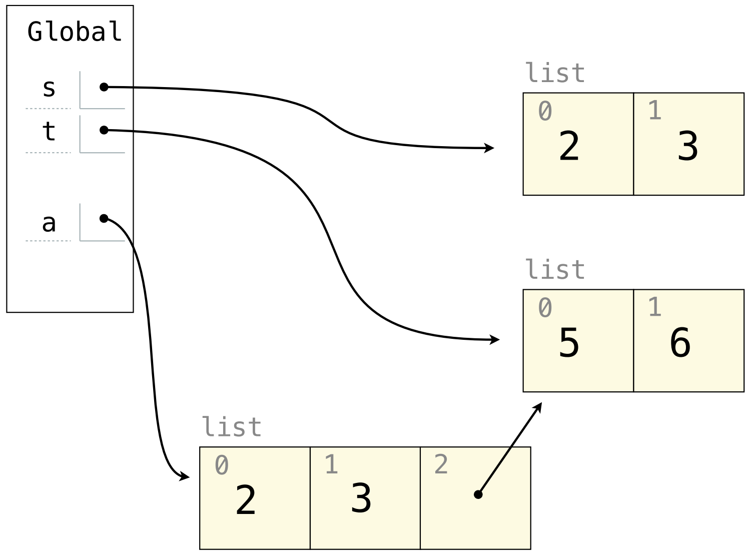
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`	



Lists in Environment Diagrams

Assume that before each example below we execute:

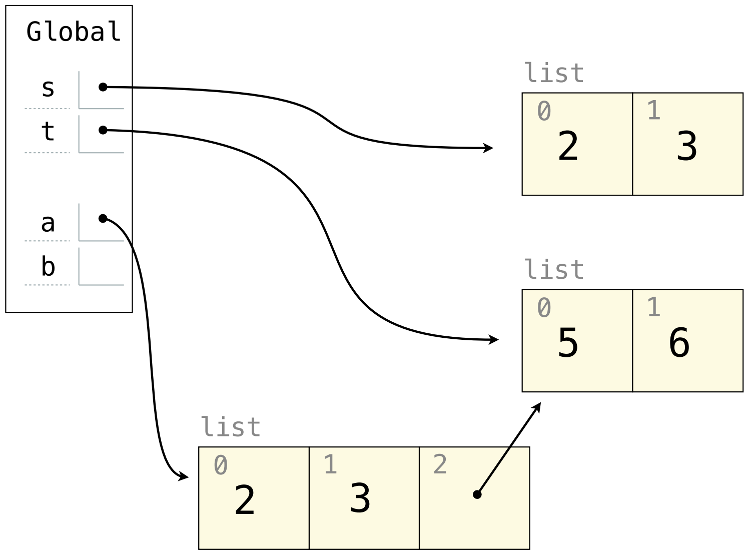
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`	



Lists in Environment Diagrams

Assume that before each example below we execute:

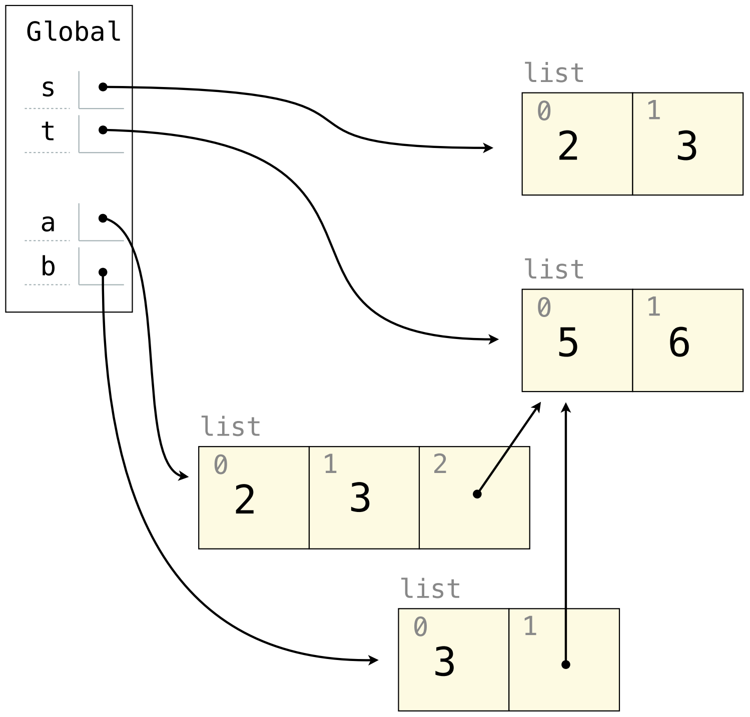
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`	



Lists in Environment Diagrams

Assume that before each example below we execute:

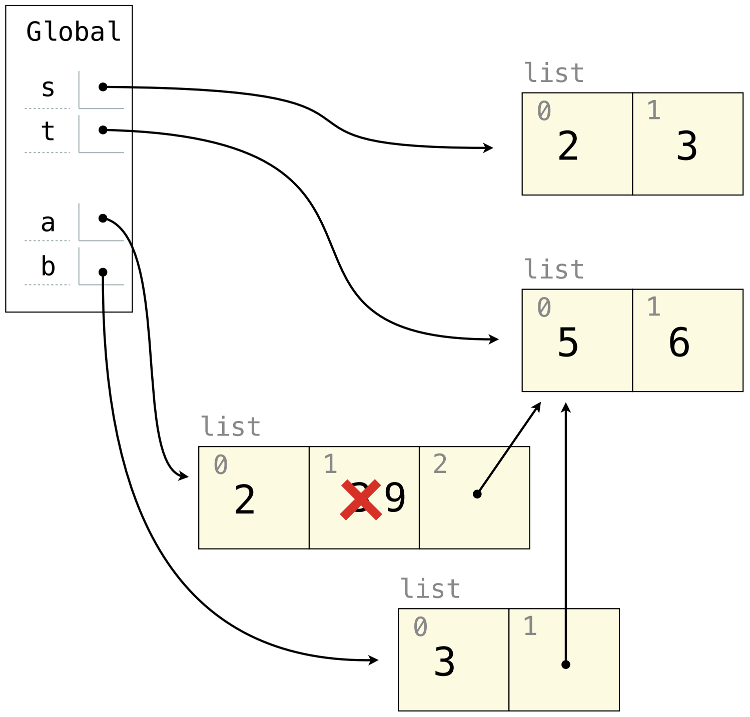
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	s.extend(t)
	t[1] = 0
		s → [2, 3, 5, 6]
		t → [5, 0]

addition & **slicing** create new lists containing existing elements	a = s + [t]
	b = a[1:]
	a[1] = 9
	b[1][1] = 0	



Lists in Environment Diagrams

Assume that before each example below we execute:

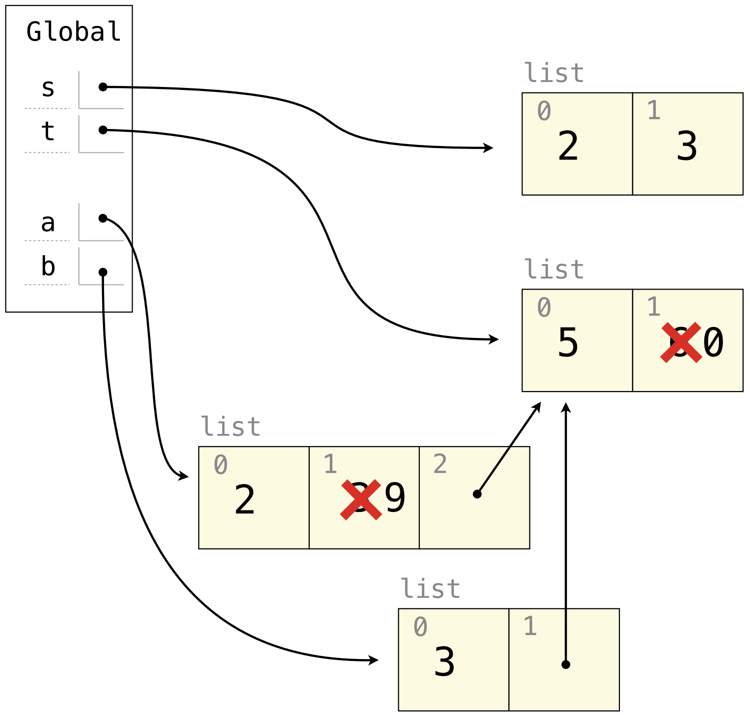
```
s = [2, 3]
t = [5, 6]
```

Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`	

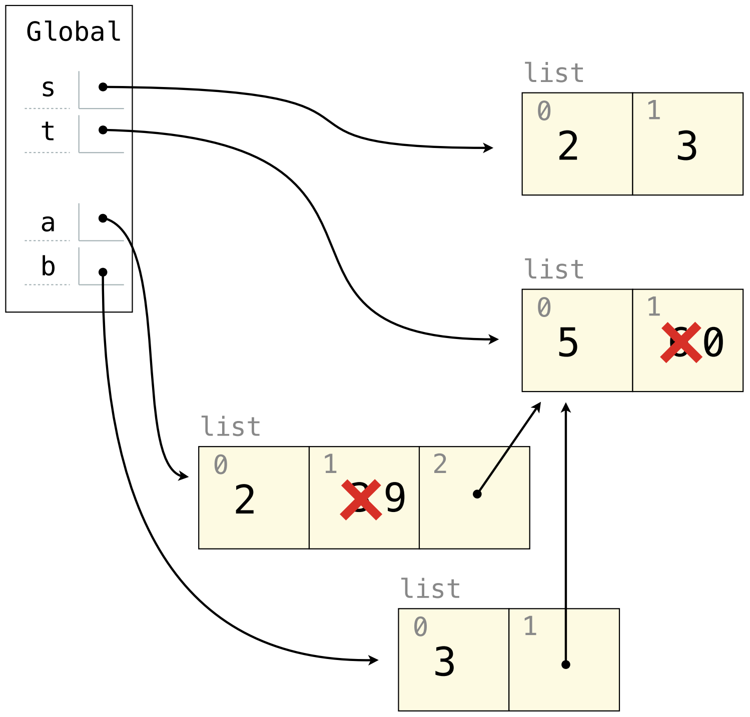


Lists in Environment Diagrams

Assume that before each example below we execute:

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

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



Lists in Environment Diagrams

Assume that before each example below we execute:

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

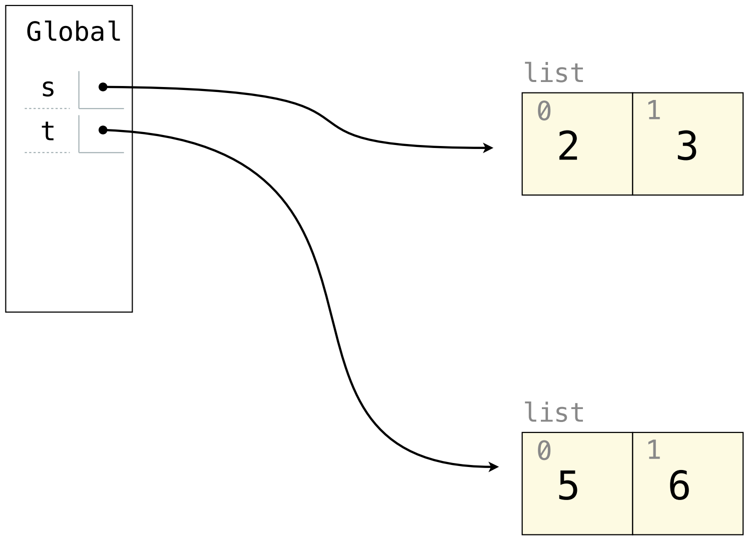
Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`
		`s → [2, 3]`
		`t → [5, 0]`
		`a → [2, 9, [5, 0]]`
		`b → [3, [5, 0]]`

The **list** function also creates a new list containing existing elements	`t = list(s)`
	`s[1] = 0`	



Lists in Environment Diagrams

Assume that before each example below we execute:

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

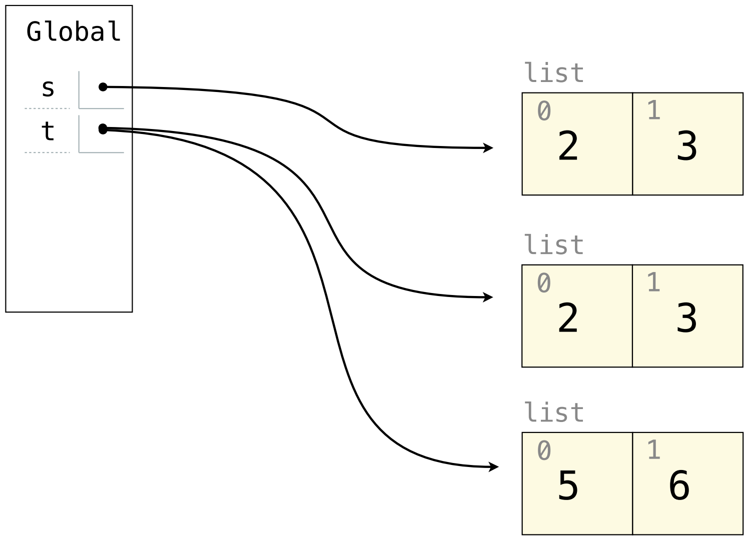
Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	s.extend(t)
	t[1] = 0
		s → [2, 3, 5, 6]
		t → [5, 0]

addition & **slicing** create new lists containing existing elements	a = s + [t]
	b = a[1:]
	a[1] = 9
	b[1][1] = 0
		s → [2, 3]
		t → [5, 0]
		a → [2, 9, [5, 0]]
		b → [3, [5, 0]]

The **list** function also creates a new list containing existing elements	t = list(s)
	s[1] = 0	



Lists in Environment Diagrams

Assume that before each example below we execute:

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

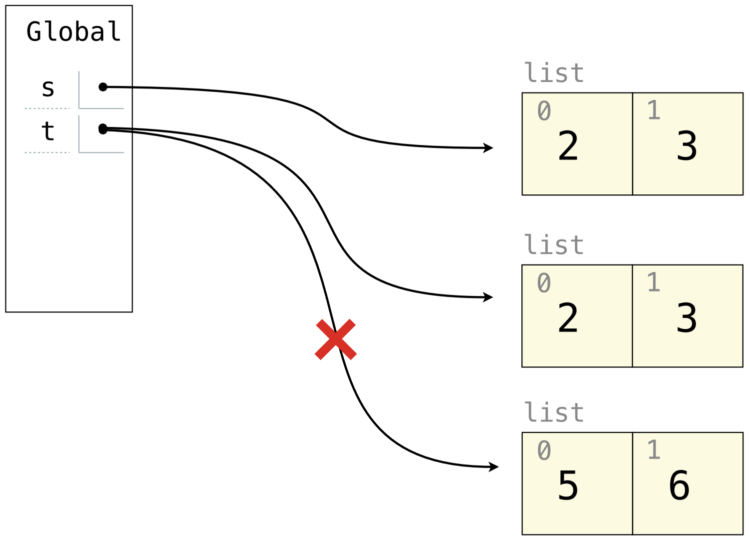
Operation	Example	Result

append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`
		`s → [2, 3]`
		`t → [5, 0]`
		`a → [2, 9, [5, 0]]`
		`b → [3, [5, 0]]`

The **list** function also creates a new list containing existing elements	`t = list(s)`
	`s[1] = 0`	



Lists in Environment Diagrams

Assume that before each example below we execute:

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

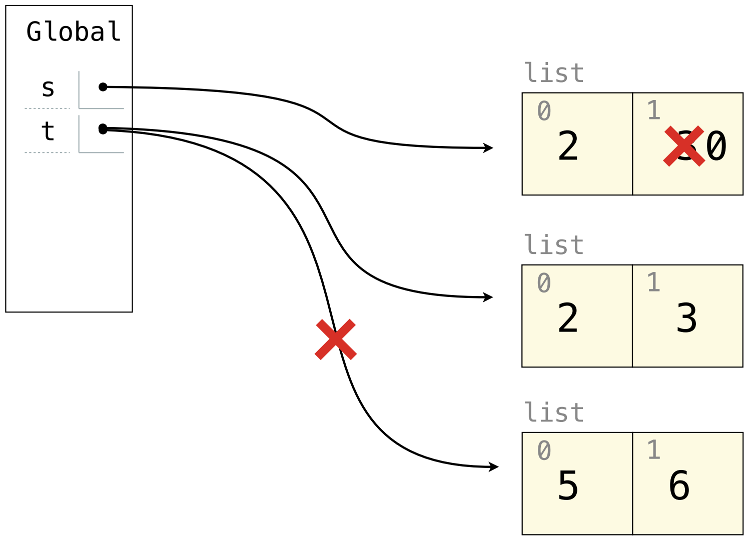
Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	s.extend(t)
	t[1] = 0
		s → [2, 3, 5, 6]
		t → [5, 0]

addition & **slicing** create new lists containing existing elements	a = s + [t]
	b = a[1:]
	a[1] = 9
	b[1][1] = 0
		s → [2, 3]
		t → [5, 0]
		a → [2, 9, [5, 0]]
		b → [3, [5, 0]]

The **list** function also creates a new list containing existing elements	t = list(s)
	s[1] = 0	



Lists in Environment Diagrams

Assume that before each example below we execute:

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

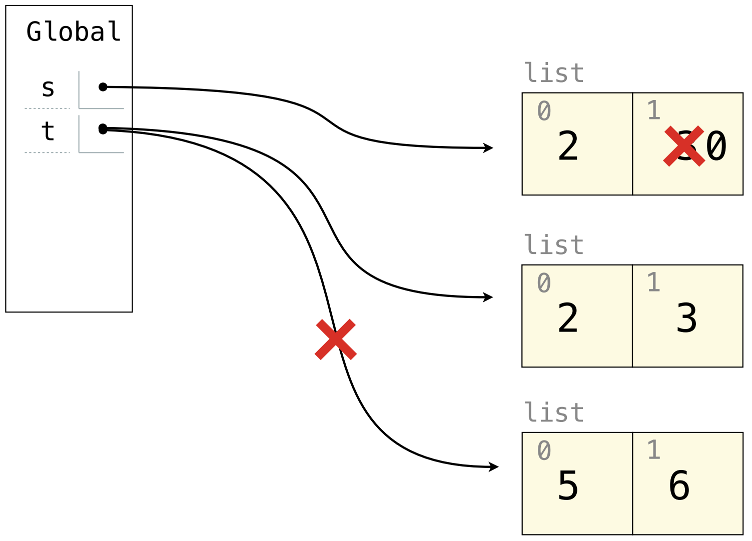
Operation	Example	Result

append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	s.extend(t)
	t[1] = 0
		s → [2, 3, 5, 6]
		t → [5, 0]

addition & **slicing** create new lists containing existing elements	a = s + [t]
	b = a[1:]
	a[1] = 9
	b[1][1] = 0
		s → [2, 3]
		t → [5, 0]
		a → [2, 9, [5, 0]]
		b → [3, [5, 0]]

The **list** function also creates a new list containing existing elements	t = list(s)
	s[1] = 0
		s → [2, 0]
		t → [2, 3]



Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result

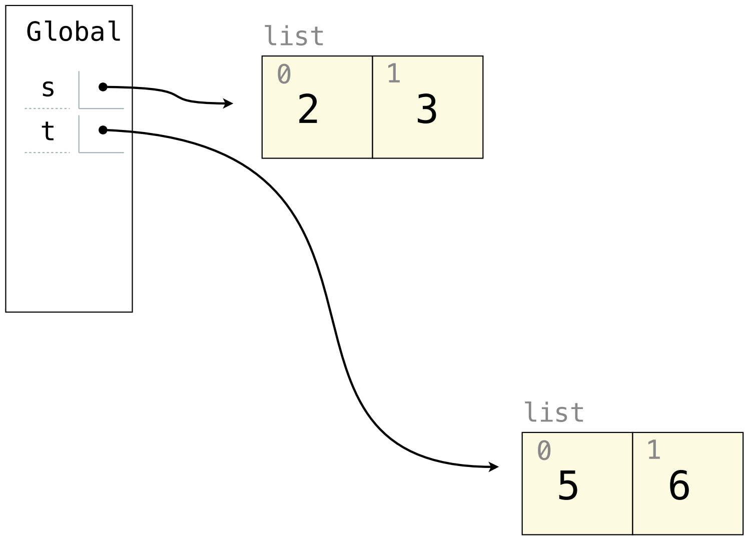
append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	s.extend(t)
	t[1] = 0
		s → [2, 3, 5, 6]
		t → [5, 0]

addition & **slicing** create new lists containing existing elements	a = s + [t]
	b = a[1:]
	a[1] = 9
	b[1][1] = 0
		s → [2, 3]
		t → [5, 0]
		a → [2, 9, [5, 0]]
		b → [3, [5, 0]]

The **list** function also creates a new list containing existing elements	t = list(s)
	s[1] = 0
		s → [2, 0]
		t → [2, 3]

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:

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

Operation	Example	Result

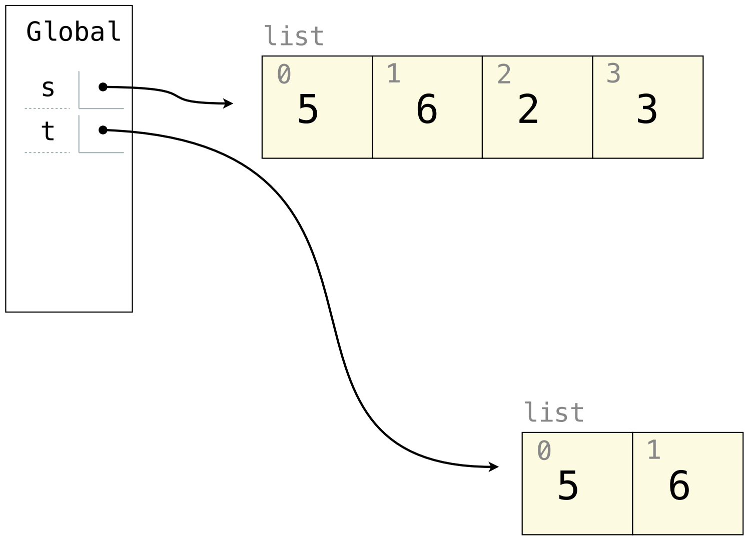
append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`
		`s → [2, 3]`
		`t → [5, 0]`
		`a → [2, 9, [5, 0]]`
		`b → [3, [5, 0]]`

The **list** function also creates a new list containing existing elements	`t = list(s)`
	`s[1] = 0`
		`s → [2, 0]`
		`t → [2, 3]`

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:

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

Operation	Example	Result

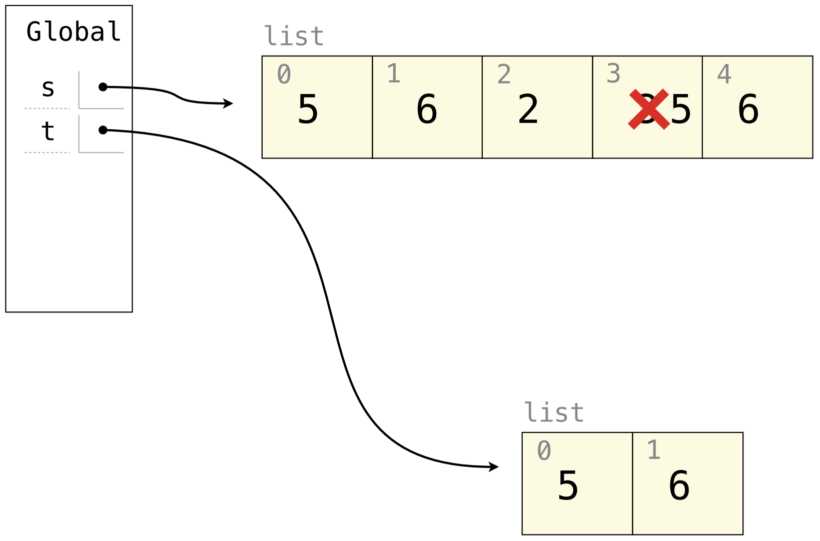
append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`
		`s → [2, 3]`
		`t → [5, 0]`
		`a → [2, 9, [5, 0]]`
		`b → [3, [5, 0]]`

The **list** function also creates a new list containing existing elements	`t = list(s)`
	`s[1] = 0`
		`s → [2, 0]`
		`t → [2, 3]`

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:

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

Operation	Example	Result

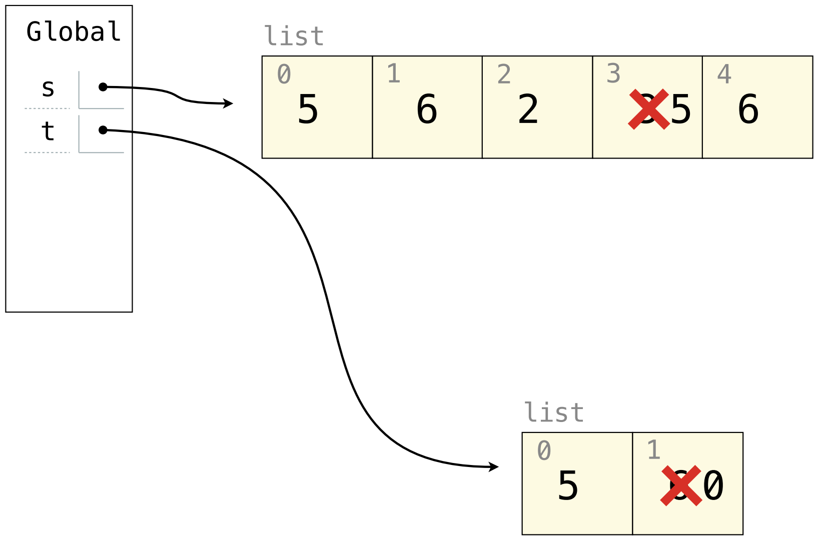
append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

extend adds all elements in one list to another list	s.extend(t)
	t[1] = 0
		s → [2, 3, 5, 6]
		t → [5, 0]

addition & **slicing** create new lists containing existing elements	a = s + [t]
	b = a[1:]
	a[1] = 9
	b[1][1] = 0
		s → [2, 3]
		t → [5, 0]
		a → [2, 9, [5, 0]]
		b → [3, [5, 0]]

The **list** function also creates a new list containing existing elements	t = list(s)
	s[1] = 0
		s → [2, 0]
		t → [2, 3]

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:

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

Operation	Example	Result

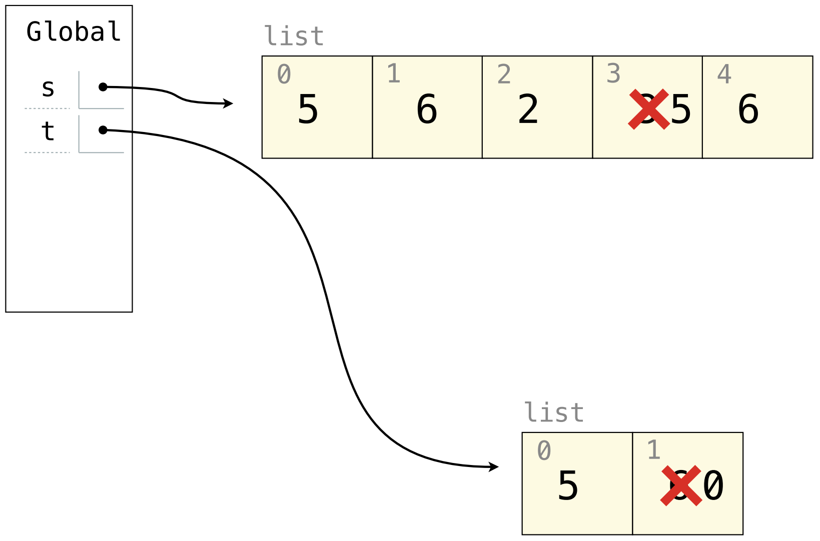
append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

extend adds all elements in one list to another list	`s.extend(t)`
	`t[1] = 0`
		`s → [2, 3, 5, 6]`
		`t → [5, 0]`

addition & **slicing** create new lists containing existing elements	`a = s + [t]`
	`b = a[1:]`
	`a[1] = 9`
	`b[1][1] = 0`
		`s → [2, 3]`
		`t → [5, 0]`
		`a → [2, 9, [5, 0]]`
		`b → [3, [5, 0]]`

The **list** function also creates a new list containing existing elements	`t = list(s)`
	`s[1] = 0`
		`s → [2, 0]`
		`t → [2, 3]`

slice assignment replaces a slice with new values	`s[0:0] = t`
	`s[3:] = t`
	`t[1] = 0`
		`s → [5, 6, 2, 5, 6]`
		`t → [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:

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

Operation	Example	Result

Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result
pop removes & returns the last element		

Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result
pop removes & returns the last element	`t = s.pop()`	

Lists in Environment Diagrams

Assume that before each example below we execute:

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

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

Lists in Environment Diagrams

Assume that before each example below we execute:

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

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

Lists in Environment Diagrams

Assume that before each example below we execute:

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

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

remove removes the first element equal to the argument	`t.extend(t)`
	`t.remove(5)`	

Lists in Environment Diagrams

Assume that before each example below we execute:

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

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

remove removes the first element equal to the argument	`t.extend(t)`
	`t.remove(5)`
		s → [2, 3]
		t → [6, 5, 6]

Lists in Environment Diagrams

Assume that before each example below we execute:

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

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

remove removes the first element equal to the argument	`t.extend(t)`
	`t.remove(5)`
		s → [2, 3]
		t → [6, 5, 6]

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]
```

Operation	Example	Result

pop removes & returns the last element	`t = s.pop()`	`s → [2]`
		`t → 3`

remove removes the first element equal to the argument	`t.extend(t)`
	`t.remove(5)`
		`s → [2, 3]`
		`t → [6, 5, 6]`

slice assignment can remove elements from a list by assigning [] to a slice.	`s[:1] = []`
	`t[0:2] = []`	

Lists in Environment Diagrams

Assume that before each example below we execute:

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

Operation	Example	Result

pop removes & returns the last element	`t = s.pop()`	`s → [2]`
		`t → 3`

remove removes the first element equal to the argument	`t.extend(t)`
	`t.remove(5)`
		`s → [2, 3]`
		`t → [6, 5, 6]`

slice assignment can remove elements from a list by assigning [] to a slice.	`s[:1] = []`
	`t[0:2] = []`
		`s → [3]`
		`t → []`

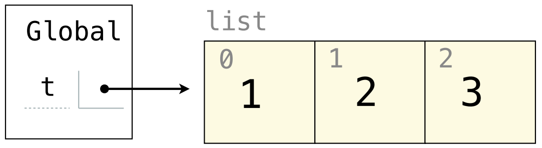
Lists in Lists in Lists in Environment Diagrams

```
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

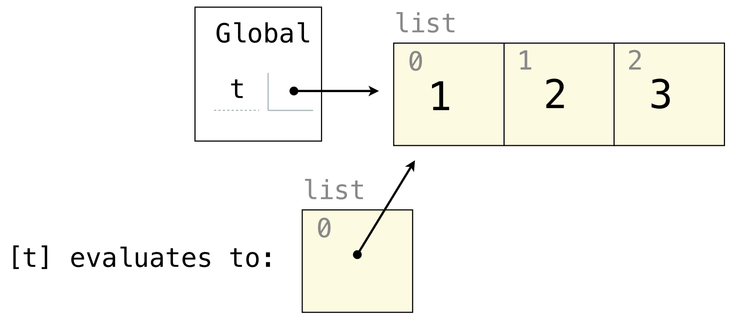
```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```



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

Lists in Lists in Lists in Environment Diagrams

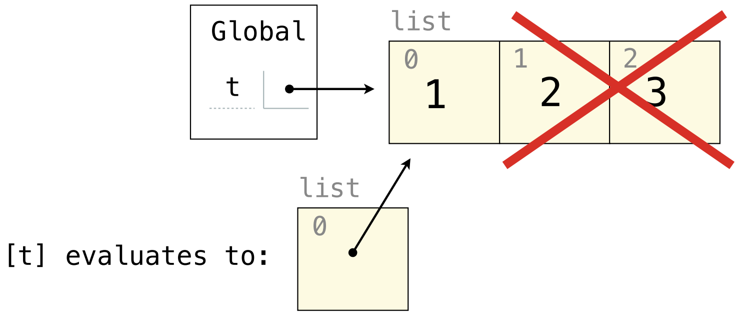
```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```



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

Lists in Lists in Lists in Environment Diagrams

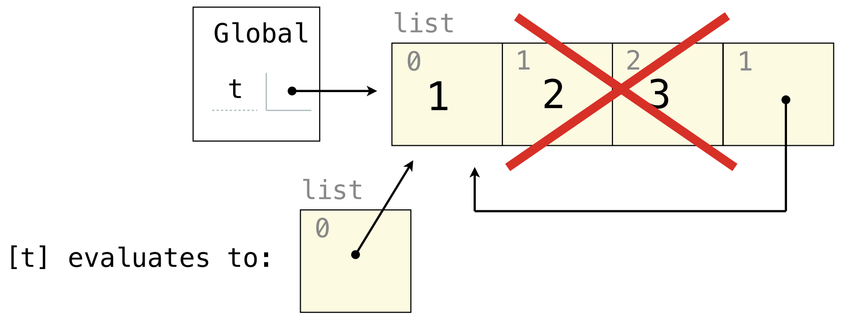
```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```



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

Lists in Lists in Lists in Environment Diagrams

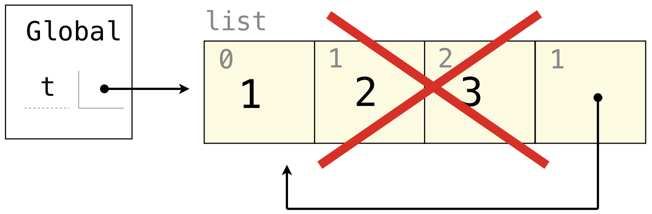
```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```



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

Lists in Lists in Lists in Environment Diagrams

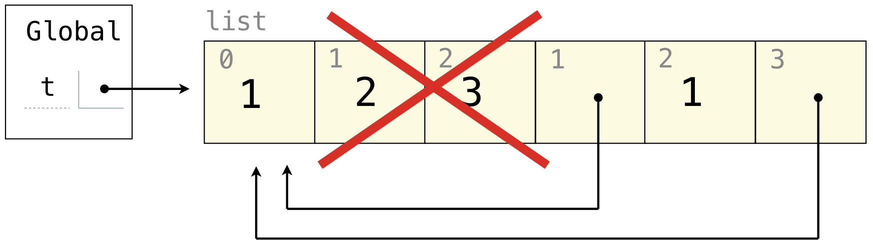
```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```



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

Lists in Lists in Lists in Environment Diagrams

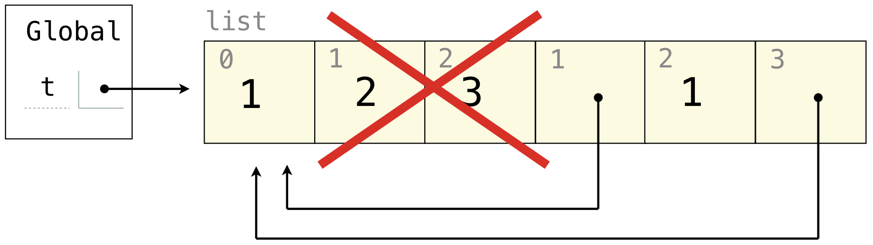
```
t = [1, 2, 3]
t[1:3] = [t]
t.extend(t)
```



```
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)
```

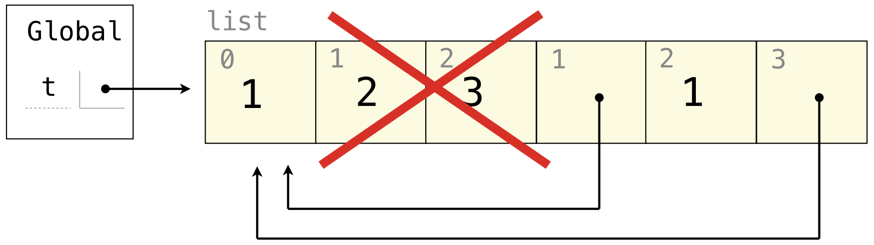


```
[1, [...], 1, [...]]
```

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

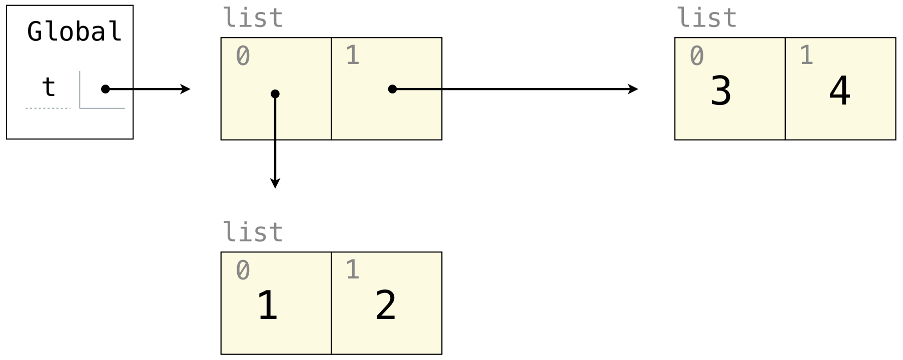
Lists in Lists in Lists in Environment Diagrams

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



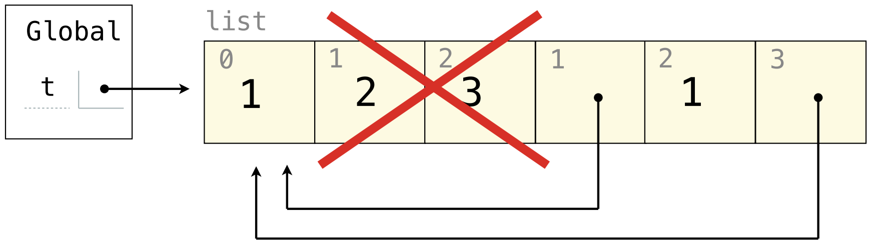
```
[1, [...], 1, [...]]
```

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



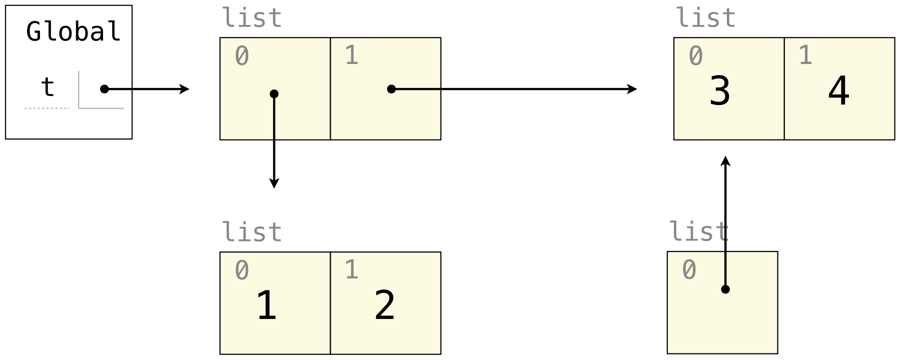
Lists in Lists in Lists in Environment Diagrams

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



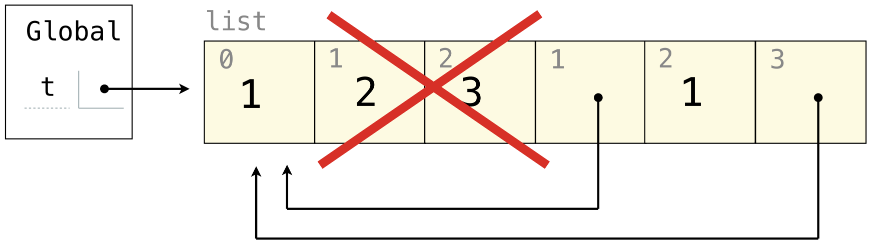
```
[1, [...], 1, [...]]
```

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



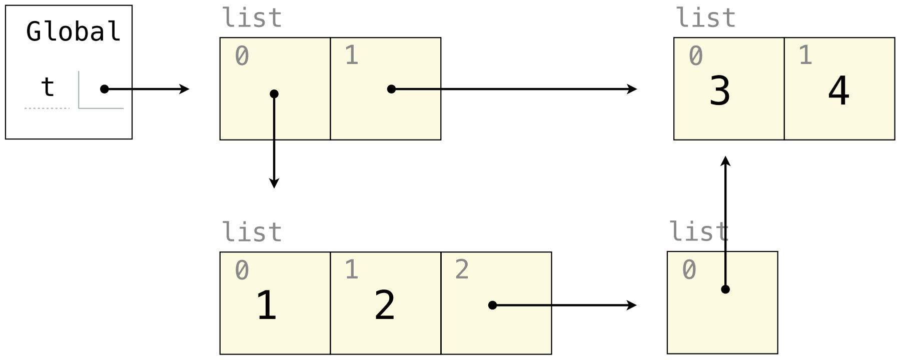
Lists in Lists in Lists in Environment Diagrams

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



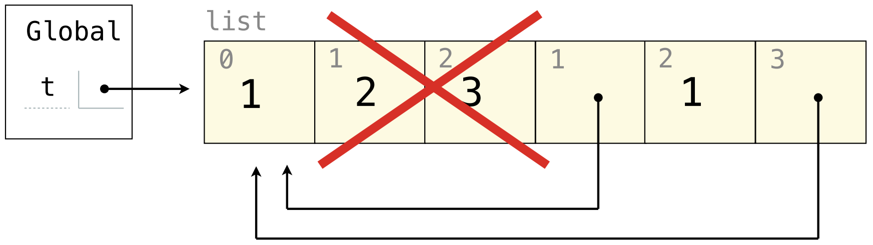
```
[1, [...], 1, [...]]
```

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



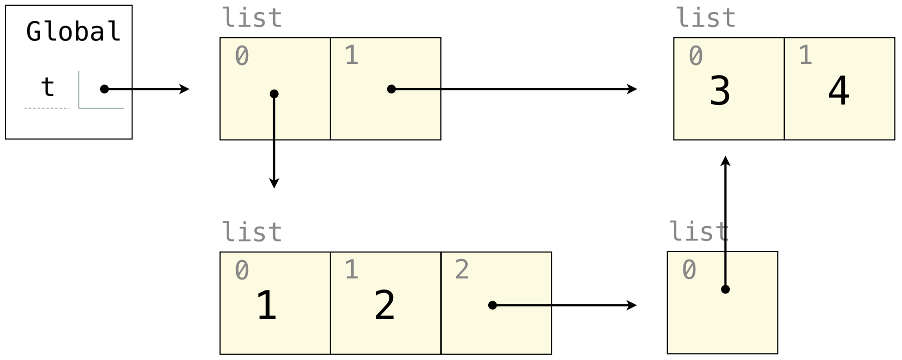
Lists in Lists in Lists in Environment Diagrams

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



```
[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'
```

Land Owners

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
```

```
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
```

```
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

```
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()

>>> jack

>>> jack.work()

>>> john.work()

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

<class Worker>

greeting: 'Sir'

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()

>>> jack

>>> jack.work()

>>> john.work()

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

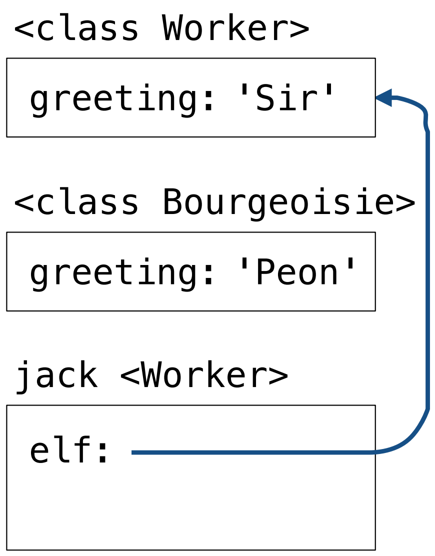
```
<class Worker>
greeting: 'Sir'

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

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()

>>> jack

>>> jack.work()

>>> john.work()

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

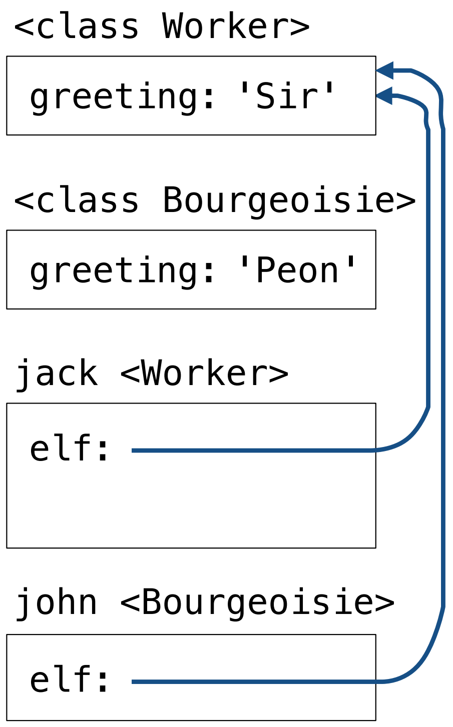


Land Owners

`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()

>>> jack

>>> jack.work()

>>> john.work()

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

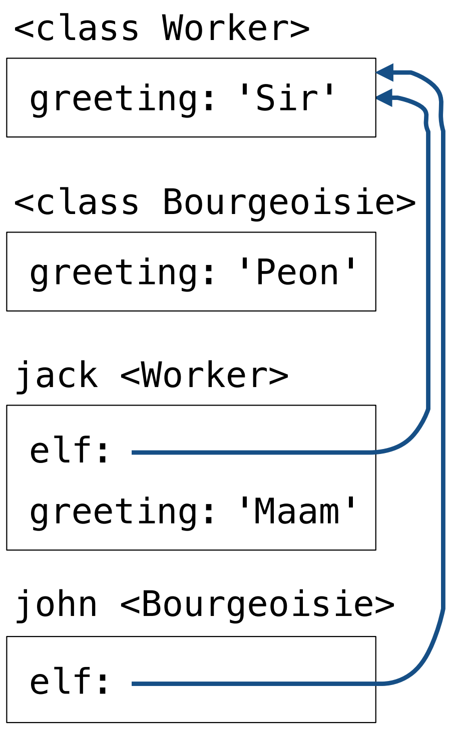


Land Owners

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()

>>> jack

>>> jack.work()

>>> john.work()

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

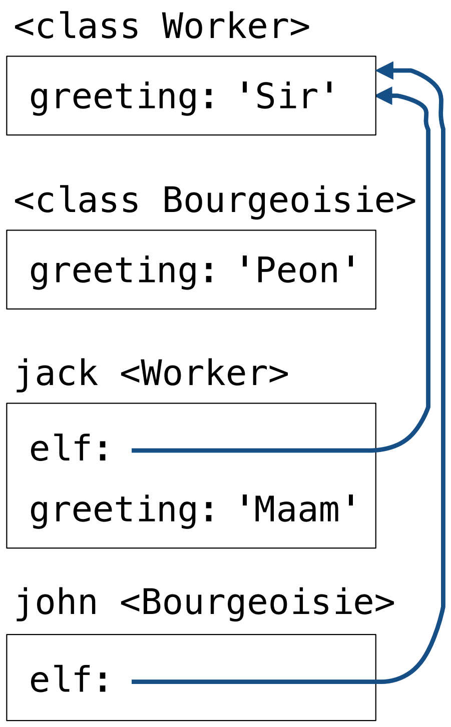


Land Owners

```
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()

>>> jack

>>> jack.work()

>>> john.work()

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

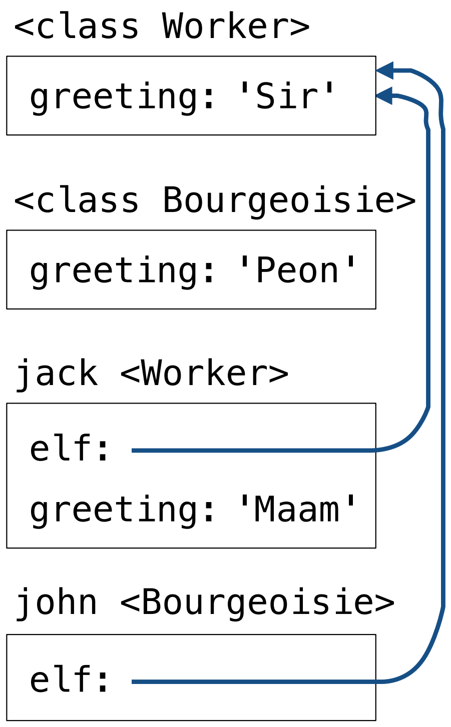


Land Owners

```
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()
'Sir, I work'

>>> jack

>>> jack.work()

>>> john.work()

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

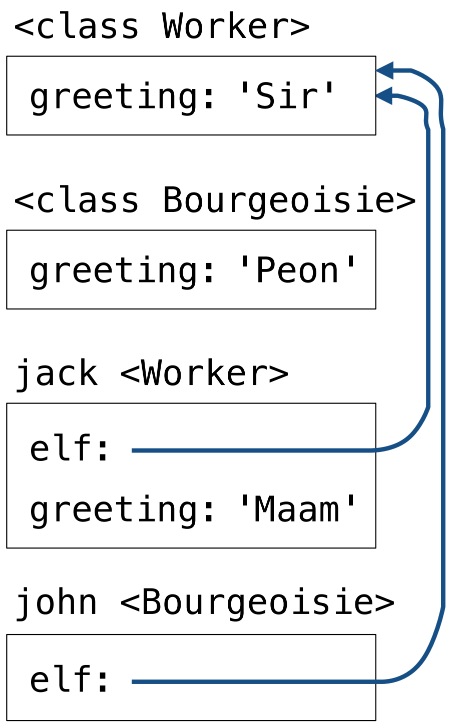


Land Owners

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()
'Sir, I work'

>>> jack

>>> jack.work()

>>> john.work()

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

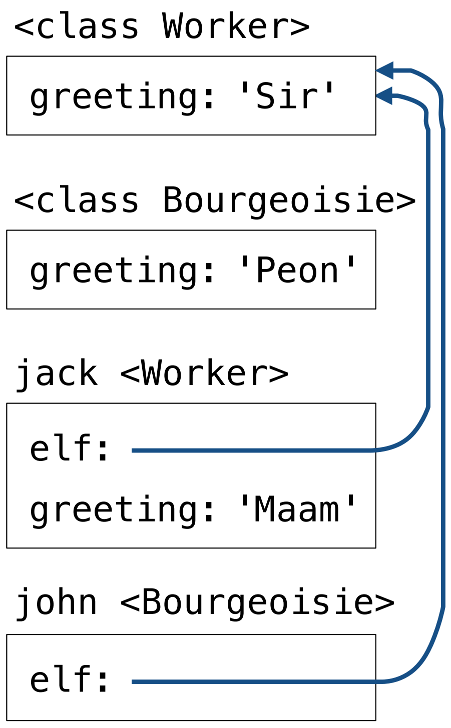


Land Owners

```
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()
'Sir, I work'

>>> jack
Peon

>>> jack.work()


>>> john.work()


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

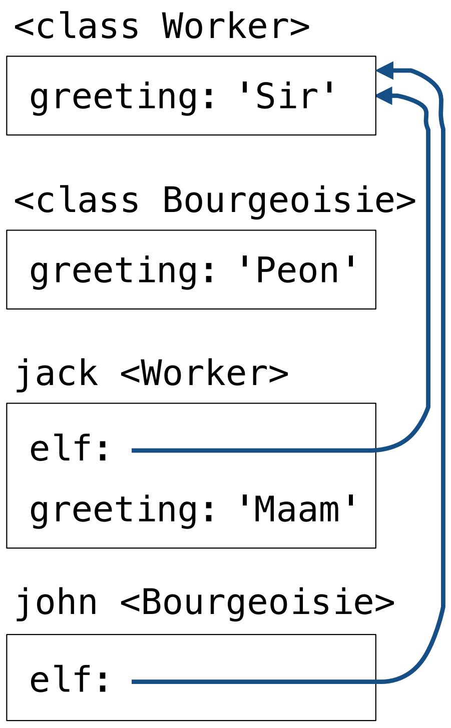


Land Owners

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)
```

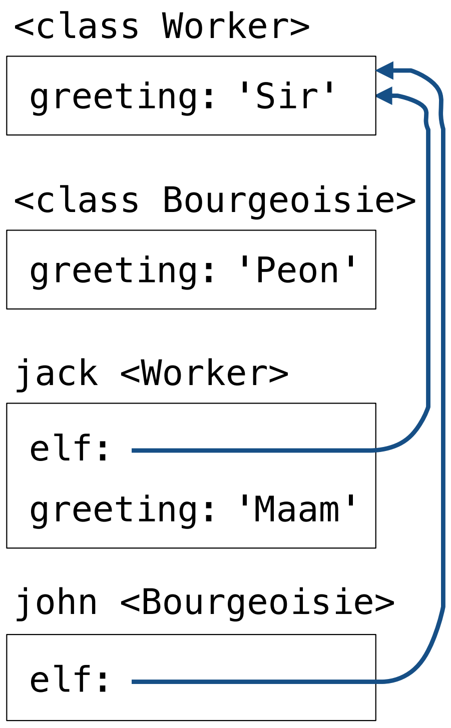


Land Owners

```
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)
```

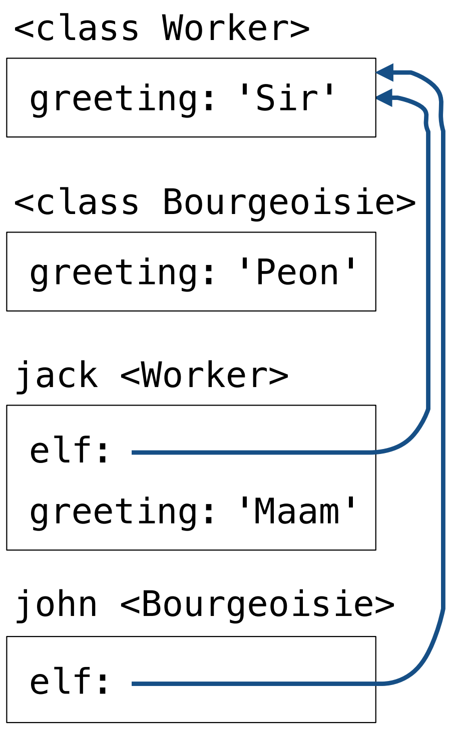


Land Owners

```
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)
```

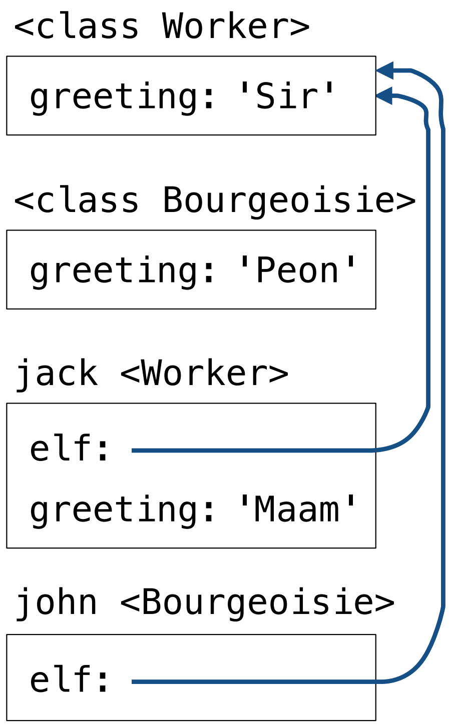


Land Owners

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)
```

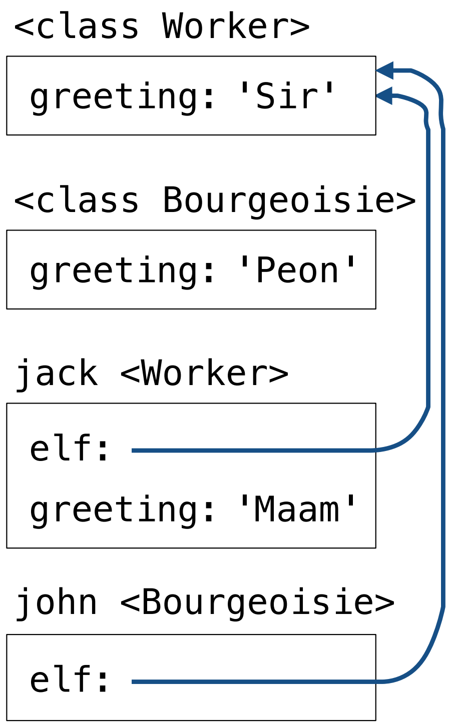


Land Owners

```
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
'I gather wealth'

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

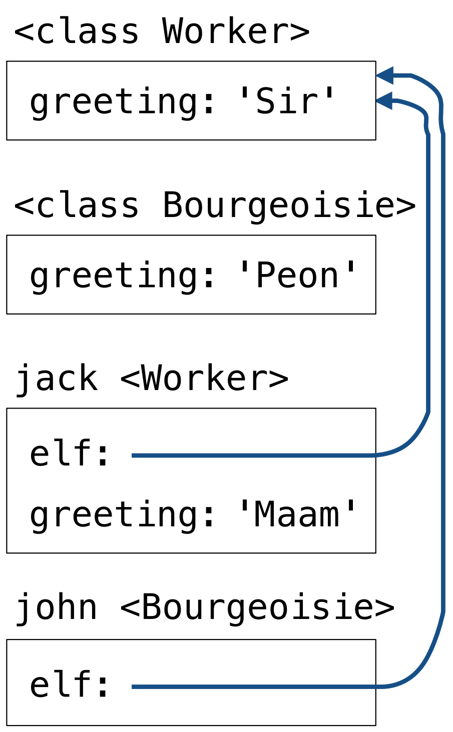


Land Owners

```
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
'I gather wealth'

>>> 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?

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

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

What's the largest sum of two adjacent elements in a list s? (Assume 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?

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

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

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

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]
  0   1   2   3   4   5
```

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]
```

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]
  0    1    2   3   4   5
```

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
```

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]
   0   1   2   3   4   5
```

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

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]
```

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]
  0   1   2   3   4   5
```

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]}
```

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]
  0   1   2   3   4   5
```

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

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?

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

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?

```
[-4, -3, -2,  3,  2,  4]  ▷  False       [4, 3, 2, 3, 2, 4]  ▷  True
```

Examples: Linked Lists

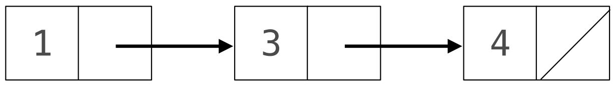
Linked List Exercises

Linked List Exercises

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

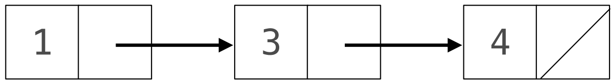
Linked List Exercises

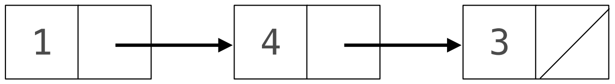
`Is a linked list s ordered from least to greatest?`



Linked List Exercises

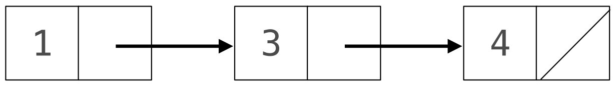
`Is a linked list s ordered from least to greatest?`

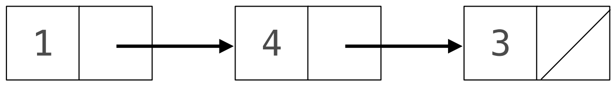




Linked List Exercises

Is a linked list s ordered from least to greatest?

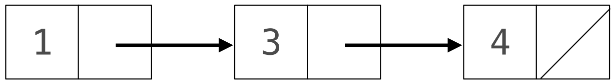


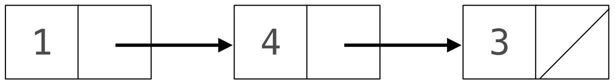


Is a linked list s ordered from least to greatest by absolute value (or a key function)?

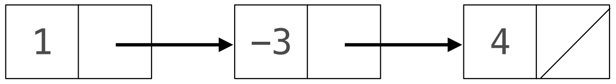
Linked List Exercises

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



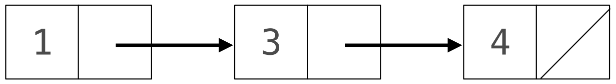


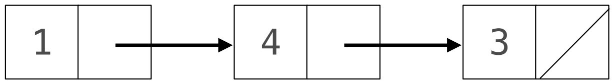
```
Is a linked list s ordered from least to greatest by absolute value (or a key function)?
```



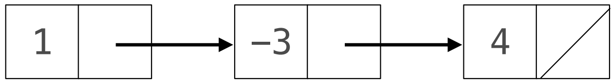
Linked List Exercises

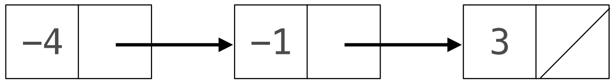
Is a linked list s ordered from least to greatest?





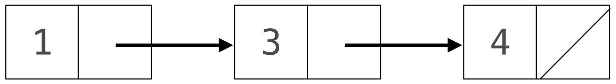
Is a linked list s ordered from least to greatest by absolute value (or a key function)?

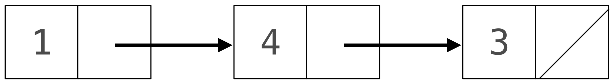




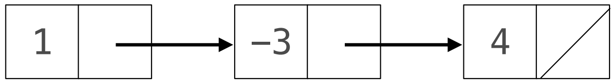
Linked List Exercises

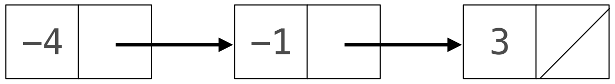
`Is a linked list s ordered from least to greatest?`





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

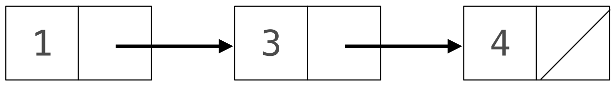


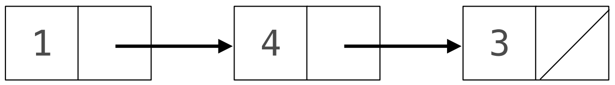


`Create a sorted Link containing all the elements of both sorted Links s & t.`

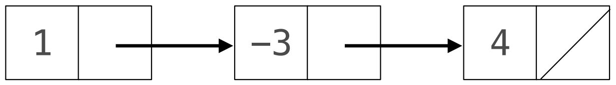
Linked List Exercises

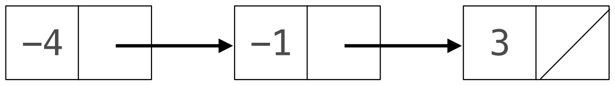
Is a linked list s ordered from least to greatest?



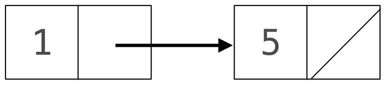


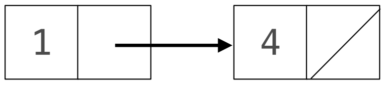
Is a linked list s ordered from least to greatest by absolute value (or a key function)?





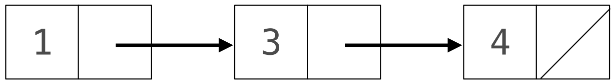
Create a sorted Link containing all the elements of both sorted Links s & t.

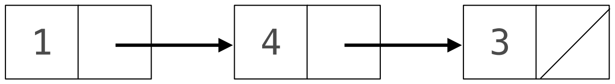




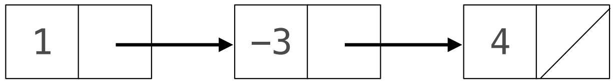
Linked List Exercises

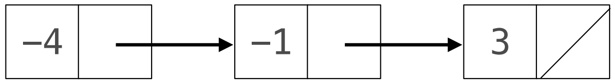
Is a linked list s ordered from least to greatest?



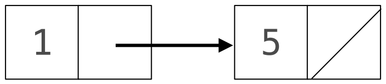


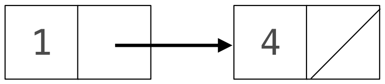
Is a linked list s ordered from least to greatest by absolute value (or a key function)?

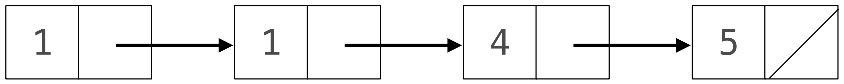




Create a sorted Link containing all the elements of both sorted Links s & t.

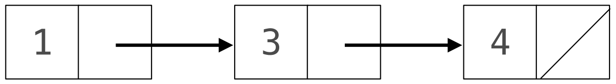


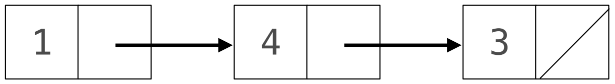




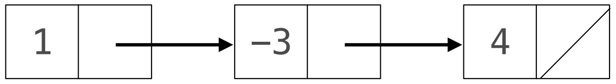
Linked List Exercises

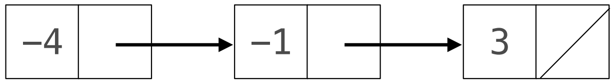
Is a linked list s ordered from least to greatest?



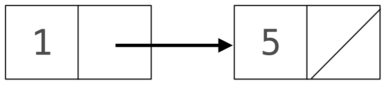


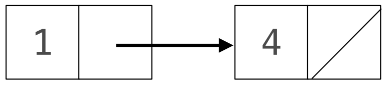
Is a linked list s ordered from least to greatest by absolute value (or a key function)?

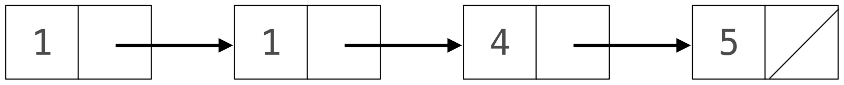




Create a sorted Link containing all the elements of both sorted Links s & t.



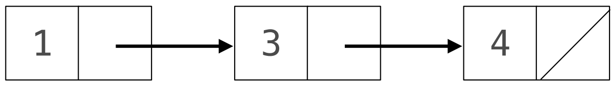


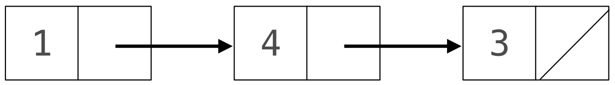


Do the same thing, but never call Link.

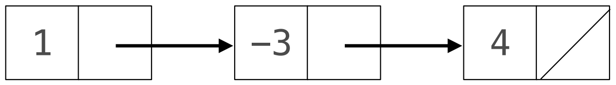
Linked List Exercises

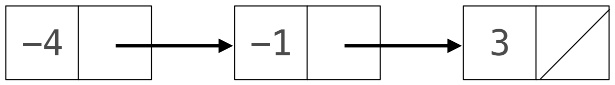
Is a linked list s ordered from least to greatest?



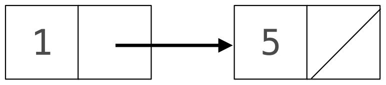


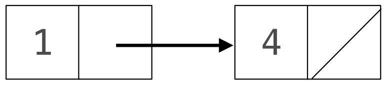
Is a linked list s ordered from least to greatest by absolute value (or a key function)?

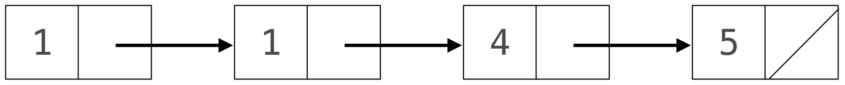




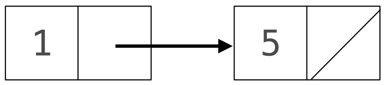
Create a sorted Link containing all the elements of both sorted Links s & t.

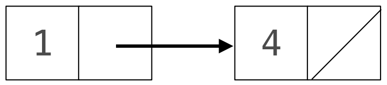






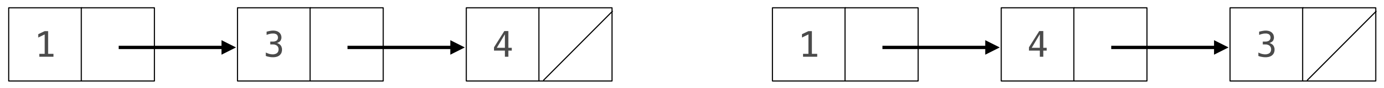
Do the same thing, but never call Link.



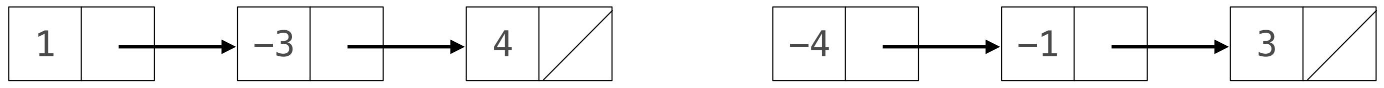


Linked List Exercises

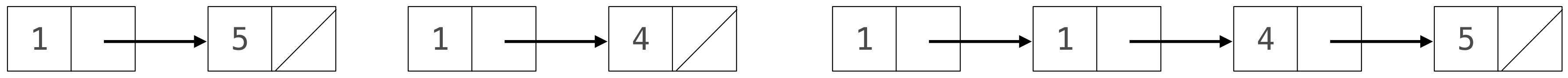
`Is a linked list s ordered from least to greatest?`



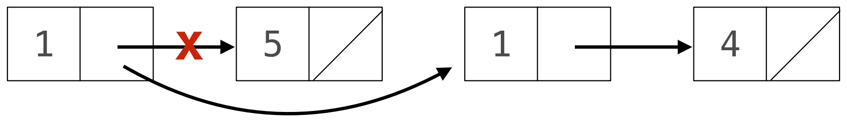
`Is a linked list s ordered from least to greatest by absolute value (or a key function)?`



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

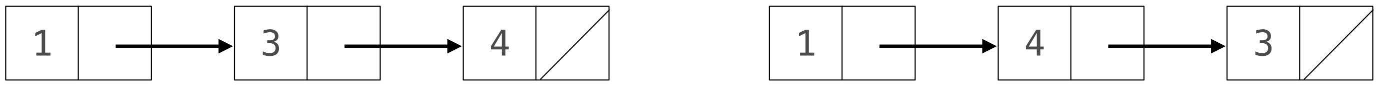


`Do the same thing, but never call Link.`

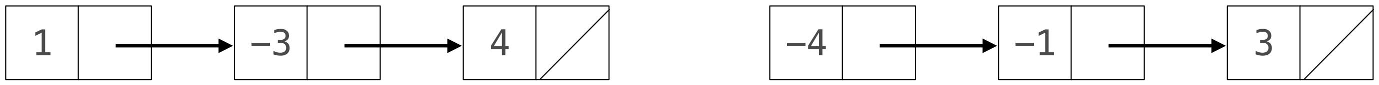


Linked List Exercises

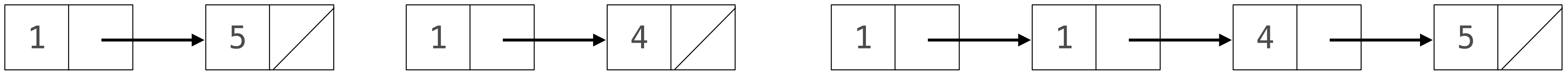
Is a linked list s ordered from least to greatest?



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.



Do the same thing, but never call Link.