Lists

Thanks, Prof. DeNero!

`

Assume that before each example below we execute:

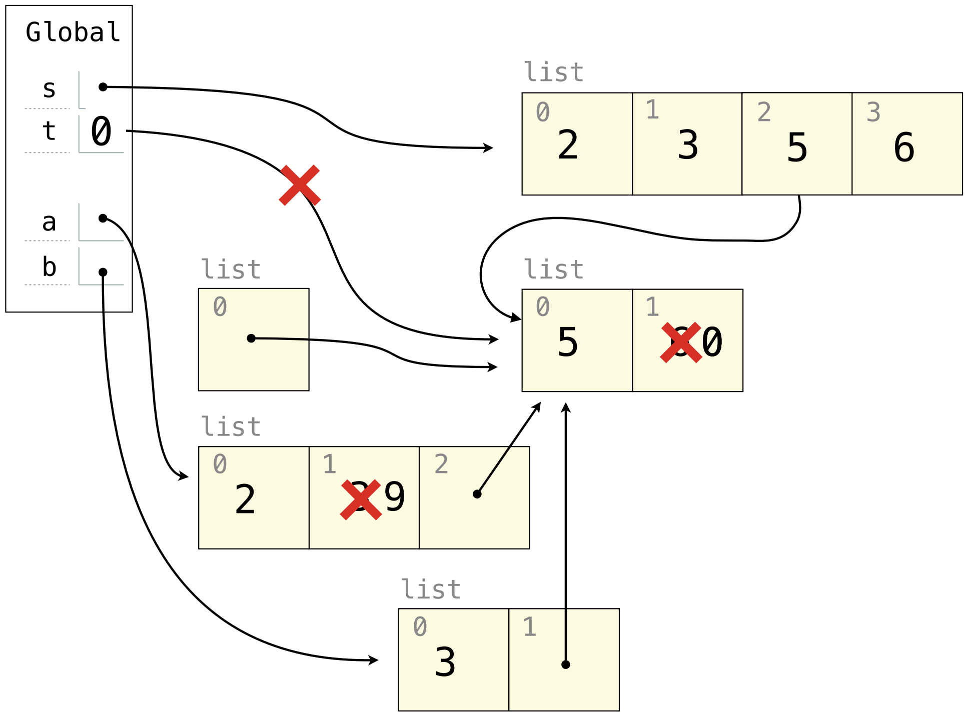
```
s = [2, 3]
t = [5, 6]
```

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]
```

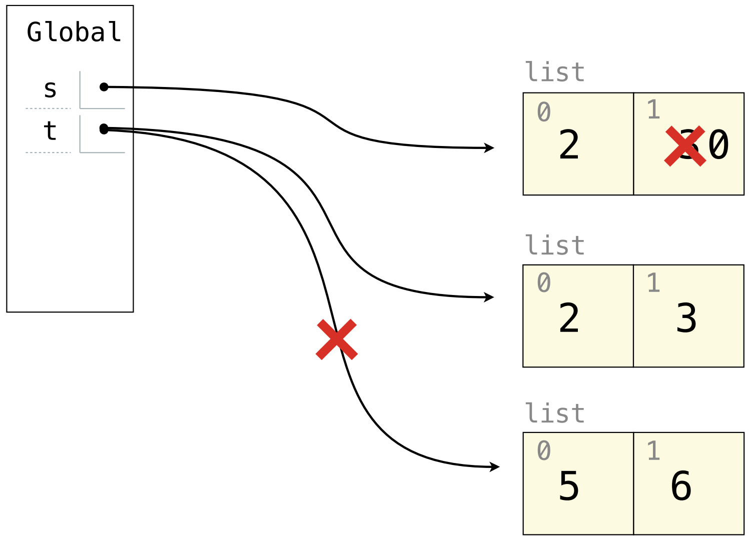
Operation	Example	Result

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`
		`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

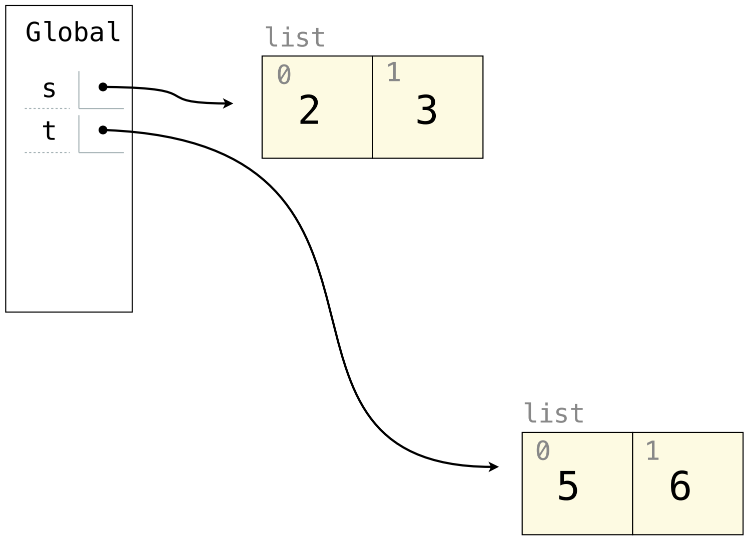
append adds one element to a list	`s.append(t)`
	`t = 0`
		`s → [2, 3, [5, 6]]`
		`t → 0`

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

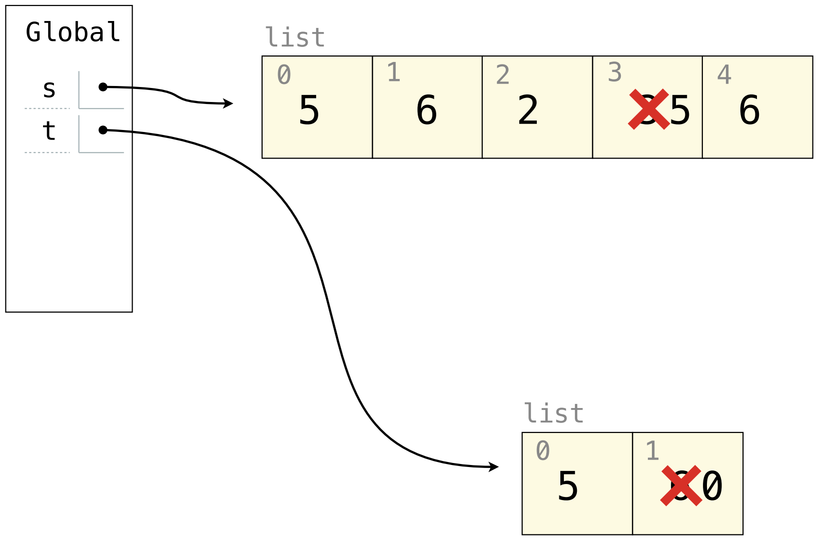
append adds one element to a list	s.append(t)
	t = 0
		s → [2, 3, [5, 6]]
		t → 0

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	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]
```

Operation	Example	Result

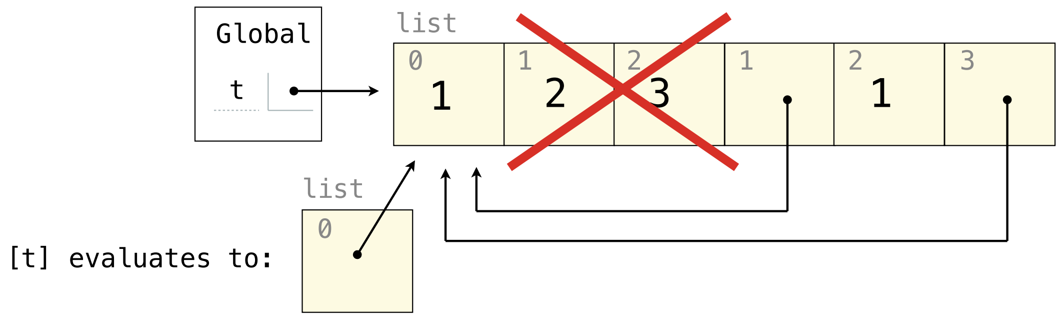
pop removes & returns the last element	`t = s.pop()`	`s → [2]`
		`t → 3`

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 → []`

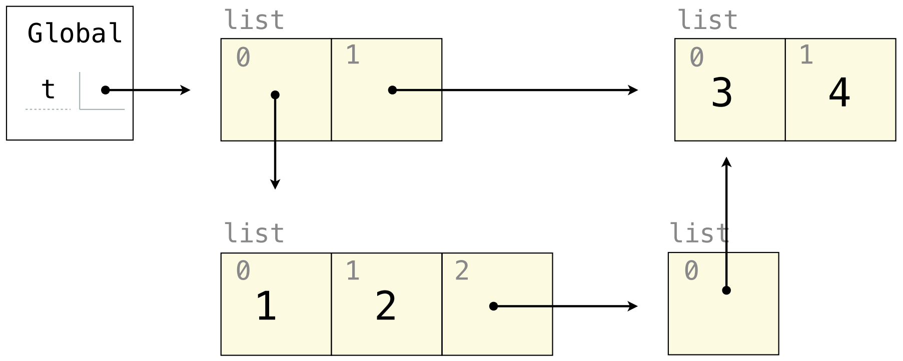
Lists in Lists in Lists in Environment Diagrams

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



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

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



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

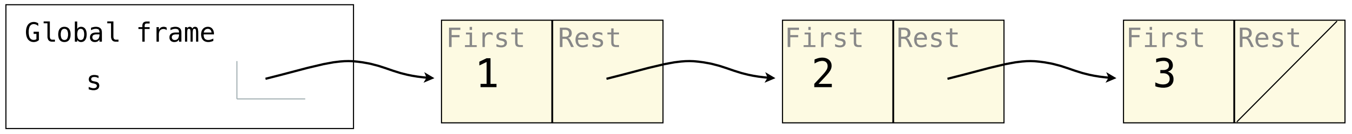
Mutable Linked Lists

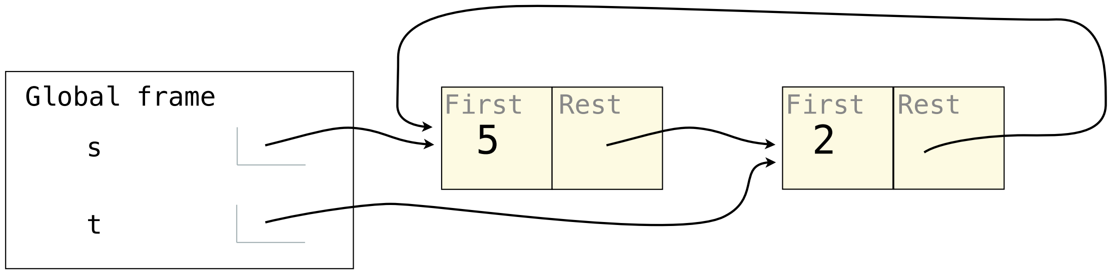
Recursive Lists Can Change

Attribute assignment statements can change first and rest attributes of a Link

The rest of a linked list can contain the linked list as a sub-list

```
>>> s = Link(1, Link(2, Link(3)))
>>> s.first = 5
>>> t = s.rest
>>> t.rest = s
>>> s.first
5
>>> s.rest.rest.rest.rest.rest.first
2
```

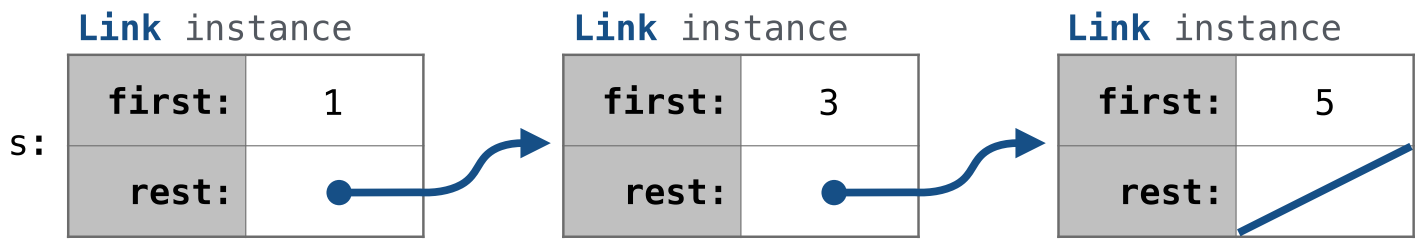




Note: The actual environment diagram is much more complicated.

Linked List Mutation Example

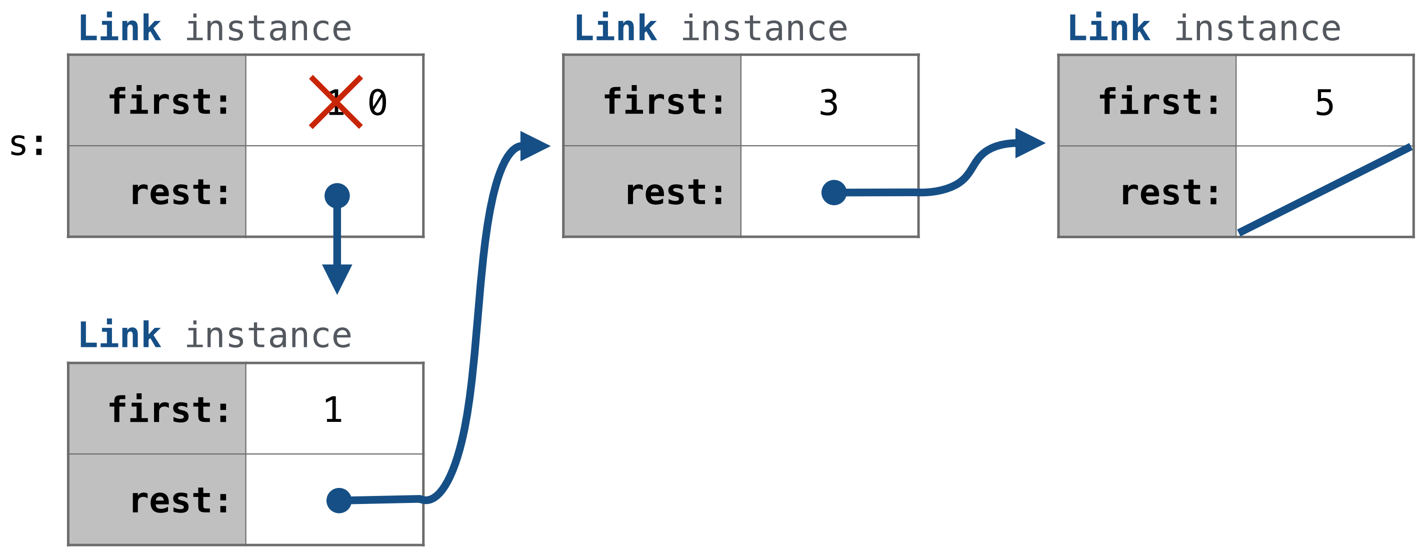
Adding to an Ordered List



```
def add(s, v):
    """Add v to an ordered list s with no repeats, returning modified s."""
    (Note: If v is already in s, then don't modify s, but still return it.)


         add(s, 0)
```

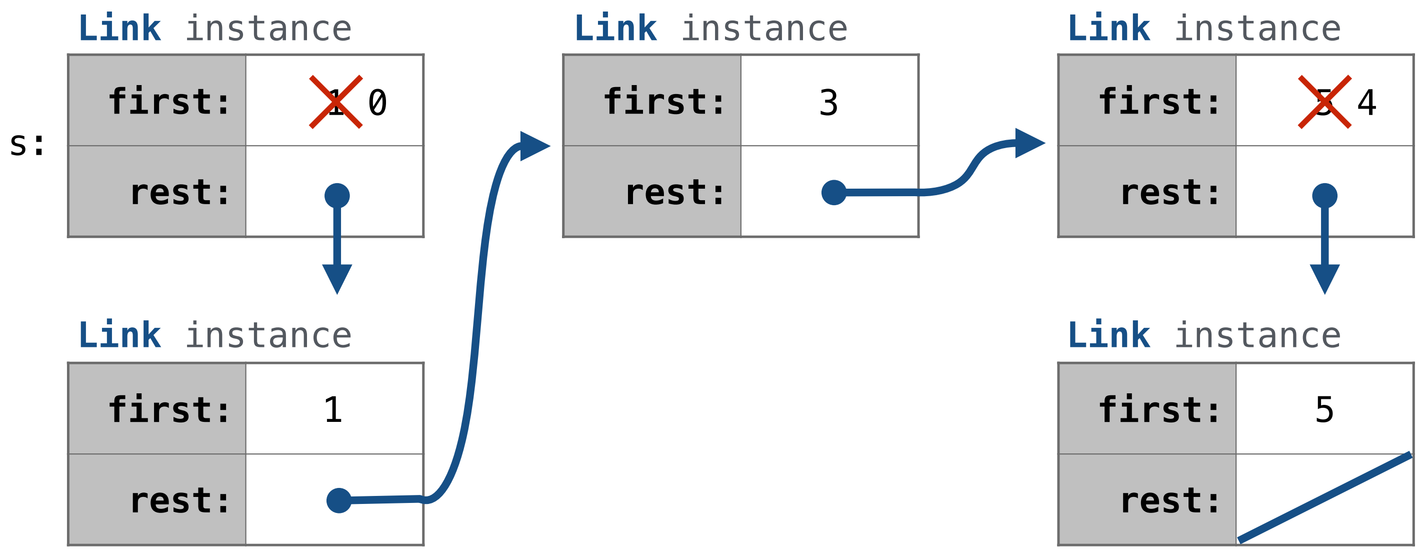
Adding to an Ordered List



```
def add(s, v):
    """Add v to an ordered list s with no repeats, returning modified s."""
    (Note: If v is already in s, then don't modify s, but still return it.)


        add(s, 0)      add(s, 3)      add(s, 4)
```

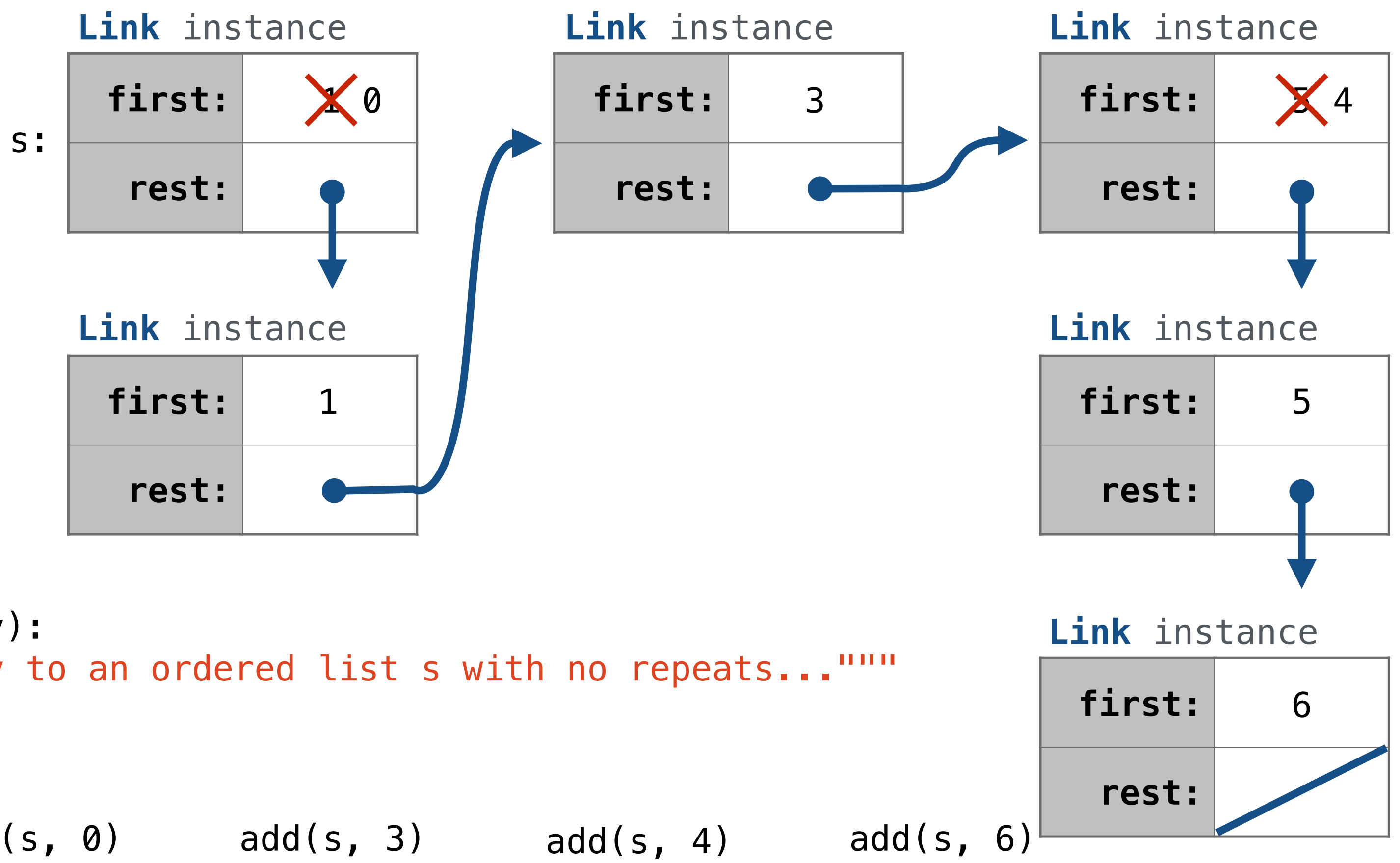
Adding to an Ordered List



```
def add(s, v):
    """Add v to an ordered list s with no repeats..."""
```

```
add(s, 0)     add(s, 3)     add(s, 4)     add(s, 6)
```

Adding to an Ordered List



```
def add(s, v):
    """Add v to an ordered list s with no repeats..."""
```

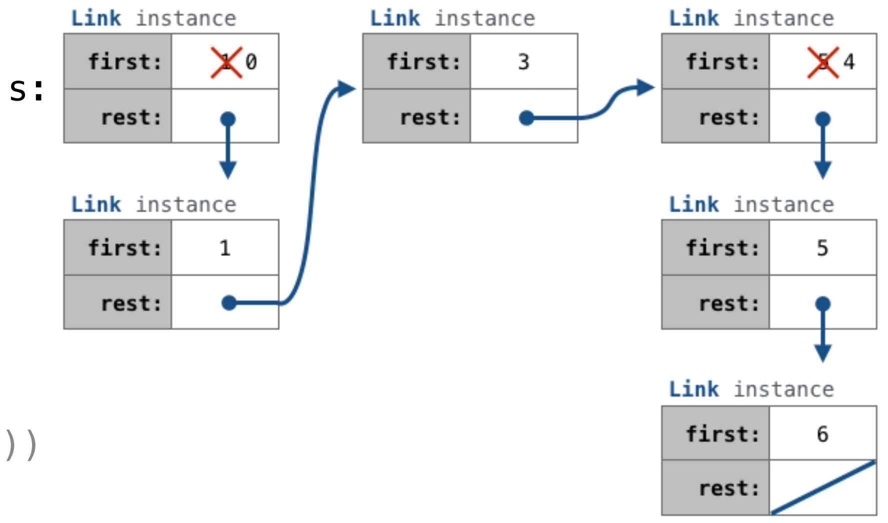
```
add(s, 0)    add(s, 3)    add(s, 4)    add(s, 6)
```

Adding to a Set Represented as an Ordered List

```
def add(s, v):
    """Add v to s, returning modified s.”””

    >>> s = Link(1, Link(3, Link(5)))
    >>> add(s, 0)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 3)
    Link(0, Link(1, Link(3, Link(5))))
    >>> add(s, 4)
    Link(0, Link(1, Link(3, Link(4, Link(5)))))
    >>> add(s, 6)
    Link(0, Link(1, Link(3, Link(4, Link(5, Link(6))
    """

    assert s is not List.empty
    if s.first > v:
                             v                            Link(s.first, s.rest)
        s.first, s.rest = _________________________ , ____________________________
    elif s.first < v and empty(s.rest):
                                  Link(v)
        s.rest = __________________________________________________________________
    elif s.first < v:
                                 add(s.rest, v)
        ___________________________________________________________________________
    return s
```



s:
Link instance
first: ~~1~~ 0
rest:
Link instance
first: 1
rest:
Link instance
first: 3
rest:
Link instance
first: ~~5~~ 4
rest:
Link instance
first: 5
rest:
Link instance
first: 6
rest: