Ordered Sets

Announcements

Sets

Sets

One more built-in Python container type

- Set literals are enclosed in braces

- Duplicate elements are removed on construction

- Sets have arbitrary order, just like dictionary entries

```
>>> s = {'one', 'two', 'three', 'four', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
>>> 'three' in s
True
>>> len(s)
4
>>> s.union({'one', 'five'})
{'three', 'five', 'one', 'four', 'two'}
>>> s.intersection({'six', 'five', 'four', 'three'})
{'three', 'four'}
>>> s
{'three', 'one', 'four', 'two'}
```
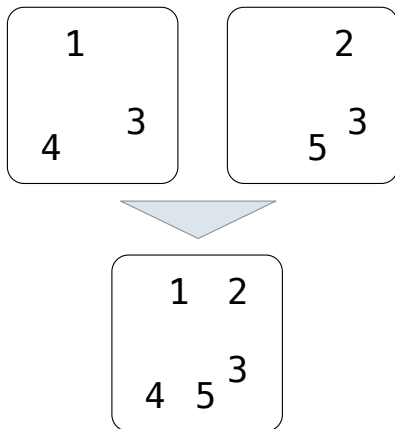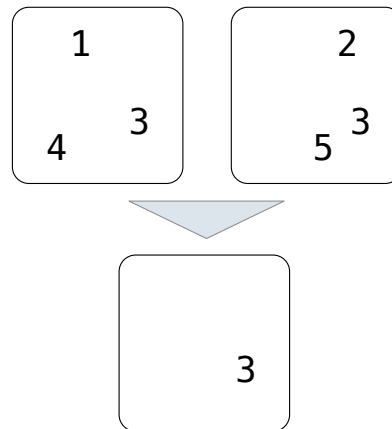
Implementing Sets

What we should be able to do with a set:

- **Membership testing:** Is a value an element of a set?
- **Union:** Return a set with all elements in set1 or set2
- **Intersection:** Return a set with any elements in set1 and set2
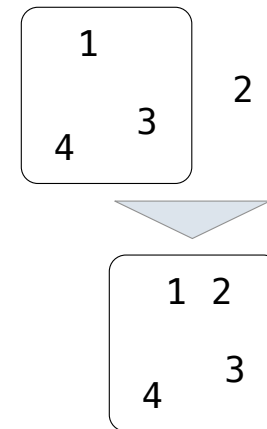- **Adjoin:** Return a set with all elements in s and a value v

Sets as Linked Lists

Sets as Unordered Sequences

Proposal 1: A set is represented by a linked list that contains no duplicate items.

Time order of growth

```
def empty(s):
    return s is Link.empty
```

$\Theta(1)$

```
def contains(s, v):
    """Return whether set s contains value v.

    >>> s = Link(1, Link(3, Link(2)))
    >>> contains(s, 2)
    True
    """
```

(Demo)

Time depends on whether
& where v appears in s.

$\Theta(n)$

In the worst case: v
does not appear in s
or
In the average case: appears
in a uniformly distributed
random location

Sets as Unordered Sequences

```
def adjoin(s, v):
    if contains(s, v):
        return s
    else:
        return Link(v, s)
```

$$\Theta(n)$$

The size of the set

```
def intersect(s, t):

    if s is Link.empty:

        return Link.empty

    rest = ____intersect(s.rest, t)____

    if contains(t, s.first):

        return __Link(s.first, rest)__

    else:

        return rest
```

$$\Theta(n^2)$$

If sets are
the same size

Sets as Ordered Linked Lists

Sets as Ordered Sequences

Proposal 2: A set is represented by a linked list with unique elements that is
            ordered from least to greatest

Parts of the program that...	Assume that sets are...	Using...
Use sets to contain values	Unordered collections	empty, contains, adjoin, intersect, union
Implement set operations	Ordered linked lists	first, rest, <, >, ==

Different parts of a program may make different assumptions about data

Searching an Ordered List

```
>>> s = Link(1, Link(3, Link(5)))
>>> contains(s, 1)
True
>>> contains(s, 2)
False
>>> t = adjoin(s, 2)
```
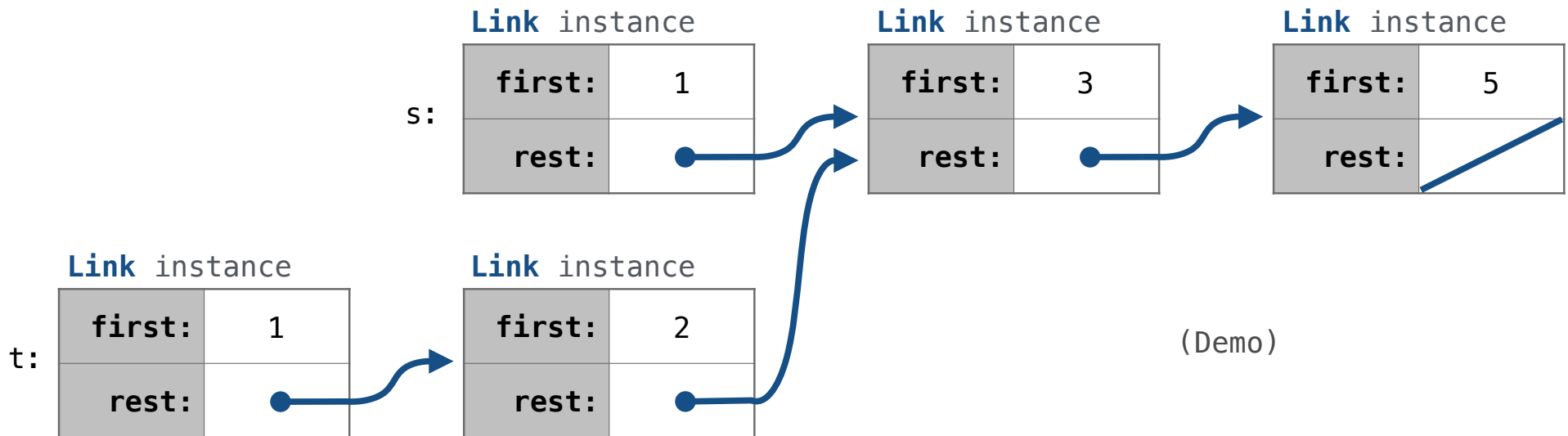
Operation	Time order of growth
contains	$\Theta(n)$
adjoin	$\Theta(n)$

(Demo)

Set Operations

Intersecting Ordered Linked Lists

Proposal 2: A set is represented by a linked list with unique elements that is *ordered from least to greatest*

```python
def intersect(s, t):
    if empty(s) or empty(t):
        return Link.empty
    else:
        e1, e2 = s.first, t.first
        if e1 == e2:
            return Link(e1, intersect(s.rest, t.rest))
        elif e1 < e2:
            return intersect(s.rest, t)
        elif e2 < e1:
            return intersect(s, t.rest)
```

Order of growth?   If s and t are sets of size n, then $\Theta(n)$ (Demo)

Set Mutation

Adding to an Ordered List

Adding to an Ordered List

Link instance

first:	~~1~~ 0
rest:	•

s:

Link instance

first:	3
rest:	•

Link instance

first:	5
rest:	

Link instance
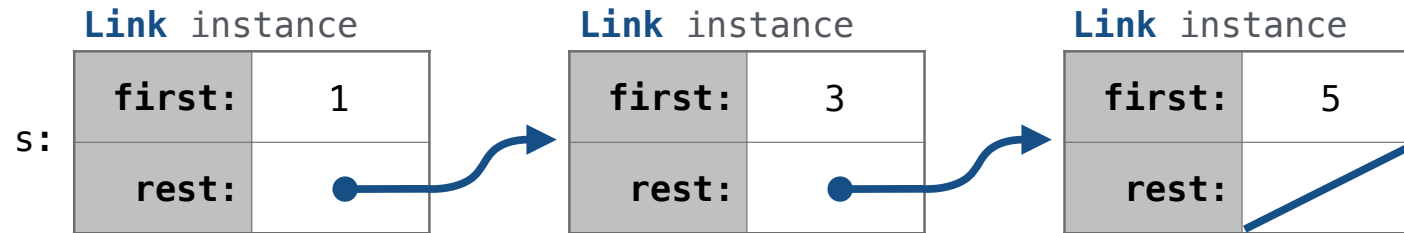
first:	1
rest:	•
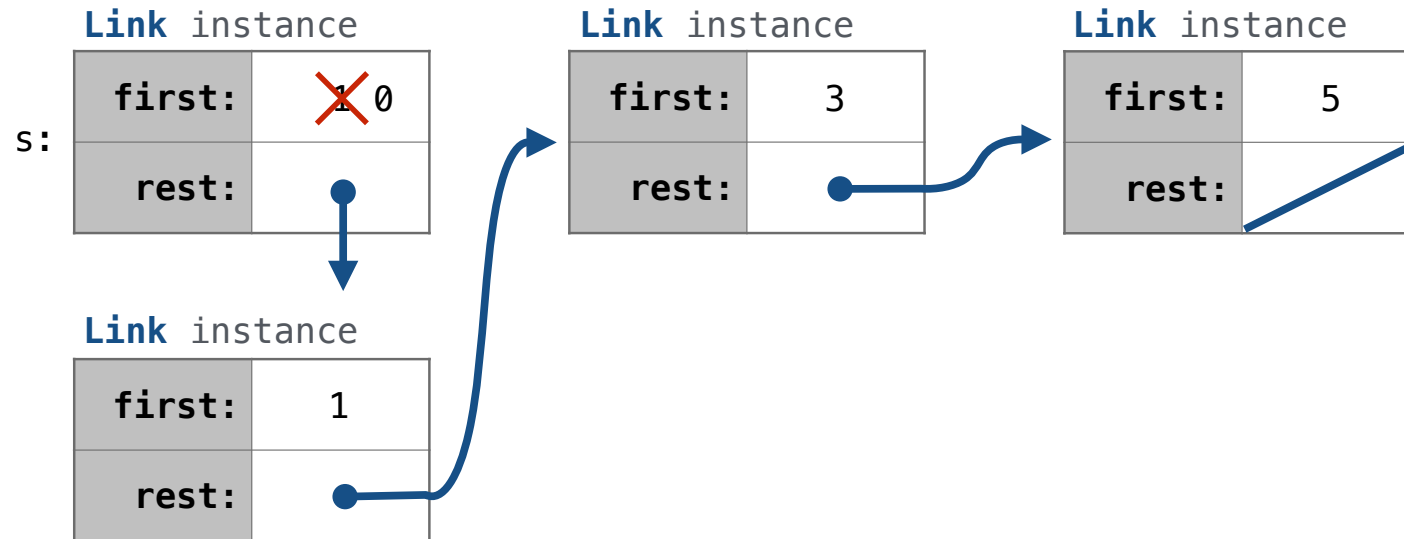
```
add(s, 3)

add(s, 4)
```

Adding to an Ordered List

Link instance

first:	❌ 0
rest:	●

s:

Link instance

first:	3
rest:	●

Link instance

first:	❌ 4
rest:	●

Link instance

first:	1
rest:	●

Link instance

first:	5
rest:	

add(s, 6)

Adding to an Ordered List

Link instance

first:	✗ 0
rest:	●

s:

Link instance

first:	1
rest:	●

Link instance

first:	3
rest:	●

Link instance

first:	✗ 4
rest:	●

Link instance

first:	5
rest:	●

Link instance

first:	6
rest:	
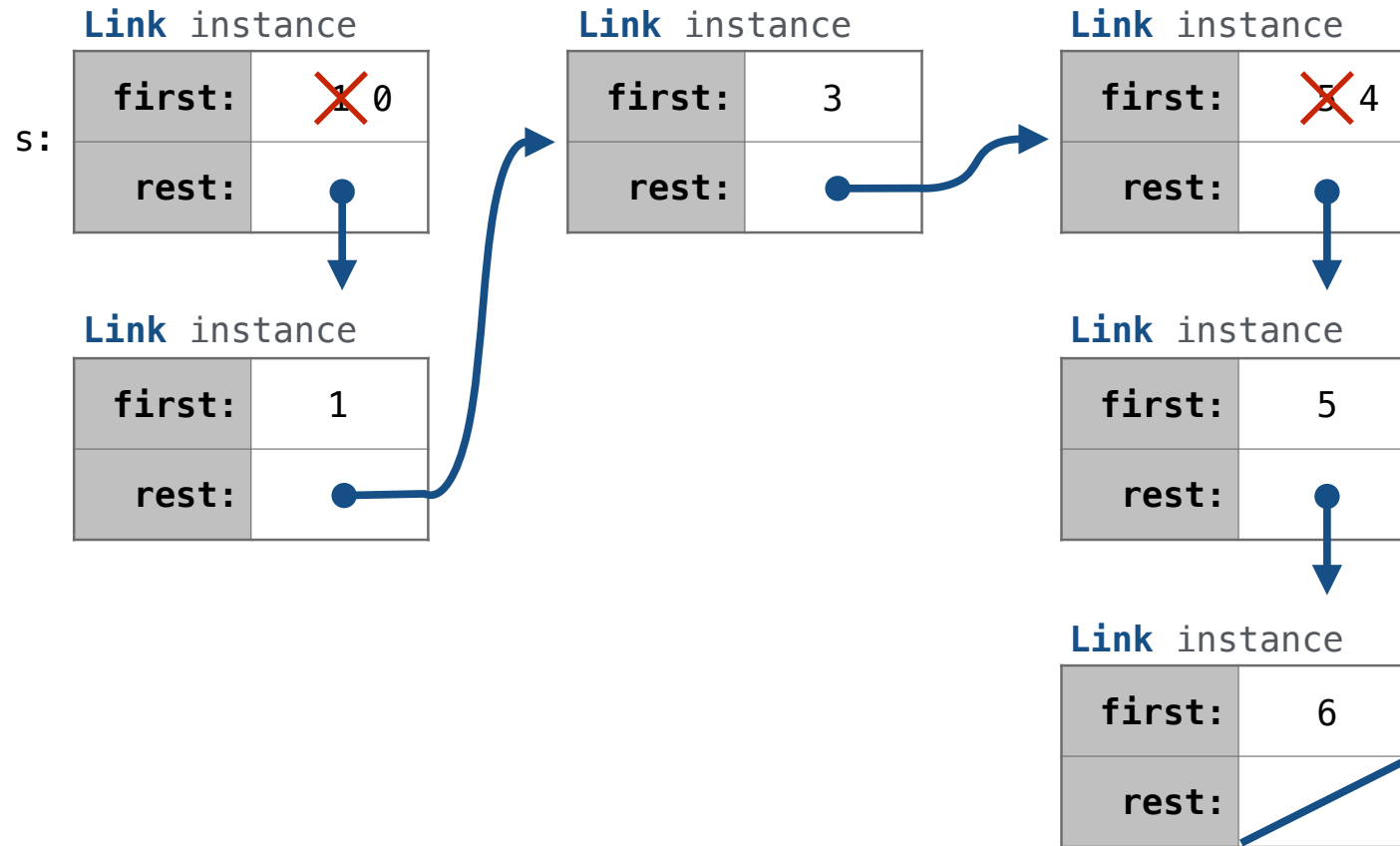
Adding to a Set Represented as an Ordered List

```python
def add(s, v):
    """Add v to a set 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))))
    """
    if empty(s): return Link(v)

    if s.first > v:
        s.first, s.rest = _____ , _____
                              v              Link(s.first, s.rest)
    elif s.first < v and empty(s.rest):
        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: 3 — rest: ●
Link instance — first: ~~5~~ 4 — rest: ●

Link instance — first: 1 — rest: ●
Link instance — first: 5 — rest: ●

Link instance — first: 6 — rest: