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

```python
>>> s = {3, 2, 1, 4, 4}
>>> 3 in s
True
>>> len(s)
4
>>> s.union({1, 5})
{1, 2, 3, 4, 5}
>>> s.intersection({6, 5, 4, 3})
{3, 4}
>>> s
{1, 2, 3, 4}
```

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**

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

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

def contains(s, v):
    # Return whether set s contains value v.
    s = Link1, Link3, Link2)
    >>> contains(s, 2)
    True

def union(set1, set2):
    ...return extend_link(set1_not_set2, set2)
```

**Sets as Unordered Sequences**

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

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

def contains(s, v):
    # Return whether set s contains value v.
    s = Link1, Link3, Link2)
    >>> contains(s, 2)
    True

def union(set1, set2):
    ...return extend_link(set1_not_set2, set2)
```

Return a linked list containing all elements in set1_not_set2 followed by all elements in set2
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.

<table>
<thead>
<tr>
<th>Parts of the program that...</th>
<th>Assume that sets are...</th>
<th>Using...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use sets to contain values</td>
<td>Unordered collections</td>
<td>empty, contains, adjoin, intersect, union</td>
</tr>
</tbody>
</table>

| Implement set operations     | Ordered linked lists    | first, rest, <, >, == |

Different parts of a program may make different assumptions about data.
Adding to an Ordered List

```python
def add(s, v):
    """Add v to a set s and return s."
    if s:
        if s.first > v:
            s.first, s.rest = __________, ____________
        elif s.first < v and empty(s.rest):
            s.rest = ____________________________________________________________________
        elif s.first < v:
            __________________________________________________________________________
    return __________
```

```
assert not empty(s), "Cannot add to an empty set."
```

```python
def intersect(set1, set2):
    if empty(set1) or empty(set2):
        return Link.empty
    else:
        e1, e2 = set1.first, set2.first
        if e1 == e2:
            return Link(e1, intersect(set1.rest, set2.rest))
        elif e1 < e2:
            return intersect(set1.rest, set2)
        elif e2 < e1:
            return intersect(set1, set2.rest)
```

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

```python
def intersect(set1, set2):
    if empty(set1) or empty(set2):
        return Link.empty
    else:
        e1, e2 = set1.first, set2.first
        if e1 == e2:
            return Link(e1, intersect(set1.rest, set2.rest))
        elif e1 < e2:
            return intersect(set1.rest, set2)
        elif e2 < e1:
            return intersect(set1, set2.rest)
```

Order of growth? $O(n)$ (Demo)