Sets

One more built-in Python container type
• Set literals are enclosed in braces
• Duplicate elements are removed on construction
• Sets are unordered, just like dictionary entries

```python
>>> s = {3, 2, 1, 4, 4}
>>> s
{1, 2, 3, 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}
```
Implementing Sets

The interface for sets

- **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
- **Adjunction**: Return a set with all elements in s and a value v

**Union**

```
 1  3

 4  5
```

```
 2  3
```

```
 1  2

 4  5  3
```

**Intersection**

```
 1  3

 4  5  3
```

```
 2

 3
```

**Adjunction**

```
 1  3

 4  5  3
```

```
 2

 3
```

```
 1  2

 4  3
```
Sets as Unordered Sequences

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

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

def set_contains(s, v):
    if empty(s):
        return False
    elif s.first == v:
        return True
    return set_contains(s.rest, v)
```

Demo
Review: Order of Growth

For a set operation that takes "linear" time, we say that

\( n \): size of the set

\( R(n) \): number of steps required to perform the operation

\[ R(n) = \Theta(n) \]

which means that there are constants \( k_1 \) and \( k_2 \) such that

\[ k_1 \cdot n \leq R(n) \leq k_2 \cdot n \]

for sufficiently large values of \( n \).
Sets as Unordered Sequences

```python
def adjoin_set(s, v):
    if set_contains(s, v):
        return s
    return Rlist(v, s)

def intersect_set(set1, set2):
    f = lambda v: set_contains(set2, v)
    return filter_rlist(set1, f)

def union_set(set1, set2):
    f = lambda v: not set_contains(set2, v)
    set1_not_set2 = filter_rlist(set1, f)
    return extend_rlist(set1_not_set2, set2)
```

Time order of growth

- $\Theta(n)$: The size of the set

- $\Theta(n^2)$: The size of the larger set

Friday, October 21, 2011
Sets as Ordered Sequences

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

```python
def set_contains2(s, v):
    if empty(s) or s.first > v:
        return False
    elif s.first == v:
        return True
    return set_contains2(s.rest, v)
```

Order of growth? $\Theta(n)$
This algorithm assumes that elements are in order.

```python
def intersect_set2(set1, set2):
    if empty(set1) or empty(set2):
        return Rlist.empty
    e1, e2 = set1.first, set2.first
    if e1 == e2:
        rest = intersect_set2(set1.rest, set2.rest)
        return Rlist(e1, rest)
    elif e1 < e2:
        return intersect_set2(set1.rest, set2)
    elif e2 < e1:
        return intersect_set2(set1, set2.rest)
```

Demo

Order of growth? $\Theta(n)$
Tree Sets

Proposal 3: A set is represented as a Tree. Each entry is:
• Larger than all entries in its left branch and
• Smaller than all entries in its right branch
Membership in Tree Sets

Set membership tests traverse the tree
• The element is either in the left or right sub-branch
• By focusing on one branch, we reduce the set by about half

```python
def set_contains3(s, v):
    if s is None:
        return False
    elif s.entry == v:
        return True
    elif s.entry < v:
        return set_contains3(s.right, v)
    elif s.entry > v:
        return set_contains3(s.left, v)
```

If 9 is in the set, it is in this branch.
Adjoining to a Tree Set

Right!  Left!  Right!  Stop!

Demo

Friday, October 21, 2011
What Did I Leave Out?

Sets as ordered sequences:
• Adjoining an element to a set
• Union of two sets

Sets as binary trees:
• Intersection of two sets
• Union of two sets

That's homework 8!

No lecture on Monday
Midterm 2 on Monday, 7pm–9pm
Good luck!