Sets in Python

- A type of collection (as are lists and tuples).
- Main differences from a list:
  - Unordered collection:
    - not indexed by number
    - printing / looping over set gives elements in no particular order
  - Collection of distinct items:
    - The same element can only appear once.
- Analogous to sets in mathematics.

Why use sets? Example.

- We have a set containing names of measurement sites which have temperature data.
- We don't care about the ordering ("site number 2" is not meaningful).
- Set operations are meaningful here, for example:
  - We also have a set containing sites with wind data.
  - Union: set of sites with *either* wind *or* temperature.
  - Intersection: sites with *both* wind *and* temperature.

How to construct sets in python

- Using `{...}` from specified items, e.g.: `{2, 3, 4}`
- Using `set(...)` from anything you can loop over, e.g.
  - `set([0, 1, 2, 3])`
  - `set('fred')` ← *loop over characters*
  - but not: `set(0, 1, 2, 3)` ← *needs 1 thing to loop over*
- For an empty set, use: `set()`
  - because `{}` means something else
- In Python 2.6 and earlier, `{...}` doesn't exist.
  Use the `set(...)` way instead

Sets are mutable

```
>>> a = {10, 11, 12}

>>> a.add(13)

>>> a.remove(11)

>>> print a
set([10, 12, 13])    ← NB not ordered

>>> a.clear()  ← remove all items
```

Find unique items in a collection

```
letters = set()
for char in 'ichthyosaur':
    letters.add(char)
print letters

set(['a', 'c', 'i', 'h', 'o', 's', 'r',
'u', 't', 'y'])
```

note h only appears once, and no particular order

- *or simply:*

```
letters = set('ichthyosaur')
```

Set operations

- `len(a)` gives the number of elements

- Many operations on two sets exist
  - comparisons (returning True or False)
  - combinations (returning another set)
  - many *operators* have equivalent *methods*
  - see following slides

Set comparisons

- return True or False

`a <= b`	`a.issubset(b)`
`a >= b`	`a.issuperset(b)`
`a < b`	*strict subset*
`a > b`	*strict superset*
`a == b`	*identical*

Set combinations

- these return another set

```
a = { 2, 3 }
b = { 3, 4 }

a | b      a.union(b)                    {2, 3, 4}

a & b      a.intersection(b)             {3}

a - b      a.difference(b)               {2}

a ^ b      a.symmetric_difference(b)     {2, 4}
```

Set operators vs methods

- operators act on two sets
- the equivalent methods act on anything you can loop over

```
set1 = { 2, 3 }
set2 = { 3, 4 }
tup = ( 3, 4 )
```

`set1 - set2` *gives* `set([2])`
but `set1 - tup` *fails with a* `TypeError`

`set1.difference(tup)` *gives* `set([2])`