

Python

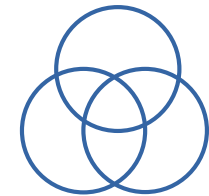
Sets

Thanks to all contributors:

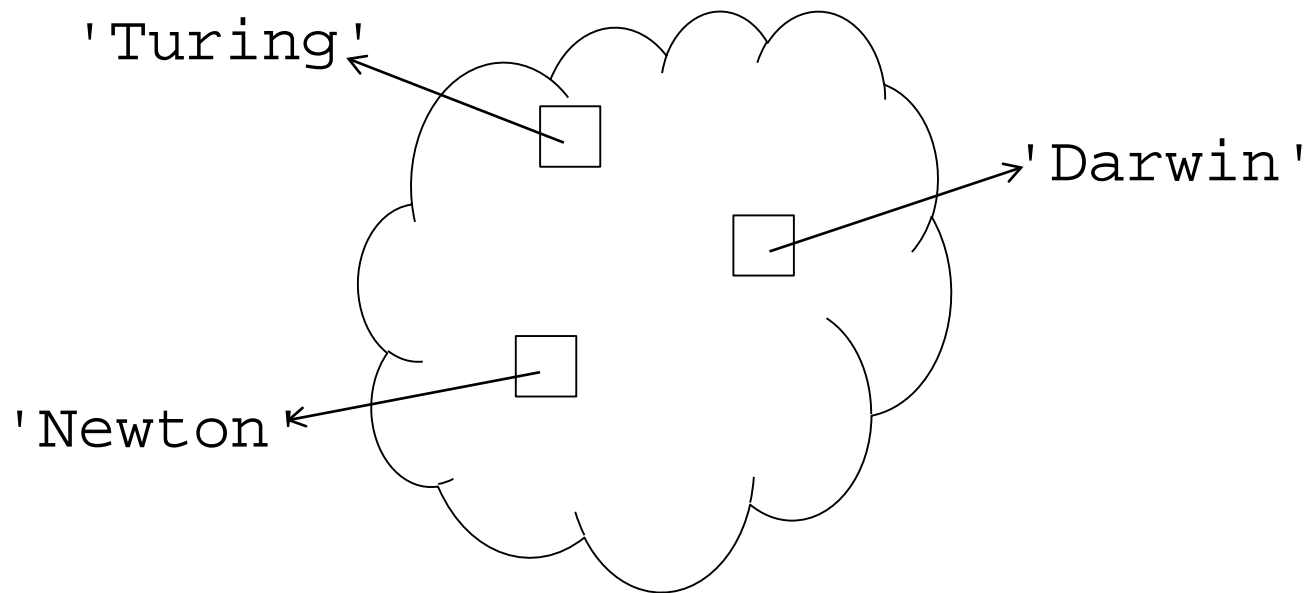
Ag Stephens, Alan Iwi and Tommy Godfrey.

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.



Note: entries are *not* in any particular order



Why use sets? An example.

- Suppose we have meteorological data at various measurement sites.
- We want to ask questions such as:
 - which sites have both wind **and** temperature data?
 - which sites have either wind **or** temperature data?
- We can store information in sets, e.g.:
 - the set of sites that have wind data
 - the set of sites that have temperature data
- Answer these questions intuitively and efficiently using Python set operations like **intersection** or **union**.

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

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
 - combinations
 - 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 combiners

- returning a new set

`a = { 2, 3 }`

`b = { 3, 4 }`

<code>a b</code>	<code>a.union(b)</code>	<code>{ 2, 3, 4 }</code>
<code>a & b</code>	<code>a.intersection(b)</code>	<code>{ 3 }</code>
<code>a - b</code>	<code>a.difference(b)</code>	<code>{ 2 }</code>
<code>a ^ b</code>	<code>a.symmetric_difference(b)</code>	<code>{ 2, 4 }</code>

Set operators vs methods

- operators will **ONLY** work on two sets
- equivalent methods will work with anything you can loop over

```
set1 = { 2, 3 }
```

```
set2 = { 3, 4 }
```

```
print(set1 - set2)
```

```
{2}
```

```
tup = ( 3, 4 )
```

```
print(set1 - tup)
```

```
TypeError
```

```
print(set1.difference(tup))
```

```
{2}
```

