THE operator MODULE

Functional Equivalents to Operators

In the last lecture we wrote code such as:

```
l = [2, 3, 4]

reduce(lambda a, b: a * b, l)
```

We used a lambda expression to create a functional version of the `*` operator

This is something that happens quite often, so the `operator` module was created

This module is a convenience module.

You can always use your own functions and lambda expressions instead.

The operator module

Arithmetic Functions

```
add(a, b)

mul(a, b)

pow(a, b)

mod(a, b)

floordiv(a, b)

neg(a)
```

and many more…

Comparison and Boolean Operators

```
lt(a, b)      gt(a, b)      eq(a, b)

le(a, b)      ge(a, b)      ne(a, b)

is_(a,b)      is_not(a,b)


and_(a, b)

or_(a,b)

not_(a,b)
```

Sequence/Mapping Operators

```
concat(s1, s2)

contains(s, val)

countOf(s, val)

getitem(s, i)

setitem(s, i, val)

delitem(s, i)
```

mutable objects

variants that use slices

Item Getters

The `itemgetter` function returns a callable

`getitem(s, i)` takes two parameters, and returns a value: `s[i]`

```
s = [1, 2, 3]
getitem(s, 1)   → 2
```

`itemgetter(i)` returns a callable which takes one parameter: a sequence object

```
f = itemgetter(1)

s = [1, 2, 3]
f(s)            → 2

s = 'python'
f(s)            → 'y'
```

Item Getters

We can pass more than one index to `itemgetter`:

```
l = [1, 2, 3, 4, 5, 6]
s = 'python'

f = itemgetter(1, 3, 4)

f(l)   → (2, 4, 5)

f(s)   → ('y', 'h', 'o')
```

Attribute Getters

The `attrgetter` function is similar to `itemgetter`, but is used to retrieve object attributes

It also returns a callable, that takes the object as an argument

Suppose `my_obj` is an object with three properties:

```
my_obj.a → 10
my_obj.b → 20
my_obj.c → 30
```

```
f = attrgetter('a')          f(my_obj)      → 10
```

```
f = attrgetter('a', 'c')     f(my_obj)      → (10, 30)
```

Can also call directly:

```
attrgetter('a', 'b', 'c')(my_obj)    → (10, 20, 30)
```

Calling another Callable

Consider the `str` class that provides the `upper()` method:

`s = 'python'` `s.upper()` → PYTHON

`f = attrgetter('upper')`

`f(s)` → returns the `upper` method of `s`
it is a callable, and can be called using `()`

`f(s)()` → PYTHON

`attrgetter('upper')(s)()` → PYTHON

Or, we can use the slightly simpler `methodcaller` function

`methodcaller('upper')('python')` → PYTHON

Basically, `methodcaller` retrieves the named attribute and calls it as well

It can also handle more arguments, as we'll in the code

Code