Operators in Python

Mathematical operators (like + and -) provided our first example of a method of combination, but we have yet to define an evaluation procedure for expressions that contain these operators.

Python expressions with infix operators each have their own evaluation procedures, but you can often think of them as short-hand for call expressions. When you see

```
>>> 2 + 3
5
```

simply consider it to be short-hand for

```
>>> add(2, 3)
5
```

Infix notation can be nested, just like call expressions. Python applies the normal mathematical rules of operator precedence, which dictate how to interpret a compound expression with multiple operators.

```
>>> 2 + 3 * 4 + 5
19
```

evaluates to the same result as

```
>>> add(add(2, mul(3, 4)), 5)
19
```

The nesting in the call expression is more explicit than the operator version, but also harder to read. Python also allows subexpression grouping with parentheses, to override the normal precedence rules or make the nested structure of an expression more explicit.

```
>>> (2 + 3) * (4 + 5)
45
```

evaluates to the same result as

```
>>> mul(add(2, 3), add(4, 5))
45
```

When it comes to division, Python provides two infix operators: `/` and `//`. The former is normal division, so that it results in a*floating point*, or decimal value, even if the divisor evenly divides the dividend:

```
>>> 5 / 4
1.25
>>> 8 / 4
2.0
```

The `//` operator, on the other hand, rounds the result down to an integer:

```
>>> 5 // 4
1
>>> -5 // 4
-2
```

These two operators are shorthand for the `truediv` and `floordiv` functions.

```
>>> from operator import truediv, floordiv
>>> truediv(5, 4)
1.25
>>> floordiv(5, 4)
1
```

You should feel free to use infix operators and parentheses in your programs. Idiomatic Python prefers operators over call expressions for simple mathematical operations.

Source : http://inst.eecs.berkeley.edu/~cs61A/book/chapters/functions.html#operators