Operators in Python

There are 7 kinds of operators in Python

1. Arithmetic
2. Assignment
3. Comparison
4. Logical
5. Identity
6. Membership
7. Bitwise

These operators do different things with different data types.

Arithmetic operators

+	Addition	x + y
-	subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor (integer) division	x // y

We usually use these with numbers.

Some examples:

integers: 1 + 1 ⇒ 2 ← integer

integer + float: 1 + 1.0 ⇒ 2.0 ← float

1 / 0 ⇒ Zero Division Error

integers: 1 * 0 ⇒ 0 integer

integer * float: 1 * 0.0 ⇒ 0.0 float

1 * x 0 ⇒ 1

1 ** 0.0 ⇒ 1.0

1 % 1 ⇒ 0

1 % 1.0 ⇒ 0.0

BUT . . .

integers 1 / 1 ⇒ 1.0 float!

In Python 3 division is float division, even with integers

Use floor division to do integer division.

integers 1 // 1 ⇒ 1 ← integer

The arithmetic operators sometimes work on other data types too.

Strings

`'a' + 'b'` ⇒ `'ab'` concatenate strings

`2 * 'a'` ⇒ `'aa'` repeat strings

integer → `2`

But these will all lead to errors

`'ab' - 'a'`

float → `2.0 * 'a'`

`'a' / 'b'`

`'a' ** 2`

None of these are allowed because they either don't make sense, or could have ambiguous meaning.

Lists and tuples

Like strings and tuples these are supported on lists

`[1, 2] + [3]` ⇒ `[1, 2, 3]` Append

`2 * [1, 2]` ⇒ `[1, 2, 1, 2]` Repeat

$(1, 2) + (3,) \Rightarrow (1, 2, 3)$

$(1, 2) * 2 \Rightarrow (1, 2, 1, 2)$

this can go before or after, the order of multiplication is not important.

Also like strings none of the other arithmetic operators are defined

Assignment operators

An assignment operator is used to assign a value to or to modify a variable.

The main operator is =

as in:

```
var = value
```

This assigns value to var.

```
a = 1

b = '5'

c = 3.0

d = 1 + 2.0    =>    d = 3.0
```

You can use an expression on the right hand side.

Self-assignment operators

Say you want to increment a variable by 1. A way to do this is:

```
a = a + 1
```

There are several self-assignment operators for tasks like this.

```
a += 1   ⇒   a = a + 1
a -= 1   ⇒   a = a - 1
a *= 2   ⇒   a = a * 2
a /= 2   ⇒   a = a / 2
a %= 2   ⇒   a = a % 2
a //=2   ⇒   a = a // 2
```

Comparison operators

we sometimes need to compare things, like are they equal, is one thing bigger or smaller than another thing?

These operators evaluate to True or False.

`x == y`	is x equal to y
`x != y`	is x not equal to y
`x > y`	is x greater than y
`x < y`	is x less than y
`x >= y`	is x greater than or equal to y
`x <= y`	is x less than or equal to y.

Caveat: Be very cautious using equality for floats, because they are not exact.

These operators also work on strings

e.g. `'a' < 'b'` is True because a comes before b.

These operators may also work on other data types.

Logical operators

The logical operators are:

and : returns True if both arguments are True

or : returns True if one argument is True

not : returns the logical opposite of the argument

Suppose
$x = 3$

then

$x > 0$ and $x < 5 \Rightarrow$ True

$x > 0$ or $x < 1 \Rightarrow$ True

not $x < 0 \Rightarrow$ True

These tables may help explain how and/or work.

and

	True	False
True	True	False
False	False	False

or

	T	F
T	T	T
F	T	F

Identity operators

There are only two identity operators

is — returns True if two things are the same object.

is not — returns True if two things are not the same object.

This is most often used as:

```
if x is None:
    do something.
```

Membership operators

There are two membership operators

in — returns True if an object is in a sequence

not in — returns True if an object is not in a sequence.

Some examples

0 in [1, 0, 2] => True

3 in [1, 0, 2] => False

for dictionaries membership is on the keys.

1 in {0: 1} => False

0 in {0: 1} => True

You can also test strings.

'ab' in 'baba' => True