PYTHON OPERATORS

We will briefly take a look at the operators and their usage.

Note that you can evaluate the expressions given in the examples using the interpreter interactively. For example, to test the expression `2 + 3`, use the interactive Python interpreter prompt:

```
>>> 2 + 3
5
>>> 3 * 5
15
>>>
```

Here is a quick overview of the available operators:

`+` (plus)

Adds two objects

`3 + 5` gives `8`. `'a' + 'b'` gives `'ab'`.

- (minus)

Gives the subtraction of one number from the other; if the first operand is absent it is assumed to be zero.

-5.2 gives a negative number and 50 - 24 gives 26.

*** (multiply)**

Gives the multiplication of the two numbers or returns the string repeated that many times.

2 * 3 gives 6. 'la' * 3 gives 'lalala'.

**** (power)**

Returns x to the power of y

3 ** 4 gives 81 (i.e. 3 * 3 * 3 * 3)

/ (divide)

Divide x by y

13 / 3 gives 4. 13.0 / 3 gives 4.333333333333333

`%` **(modulo)**

Returns the remainder of the division

`13 % 3` gives `1`. `-25.5 % 2.25` gives `1.5`.

`<<` **(left shift)**

Shifts the bits of the number to the left by the number of bits specified. (Each number is represented in memory by bits or binary digits i.e. 0 and 1)

`2 << 2` gives `8`. `2` is represented by `10` in bits.

Left shifting by 2 bits gives `1000` which represents the decimal `8`.

`>>` **(right shift)**

Shifts the bits of the number to the right by the number of bits specified.

`11 >> 1` gives `5`.

`11` is represented in bits by `1011` which when right shifted by 1 bit gives `101`which is the decimal `5`.

`&` **(bit-wise AND)**

Bit-wise AND of the numbers

5 & 3 gives 1.

| (bit-wise OR)

Bitwise OR of the numbers

5 | 3 gives 7

^ (bit-wise XOR)

Bitwise XOR of the numbers

5 ^ 3 gives 6

~ (bit-wise invert)

The bit-wise inversion of x is -(x+1)

~5 gives -6.

< (less than)

Returns whether x is less than y. All comparison operators return True or False. Note the capitalization of these names.

5 < 3 gives False and 3 < 5 gives True.

Comparisons can be chained arbitrarily: `3 < 5 < 7` gives `True`.

`>` **(greater than)**

Returns whether x is greater than y

`5 > 3` returns `True`. If both operands are numbers, they are first converted to a common type. Otherwise, it always returns `False`.

`⇐` **(less than or equal to)**

Returns whether x is less than or equal to y

`x = 3; y = 6; x ⇐ y` returns `True`.

`>=` **(greater than or equal to)**

Returns whether x is greater than or equal to y

`x = 4; y = 3; x >= 3` returns `True`.

`==` **(equal to)**

Compares if the objects are equal

`x = 2; y = 2; x == y` returns `True`.

x = 'str'; y = 'stR'; x == y returns False.

x = 'str'; y = 'str'; x == y returns True.

!= **(not equal to)**

Compares if the objects are not equal

x = 2; y = 3; x != y returns True.

not **(boolean NOT)**

If x is True, it returns False. If x is False, it returns True.

x = True; not x returns False.

and **(boolean AND)**

x and y returns False if x is False, else it returns evaluation of y

x = False; y = True; x and y returns False since x is False. In this case, Python will not evaluate y since it knows that the left hand side of the 'and' expression is False which implies that the whole expression will be False irrespective of the other values. This is called short-circuit evaluation.

`or` **(boolean OR)**

If x is `True`, it returns True, else it returns evaluation of y

`x = True; y = False; x or y` returns `True`. Short-circuit evaluation applies here as well.

Shortcut for math operation and assignment

It is common to run a math operation on a variable and then assign the result of the operation back to the variable, hence there is a shortcut for such expressions:

```
a = 2
a = a * 3
```

can be written as:

```
a = 2
a *= 3
```

Notice that `var = var operation expression` becomes `var operation= expression`.

Source: http://www.swaroopch.com/notes/python/