EVALUATION ORDER AND EXPRESSIONS

If you had an expression such as `2 + 3 * 4`, is the addition done first or the multiplication? Our high school maths tells us that the multiplication should be done first. This means that the multiplication operator has higher precedence than the addition operator.

The following table gives the precedence table for Python, from the lowest precedence (least binding) to the highest precedence (most binding). This means that in a given expression, Python will first evaluate the operators and expressions lower in the table before the ones listed higher in the table.

The following table, taken from the Python reference manual, is provided for the sake of completeness. It is far better to use parentheses to group operators and operands appropriately in order to explicitly specify the precedence. This makes the program more readable. See Changing the Order of Evaluation below for details.

`lambda`

Lambda Expression

if - else

Conditional expression

or

Boolean OR

and

Boolean AND

not x

Boolean NOT

in, not in, is, is not, <, ⇐, >, >=, !=, ==

Comparisons, including membership tests and identity tests

|

Bitwise OR

^

Bitwise XOR

&

Bitwise AND

<<, >>

Shifts

+, -

Addition and subtraction

*, /, //, %

Multiplication, Division, Floor Division and Remainder

+x, -x, ~x

Positive, Negative, bitwise NOT

**

Exponentiation	
	`x[index], x[index:index], x(arguments…), x.attribute`
Subscription, slicing, call, attribute reference	
	`(expressions…), [expressions…], {key: value…}, {expressions…}`
Binding or tuple display, list display, dictionary display, set display	

The operators which we have not already come across will be explained in later chapters.

Operators with the *same precedence* are listed in the same row in the above table. For example, `+` and `-` have the same precedence.

Changing the Order Of Evaluation

To make the expressions more readable, we can use parentheses. For example, `2 + (3 * 4)` is definitely easier to understand than `2 + 3 * 4` which requires knowledge of the operator precedences. As with everything else, the parentheses should be used reasonably (do not overdo it) and should not be redundant, as in `(2 + (3 * 4))`.

There is an additional advantage to using parentheses - it helps us to change the order of evaluation. For example, if you want addition to be evaluated before multiplication in an expression, then you can write something like `(2 + 3) * 4`.

Associativity

Operators are usually associated from left to right. This means that operators with the same precedence are evaluated in a left to right manner. For example, `2 + 3 + 4` is evaluated as `(2 + 3) + 4`. Some operators like assignment operators have right to left associativity i.e. `a = b = c` is treated as `a = (b = c)`.

Expressions

Example (save as expression.py):

```
length = 5
breadth = 2

area = length * breadth
print 'Area is', area
print 'Perimeter is', 2 * (length + breadth)
```

Output:

```
$ python expression.py

Area is 10

Perimeter is 14
```

How It Works

The length and breadth of the rectangle are stored in variables by the same name. We use these to calculate the area and perimeter of the rectangle with the help of expressions. We store the result of the expression `length * breadth` in the variable area and then print it using the print function. In the second case, we directly use the value of the expression `2 * (length + breadth)` in the print statement.

Also, notice how Python *pretty-prints* the output. Even though we have not specified a space between `'Area is'` and the variable `area`, Python puts it for us so that we get a clean nice output and the program is much more readable this way (since we don't need to worry about spacing in the strings we use for output). This is an example of how Python makes life easy for the programmer.

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