Operator Precedence and Associativity in Python Expressions

The following table lists the operators in decreasing order of precedence with their associativity.

S. No.	Operator	Description	Associativity
1	`{key: expr, …}`	Creating Dictionary	NA
2	`{expr, …}`	Set Creation	NA
3	`[expr, …]`	List Creation	NA
4	`(expr, …)`	Tuple Creation or Just Parenthesis	NA
5	`f(expr, …)`	Function Call	Left to Right
6	`x[index: index]`	Slicing	Left to Right
7	`x[index]`	Indexing	Left to Right
8	`x.attr`	Attribute Reference	Left to Right
9	`x ** y`	Exponentiation (*y raised to power x*)	Right to Left
10	`~x`	Bitwise NOT	NA
11	`+x, -x`	Unary Plus and Minus	NA
12	`x*y, x/y, x//y, x%y`	Multiplication, Division, Truncating Division and Remainder	Left to Right
13	`x+y, x-y`	Addition and Subtraction	Left to Right
14	`x<<y, x>>y`	Left Shift and Right Shift	Left to Right
15	`x&y`	Bitwise AND	Left to Right
16	`x^y`	Bitwise XOR	Left to Right
17	`x|y`	Bitwise OR	Left to Right
18	`x<y, x<=y, x>y, x>=y, x==y, x!=y, x<>y`(v2 Only)	Comparisons (less than, less than or equal, greater than, greater than or equal, equality, inequality.	NA
19	`x is y, x is not y`	Identity Tests	NA
20	`x in y, x not in y`	Membership Tests	NA
21	`not x`	Boolean NOT	NA
22	`x and y`	Boolean AND	Left to Right
23	`x or y`	Boolean OR	Left to Right
24	`x if expr else y`	Ternary Operator	NA
25	`lambda arg, …: expr`	Anonymous Simple Function	NA
26	`yield x`	Generator Function Send Protocol	NA

Source:
Python in a Nutshell, Alex Martelli, Anna Ravenscroft & Steve Holden, 3rd Edition

Version Differences

Some notes about version differences related to operators mentioned above.

1. In `Python 2.X`, value inequality can be written as either `X != Y` or `X <> Y`. In `Python 3.X`, the latter of these options is removed because it is redundant. In either version, best practice is to use `X != Y` for all value inequality tests.
2. In `Python 2.X`, a backquotes expression `` `X` `` works the same as `repr(X)` and converts objects to display `strings`. Due to its obscurity, this `expression` is removed in `Python 3.X`; use the more readable `str` and `repr` built-in functions, described in "Numeric Display Formats."
3. The `X // Y` floor division expression always truncates fractional remainders in both `Python 2.X` and `3.X`. The `X / Y` expression performs true division in `3.X` (retaining remainders) and classic division in `2.X` (truncating for integers).
4. The syntax `[...]` is used for both `list` literals and `list` comprehension `expressions`. The latter of these performs an implied loop and collects `expression` results in a new list.
5. The syntax `(...)` is used for `tuples` and `expression` grouping, as well as generator expressions—a form of list comprehension that produces results on demand, instead of building a result `list`. The parentheses may sometimes be omitted in all three contexts.
6. The syntax `{...}` is used for dictionary literals, and in `Python 3.X` and `2.7` for set literals and both dictionary and set comprehensions.
7. The `yield` and ternary `if/else` selection expressions are available in `Python 2.5` and later. The former returns send`(...)` arguments in `generators`; the latter is shorthand for a multiline if statement. `yield` requires parentheses if not alone on the right side of an assignment statement.
8. Comparison operators may be chained: `X < Y < Z` produces the same result as `X < Y` and `Y < Z`.
9. In recent `Pythons`, the slice expression `X[I:J:K]` is equivalent to indexing with a slice object: `X[slice(I, J, K)]`.
10. In `Python 2.X`, magnitude comparisons of mixed types are allowed, and convert numbers to a common type, and order other mixed types according to type names. In `Python 3.X`, nonnumeric mixed-type magnitude comparisons are not allowed and raise exceptions; this includes sorts by proxy.
11. Magnitude comparisons for dictionaries are also no longer supported in `Python 3.X` (though equality tests are); comparing `sorted(aDict.items())` is one possible replacement.

Source:
Learning Python, Mark Lutz, 5th Edition