notation and operator symbols. For instance, to add two numbers `X` and `Y` you would say `X + Y`, which tells Python to apply the `+` operator to the values named by `X` and `Y`. The result of the expression is the sum of `X` and `Y`, another number object.

Table 5-2 lists all the operator expressions available in Python. Many are self-explanatory; for instance, the usual mathematical operators (`+`, `-`, `*`, `/`, and so on) are supported. A few will be familiar if you've used other languages in the past: `%` computes a division remainder, `<<` performs a bitwise left-shift, `&` computes a bitwise AND result, and so on. Others are more Python-specific, and not all are numeric in nature: for example, the `is` operator tests object identity (i.e., address in memory, a strict form of equality), and `lambda` creates unnamed functions.

Table 5-2. Python expression operators and precedence

Operators	Description
`yield x`	Generator function `send` protocol
`lambda args: expression`	Anonymous function generation
`x if y else z`	Ternary selection (`x` is evaluated only if `y` is true)
`x or y`	Logical OR (`y` is evaluated only if `x` is false)
`x and y`	Logical AND (`y` is evaluated only if `x` is true)
`not x`	Logical negation

`x in y, x not in y`
`x is y, x is not y`
`x < y, x <= y, x > y, x >= y`
`x == y, x != y`
	Membership (iterables, sets)
	Object identity tests
	Magnitude comparison, set subset and superset
	Value equality operators

`x | y`	Bitwise OR, set union
`x ^ y`	Bitwise XOR, set symmetric difference
`x & y`	Bitwise AND, set intersection
`x << y, x >> y`	Shift `x` left or right by `y` bits
`x + y` `x - y`	Addition, concatenation Subtraction, set difference

`x * y`
`x % y`
`x / y, x // y`
	Multiplication, repetition
	Remainder, format
	Division: true and floor

`-x, +x` `~x`	Negation, identity Bitwise NOT (inversion)
`x ** y`	Power (exponentiation)

Operators	Description

`x[i]`
`x[i:j:k]`
`x(...)`
`x.attr`
	Indexing (sequence, mapping, others)
	Slicing
	Call (function, method, class, other callable)
	Attribute reference

`(...)`
`[...]`
`{...}`
	Tuple, expression, generator expression
	List, list comprehension
	Dictionary, set, set and dictionary comprehensions

Since this book addresses both Python 2.X and 3.X, here are some notes about version differences and recent additions related to the operators in Table 5-2:

- 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.
- 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."
- 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). See "Division: Classic, Floor, and True" on page 151.
- 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. See Chapter 4, Chapter 14, and Chapter 20 for examples.
- 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. See Chapter 4 and Chapter 20 for examples. The parentheses may sometimes be omitted in all three contexts. When a tuple's parentheses are omitted, the *comma* separating its items acts like a lowest-precedence operator if not otherwise significant.
- The syntax `{...}` is used for dictionary literals, and in Python 3.X and 2.7 for set literals and both dictionary and set comprehensions. See the set coverage in this chapter as well as Chapter 4, Chapter 8, Chapter 14, and Chapter 20 for examples.
- 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.