# OPERATORS, VARIABLES, ASSIGNMENT

## OPERATORS:

The standard mathematical operators that you are familiar with work the same way in Python as in most other languages.

+ - * / // % **

Addition, subtraction, multiplication, division, and modulus (remainder) are all part of the standard set of operators. Python has two division operators, a single slash character for classic division and a double slash for "floor" division (rounds down to nearest whole number). Classic division means that if the operands are both integers, it will perform floor division, while for floating point numbers, it represents true division. If true division is enabled, then the division operator will always perform that operation, regardless of operand types.

There is also an exponentiation operator, the double star/asterisk ( ** ). Although we are emphasizing the mathematical nature of these operators, please note that some of these operators are overloaded for use with other data types as well, for example, strings and lists. Let us look at an example:

```
>>> print -2 * 4 + 3 ** 2
1
```

As you can see, the operator precedence is what you expect: + and - are at the bottom, followed by *, /, //, and %; then comes the unary + and - , and finally, we have ** at the top. ((3 ** 2) is calculated first, followed by (-2 * 4), then both results are summed together.)

Python also provides the standard comparison operators, which return a Boolean value indicating the truthfulness of the expression:

< <= > >= == != <>

Trying out some of the comparison operators we get:

```
>>> 2 < 4
True
>>> 2 == 4
False
>>> 2 > 4
False
>>> 6.2 <= 6
False
>>> 6.2 <= 6.2
True
>>> 6.2 <= 6.20001
True
```

Python currently supports two "not equal" comparison operators, != and <>. These are the C-style and ABC/Pascal-style notations. The latter is slowly being phased out, so we recommend against its use. Python also provides the expression conjunction operators:

```
And   or   not
```

We can use these operations to chain together arbitrary expressions and logically combine the Boolean results:

```
>>> 2 < 4 and 2 == 4
False
>>> 2 > 4 or 2 < 4
True
>>> not 6.2 <= 6
True
```

```
>>> 3 < 4 < 5
True
```

The last example is an expression that may be invalid in other languages, but in Python it is really a short way of saying: >>> 3 < 4 and 4 < 5.

## VARIABLES AND ASSIGNMENT

Rules for variables in Python are the same as they are in most other high-level languages inspired by (or more likely, written in) C. They are simply identifier names with an alphabetic first character "alphabetic" meaning upper-or lowercase letters, including the underscore ( _ ). Any additional characters may be alphanumeric or underscore. Python is case-sensitive, meaning that the identifier "cAsE" is different from "CaSe."

Python is dynamically typed, meaning that no pre-declaration of a variable or its type is necessary. The type (and value) are initialized on assignment. Assignments are performed using the equal sign.

```
>>> counter = 0
>>> miles = 1000.0
>>> name = 'Bob'
>>> counter = counter + 1
>>> kilometers = 1.609 * miles
>>> print '%f miles is the same as %f km' % (miles, kilometers)
1000.000000 miles is the same as 1609.000000 km
```

We have presented five examples of variable assignment. The first is an integer assignment followed by one each for floating point numbers, one for strings, an increment statement for integers, and finally, a floatingpoint operation and assignment.

Python also supports augmented assignment, statements that both refer to and assign values to variables. You can take the following expression ...

```
n = n * 10
```

...and use this shortcut instead:

```
n *= 10
```

Python does not support increment and decrement operators like the ones in C: n++ or --n. Because + and -- are also unary operators, Python will interpret --n as -(-n) == n, and the same is true for ++n.