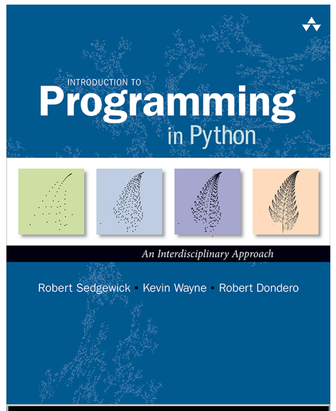

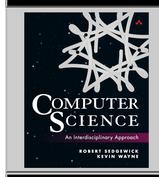

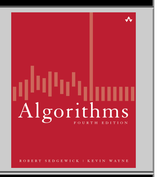

Appendix A: Python Operator Precedence

Python has well-defined rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators. For example, multiplication and division have a higher precedence than addition and subtraction. Precedence rules can be overridden by explicit parentheses.

Precedence Order

When two operators share an operand, the operator with the higher *precedence* goes first. For example, since multiplication has a higher precedence than addition, `a + b * c` is treated as `a + (b * c)`, and `a * b + c` is treated as `(a * b) + c`.

Associativity

When two operators share an operand and the operators have the same precedence, then the expression is evaluated according to the *associativity* of the operators. For example, since the `**` operator has right-to-left associativity, `a ** b ** c` is treated as `a ** (b ** c)`. On the other hand, since the `/` operator has left-to-right associativity, `a / b / c` is treated as `(a / b) / c`.

Precedence and Associativity of Python Operators

The Python documentation on operator precedence contains a table that shows all Python operators from lowest to highest precedence, and notes their associativity. Most programmers do not memorize them all, and those that do still use parentheses for clarity.

Order of Evaluation

In Python, the left operand is always evaluated before the right operand. That also applies to function arguments.

Python uses *short circuiting* when evaluating expressions involving the `and` or `or` operators. When using those operators, Python does not evaluate the second operand unless it is necessary to resolve the result. That allows statements such as `if (s != None) and (len(s) < 10): ...` to work reliably.

Precedence Order Gone Awry

Sometimes the precedence order defined in a language do not conform with mathematical norms. For example, in Microsoft Excel, -a^b is interpreted as (-a)^b instead of -(a^b). So -1^2 is equal to 1 instead of -1, which is the values most mathematicians would expect. Microsoft acknowledges this quirk as a "design choice". One wonders whether the programmer was relying on the C precedence order in which unary operators have higher precedence than binary operators. This rule agrees with mathematical conventions for all C operators, but fails with the addition of the exponentiation operator. Once the order was established in Microsoft Excel 2.0, it could not easily be changed without breaking backward compatibility.

Exercises

1. Add parentheses to the following expression to make the order of evaluation more clear.

```
year % 4 == 0 and year % 100 != 0 or year % 400 == 0
```

Answer: leapyearvariety.py shows a variety of equivalent expressions, including the following reasonable alternative.

```
((year % 4 == 0) and (year % 100 != 0)) or (year % 400 == 0)
```