

Chapter IV-1 — Working with Commands

For example:

```
wave1 = 10
```

sets each Y value of the wave1 equal to 10, whereas:

```
wave1 += 10
```

adds 10 to each Y value of wave1. This is equivalent to:

```
wave1 = wave1 + 10
```

The assignment operators `=`, `:=` and `+=` work with string assignment statements but `-=`, `*=` and `/=` do not. For example:

```
String str1; str1 = "Today is "; str1 += date(); Print str1
```

prints something like "Today is Fri, Mar 31, 2000".

For more information on the `:=` operator, see Chapter IV-9, **Dependencies**.

Operators

Here is a complete list of the operators that Igor supports in the expression part of an assignment statement in order of precedence:

Operator	Effect
<code>++</code> <code>--</code>	Prefix and postfix increment and decrement. Require Igor Pro 7 or later. Available only for local variables in user-defined functions.
<code>^</code> <code><<</code> <code>>></code>	Exponentiation, bitwise left shift, bitwise right shift. Shifts require Igor Pro 7 or later.
<code>-</code> <code>!</code> <code>~</code>	Negation, logical complement, bitwise complement.
<code>*</code> <code>/</code>	Multiplication, division.
<code>+</code> <code>-</code>	Addition or string concatenation, subtraction.
<code>==</code> <code>!=</code> <code>></code> <code><</code> <code>>=</code> <code><=</code>	Comparison operators.
<code>&</code> <code> </code> <code>%^</code>	Bitwise AND, bitwise OR, bitwise XOR.
<code>&&</code> <code> </code> <code>?</code> <code>:</code>	Logical AND, logical OR, conditional operator.
<code>\$</code>	Substitute following string expression as name.

Comparison operators do not work with NaN parameters because, by definition, NaN compared to anything, even another NaN, is false. Use `numtype` to test if a value is NaN.

Comparison operators, bitwise AND, bitwise OR, bitwise XOR associate right to left. Therefore `a==b>=c` means `(a==(b>=c))`. For example, `2==1>= 0` evaluates to 0, not 1.

All other binary operators associate left to right.

Aside from the precedence of common operators that everyone intuitively knows, it is useful to include parentheses to avoid confusion. You may know the precedence and associativity but someone reading your code may not.

Unary negation changes the sign of its operand. Logical complementation changes nonzero operands to zero and zero operands to 1. Bitwise complementation converts its operand to an unsigned integer by truncation and then changes it to its binary complement.

Exponentiation raises its left-hand operand to a power specified by its right-hand operand. That is, 3^2 is written as `3^2`. In an expression `a^b`, if the result is assigned to a real variable or wave, then `a` must not be negative if `b` is not an integer. If the result is used in a complex expression, any combination of negative `a`, fractional `b` or complex `a` or `b` is allowed.

If the exponent is an integer, Igor evaluates the expression using only multiplication. There is no need to write a^2 as $a*a$ to get efficient evaluation — Igor does the equivalent automatically. If, on the other hand, the exponent is not an integer then the evaluation is performed using logarithms, hence the restriction on negative a in a real expression.

Logical OR (`||`) and logical AND (`&&`) determine the truth or falseness of pairs of expressions. The AND operation returns true only when both expressions are true; OR will return true if *either* is true. As in C, true is *any* nonzero value, and false is zero. The operations are undefined for NaNs. These operators are not available in complex expressions.

The logical operators are evaluated from left to right, and an operand will not be evaluated if it is not necessary. For the example:

```
if(MyFunc1() && MyFunc2())
```

when `MyFunc1()` returns false (zero), then `MyFunc2()` will not be evaluated because the entire expression is already false. This can produce unexpected consequences when the right-hand expression has side effects, such as creating waves or setting global values.

Bitwise AND (`&`), OR (`|`), and XOR (`%^`) convert their operands to an unsigned integer by truncation and then return their binary AND, OR or exclusive OR.

Bitwise shifts, `<<` and `>>`, return the lefthand operand shifted by the specified number of bits. They require Igor Pro 7 or later. The lefthand operation is truncated to an integer before shifting.

The conditional operator (`? :`) is a shorthand form of an if-else-endif expression. In the statement:

```
<expression> ? <TRUE> : <FALSE>
```

the first operand, `<expression>`, is the test condition; if it is nonzero then Igor evaluates the `<TRUE>` operand; otherwise `<FALSE>` is evaluated. Only one operand is evaluated according to the test condition. This is the same as if you had written:

```
if( <expression> )
    <TRUE>
else
    <FALSE>
endif
```

The `“:”` character in the conditional operator must always be separated from the two adjacent operands with a space. If you omit either space, you will get an error (“No such data folder”) because the expression can also be interpreted as part of a data folder path. To be safe, always separate the operands from the operator symbols with a space.

The operands must be numeric; for strings, use the **SelectString** function. When using complex expressions with the conditional operator, only the real portion is used when the operator evaluates the expression.

The conditional operator can easily cause confusion, so you should exercise caution when using it. For example, it is unclear from simple inspection what Igor may return for

```
1 ? 2 : 3 ? 4 : 5
```

(4 in this case), whereas

```
1 ? 2 : (3 ? 4 : 5)
```

will return 2. Always use parentheses to remove any ambiguity.

The comparison operators return 1 if the result of the comparison is true or 0 if it is false. For example, the `==` operator returns 1 if its operands are equal or 0 if they are not equal. The `!=` operator returns the opposite. Because comparison operators return the values 1 or 0 they can be used in interesting ways. The assignment:

```
wave1 = sin(x)*(x<=50) + cos(x)*(x>50)
```

sets `wave1` so that it is a sine wave below $x=50$ and a cosine wave above $x=50$. See also **Example: Comparison Operators and Wave Synthesis** on page II-82.