

Numbers and Math

This page covers number conversion, number formatting, and the Math object.

Topics on this page:

- [Numbers in DG Script](#)
- [Converting values to numbers: Number, parseInt, parseNumber, isNaN](#)
- [numberFormat and number format patterns](#)
- [Math functions](#)
- [Number constants](#)

See also: [Strings](#), [Date and Time](#).

Numbers in DG Script

You can write numbers as integers (42), decimals (3.14), with an exponent (1.5e3), or in hexadecimal (0xFF).

There are no global NaN or Infinity constants. When you need them, use `Number("NaN")` and `Number("Infinity")`, or compute `1 / 0`.

When `+` has a number on the left, the right side is converted to a number, so `1 + "2"` is 3. See [Joining strings](#).

Tip

To check whether a value is a number, use `typeof`:

```
val = 123;  
if (typeof val == "number") return "It's Numeric!";
```

Converting Values to Numbers

Number(value)

Converts a value to a number and returns it.

- A string that contains only a number (for example "12.12", "-5", "1e3") is converted. Any other string, including "" and "12px", gives NaN.
- true gives 1 and false gives 0.
- null and other values give NaN.

```
print(Number("12.12"), "\n");  
print(Number("12px"), "\n");  
print(Number(true), "\n");  
return Number(null);
```

Print output:

```
12.12  
NaN  
1
```

Output: NaN

parseInt(value, radix)

Converts a value to a whole number by dropping the fraction. Unlike JavaScript, a value that cannot be converted gives **0**, not NaN, and trailing text is not allowed (parseInt("12px") is 0).

When you pass an integer radix, the value is read as a whole number in that base, such as 16 for hexadecimal.

```
print(parseInt("12.7"), "\n");  
print(parseInt(-3.9), "\n");  
print(parseInt("abc"), "\n");  
return parseInt("ff", 16);
```

Print output:

```
12  
-3  
0
```

Output: 255

parseFloat(str, default)

Finds the first number inside a string and returns it. Use it to read numbers that contain units, currency symbols, or thousands separators.

- Commas inside the number are removed, so "1,234.5" gives 1234.5.
- A string that starts with # or 0x is read as hexadecimal. The rest of the string must be valid hex digits, otherwise an error occurs.
- If no number is found, or the string is empty, the result is default (NaN when omitted).
- str must be a string. Passing a number returns NaN; use Number() for numbers.

```
print(parseFloat("$12.11"), "\n");  
print(parseFloat("Total: 1,234.5 kg"), "\n");  
print(parseFloat("#ff"), "\n");  
print(parseFloat("n/a"), "\n");  
return parseFloat("n/a", 0);
```

Print output:

```
12.11  
1234.5  
255  
NaN
```

Output: 0

isNaN(value)

Returns true only if the value is the number NaN. The value is not converted first, so any string, including "abc", returns false. This is different from JavaScript, where isNaN("abc") is true.

To check whether a string contains a valid number, convert it first: isNaN(Number(str)).

```
print(isNaN(Number("abc")), "\n");  
print(isNaN("abc"), "\n");  
print(isNaN("12"), "\n");  
return isNaN(null);
```

Print output:

```
true  
false  
false
```

Output: false

numberFormat(value, pattern, decimalSeparator, groupSeparator)

Formats a number as a string using a number format pattern.

- **value** — a number, or a string that `Number()` can convert. For any other value the result is an empty string.
- **pattern** — a number format pattern (see below).
- **decimalSeparator, groupSeparator** — optional. Use them to change the decimal and grouping characters, for example for European formats. You can pass only `decimalSeparator`.

If the pattern is invalid, the result is an empty string.

```
print(numberFormat(123.456789, "$#.00"), "\n");
print(numberFormat(1234567.891, "#,##0.00"), "\n");
print(numberFormat(5, "000"), "\n");
print(numberFormat(0.256, "0.0%"), "\n");
print(numberFormat("12.5", "0.00"), "\n");
print(numberFormat(-1234.5, "#,##0.00;(#,##0.00)"), "\n");
return numberFormat(1234.5, "#,##0.00", ",", ".");
```

Print output:

```
$123.46
1,234,567.89
005
25.6%
12.50
(1,234.50)
```

Output: 1.234,50

Number Format Patterns

Number format patterns use the ICU pattern syntax. The same patterns are used by the number formatting dataflow blocks and by bindings.

Character	Meaning
Zero (0)	A digit that is always shown, even if it is a non-significant zero

Character	Meaning
Number sign (#)	A digit that is omitted if it is a non-significant zero. Note that #.00 formats 0.5 as .50; use 0.00 to get 0.50
Decimal (.)	The decimal separator
Hyphen (-)	The minus sign
Comma (,)	The grouping separator. The grouping size is the number of digits between the last comma and the end of the integer part
The letter E	Separates the number and its exponent in scientific notation, for example 0.00E0
Plus sign (+)	Used after E to show a plus sign on positive exponents
Percent (%)	Multiplies the number by 100 and shows it as a percentage
Per mille (‰ or \u2030)	Multiplies the number by 1000 and shows it as per mille
Currency sign (¤ or \u00A4)	Replaced by the currency symbol
Single quotes (") Quote special characters so they appear literally, for example "'#'"	
Semicolon (;)	Separates the positive and negative patterns

Other characters, such as \$ or a space, are copied to the result as they are. The special pattern ,0.00 is treated as ,###.00.

Math Functions

Math functions accept only numbers. They do **not** convert strings: `Math.abs("-5")` is NaN. Convert with `Number()` first.

Function	Returns
<code>Math.abs(x)</code>	The absolute value of x
<code>Math.min(a, b, ...)</code>	The smallest argument. Arguments that are not numbers are ignored. With no numeric arguments the result is NaN
<code>Math.max(a, b, ...)</code>	The largest argument. Arguments that are not numbers are ignored. With no numeric arguments the result is NaN
<code>Math.sin(x)</code> , <code>Math.cos(x)</code> , <code>Math.tan(x)</code>	Sine, cosine, tangent of x in radians
<code>Math.asin(x)</code> , <code>Math.acos(x)</code> , <code>Math.atan(x)</code>	Arcsine, arccosine, arctangent, in radians
<code>Math.atan2(y, x)</code>	The angle in radians of the point (x, y)
<code>Math.ceil(x)</code>	x rounded up to an integer
<code>Math.floor(x)</code>	x rounded down to an integer
<code>Math.round(x)</code>	x rounded to the nearest integer. Halves are rounded away from zero : <code>Math.round(-2.5)</code> is -3 (JavaScript gives -2)

Function	Returns
<code>Math.exp(x)</code>	e raised to the power x
<code>Math.log(x)</code>	The natural (base-e) logarithm of x
<code>Math.sqrt(x)</code>	The square root of x
<code>Math.pow(x, y)</code>	x raised to the power y
<code>Math.random()</code>	A random number from 0 (inclusive) to 1 (exclusive)

`Math.ceil` and `Math.round` cause an error when the value is NaN or infinite. Check the value with `isNaN` first if it may not be a valid number. `Math.floor(NaN)` returns NaN.

```
print(Math.abs(-47), "\n");
print(Math.min(-2, -1, 0, 1, 2), " ", Math.max(-2, -1, 0, 1, 2), "\n");
print(Math.max(1, "5"), "\n");
print(Math.sin(Math.PI / 2), " ", Math.cos(Math.PI), "\n");
print(Math.atan2(1, 1) == Math.PI / 4, "\n");
print(Math.ceil(1.2), " ", Math.floor(1.7), " ", Math.round(1.5), " ",
Math.round(-2.5), "\n");
print(Math.sqrt(9), " ", Math.pow(2, 3), "\n");
return Math.log(10);
```

Print output:

```
47
-2 2
1
1 -1
true
2 1 2 -3
3 8
```

Output: 2.302585092994046

To round to a number of decimal places, multiply, round, and divide, or use `numberFormat` when you need a string:

```
x = 3.14159;
print(Math.round(x * 100) / 100, "\n");
return numberFormat(x, "0.00");
```

Print output:

```
3.14
```

Output: 3.14

Tip

The following example gets a random value within a range:

```
function getRandom(min, max) {  
    return Math.random() * (max - min) + min;  
}  
  
return getRandom(50, 75);
```

Number Constants

Constant	Value
Math.PI	3.141592653589793, the constant pi
Math.E	2.718281828459045, the constant e
Math.LN2	0.6931471805599453, the natural logarithm of two
Math.LN10	2.302585092994046, the natural logarithm of ten
Math.LOG2E	1.4426950408889634, the base-2 logarithm of e
Math.LOG10E	0.4342944819032518, the base-10 logarithm of e
Math.SQRT2	1.4142135623730951, the square root of two
Math.SQRT1_2	0.7071067811865476, the square root of one-half

The Math object is read-only. Assigning to it, for example `Math.PI = 3`, causes an error.

From:

<http://wiki.dglogik.com/> - **DGLogik**

Permanent link:

http://wiki.dglogik.com/dglux5_wiki:dgscript:numbers_math

Last update: **2026/10/22 17:28**

