# Arithmetic operators

The 18 operators of eZTemplateArithmeticOperator (sum, sub, mul, div, mod, inc, dec, min, max, abs, ceil, floor, round, int, float, count, roman, rand) with their parameters, input, return value and examples rendered on...

Source file: `doc/eng-us/help/template-operators/arithmetic-operators.md` Last changed 2026-11-04
https://doc.exponential.earth/learninminutes/exphelp/viewer/help/template-operators/arithmetic-operators.md

The template language has no infix arithmetic: `{1 + 2}` is not an expression. Every calculation is an operator call. All 18 arithmetic operators are registered by `eZTemplateArithmeticOperator` (`lib/eztemplate/classes/eztemplatearithmeticoperator.php`) through `lib/eztemplate/classes/eztemplateautoload.php`.

Most of them take their operands either from the piped input (`10|sum( 5 )`) or from the parameters (`sum( 10, 5 )`). Where an operator behaves differently, its section says so. The examples were rendered on a development installation by the template interpreter and then again by the template compiler; the compiled form of `sum`, `sub`, `mul` and `div` calls the same `eZTemplateArithmeticOperator::calculate()`. The one difference found, in `min`, is shown in its section.

## Contents

1. How operands are read
2. sum and sub
3. mul, div and mod
4. inc and dec
5. min and max
6. abs, ceil, floor, round, int and float
7. count
8. roman and rand
9. A worked example: number of pages
10. Related pages

## How operands are read

`sum`, `sub`, `mul` and `div` turn every operand into a number with `eZTemplateArithmeticOperator::numericOperand()`: integers and floats stay as they are, a numeric string becomes its number, a string that starts with a number becomes that number (`'7 days'` is 7), any other string and `null` become 0, a boolean becomes 0 or 1, an array becomes 0 when it is empty and 1 otherwise, and an object becomes 0. Only these four operators use that rule; the others use PHP's own conversion.

## sum and sub

### sum

Adds all operands: the piped input (when there is one) and every parameter.

| Parameter | Type | Required | Default |
|---|---|---|---|
| any number of values | number or numeric string | no | none |

Input: optional; when present it is the first operand. Returns an integer or a float. Generated reference: sum.

### sub

Takes every following operand from the first one. The first operand is the piped input, or the first parameter when nothing is piped.

| Parameter | Type | Required | Default |
|---|---|---|---|
| any number of values | number or numeric string | no | none |

Input: optional. Returns an integer or a float. Generated reference: sub.

**Template**

```
sum( 2, 3, 4 ) = {sum( 2, 3, 4 )}
10|sum( 5 ) = {10|sum( 5 )}
sub( 10, 3, 2 ) = {sub( 10, 3, 2 )}
'7 days'|sum( 1 ) = {'7 days'|sum( 1 )}
sum( 'abc', 2 ) = {sum( 'abc', 2 )}
```

Output:

```
sum( 2, 3, 4 ) = 9
10|sum( 5 ) = 15
sub( 10, 3, 2 ) = 5
'7 days'|sum( 1 ) = 8
sum( 'abc', 2 ) = 2
```

## mul, div and mod

### mul

Multiplies the input (when there is one) by every parameter. With no parameter at all it logs the warning "Requires at least 1 parameter value" and leaves the value unchanged.

| Parameter | Type | Required | Default |
|---|---|---|---|
| values | number | yes, at least one | none |

Input: optional. Returns an integer or a float. Generated reference: mul.

### div

Divides the first operand by each following one, in order. Dividing by zero gives 0 instead of an error. Needs at least one parameter, like `mul`.

| Parameter | Type | Required | Default |
|---|---|---|---|
| values | number | yes, at least one | none |

Input: optional; when present it is the dividend. Returns an integer or a float. Generated reference: div.

### mod

The remainder of an integer division (PHP `%`). With one parameter the input is the dividend and the parameter the divisor; with two parameters they are dividend and divisor and the input is ignored. Without a parameter it logs "Missing dividend and divisor". There is no zero check: test the divisor first.

| Parameter | Type | Required | Default |
|---|---|---|---|
| dividend or divisor | integer | yes | none |
| divisor | integer | no | none |

Returns an integer. Generated reference: mod.

**Template**

```
mul( 4, 2.5 ) = {mul( 4, 2.5 )}
div( 10, 4 ) = {div( 10, 4 )}
div( 10, 0 ) = {div( 10, 0 )}
100|div( 2, 5 ) = {100|div( 2, 5 )}
mod( 10, 4 ) = {mod( 10, 4 )}
10|mod( 3 ) = {10|mod( 3 )}
```

Output:

```
mul( 4, 2.5 ) = 10
div( 10, 4 ) = 2.5
div( 10, 0 ) = 0
100|div( 2, 5 ) = 10
mod( 10, 4 ) = 2
10|mod( 3 ) = 1
```

## inc and dec

### inc

Returns the value plus one: the piped input, or the named parameter `value` when nothing is piped. The variable itself is not changed; use `{set $i = $i|inc}` to store the result.

| Parameter | Type | Required | Default |
|---|---|---|---|
| value | mixed | no | false |

Generated reference: inc.

### dec

The same, minus one.

| Parameter | Type | Required | Default |
|---|---|---|---|
| value | mixed | no | false |

Generated reference: dec.

**Template**

```
{def $i = 5}
$i|inc = {$i|inc}, $i|dec = {$i|dec}, $i is still {$i}
inc( 9 ) = {inc( 9 )}
dec( 1 ) = {dec( 1 )}
```

Output:

```
$i|inc = 6, $i|dec = 4, $i is still 5
inc( 9 ) = 10
dec( 1 ) = 0
```

## min and max

### max

Returns the largest value. The piped input, when there is one, takes part in the comparison; otherwise the first parameter starts it. Needs at least one parameter.

| Parameter | Type | Required | Default |
|---|---|---|---|
| values | mixed | yes, at least one | none |

Generated reference: max.

### min

Returns the smallest value. Needs at least one parameter. Use it in the call form, `min( $a, $b )`: with a piped input the interpreter and the compiled template disagree. The interpreter ( `modify()`) compares only the parameters, so `7|min( 9, 8 )` is 8 there; the compiled template includes the input and gives 7.

| Parameter | Type | Required | Default |
|---|---|---|---|
| values | mixed | yes, at least one | none |

Generated reference: min.

**Template**

```
max( 3, 9, 4 ) = {max( 3, 9, 4 )}
min( 3, 9, 4 ) = {min( 3, 9, 4 )}
7|max( 3, 5 ) = {7|max( 3, 5 )}
7|min( 9, 8 ) = {7|min( 9, 8 )}
```

Output:

```
max( 3, 9, 4 ) = 9
min( 3, 9, 4 ) = 3
7|max( 3, 5 ) = 7
7|min( 9, 8 ) = 8
```

The same template compiled on a development installation (template compiling is enabled in `site.ini`, `[TemplateSettings] TemplateCompile`) printed `7|min( 9, 8 ) = 7` on the last line; the other lines were identical:

Output with the template compiler (compiled into a scratch folder):

```
max( 3, 9, 4 ) = 9
min( 3, 9, 4 ) = 3
7|max( 3, 5 ) = 7
7|min( 9, 8 ) = 7
```

## abs, ceil, floor, round, int and float

Each takes one value: the first parameter when there is one, else the piped input.

| Operator | Does | Returns |
|---|---|---|
| `abs` | PHP `abs()` | number |
| `ceil` | PHP `ceil()`, rounds up | float |
| `floor` | PHP `floor()`, rounds down | float |
| `round` | PHP `round()` with no precision: to a whole number, halves away from zero | float |
| `int` | PHP `(int)` cast: `'42abc'` is 42, `9.99` is 9 | integer |
| `float` | PHP `(float)` cast | float |

None of them has named parameters. `round` has no precision parameter; for two decimals use `mul( $x, 100 )|round|div( 100 )` or the locale operators (`l10n( 'number' )`). Generated references: abs, ceil, floor, round, int, float.

**Template**

```
-4.6|abs = {-4.6|abs}
4.2|ceil = {4.2|ceil}
4.8|floor = {4.8|floor}
4.5|round = {4.5|round}
round( 2.4 ) = {round( 2.4 )}
'42abc'|int = {'42abc'|int}
'3.75'|float = {'3.75'|float}
int( 9.99 ) = {int( 9.99 )}
```

Output:

```
-4.6|abs = 4.6
4.2|ceil = 5
4.8|floor = 4
4.5|round = 5
round( 2.4 ) = 2
'42abc'|int = 42
'3.75'|float = 3.75
int( 9.99 ) = 9
```

# count

## count

Counts the first parameter, or the piped input when there is no parameter:

- an array: its number of elements;
- an object with an `attributes()` method (nodes, content objects, attributes): the number of attribute names it offers to templates;
- a string: its length in bytes (`strlen`), so a multibyte character counts more than once;
- anything else: 0.

A second parameter logs an "extra parameters" warning.

| Parameter | Type | Required | Default |
|---|---|---|---|
| value | mixed | no | the input |

Generated reference: count.

**Template**

```
{def $node = fetch( 'content', 'node', hash( 'node_id', 89 ) )}
array( 1, 2, 3 )|count = {array( 1, 2, 3 )|count}
'Hello'|count = {'Hello'|count}
count( hash( 'a', 1, 'b', 2 ) ) = {count( hash( 'a', 1, 'b', 2 ) )}
$node.object.data_map|count = {$node.object.data_map|count}
$node|count (attributes of the node object) = {$node|count}
42|count = {42|count}
```

Output:

```
array( 1, 2, 3 )|count = 3
'Hello'|count = 5
count( hash( 'a', 1, 'b', 2 ) ) = 2
$node.object.data_map|count = 7
$node|count (attributes of the node object) = 50
42|count = 0
```

# roman and rand

## roman

Writes a positive integer as a Roman numeral, built by `buildRoman()`. The value is the named parameter `value` or the piped input. Zero and negative numbers give an empty string.

| Parameter | Type | Required | Default |
|---|---|---|---|
| value | mixed | no | `false` (use the input) |

Generated reference: roman.

### rand

With two parameters, a random integer between them, inclusive (PHP `mt_rand( min, max )`); with any other number of parameters, `mt_rand()` without bounds. Inside a cache-block or a view-cached template the value stays the same until the cache is rebuilt.

| Parameter | Type | Required | Default |
|---|---|---|---|
| minimum | integer | no | none |
| maximum | integer | no | none |

Generated reference: rand.

**Template**

```
{def $r = rand( 1, 6 )}
4|roman = {4|roman}
roman( 1999 ) = {roman( 1999 )}
2026|roman = {2026|roman}
rand( 1, 6 ) is between 1 and 6: {and( $r|ge( 1 ), $r|le( 6 ) )|choose( 'no', 'yes' )}
```

Output:

```
4|roman = IV
roman( 1999 ) = MCMXCIX
2026|roman = MMXXVI
rand( 1, 6 ) is between 1 and 6: yes
```

## A worked example: number of pages

The Recipes category on a development installation (node 144), four items per page:

**Template**

```
{def $total = fetch( 'content', 'list_count', hash( 'parent_node_id', 144 ) )
     $limit = 4
     $pages = ceil( div( $total, $limit ) )}
{$total} items under Recipes, {$limit} per page: {$pages} pages
```

Output:

```
18 items under Recipes, 4 per page: 5 pages
```

## Related pages

- Math operators: the designer's overview, with paging and grid recipes
- Logic operators for comparing the results
- Pagination
- All template operators (generated)
- Free documentation: this page is free to copy, translate and improve under the GNU FDL 1.3.

### Become an Exponential Partner

Partners share their information, provide funding, contribute back and develop the products with us, and that is how Exponential grows. A partner website and dashboard are planned for mid-2027: please become a partner today to speed up our growth rate and land new features faster.

Partners: start here | How to apply

Exponential Copyright © 1998-2026 Exponential Foundation and 7x
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".
This documentation is free documentation, licensed under the GNU Free Documentation License 1.3 or later:
https://www.gnu.org/licenses/fdl-1.3.html.