Amtiri

AMTIRI SCRIPT

Operators, literals and structures

Generated on November 25, 2026

Values

Decimal

`double`

A decimal number. Written with a point, never a comma, whatever the display locale.

Returns `double`: The number as written.

Examples

```
2.5 # 2.5
1.5e3 # 1500.0
```

False

`false`

The boolean a comparison yields when it does not hold. Written bare and lower case, never quoted.

Returns `boolean`: False.

Examples

```
false # false
PUMP_ON == false # false
```

Null

`null`

The absence of a value. Use it as the fallback of a lookup, and read it back with nvl.

Returns `any`: Nothing - null itself.

Examples

```
null # null
nvl(null, 0) # 0
```

Number

`integer`

A whole number written straight into the script. It is a 32-bit integer: one above 2147483647 does not compile, so write a larger value with a decimal point. A minus before a number is the unary minus, so -2147483648 does not compile either: write -2147483647 - 1.

Returns `int`: The number as written.

Examples

```
42 # 42
COUNT + 1 # 4.0
2147483647 # 2147483647
```

Text

`string`

A fixed piece of text, in double quotes. Quotes and backslashes inside it are escaped with a backslash.

Returns `string`: The text as written.

Examples

```
"P-101" # "P-101"
"Pump \"A\" stopped" # "Pump \"A\" stopped"
"Tank " + TAG # "Tank P-101"
```

True

`true`

The boolean a comparison yields when it holds. Written bare and lower case, never quoted.

Returns `boolean`: True.

Examples

```
true # true
not(true) # false
```

Operators

Add

`+` · `left: any + right: any...`

Adds numbers. When the left value is text it joins the others on as text instead: "a" + 1 gives "a1", while 1 + "a" fails.

Parameters

Name	Type	Description
`left`	`any`	The value on the left of the operator.
`right`	`any` (1 or more)	A further value to add. The operator chains, so a + b + c is one expression.

Returns `any`: The sum, or the joined text.

Examples

```
LEVEL + 7.5 # 50.0
COUNT + 1 # 4.0
"Tank " + TAG # "Tank P-101"
TAG + 1 # "P-1011"
1 + "a" # error
```

And (short-circuit)

`&&` · `left: boolean && right: boolean...`

True only when both sides are true. The right side is only evaluated if the left one holds, so it can guard a condition that is only valid once the left one holds.

Parameters

Name	Type	Description
`left`	`boolean`	The condition on the left of the operator.
`right`	`boolean` (1 or more)	A further condition. The operator chains, so a && b && c is one expression.

Returns `boolean`: True when both sides are true.

Examples

```
PUMP_ON && LEVEL > 40 # true
false && NEVER_WRITTEN > 0 # false
```

Divide

`/` · `left: double / right: double...`

Divides the left value by the right one. Dividing by zero yields an infinite value rather than failing - guard it with ivl. When the left value is text it joins the others on as text instead, as + does.

Parameters

Name	Type	Description
`left`	`double`	The value to divide.
`right`	`double` (1 or more)	A further value to divide by. Chains left to right, so a / b / c divides twice.

Returns `double`: The quotient.

Examples

```
LEVEL / 2 # 21.25
1 / 0 # Infinity
```

Multiply

`*` · `left: double * right: double...`

Multiplies numbers. When the left value is text it joins the others on as text instead, as + does: "ab" * 3 gives "ab3".

Parameters

Name	Type	Description
`left`	`double`	The value on the left of the operator.
`right`	`double` (1 or more)	A further factor. The operator chains, so a * b * c is one expression.

Returns `double`: The product.

Examples

```
COUNT * 2 # 6.0
LEVEL * 1.8 + 32 # 108.5
```

Negate

`-` · `-value: double`

Flips the sign of a number and keeps its kind: a whole number stays whole and a decimal stays decimal. Anything that is not a number gives null. This is the unary minus, distinct from the subtraction operator that takes two values.

Parameters

Name	Type	Description
`value`	`double`	The number whose sign is flipped.

Returns `double`: The value with the opposite sign, of the same kind.

Examples

```
-LEVEL # -42.5
-COUNT # -3
-(LEVEL - 50) # 7.5
```

Or (short-circuit)

`||` · `left: boolean || right: boolean...`

True when either side is true. The right side is only evaluated if the left one is false.

Parameters

Name	Type	Description
`left`	`boolean`	The condition on the left of the operator.
`right`	`boolean` (1 or more)	A further condition. The operator chains, so a || b || c is one expression.

Returns `boolean`: True when either side is true.

Examples

```
PUMP_ON || LEVEL > 90 # true
true || NEVER_WRITTEN > 0 # true
```

Power

`^` · `left: double ^ right: double...`

Raises the left value to the power of the right one.

Parameters

Name	Type	Description
`left`	`double`	The base.
`right`	`double` (1 or more)	A further exponent. Chains in the order written, so a ^ b ^ c raises twice.

Returns `double`: The power.

Examples

```
2 ^ 3 # 8.0
2 ^ 3 ^ 2 # 64.0
COUNT ^ 2 # 9.0
```

Subtract

`-` · `left: double - right: double...`

Subtracts the right value from the left one. Distinct from the unary minus, which takes a single value.

Parameters

Name	Type	Description
`left`	`double`	The value to subtract from.
`right`	`double` (1 or more)	A further value to subtract. Chains left to right, so a - b - c subtracts both.

Returns `double`: The difference.

Examples

```
LEVEL - 2.5 # 40.0
100 - LEVEL - 7.5 # 50.0
```

Comparisons

Equals

`==` · `left: any == right: any`

True when both values match. Always written doubled: a single = is not a comparison. Comparing a number with text is an error.

Parameters

Name	Type	Description
`left`	`any`	The value on the left of the comparison.
`right`	`any`	The value on the right of the comparison.

Returns `boolean` : True when the comparison holds.

Examples

```
COUNT == 3 # true
TAG == "P-101" # true
1.0 == 1 # true
```

Greater than

`>` · `left: any > right: any`

True when the left value sorts after the right one. Comparing a number with text is an error.

Parameters

Name	Type	Description
`left`	`any`	The value on the left of the comparison.
`right`	`any`	The value on the right of the comparison.

Returns `boolean` : True when the comparison holds.

Examples

```
LEVEL > 40 # true
TAG > "P-100" # true
```

Greater than or equal

`>=` · `left: any >= right: any`

True when the left value sorts after the right one, or matches it. Comparing a number with text is an error.

Parameters

Name	Type	Description
`left`	`any`	The value on the left of the comparison.
`right`	`any`	The value on the right of the comparison.

Returns `boolean`: True when the comparison holds.

Examples

```
COUNT >= 3 # true
```

Less than

`<` · `left: any < right: any`

True when the left value sorts before the right one. Comparing a number with text is an error.

Parameters

Name	Type	Description
`left`	`any`	The value on the left of the comparison.
`right`	`any`	The value on the right of the comparison.

Returns `boolean`: True when the comparison holds.

Examples

```
LEVEL < SETPOINTS.low # false
SETPOINTS.low < LEVEL < SETPOINTS.high # true
```

Less than or equal

`<=` · `left: any <= right: any`

True when the left value sorts before the right one, or matches it. Comparing a number with text is an error.

Parameters

Name	Type	Description
`left`	`any`	The value on the left of the comparison.
`right`	`any`	The value on the right of the comparison.

Returns `boolean`: True when the comparison holds.

Examples

```
LEVEL <= 42.5 # true
```

Not equal

`<>` · `left: any <> right: any`

True when the two values differ. The spelling to prefer in new scripts. Comparing a number with text is an error.

Parameters

Name	Type	Description
`left`	`any`	The value on the left of the comparison.
`right`	`any`	The value on the right of the comparison.

Returns `boolean`: True when the comparison holds.

Examples

```
TAG <> "P-102" # true
```

Not equal (C style)

`!=` · `left: any != right: any`

True when the two values differ. Accepted as a synonym of <> so scripts written in C style keep working. Comparing a number with text is an error.

Parameters

Name	Type	Description
`left`	`any`	The value on the left of the comparison.
`right`	`any`	The value on the right of the comparison.

Returns `boolean`: True when the comparison holds.

Examples

```
COUNT != 3 # false
```

Structures

Field

`.` · `record: object.field: name`

Reads one named field out of a record - a control function's state, a database row, an inline object. The name is written bare, never in quotes, and a field the record does not hold is an error rather than a null.

Parameters

Name	Type	Description
`record`	`object`	The record the field is read from.
`field`	`name`	The name of the field to read.

Returns `any`: Whatever that field holds.

Examples

```
SETPOINTS.high # 90
{level: LEVEL, alarm: false}.level # 42.5
```

Item at

`[]` · `list: [][position: int]`

Reads one item out of a list by its position, counting from zero. Ask length how many there are before reaching past the end.

Parameters

Name	Type	Description
`list`	`[]`	The list the item is read from.
`position`	`int`	Which item to read, counting from zero.

Returns `any`: The item at that position.

Examples

```
REGISTERS[2] # 65535
[10, 20, 30][0] # 10
REGISTERS[4] # error
```

List

`array`

An ordered list written inline, [a, b, c]. Read an item back by index, and count them with length.

Returns `[]`: The list, holding its items in order.

Examples

```
[LEVEL, COUNT, TAG] # [42.5, 3, "P-101"]
[] # []
[1, [2, 3]] # [1, [2, 3]]
```

Object

`object`

A record of named fields written inline, {key: value}. This is how a control function's settings and state are built.

Returns `object`: The record, holding its named fields.

Examples

```
{level: LEVEL, alarm: LEVEL > 90} # {alarm: false, level: 42.5}
{} # {}
{low = 10, high = 90} # {high: 90, low: 10}
```

This page describes Amtiri Script 5.5.0 as shipped with Data Orchester engine 6.22.0 and Site 1.27.0.