mdn web docs_

Logical OR (||)

Baseline Widely available

The **logical OR (`||`)** (logical disjunction) operator for a set of operands is true if and only if one or more of its operands is true. It is typically used with boolean (logical) values. When it is, it returns a Boolean value. However, the `||` operator actually returns the value of one of the specified operands, so if this operator is used with non-Boolean values, it will return a non-Boolean value.

Try it

JavaScript Demo: Expressions - Logical OR

```js
const a = 3;
const b = -2;

console.log(a > 0 || b > 0);
// Expected output: true

```

Run ›

Reset

Syntax

JS

```js
x || y
```

Description

If `x` can be converted to `true`, returns `x`; else, returns `y`.

If a value can be converted to `true`, the value is so-called truthy. If a value can be converted to `false`, the value is so-called falsy.

Examples of expressions that can be converted to false are:

- `null`;
- `NaN`;
- `0`;
- empty string (`""` or `''` or ``` `` ```);
- `undefined`.

Even though the `||` operator can be used with operands that are not Boolean values, it can still be considered a boolean operator since its return value can always be converted to a boolean primitive. To explicitly convert its return value (or any expression in general) to the corresponding boolean value, use a double NOT operator or the `Boolean()` constructor.

Short-circuit evaluation

The logical OR expression is evaluated left to right, it is tested for possible "short-circuit" evaluation using the following rule:

`(some truthy expression) || expr` is short-circuit evaluated to the truthy expression.

Short circuit means that the `expr` part above is **not evaluated**, hence any side effects of doing so do not take effect (e.g., if `expr` is a function call, the calling never takes place). This happens because the value of the operator is already determined after the evaluation of the first operand. See example:

JS

```
function A() {
  console.log("called A");
  return false;
}
function B() {
  console.log("called B");
  return true;
}

console.log(B() || A());
// Logs "called B" due to the function call,
// then logs true (which is the resulting value of the operator)
```

Operator precedence

The following expressions might seem equivalent, but they are not, because the `&&` operator is executed before the `||` operator (see operator precedence).

JS

```
true || false && false; // returns true, because && is executed first
(true || false) && false; // returns false, because grouping has the highest
precedence
```

Examples

Using OR

The following code shows examples of the `||` (logical OR) operator.

JS

```
true || true; // t || t returns true
false || true; // f || t returns true
true || false; // t || f returns true
false || 3 === 4; // f || f returns false
"Cat" || "Dog"; // t || t returns "Cat"
false || "Cat"; // f || t returns "Cat"
"Cat" || false; // t || f returns "Cat"
"" || false; // f || f returns false
```

```
false || ""; // f || f returns ""
false || varObject; // f || object returns varObject
```

> **Note:** If you use this operator to provide a default value to some variable, be aware that any *falsy* value will not be used. If you only need to filter out `null` or `undefined`, consider using the nullish coalescing operator.

Conversion rules for booleans

Converting AND to OR

The following operation involving **booleans**:

JS

```
bCondition1 && bCondition2
```

is always equal to:

JS

```
!(!bCondition1 || !bCondition2)
```

Converting OR to AND

The following operation involving **booleans**:

JS

```
bCondition1 || bCondition2
```

is always equal to:

JS

```
!(!bCondition1 && !bCondition2)
```

Removing nested parentheses

As logical expressions are evaluated left to right, it is always possible to remove parentheses from a complex expression following some rules.

The following composite operation involving **booleans**:

JS

```
bCondition1 && (bCondition2 || bCondition3)
```

is always equal to:

JS

```
!(!bCondition1 || !bCondition2 && !bCondition3)
```

Specifications

Specification
ECMAScript Language Specification # prod-LogicalORExpression

Browser compatibility

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android	WebView on iOS
Logical OR (||)	1	12	1	3	1	18	4	10.1	1	1.0	4.4	1

Tip: you can click/tap on a cell for more information.

Full support

See also

- Nullish coalescing operator (`??`)
- `Boolean`
- Truthy
- Falsy

Help improve MDN

Was this page helpful to you?

Yes No

Learn how to contribute.

This page was last modified on Jul 25, 2024 by MDN contributors.