developer.mozilla.org

Nullish coalescing operator (??) - JavaScript | MDN

5-6 minutos

The **nullish coalescing (`??`)** operator is a logical operator that returns its right-hand side operand when its left-hand side operand is `null` or `undefined`, and otherwise returns its left-hand side operand.

Try it

Syntax

Description

The nullish coalescing operator can be seen as a special case of the logical OR (`||`) operator. The latter returns the right-hand side operand if the left operand is *any* falsy value, not only `null` or `undefined`. In other words, if you use `||` to provide some default value to another variable `foo`, you may encounter unexpected behaviors if you consider some falsy values as usable (e.g., `''` or `0`). See below for more examples.

The nullish coalescing operator has the fifth-lowest operator precedence, directly lower than `||` and directly higher than the conditional (ternary) operator.

It is not possible to combine both the AND (`&&`) and OR operators (`||`) directly with `??`. A syntax error will be thrown in such cases.

```
null || undefined ?? "foo"; // raises a SyntaxError
true && undefined ?? "foo"; // raises a SyntaxError
```

Instead, provide parenthesis to explicitly indicate precedence:

```
(null || undefined) ?? "foo"; // returns "foo"
```

Examples

Using the nullish coalescing operator

In this example, we will provide default values but keep values other than `null` or `undefined`.

```
const nullValue = null;
const emptyText = ""; // falsy
const someNumber = 42;

const valA = nullValue ?? "default for A";
const valB = emptyText ?? "default for B";
const valC = someNumber ?? 0;

console.log(valA); // "default for A"
console.log(valB); // "" (as the empty string is not
null or undefined)
console.log(valC); // 42
```

Assigning a default value to a variable

Earlier, when one wanted to assign a default value to a variable, a common pattern was to use the logical OR operator (||):

```
let foo;

// foo is never assigned any value so it is still
undefined
```

```
const someDummyText = foo || "Hello!";
```

However, due to `||` being a boolean logical operator, the left-hand-side operand was coerced to a boolean for the evaluation and any *falsy* value (including `0`, `''`, `NaN`, `false`, etc.) was not returned. This behavior may cause unexpected consequences if you consider `0`, `''`, or `NaN` as valid values.

```
const count = 0;
const text = "";

const qty = count || 42;
const message = text || "hi!";
console.log(qty); // 42 and not 0
console.log(message); // "hi!" and not ""
```

The nullish coalescing operator avoids this pitfall by only returning the second operand when the first one evaluates to either `null` or `undefined` (but no other falsy values):

```
const myText = ""; // An empty string (which is also a
falsy value)

const notFalsyText = myText || "Hello world";
console.log(notFalsyText); // Hello world

const preservingFalsy = myText ?? "Hi neighborhood";
console.log(preservingFalsy); // '' (as myText is
neither undefined nor null)
```

Short-circuiting

Like the 'OR' and 'AND' logical operators, the right-hand side expression is not evaluated if the left-hand side proves to be neither `null` nor `undefined`.

```
function a() {
  console.log("a was called");
  return undefined;
}
function b() {
  console.log("b was called");
  return false;
}
function c() {
  console.log("c was called");
  return "foo";
}

console.log(a() ?? c());
// Logs "a was called" then "c was called" and then
"foo"
// as a() returned undefined so both expressions are
evaluated

console.log(b() ?? c());
// Logs "b was called" then "false"
// as b() returned false (and not null or undefined),
the right
// hand side expression was not evaluated
```

Relationship with the optional chaining operator (?.)

The nullish coalescing operator treats `undefined` and `null` as specific values. So does the optional chaining operator (`?.`), which is useful to access a property of an object which may be `null` or `undefined`. Combining them, you can safely access a property of an object which may be nullish and provide a default value if it is.

```
const foo = { someFooProp: "hi" };

console.log(foo.someFooProp?.toUpperCase() ?? "not
available"); // "HI"
console.log(foo.someBarProp?.toUpperCase() ?? "not
available"); // "not available"
```

Specifications

Specification
ECMAScript Language Specification # prod-CoalesceExpression

Browser compatibility

Report problems with this compatibility data on GitHub

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	
Nullish coalescing operator (??)							

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

Full support

The compatibility table on this page is generated from structured data. If you'd like to contribute to the data, please check out https://github.com/mdn/browser-compat-data and send us a pull request.

See also