EDUFY

Logical operators: &&, || and !

Maximilian Bahrami · 10 November 2026

Combine conditions with && (and), || (or) and ! (not), use brackets when you mix them, fill in default values with ||, and avoid the range mistake 10 < age < 20.

Real decisions rarely depend on one thing. "Enable the Sign up button if the name is filled in **and** the terms are ticked." "Show free shipping if the order is over 50 € **or** the customer is a member." The three **logical operators** &&, || and ! are how you write that. You've used them a little already. By the end of this lesson you'll know exactly how each one works, how to combine several without surprises, and two shortcuts frontend developers use every day.

The three operators

Operator	Name	Gives true when…
a && b	AND	**both** a and b are true
a || b	OR	**at least one** of them is true
!a	NOT	a is false (it flips the value)

&& (AND): everything must be true

```
console.log(true && true); // true
console.log(true && false); // false
console.log(false && true); // false
console.log(false && false); // false
```

Only **one** result is true: when both sides are true. Think of a door with two locks: you need both keys.

```
const hasTicket = true;
const hasPassport = false;

if (hasTicket && hasPassport) {
console.log("You can board the plane");
} else {
```

```
console.log("Sorry, you can't board");
}
```

|| (OR): one is enough

```
console.log(true || true); // true
console.log(true || false); // true
console.log(false || true); // true
console.log(false || false); // false
```

Only **one** result is false: when both sides are false. Think of a house with two doors: one open door is enough to get in.

```
const total = 35;
const isMember = true;

if (total >= 50 || isMember) {
console.log("Free shipping!");
} else {
console.log("Shipping: 5 €");
}
```

! (NOT): flip it

! turns true into false and false into true. It's great for checking that something is **not** the case:

```
const isLoggedIn = false;

if (!isLoggedIn) {
console.log("Please sign in");
}

console.log(!true); // false
console.log(!false); // true
```

More than two conditions

You can chain as many as you need. With only &&, **all** must be true. With only ||, **one** is enough:

```
const name = "Sara";
const age = 16;
const acceptedTerms = true;

const canSignUp = name !== "" && age >= 13 && acceptedTerms;
console.log("Can sign up:", canSignUp); // true
```

When you **mix** && and ||, use brackets, like in maths. They make the order clear for JavaScript and for anyone reading your code:

```
const isWeekend = true;
const isHoliday = false;
const isSunny = true;

// A day out: on a day off, and only if it's sunny
const dayOut = (isWeekend || isHoliday) && isSunny;
console.log("Day out:", dayOut); // true
```

Without brackets

> JavaScript does && before ||, the way maths does × before +. So a || b && c means a || (b && c). That's easy to get wrong, so whenever you mix the two, **write the brackets yourself**.

isAdmin is false, isOwner is true, isBlocked is true. What is (isAdmin || isOwner) && !isBlocked?

- true
- ✓ false

> (isAdmin || isOwner) is true, because the user is the owner. But !isBlocked is false, because the user **is** blocked. true && false is false.

A sign-up form

Here's how this looks on a real page. The button only works when **all** the rules are met. Press **Run**, then type a name and tick the box:

```
<!DOCTYPE html>
<html lang="en">
<body style="padding: 16px; font-family: sans-serif">
<input id="name" placeholder="Your name" />
<label>
<input id="terms" type="checkbox" /> I accept the terms
</label>
<p><button id="signup" disabled>Sign up</button></p>
<script>
const nameBox = document.getElementById("name");
const terms = document.getElementById("terms");
const button = document.getElementById("signup");

function update() {
const hasName = nameBox.value.trim() !== "";
const canSignUp = hasName && terms.checked;

button.disabled = !canSignUp;
console.log("name:", hasName, "terms:", terms.checked);
```

```
}

nameBox.addEventListener("input", update);
terms.addEventListener("change", update);
</script>
</body>
</html>
```

button.disabled = !canSignUp uses ! to flip the answer: when you **can** sign up, the button is **not** disabled.

Shortcut 1: || for a default value

|| doesn't only work with true and false. It gives back the **first truthy value** it finds. Developers use that to fill in a value when one is missing:

```
const typedName = "";
const name = typedName || "Guest";
console.log("Hello, " + name); // Hello, Guest

const typedName2 = "Omar";
const name2 = typedName2 || "Guest";
console.log("Hello, " + name2); // Hello, Omar
```

Read it as: use typedName, or if it's empty, use "Guest".

Careful with 0

> 0 is falsy too, so 0 || 10 gives 10. If 0 is a real answer (a score, a price), don't use || for a default.

Shortcut 2: JavaScript stops early

&& and || stop as soon as they know the answer:

- false && ...: the answer is already false, so the right side is **never checked**.
- true || ...: the answer is already true, so the right side is **never checked**.

This lets you check that something exists before you use it:

```
const user = null; // nobody is signed in

if (user && user.name === "Sara") {
console.log("Hi Sara");
} else {
console.log("No user, or not Sara");
}
```

Without user &&, the code would try to read .name of null and crash. With it, JavaScript sees null (falsy), stops, and never touches user.name.

Common mistakes

Writing a range like in maths

Wrong

```
if (10 < age < 20) { ... }
```

Right

```
if (age > 10 && age < 20) { ... }
```

JavaScript reads 10 < age < 20 from left to right: first 10 < age gives true or false, and then it checks true < 20 or false < 20. true counts as 1 and false as 0, and both are less than 20. So it's **always** true, even for an age of 50. Write both comparisons in full with &&.

Leaving out the second comparison

Wrong

```
if (age > 12 && < 20) { ... }
```

Right

```
if (age > 12 && age < 20) { ... }
```

Each side of && must be a full condition on its own. < 20 alone isn't one, so this is an error.

One & or | instead of two

Wrong

```
if (hasTicket & hasPassport) { ... }
```

Right

```
if (hasTicket && hasPassport) { ... }
```

& and | are different operators, for working with numbers bit by bit. They sometimes seem to work by accident. For logic, always write two: && and ||.

Exercise

Can you watch the film?

A cinema has a rule for a 15+ film: you can watch it if you're **15 or older**, or if you're **11 or older and come with a parent**.

Write a function canWatch(age, withParent) that **returns** true or false. Test it with:

- canWatch(16, false): should be true
- canWatch(12, true): should be true
- canWatch(12, false): should be false
- canWatch(9, true): should be false

Answer

```
function canWatch(age, withParent) {
return age >= 15 || (age >= 11 && withParent);
}

console.log(canWatch(16, false)); // true
console.log(canWatch(12, true)); // true
console.log(canWatch(12, false)); // false
console.log(canWatch(9, true)); // false
```

The brackets group the second rule: "11 or older **and** with a parent". Then || says either rule is enough. You don't need an if: the whole expression already is true or false, so you can return it directly.

Key points

- && is true only when **both** sides are true; || is true when **at least one** is; ! flips a value.
- When you mix && and ||, write brackets: (a || b) && c.
- value || "default" fills in a missing value, but watch out for 0.
- && and || stop as soon as they know the answer, so user && user.name is safe.
- Write ranges in full: age > 10 && age < 20, never 10 < age < 20.