EDUFY

const: names that keep their value (and when to use let)

Maximilian Bahrami · 10 November 2026

What const means, how it differs from let, the simple rule for choosing between them, how const works with page elements and lists, and the three mistakes beginners make most.

Almost every line of JavaScript you write starts by giving a value a name. You do that with const or let. By the end of this lesson you'll know what const means, how it's different from let, and the simple rule developers use to pick one, so your code is easier to read and has fewer bugs.

const: a name that keeps its value

const is short for **constant**: something that stays the same. You give it a name and a value, and from then on that name always means that value:

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

console.log(name); // "Sara"
console.log(age); // 16
```

Think of it as a box with a label. With const, you put something in the box once and close the lid.

Try to change it

What happens if you give a const a new value? Press **Run**:

```
const city = "Stockholm";
console.log(city);

city = "Paris"; // not allowed
console.log(city);
```

JavaScript stops with an error: **Assignment to constant variable**. "Assignment" means giving a name a value with =. That's the whole point of const: it protects the value from being changed by mistake.

let: a name that can change

When a value **has** to change, use let instead:

```
let score = 0;
console.log(score); // 0

score = 10;
console.log(score); // 10

score = score + 5;
console.log(score); // 15
```

	const	let
Can you give it a new value?	No	Yes
Must it get a value right away?	Yes	No
Use it for	values that stay the same	counters, scores, totals

The rule developers use

Start with const

> Write const every time. Only change it to let when you see that the value needs to change.

Most names in real code never change, so most of them are const. When another developer (or you, next week) sees let, it's a signal: watch out, this one changes.

Which one should be let?

> • const siteName = "EDUFY";
>
> • const button = document.getElementById("save");
>
> • ✓ const clicks = 0; (it goes up on every click)
>
> • const price = 25;

> clicks goes up by one on every click, so it gets a new value: it has to be let. The others are set once and stay the same.

const on a web page

In frontend code you use const all the time for the parts of the page you work with. The button is always the same button, so it's const. The number of clicks changes, so it's let. Press **Run**, then click the button:

```
<!DOCTYPE html>
<html lang="en">
<body style="padding: 16px; font-family: sans-serif">
<button id="like">Like</button>
<p id="count">0 likes</p>
<script>
const button = document.getElementById("like");
const text = document.getElementById("count");
let likes = 0;

button.addEventListener("click", () => {
likes = likes + 1;
text.textContent = likes + " likes";
console.log("likes:", likes);
});
</script>
</body>
</html>
```

const with a list

A const list can still get new items. const only means the name always points to **the same list**. What's inside the list can change:

```
const fruits = ["apple", "banana"];

fruits.push("cherry"); // add an item: allowed
console.log(fruits); // ["apple", "banana", "cherry"]
```

What you can't do is swap it for a whole new list with =:

```
const fruits = ["apple", "banana"];
fruits = ["melon"]; // error: a new value
```

So for a to-do list or a shopping cart, const is the right choice: you keep adding to the same list.

What about var?

In older code and old tutorials you'll see var. It's the old way to make a variable, from before const and let existed, and it has some confusing behaviour. You don't need it: use const and let.

Common mistakes

Using const for something that changes

Wrong

```
const count = 0;
count = count + 1; // error
```

Right

```
let count = 0;
count = count + 1; // 1
```

If you get **Assignment to constant variable**, look at the name in the error. Either the value really should change (make it let), or you changed it by mistake (remove that line).

Making a const without a value

Wrong

```
const total;
total = 50;
```

Right

```
const total = 50;
```

A const can never get a value later, so it must get one on the same line. If you really don't know the value yet, use let.

Writing const twice for the same name

Wrong

```
const color = "red";
const color = "blue"; // error
```

Right

```
let color = "red";
color = "blue";
```

Write const or let only the **first** time you make a name. After that, use the name on its own.

Exercise

A shopping list

> Make a page with a text box, an **Add** button and an empty list.

> 1. Get the text box, the button and the list with document.getElementById. Should they be const or let?
> 2. Make an empty array called items.
> 3. On a click, add the text to items with push, show it in the list, and print items in the console.

Answer

```
<!DOCTYPE html>
<html lang="en">
<body style="padding: 16px; font-family: sans-serif">
<input id="item" placeholder="Milk, bread..." />
<button id="add">Add</button>
<ul id="list"></ul>
<script>
const input = document.getElementById("item");
const button = document.getElementById("add");
const list = document.getElementById("list");
const items = [];

button.addEventListener("click", () => {
const text = input.value;
if (text === "") return;

items.push(text);
const li = document.createElement("li");
li.textContent = text;
list.appendChild(li);
input.value = "";
console.log(items);
});
</script>
</body>
</html>
```

> Everything here is const. The page parts never change, and items is always the same list, it just gets longer. text is const too: every click makes a new text, it's never changed.

Key points

> • const gives a name a value that can't be changed with =.

> • let is for values that change, like counters and scores.

- Start with const, and change to let only when the value has to change.
- A const must get its value on the same line.
- A const list can get new items with push; it just can't be swapped for another list. Skip var.