

const



The **const** declaration declares block-scoped local variables. The value of a constant can't be changed through reassignment using the <u>assignment operator</u>, but if a constant is an <u>object</u>, its properties can be added, updated, or removed.

Try it

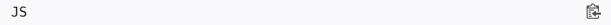
```
JavaScript Demo: const declaration
1 const number = 42;
2
3 try {
4 number = 99;
5 } catch (err) {
    console.log(err);
6
7
     // Expected output: TypeError: invalid assignment to const 'number'
8
     // (Note: the exact output may be browser-dependent)
9 }
10
11 console.log(number);
12 // Expected output: 42
13
```

Reset

Reset

See what I can help you with!

Syntax



```
const name1 = value1;
const name1 = value1, name2 = value2;
const name1 = value1, name2 = value2, /* ..., */ nameN = valueN;
```

nameN

The name of the variable to declare. Each must be a legal JavaScript <u>identifier</u> or a <u>destructuring binding pattern</u>.

valueN

Initial value of the variable. It can be any legal expression.

Description

The const declaration is very similar to <a>let:

- const declarations are scoped to blocks as well as functions.
- const declarations can only be accessed after the place of declaration is reached (see <u>temporal dead zone</u>). For this reason, const declarations are commonly regarded as <u>non-hoisted</u>.
- const declarations do not create properties on globalThis when declared at the top level of a script.
- const declarations cannot be <u>redeclared</u> by any other declaration in the same scope.
- const begins <u>declarations</u>, not <u>statements</u>. That means you cannot use a lone <u>const</u> declaration as the body of a block (which makes sense, since there's no way to access the variable).

```
if (true) const a = 1; // SyntaxError: Lexical declaration cannot appear in a single-statement context
```

An initializer for a constant is required. You must specify its value in the same declaration. (This makes sense, given that it can't be changed later.)

JS 🚉

const F00; // SyntaxError: Missing initializer in const declaration



The const declaration creates an immutable reference to a value. It does *not* mean the value it holds is immutable — just that the variable identifier cannot be reassigned. For instance, in the case where the content is an object, this means the object's contents (e.g., its properties) can be altered. You should understand const declarations as "create a variable whose *identity* remains constant", not "whose *value* remains constant" — or, "create immutable <u>bindings</u>", not "immutable values".

Many style guides (including MDN's) recommend using const over <u>let</u> whenever a variable is not reassigned in its scope. This makes the intent clear that a variable's type (or value, in the case of a primitive) can never change. Others may prefer <u>let</u> for non-primitives that are mutated.

The list that follows the const keyword is called a <u>binding</u> list and is separated by commas, where the commas are <u>not comma operators</u> and the <u>= signs are not assignment operators</u>. Initializers of later variables can refer to earlier variables in the list.

Examples

Basic const usage

Constants can be declared with uppercase or lowercase, but a common convention is to use all-uppercase letters, especially for primitives because they are truly immutable.

```
JS

// define MY_FAV as a constant and give it the value 7

const MY_FAV = 7;

See what I can help you with!

console.log("my favorite number is: " + MY_FAV);

JS
```

```
// Re-assigning to a constant variable throws an error
MY_FAV = 20; // TypeError: Assignment to constant variable

// Redeclaring a constant throws an error
const MY_FAV = 20; // SyntaxError: Identifier 'MY_FAV' has already been
declared
var MY_FAV = 20; // SyntaxError: Identifier 'MY_FAV' has already been
declared
let MY_FAV = 20; // SyntaxError: Identifier 'MY_FAV' has already been
declared
```

Block scoping

It's important to note the nature of block scoping.

```
const MY_FAV = 7;

if (MY_FAV === 7) {
    // This is fine because it's in a new block scope
    const MY_FAV = 20;
    console.log(MY_FAV); // 20

    // var declarations are not scoped to blocks so this throws an error
    var MY_FAV = 20; // SyntaxError: Identifier 'MY_FAV' has already been
declared
}

console.log(MY_FAV); // 7
```

const in objects and arrays

const also works on objects and arrays. Attempting to overwrite the object throws an error "Assignment to constant variable".

```
Const MY_OBJECT = { key: "value" };

MY_OBJECT = { OTHER_KEY: "value" };
```

However, object keys are not protected, so the following statement is executed without problem.

```
JS

MY_OBJECT.key = "otherValue";
```

You would need to use <a>Object.freeze() to make an object immutable.

The same applies to arrays. Assigning a new array to the variable throws an error "Assignment to constant variable".

```
JS

const MY_ARRAY = [];
MY_ARRAY = ["B"];
```

Still, it's possible to push items into the array and thus mutate it.

```
JS
MY_ARRAY.push("A"); // ["A"]
```

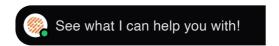
Declaration with destructuring

The left-hand side of each = can also be a binding pattern. This allows creating multiple variables at once.

```
JS

const result = /(a+)(b+)(c+)/.exec("aaabcc");
const [, a, b, c] = result;
console.log(a, b, c); // "aaa" "b" "cc"
```

For more information, see <u>Destructuring</u>.



Specifications