Block Scope

**block**

A **block statement** is used to group zero or more statements. The block is delimited by a pair of braces ("curly brackets") and contains a list of zero or more statements and declarations.

[**Try it**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block#try_it)

[**Syntax**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block#syntax)

{

StatementList

}

Copy to Clipboard

StatementList

Statements and declarations grouped within the block statement.

[**Description**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block#description)

The block statement is often called the *compound statement* in other languages. It allows you to use multiple statements where JavaScript expects only one statement. Combining statements into blocks is a common practice in JavaScript, especially when used in association with control flow statements like [if...else](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else) and [for](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for). The opposite behavior is possible using an [empty statement](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/Empty), where you provide no statement, although one is required.

In addition, combined with block-scoped declarations like [let](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let), [const](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const), and [class](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/class), blocks can prevent temporary variables from polluting the global namespace, just like [IIFEs](https://developer.mozilla.org/en-US/docs/Glossary/IIFE) do.

[**Block scoping rules with var or function declaration in non-strict mode**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block#block_scoping_rules_with_var_or_function_declaration_in_non-strict_mode)

Variables declared with var or created by [function declarations](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/function) in non-strict mode **do not** have block scope. Variables introduced within a block are scoped to the containing function or script, and the effects of setting them persist beyond the block itself. For example:

var x = 1;

{

var x = 2;

}

console.log(x); // 2

Copy to Clipboard

This logs 2 because the var x statement within the block is in the same scope as the var x statement before the block.

In non-strict code, function declarations inside blocks behave strangely. Do not use them.

[**Block scoping rules with let, const, class, or function declaration in strict mode**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block#block_scoping_rules_with_let_const_class_or_function_declaration_in_strict_mode)

By contrast, identifiers declared with [let](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let), [const](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const), and [class](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/class) do have block scope:

let x = 1;

{

let x = 2;

}

console.log(x); // 1

Copy to Clipboard

The x = 2 is limited in scope to the block in which it was defined.

The same is true of const:

const c = 1;

{

const c = 2;

}

console.log(c); // 1; does not throw SyntaxError

Copy to Clipboard

Note that the block-scoped const c = 2 *does not* throw a SyntaxError: Identifier 'c' has already been declared because it can be declared uniquely within the block.

In [strict mode](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Strict_mode), function declarations inside blocks are scoped to that block and are hoisted.

"use strict";

{

foo(); // Logs "foo"

function foo() {

console.log("foo");

}

}

foo(); // ReferenceError: foo is not defined

Copy to Clipboard

[**Examples**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block#examples)

[**Using a block statement as the body of a for loop**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block#using_a_block_statement_as_the_body_of_a_for_loop)

A [for](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/for) loop accepts a single statement as its body.

for (let i = 0; i < 10; i++) console.log(i);

Copy to Clipboard

If you want to use more than one statement in the loop body, you can group them into one block statement:

for (let i = 0; i < 10; i++) {

console.log(i);

console.log(i \*\* 2);

}

Copy to Clipboard

[**Using a block statement to encapsulate data**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/block#using_a_block_statement_to_encapsulate_data)

let and const declarations are scoped to the containing block. This allows you to hide data from the global scope without wrapping it in a function.

let sector;

{

// These variables are scoped to this block and are not

// accessible after the block

const angle = Math.PI / 3;

const radius = 10;

sector = {

radius,

angle,

area: (angle / 2) \* radius \*\* 2,

perimeter: 2 \* radius + angle \* radius,

};

}

console.log(sector);

// {

// radius: 10,

// angle: 1.0471975511965976,

// area: 52.35987755982988,

// perimeter: 30.471975511965976

// }

console.log(typeof radius); // "undefined"

Copy to Clipboard