JavaScript Let

❮ Previous Next ❯

ECMAScript 2015

ES2015 introduced two important new JavaScript keywords: `let` and `const`.

These two keywords provide **Block Scope** variables (and constants) in JavaScript.

Before ES2015, JavaScript had only two types of scope: **Global Scope** and **Function Scope**.

Global Scope

Variables declared **Globally** (outside any function) have **Global Scope**.

Example

```
var carName = "Volvo";

// code here can use carName

function myFunction() {
  // code here can also use carName
}
```

Try it Yourself »

Global variables can be accessed from anywhere in a JavaScript program.

Function Scope

Variables declared **Locally** (inside a function) have **Function Scope**.

Example

```
// code here can NOT use carName

function myFunction() {
  var carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Try it Yourself »

Local variables can only be accessed from inside the function where they are declared.

JavaScript Block Scope

Variables declared with the `var` keyword can not have **Block Scope**.

Variables declared inside a block **{}** can be accessed from outside the block.

Example

```
{
  var x = 2;
}
// x CAN be used here
```

Before ES2015 JavaScript did not have **Block Scope**.

Variables declared with the `let` keyword can have Block Scope.

Variables declared inside a block **{}** can not be accessed from outside the block:

Example

```
{
  let x = 2;
}
// x can NOT be used here
```

Redeclaring Variables

Redeclaring a variable using the `var` keyword can impose problems.

Redeclaring a variable inside a block will also redeclare the variable outside the block:

Example

```
var x = 10;
// Here x is 10
{
  var x = 2;
  // Here x is 2
}
// Here x is 2
```

Try it Yourself »

Redeclaring a variable using the `let` keyword can solve this problem.

Redeclaring a variable inside a block will not redeclare the variable outside the block:

Example

```
var x = 10;
// Here x is 10
{
  let x = 2;
  // Here x is 2
}
// Here x is 10
```

Try it Yourself »

Browser Support

The `let` keyword is not fully supported in Internet Explorer 11 or earlier.

The following table defines the first browser versions with full support for the `let` keyword:

Chrome 49	IE / Edge 12	Firefox 44	Safari 11	Opera 36
Mar, 2016	Jul, 2015	Jan, 2015	Sep, 2017	Mar, 2016

Loop Scope

Using `var` in a loop:

Example

```
var i = 5;
for (var i = 0; i < 10; i++) {
  // some statements
}
// Here i is 10
```

Try it Yourself »

Using `let` in a loop:

Example

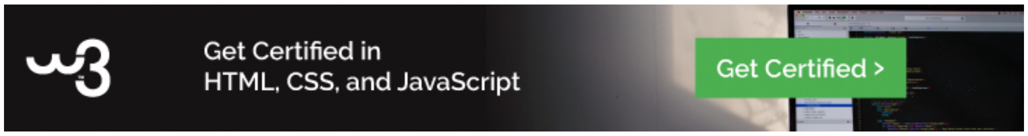
```
let i = 5;
for (let i = 0; i < 10; i++) {
  // some statements
}
// Here i is 5
```

Try it Yourself »

In the first example, using `var`, the variable declared in the loop redeclares the variable outside the loop.

In the second example, using `let`, the variable declared in the loop does not redeclare the variable outside the loop.

When `let` is used to declare the i variable in a loop, the i variable will only be visible within the loop.



Function Scope

Variables declared with `var` and `let` are quite similar when declared inside a function.

They will both have **Function Scope**:

```
function myFunction() {
  var carName = "Volvo";   // Function Scope
}
```

```
function myFunction() {
  let carName = "Volvo";   // Function Scope
}
```

Global Scope

Variables declared with `var` and `let` are quite similar when declared outside a block.

They will both have **Global Scope**:

```
var x = 2;       // Global scope
```

```
let x = 2;       // Global scope
```

Global Variables in HTML

With JavaScript, the global scope is the JavaScript environment.

In HTML, the global scope is the window object.

Global variables defined with the `var` keyword belong to the window object:

Example

```
var carName = "Volvo";
// code here can use window.carName
```

Try it Yourself »

Global variables defined with the `let` keyword do not belong to the window object:

Example

```
let carName = "Volvo";
// code here can not use window.carName
```

Try it Yourself »

Redeclaring

Redeclaring a JavaScript variable with `var` is allowed anywhere in a program:

Example

```
var x = 2;
```

```
// Now x is 2

var x = 3;

// Now x is 3
```

Try it Yourself »

Redeclaring a `var` variable with `let`, in the same scope, or in the same block, is not allowed:

Example

```
var x = 2;       // Allowed
let x = 3;       // Not allowed

{
  var x = 4;   // Allowed
  let x = 5   // Not allowed
}
```

Redeclaring a `let` variable with `let`, in the same scope, or in the same block, is not allowed:

Example

```
let x = 2;       // Allowed
let x = 3;       // Not allowed

{
  let x = 4;   // Allowed
  let x = 5;   // Not allowed
}
```

Redeclaring a `let` variable with `var`, in the same scope, or in the same block, is not allowed:

Example

```
let x = 2;       // Allowed
var x = 3;       // Not allowed

{
  let x = 4;   // Allowed
  var x = 5;   // Not allowed
}
```

Redeclaring a variable with `let`, in another scope, or in another block, is allowed:

Example

```
let x = 2;       // Allowed

{
  let x = 3;   // Allowed
}

{
  let x = 4;   // Allowed
}
```

Try it Yourself »

Hoisting

Variables defined with `var` are **hoisted** to the top (if you don't know what Hoisting is, read our Hoisting Chapter).

You can use a variable before it is declared:

Example

```
// you CAN use carName here
var carName;
```

Try it Yourself »

Variables defined with `let` are not hoisted to the top.

Using a `let` variable before it is declared will result in a `ReferenceError`.

The variable is in a "temporal dead zone" from the start of the block until it is declared:

Example

```
// you can NOT use carName here
let carName;
```

❮ Previous Next ❯