MDN Web Docs
moz://a

Hoisting

JavaScript **Hoisting** refers to the process whereby the interpreter allocates memory for variable and function declarations prior to execution of the code. Declarations that are made using `var` are initialized with a default value of `undefined`. Declarations made using `let` and `const` are not initialized as part of hoisting.

Conceptually hoisting is often presented as the interpreter "splitting variable declaration and initialization, and moving (just) the declarations to the top of the code". This allows variables to appear in code before they are defined. Note however, that any variable initialization in the original code will not happen until the line of code is executed.

> **Note:** The term hoisting is not used in any normative specification prose prior to ECMAScript® 2015 Language Specification . Hoisting was thought up as a general way of thinking about how execution contexts (specifically the creation and execution phases) work in JavaScript.

Technical example

One of the advantages of JavaScript putting function declarations into memory before it executes any code segment is that it allows you to use a function before you declare it in your code. For example:

```
function catName(name) {
  console.log("My cat's name is " + name);
}

catName("Tiger");

/*
The result of the code above is: "My cat's name is Tiger"
*/
```

The above code snippet is how you would expect to write the code for it to work. Now, let's see what happens when we call the function before we write it:

```
catName("Chloe");

function catName(name) {
  console.log("My cat's name is " + name);
}
/*
The result of the code above is: "My cat's name is Chloe"
*/
```

Even though we call the function in our code first, before the function is written, the code still works, because this is how context execution works in JavaScript.

Hoisting works well with other data types and variables. The variables can be initialized and used before they are declared.

Only declarations are hoisted

JavaScript only hoists declarations, not initializations. If a variable is used in code and then declared and initialized, the value when it is used will be its default initialization (`undefined` for a variable declared using `var`, otherwise uninitialized). For example:

```
console.log(num); // Returns 'undefined' from hoisted var declaration (not
var num; // Declaration
num = 6; // Initialization
```

The example below only has initialization. No hoisting happens so trying to read the variable results in `ReferenceError` exception.

```
console.log(num); // Throws ReferenceError exception
num = 6; // Initialization
```

Below are more examples demonstrating hoisting.

```
// Example 1
// Only y is hoisted
```

```
x = 1; // Initialize x, and if not already declared, declare it - but no hoist
console.log(x + " " + y); // '1 undefined'
// This prints value of y as undefined as JavaScript only hoists declarations
var y = 2; // Declare and Initialize y


// Example 2
// No hoisting, but since initialization also causes declaration (if not alrea

a = 'Cran'; // Initialize a
b = 'berry'; // Initialize b

console.log(a + "" + b); // 'Cranberry'
```

Let and const hoisting

Variables declared with `let` and `const` are also hoisted, but unlike for `var` the variables are not initialized with a default value of `undefined`. Until the line in which they are initialized is executed, any code that accesses these variables will throw an exception.

For information and examples see `let` > temporal dead zone.

See also

- var statement — MDN
- function statement — MDN

Last modified: Oct 8, 2021, by MDN contributors