**JavaScript Hoisting**

[**https://www.w3schools.com/js/js\_hoisting.asp**](https://www.w3schools.com/js/js_hoisting.asp)

Hoisting is JavaScript's default behavior of moving declarations to the top.

**JavaScript Declarations are Hoisted**

In JavaScript, a variable can be declared after it has been used.

In other words; a variable can be used before it has been declared.

**Example 1** gives the same result as **Example 2**:

**Example 1**

x = 5; // Assign 5 to x  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x;                     // Display x in the element  
  
var x; // Declare x

**Example 2**

var x; // Declare x  
x = 5; // Assign 5 to x  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x;                     // Display x in the element

To understand this, you have to understand the term "hoisting".

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope (to the top of the current script or the current function).

**The let and const Keywords**

Variables defined with let and const are hoisted to the top of the block, but not *initialized*.

Meaning: The block of code is aware of the variable, but it cannot be used until it has been declared.

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**

This will result in a ReferenceError:

carName = "Volvo";  
let carName;

Using a const variable before it is declared, is a syntax errror, so the code will simply not run.

**Example**

This code will not run.

carName = "Volvo";  
const carName;

Read more about let and const in [JS Let / Const](https://www.w3schools.com/js/js_let.asp).

ADVERTISEMENT

**JavaScript Initializations are Not Hoisted**

JavaScript only hoists declarations, not initializations.

**Example 1** does **not** give the same result as **Example 2**:

**Example 1**

var x = 5; // Initialize x  
var y = 7; // Initialize y  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x + " " + y;           // Display x and y

**Example 2**

var x = 5; // Initialize x  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x + " " + y;           // Display x and y  
  
var y = 7; // Initialize y

Does it make sense that y is undefined in the last example?

This is because only the declaration (var y), not the initialization (=7) is hoisted to the top.

Because of hoisting, y has been declared before it is used, but because initializations are not hoisted, the value of y is undefined.

Example 2 is the same as writing:

**Example**

var x = 5; // Initialize x  
var y;     // Declare y  
  
elem = document.getElementById("demo"); // Find an element   
elem.innerHTML = x + " " + y;           // Display x and y  
  
y = 7;    // Assign 7 to y

**Declare Your Variables At the Top !**

Hoisting is (to many developers) an unknown or overlooked behavior of JavaScript.

If a developer doesn't understand hoisting, programs may contain bugs (errors).

To avoid bugs, always declare all variables at the beginning of every scope.

Since this is how JavaScript interprets the code, it is always a good rule.

JavaScript in strict mode does not allow variables to be used if they are not declared.  
Study **"use strict"** in the next chapter.