



# JavaScript Variables

Check this out if you dream of running a solo Internet business

**Published Feb 15 2018**

A variable is a literal assigned to an identifier, so you can reference and use it later in the program. Learn how to declare one with JavaScript

- Introduction to JavaScript Variables
- Using **var**
- Using **let**
- Using **const**

## Introduction to JavaScript Variables

A variable is a literal assigned to an identifier, so you can reference and use it later in the program.

Variables in JavaScript do not have any type attached. Once you assign a specific literal type to a variable, you can later reassign the variable to host any other type, without type errors or any issue.

This is why JavaScript is sometimes referred to as “untyped”.

A variable must be declared before you can use it. There are 3 ways to do this, using **var**, **let** or **const**, and those 3 ways differ in how you can interact with the variable later on.

## Using var

Until ES2015, **var** was the only construct available for defining variables.

```
var a = 0
```

If you forget to add **var** you will be assigning a value to an undeclared variable, and the results might vary.

In modern environments, with strict mode enabled, you will get an error. In older environments (or with strict mode disabled) this will initialize the variable and assign it to the global object.

If you don't initialize the variable when you declare it, it will have the **undefined** value until you assign a value to it.

```
var a //typeof a === 'undefined'
```

You can redeclare the variable many times, overriding it:

```
var a = 1
var a = 2
```

You can also declare multiple variables at once in the same statement:

```
var a = 1, b = 2
```

The **scope** is the portion of code where the variable is visible.

A variable initialized with **var** outside of any function is assigned to the global object, has a global scope and is visible everywhere. A variable initialized with **var** inside a function is assigned to that function, it's local and is visible only inside it, just like a function parameter.

Any variable defined in a function with the same name as a global variable takes precedence over the global variable, shadowing it.

It's important to understand that a block (identified by a pair of curly braces) does not define a new scope. A new scope is only created when a function is created, because **var** does not have block scope, but function scope.

Inside a function, any variable defined in it is visible throughout all the function code, even if the variable is declared at the end of the function it can still be referenced in the beginning, because JavaScript before executing the code actually *moves all variables on top* (something that is called **hoisting**). To avoid confusion, always declare variables at

the beginning of a function.

## Using **let**

**let** is a new feature introduced in ES2015 and it's essentially a block scoped version of **var**. Its scope is limited to the block, statement or expression where it's defined, and all the contained inner blocks.

Modern JavaScript developers might choose to only use **let** and completely discard the use of **var**.

If **let** seems an obscure term, just read **let color = 'red'** as *let the color be red* and it all makes much more sense

Defining **let** outside of any function - contrary to **var** - does not create a global variable.

Using **var** in the top level defines a global variable that's (in the browser) is added to the **window** object. A **let** (and **const**) declaration outside of a block still create a variable that's available across the app code, but it is not assigned to **window**.

## Using **const**

Variables declared with **var** or **let** can be changed later on in the program, and reassigned. Once a **const** is initialized, its value can never be changed again, and it can't be reassigned to a different value.

```
const a = 'test'
```

We can't assign a different literal to the **a** const. We can however mutate **a** if it's an object that provides methods that mutate its contents.


**const** does not provide immutability, just makes sure that the reference can't be changed.

**const** has block scope, same as **let**.

Modern JavaScript developers might choose to always use **const** for variables that don't need to be reassigned later in the program.

Why? Because we should always use the simplest construct available to avoid making errors down the road.

→ You can follow me on Twitter / X

 → Every year I organize a coding BOOTCAMP to teach you how to build modern web applications (next edition JANUARY 2024!)

→ Go to SOLO LAB if you've always dreamed, one day, to start and run an Internet Business as a solo person

(NEW COHORT SIGNUPS OPEN AT THE END OF SEPTEMBER, join the waiting list to stay in the loop)

I wrote 16 free books, read them online or download the pdf/epub:

→ C Handbook

→ Command Line Handbook

→ CSS Handbook

→ Express Handbook

→ Go Handbook

→ HTML Handbook

→ JS Handbook

→ Laravel Handbook

→ Next.js Handbook

→ Node.js Handbook

→ PHP Handbook

→ Python Handbook

→ React Handbook

→ SQL Handbook

→ Svelte Handbook

→ Swift Handbook