# JavaScript

**What is JavaScript?**

* JavaScript, often abbreviated JS, is a programming language that is one of the core technologies of the World Wide Web, alongside HTML and CSS. It lets us add interactivity to pages e.g. you might have seen sliders, alerts, click interactions, popups, etc on different websites — all of that is built using JavaScript.
* JavaScript was initially created to “make web pages alive”.
* The programs in this language are called *scripts*.
* JavaScript can execute not only in the browser, but also on the server, or actually on any device that has a special program called [the JavaScript engine](https://en.wikipedia.org/wiki/JavaScript_engine).

The browser has an embedded engine sometimes called a “JavaScript virtual machine”.

Different engines have different “codenames”. For example:

* [V8](https://en.wikipedia.org/wiki/V8_(JavaScript_engine)) – in Chrome, Opera and Edge.
* [SpiderMonkey](https://en.wikipedia.org/wiki/SpiderMonkey) – in Firefox.
* …There are other codenames like “Chakra” for IE, “JavaScriptCore”, “Nitro” and “SquirrelFish” for Safari, etc.

**How do engines work?**

Engines are complicated. But the basics are easy.

1. The engine (embedded if it’s a browser) reads (“parses”) the script.
2. Then it converts (“compiles”) the script to machine code.
3. And then the machine code runs, pretty fast.

The engine applies optimizations at each step of the process. It even watches the compiled script as it runs, analyzes the data that flows through it, and further optimizes the machine code based on that knowledge.

**What can in-browser JavaScript do?**

Modern JavaScript is a “safe” programming language. It does not provide low-level access to memory or the CPU, because it was initially created for browsers which do not require it.

JavaScript’s capabilities greatly depend on the environment it’s running in. For instance, [Node.js](https://wikipedia.org/wiki/Node.js) supports functions that allow JavaScript to read/write arbitrary files, perform network requests, etc.

In-browser JavaScript can do everything related to webpage manipulation, interaction with the user, and the webserver.

For instance, in-browser JavaScript is able to:

* Add new HTML to the page, change the existing content, modify styles.
* React to user actions, run on mouse clicks, pointer movements, key presses.
* Send requests over the network to remote servers, download and upload files (so-called [AJAX](https://en.wikipedia.org/wiki/Ajax_(programming)) and [COMET](https://en.wikipedia.org/wiki/Comet_(programming)) technologies).
* Get and set cookies, ask questions to the visitor, show messages.
* Remember the data on the client-side (“local storage”).

**What makes JavaScript unique?**

* Full integration with HTML/CSS.
* Simple things are done simply.
* Supported by all major browsers and enabled by default.

JavaScript is the only browser technology that combines these three things.

That’s what makes JavaScript unique. That’s why it’s the most widespread tool for creating browser interfaces.

That said, JavaScript can be used to create servers, mobile applications, etc.

**JavaScript Fundamentals**

**The “script” tag**

JavaScript programs can be inserted almost anywhere into an HTML document using the <script> tag.

For instance:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Script Tag</title>

</head>

<body>

    <p>before script </p>

    <script> alert("Hello world")</script>

    <p>after script </p>

</body>

</html>

**External scripts**

If we have a lot of JavaScript code, we can put it into a separate file.

Script files are attached to HTML with the **src attribute**:

<script src="/path/to/script.js"></script>

Here, /path/to/script.js is an absolute path to the script from the site root. One can also provide a relative path from the current page. For instance, src="script.js", just like src="./script.js", would mean a file "script.js" in the current folder.

We can give a full URL as well. For instance:

<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash.js/4.17.11/lodash.js"></script>

To attach several scripts, use multiple tags:

<script src="/js/script1.js"></script>

<script src="/js/script2.js"></script>

…

**Please note:**

As a rule, only the simplest scripts are put into HTML. More complex ones reside in separate files.

The benefit of a separate file is that the browser will download it and store it in its [cache](https://en.wikipedia.org/wiki/Web_cache).

Other pages that reference the same script will take it from the cache instead of downloading it, so the file is actually downloaded only once.

That reduces traffic and makes pages faster.

**If src is set, the script content is ignored.**

A single <script> tag can’t have both the src attribute and code inside.

This won’t work:

<script src="file.js">

alert(1); // the content is ignored, because src is set

</script>

We must choose either an external <script src="…"> or a regular <script> with code.

The example above can be split into two scripts to work:

<script src="file.js"></script>

<script>

alert(1);

</script>

**The modern mode, "use strict"**

* The "use strict" directive was new in ECMAScript version 5.
* It is not a statement, but a literal expression, ignored by earlier versions of JavaScript.
* The purpose of "use strict" is to indicate that the code should be executed in "strict mode".
* With strict mode, you can not, for example, use undeclared variables.

"use strict";

x = 3.14; // This will cause an error because x is not declared

**Why Strict Mode?**

Strict mode makes it easier to write "secure" JavaScript.

Strict mode changes previously accepted "bad syntax" into real errors.

**Variables**

Most of the time, a JavaScript application needs to work with information. Here are two examples:

An online shop – the information might include goods being sold and a shopping cart.

A chat application – the information might include users, messages, and much more.

Variables are used to store this information.

A [variable](https://en.wikipedia.org/wiki/Variable_(computer_science)) is a “named storage” for data. We can use variables to store goodies, visitors, and other data.

To create a variable in JavaScript, use the let keyword.

The statement below creates (in other words: *declares*) a variable with the name “message”:

Eg :-

Let message;

To assign data to the variable , we use assignment operator =

Let message =”Hello”

**Case matters**

Variables named apple and APPLE are two different variables.

**Reserved names**

There is a [list of reserved words](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Keywords), which cannot be used as variable names because they are used by the language itself.

For example: let, class, return, and function are reserved.

The code below gives a syntax error:

let let = 5; // can't name a variable "let", error!

let return = 5; // also can't name it "return", error!

**Constants**

To declare a constant (unchanging) variable, use const instead of let:

const myBirthday = '18.04.1982';

Variables declared using const are called “constants”. They cannot be reassigned. An attempt to do so would cause an error:

const myBirthday = '18.04.1982';

myBirthday = '01.01.2001'; // error, can't reassign the constant!

When a programmer is sure that a variable will never change, they can declare it with const to guarantee and clearly communicate that fact to everyone.

**“The old var”**

Var keyword is used in old scripts to declare a variable, but in modern scripts it is not used. We need to understand difference between var and let because to avoid errors while migrating scripts from var to let.

**Var has no block Scope**

Variables declared with var are either function scopes or block scoped. These are visible through the blocks.

For instance:

**if (true) {**

**var test = true; // use "var" instead of "let"**

**}**

**alert(test); // true, the variable lives after if**

As var ignores code blocks, we’ve got a global variable test.

If we used let test instead of var test, then the variable would only be visible inside if:

**if (true) {**

**let test = true; // use "let"**

**}**

**alert(test); // ReferenceError: test is not defined**

The same thing for loops: var cannot be block- or loop-local:

**for (var i = 0; i < 10; i++) {**

**var one = 1;**

**// ...**

**}**

**alert(i); // 10, "i" is visible after loop, it's a global variable**

**alert(one); // 1, "one" is visible after loop, it's a global variable**

If a code block is inside a function, then var becomes a function-level variable:

**function sayHi() {**

**if (true) {**

**var phrase = "Hello";**

**}**

**alert(phrase); // works**

**}**

**sayHi();**

**alert(phrase); // ReferenceError: phrase is not defined**

**Var tolerates redeclaration:-**

If we declare the same variable with let twice in the same scope, that’s an error:

let user;

let user; // SyntaxError: 'user' has already been declared

With var, we can redeclare a variable any number of times. If we use var with an already-declared variable, it’s just ignored:

var user = "Pete";

var user = "John"; // this "var" does nothing (already declared)

// ...it doesn't trigger an error

alert(user); // John

for more: visit <https://javascript.info/var>