Engineer

JavaScript

the language browsers run

What it is

JavaScript is a dynamically typed programming language, standardised as ECMAScript, that every web browser can execute and that also runs on servers and build machines through runtimes such as Node.js. People shorten it to JS. The standard's yearly editions are named ES2015, ES2024 and so on, the older numbering (ES5, ES6) is still heard, and "vanilla JavaScript" means the language with no framework on top. Apart from the name and some curly-brace syntax, it shares little with Java.

Brendan Eich wrote the first version at Netscape, and it shipped in the Navigator 2.0 browser. Ecma International adopted the first edition of the standard, ECMA-262, in June 1997, and its committee TC39 has published a new edition every year since ECMAScript 2016. A proposal joins the specification only at stage 4 of the committee's process, which requires implementations that already ship, so the text and the engines seldom disagree for long.

The language says very little about the outside world. ECMA-262 defines the syntax, the types, objects, promises and modules, and leaves input and output to whatever hosts it. In a browser that environment is the web platform: the document, events, timers and network requests, specified mostly by the HTML Standard and its sibling standards. Node.js supplies files, sockets and processes instead. That split explains a common surprise: code can be perfectly valid JavaScript and still break when it moves from a page to a server, because the language travelled and its environment stayed behind.

In a browser, a page's scripts share one main thread with layout, painting and input. A task runs to completion before the browser can respond to the next click. Promises and `async` functions keep waiting off that thread, while

computation stays on it unless it is moved to a Web Worker. Numbers are IEEE 754 double-precision values, so whole numbers are exact only up to 2^53 − 1, and anything larger needs `BigInt` .

Use the concept in a decision

Put “What is JavaScript?” to work in a real decision. We can help relate the concept to the system you are building or buying. Interfaces that load quickly, work on a phone and behave the same in every browser your customers happen to use.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Directed full-stack GIS map development — full-stack mapping, from database to browser

Directed full-stack GIS map development Get in touch →

Why it matters

The real decision is how much of it to send to the people reading your site. JavaScript is the language that makes a page behave without a round trip to a server, and every kilobyte of it is downloaded, parsed, compiled and run on a device you do not control. The bundle that feels instant on a developer’s laptop can hold a mid-range phone busy for seconds, and the reader pays for that in battery and patience.

Script is also where a page is most exposed. An attacker’s script, if it executes, runs with the page’s authority: it can read what the reader types and send it anywhere the page may send. That is the risk Content Security Policy exists to contain. Shipping no executable JavaScript removes the site’s own browser scripts as an attack surface; untrusted input still needs safe handling and a restrictive policy. The package ecosystem adds a second exposure. A typical front-end build pulls in hundreds of transitive dependencies, and each one is code that executes with the same rights as yours.

The other side of the ledger is real. Running one language on both sides of a system lets the browser and the server share a validation rule or a data shape, and some products need a lot of script: offline state, drag and drop, maps, live collaboration. The useful question is asked per feature: does this need to happen in the reader’s browser without asking a server? If it does not, render it on the server or at build time and ship HTML.

The related choice is JavaScript vs TypeScript. TypeScript adds a type checker and a build step and compiles back to plain JavaScript. For a codebase several people change every week, the checker usually pays for itself; for a hundred-line enhancement on a static page, JavaScript and a strict linter are enough.

What it looks like in practice

This site delivers static pages without executable JavaScript to its readers. Its search code is retained but disabled; JavaScript still runs in the build and checking tools. Three pages show the earlier browser exception and the work done outside the browser.

An earlier three-language version had no runtime backend and delivered a single first-party script. The Hugo build record describes that scoped exception: a portfolio filter whose controls stayed hidden until the script ran. Its 108 template files, including 53 partials and 18 shortcodes, did their work when the site was built. The record also names the cost of that choice at the time: interactive features had to be resolved in the build, and a search index was costed and parked rather than shipped. Those figures and that exception describe the earlier implementation, not the code now delivered to a reader.

The same content is published over Gemini and Gopher, two protocols with no place for a script at all. One build wrote 777 documents in each format at the recorded implementation stage from the same source as the pages. A separate build comparison confirmed zero HTML bytes changed when those output formats were added; the result does not cover later editorial or visual changes. The site argues that a document does not need a megabyte of script to be read, and a format that cannot hold one is the plainest way to show it.

Most of the JavaScript here lives in the tooling. At the implementation stage recorded by the linked work, the site's quality gate comprised thirty-two scripts and 10,242 lines of JavaScript, then the largest body of code in the repository. For a while it was the one body of code that nothing formatted, linted or type-checked. Bringing it under a formatter and a linter began by fitting the formatter to the code: the repository's four-space default would have rewritten 2,173 lines of the largest checker, and a two-space override for those files left 38 lines of real drift. The linter then raised 13 findings, and all 13 were fixed with none suppressed. A bug in a checker shows itself as a check that passes when it should fail, so the gate's own code gets the treatment it gives everything else.

Where it goes wrong

The failures below follow from how the language is built, and each has a known remedy.

The page renders, then ignores taps for a second or more. A long task, such as hydrating a large bundle, parsing a big JSON response or sorting thousands of rows, holds the one thread that handles input. Ship less code for the first interaction by splitting bundles per route and deferring the rest, break long work into pieces that yield back to the event loop, and move heavy computation into a Web Worker. Measure on a slow phone rather than on the laptop that wrote the code.

Prices and identifiers come out slightly wrong. Every number is a binary floating-point value, so decimal fractions round and very large integers collapse into their neighbours. A 64-bit database key sent as a JSON number is the usual victim:

```
0.1 + 0.2 === 0.3; // false
Number.MAX_SAFE_INTEGER; // 9007199254740991
9007199254740993 === 9007199254740992; // true
```

Keep money in integer minor units such as cents, send large identifiers as strings, and use `BigInt` where the arithmetic has to be exact.

A comparison lets the wrong value through. The `==` operator converts its operands before comparing them, so `"" == 0` and `"0" == false` are both true. Use `===` throughout and let the linter's `eqeqeq` rule enforce it, so the conversion is something the code asks for out loud.

Text from a user ends up running as code. Assigning untrusted text to `innerHTML`, building markup by string concatenation or handing input to `eval` turns data into script. Write text with `textContent`, send a Content Security Policy that admits scripts only by nonce or hash, and keep the dependency list short and pinned in a lockfile, reading what each update pulls in.

Code that works in the browser crashes on the server. The language moved and its environment did not: `window` and `document` exist only in a browser, and Node.js reads a file as CommonJS or as an ES module depending on settings the browser never sees. Keep runtime-specific calls at the edges, set `"type": "module"` explicitly in `package.json`, and run each piece of code in the environment it will actually run in.

Further reading

- ECMAScript Language Specification — the living draft of ECMA-262, the text every engine implements, updated as proposals reach stage 4.
- The TC39 Process — the stages a proposal passes through before it becomes part of the language, and what each one demands.
- ECMA-262 at Ecma International — the published yearly editions, for when a project has to name the version it targets.
- HTML Standard: event loops — how a browser schedules tasks, microtasks and rendering around a page's scripts.
- HTML Standard: the script element — classic and module scripts, and how `async` and `defer` change when they run.



Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is JavaScript?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/javascript/