WIKIPEDIA
The Free Encyclopedia

JavaScript

JavaScript (/ˈdʒɑːvəskrɪpt/), often abbreviated as **JS**, is a programming language and core technology of the Web, alongside HTML and CSS. Ninety-nine percent of websites use JavaScript on the client side for webpage behavior.[10]

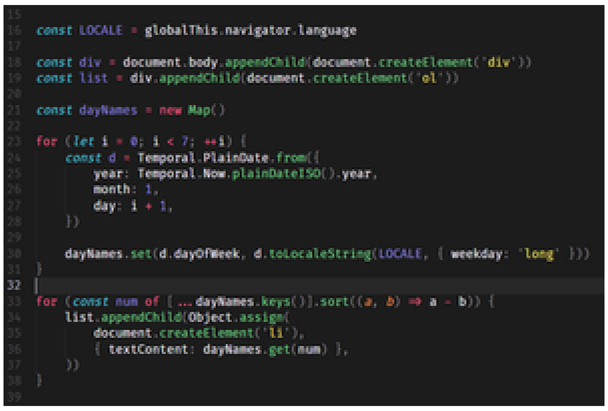
Web browsers have a dedicated JavaScript engine that executes the client code. These engines are also utilized in some servers and a variety of apps. The most popular runtime system for non-browser usage is Node.js.

JavaScript is a high-level, often just-in-time compiled language that conforms to the ECMAScript standard.[11] It has dynamic typing, prototype-based object-orientation, and first-class functions. It is multi-paradigm, supporting event-driven, functional, and imperative programming styles. It has application programming interfaces (APIs) for working with text, dates, regular expressions, standard data structures, and the Document Object Model (DOM).

The ECMAScript standard does not include any input/output (I/O), such as networking, storage, or graphics facilities. In practice, the web browser or other runtime system provides JavaScript APIs for I/O.

Although Java and JavaScript are similar in name and syntax, the two languages are distinct and differ greatly in design.

JavaScript



```
const LOCALE = globalThis.navigator.language

const div = document.body.appendChild(document.createElement('div'))
const list = div.appendChild(document.createElement('ol'))

const dayNames = new Map()

for (let i = 0; i < 7; ++i) {
    const d = Temporal.PlainDate.from({
        year: Temporal.Now.plainDateISO().year,
        month: 1,
        day: i + 1,
    })

    dayNames.set(d.dayOfWeek, d.toLocaleString(LOCALE, { weekday: 'long' }))
}

for (const num of [...dayNames.keys()].sort((a, b) => a - b)) {
    list.appendChild(Object.assign(
        document.createElement('li'),
        { textContent: dayNames.get(num) },
    ))
}
```

Screenshot of JavaScript source code

Paradigm	Multi-paradigm: event-driven, functional, imperative, procedural, object-oriented
Designed by	Brendan Eich of Netscape initially; others have also contributed to the ECMAScript standard
First appeared	4 December 1995[1]
Stable release	ECMAScript 2024[2] / June 2024
Preview release	ECMAScript 2025[3] / 27 March 2024
Typing discipline	Dynamic, weak, duck
Memory management	Garbage collection
Filename extensions	`.js` • `.cjs` • `.mjs`[4]
Website	ecma-international.org/publications-and-standards/standards/ecma-262/ (https://ecma-international.org/publications-and-standards/standards/ecma-262/)

Major implementations

V8, JavaScriptCore, SpiderMonkey, Chakra

History

Creation at Netscape

The first popular web browser with a graphical user interface, Mosaic, was released in 1993. Accessible to non-technical people, it played a prominent role in the rapid growth of the early World Wide Web.[12] The lead developers of Mosaic then founded the Netscape corporation, which released a more polished browser, Netscape Navigator, in 1994. This quickly became the most-used.[13]

Influenced by
Java,[5][6] Scheme,[6] Self,[7] AWK,[8] HyperTalk[9]
Influenced
ActionScript, ArkTS, AssemblyScript, CoffeeScript, Dart, Haxe, JS++, Opa, TypeScript
JavaScript at Wikibooks

During these formative years of the Web, web pages could only be static, lacking the capability for dynamic behavior after the page was loaded in the browser. There was a desire in the flourishing web development scene to remove this limitation, so in 1995, Netscape decided to add a programming language to Navigator. They pursued two routes to achieve this: collaborating with Sun Microsystems to embed the Java language, while also hiring Brendan Eich to embed the Scheme language.[6]

The goal was a "language for the masses",[14] "to help nonprogrammers create dynamic, interactive Web sites".[15] Netscape management soon decided that the best option was for Eich to devise a new language, with syntax similar to Java and less like Scheme or other extant scripting languages.[5][6] Although the new language and its interpreter implementation were called LiveScript when first shipped as part of a Navigator beta in September 1995, the name was changed to JavaScript for the official release in December.[6][1][16][17]

The choice of the JavaScript name has caused confusion, implying that it is directly related to Java. At the time, the dot-com boom had begun and Java was a popular new language, so Eich considered the JavaScript name a marketing ploy by Netscape.[14]

Adoption by Microsoft

Microsoft debuted Internet Explorer in 1995, leading to a browser war with Netscape. On the JavaScript front, Microsoft created its own interpreter called JScript.[18]

Microsoft first released JScript in 1996, alongside initial support for CSS and extensions to HTML. Each of these implementations was noticeably different from their counterparts in Netscape Navigator.[19][20] These differences made it difficult for developers to make their websites work well in both browsers, leading to widespread use of "best viewed in Netscape" and "best viewed in Internet Explorer" logos for several years.[19][21]

The rise of JScript

In November 1996, Netscape submitted JavaScript to Ecma International, as the starting point for a standard specification that all browser vendors could conform to. This led to the official release of the first ECMAScript language specification in June 1997.

The standards process continued for a few years, with the release of ECMAScript 2 in June 1998 and ECMAScript 3 in December 1999. Work on ECMAScript 4 began in 2000.[18]

Brendan Eich later said of this period: "It's still kind of a sidekick language. It's considered slow or annoying. People do pop-ups or those scrolling

However, the effort to fully standardize the language was undermined by Microsoft gaining an increasingly dominant position in the browser market. By the early 2000s, Internet Explorer's market share reached 95%.[22] This meant that JScript became the de facto standard for client-side scripting on the Web.

messages in the old status bar at the bottom of your old browser."[14]

Microsoft initially participated in the standards process and implemented some proposals in its JScript language, but eventually it stopped collaborating on ECMA work. Thus ECMAScript 4 was mothballed.

Growth and standardization

During the period of Internet Explorer dominance in the early 2000s, client-side scripting was stagnant. This started to change in 2004, when the successor of Netscape, Mozilla, released the Firefox browser. Firefox was well received by many, taking significant market share from Internet Explorer.[23]

In 2005, Mozilla joined ECMA International, and work started on the ECMAScript for XML (E4X) standard. This led to Mozilla working jointly with Macromedia (later acquired by Adobe Systems), who were implementing E4X in their ActionScript 3 language, which was based on an ECMAScript 4 draft. The goal became standardizing ActionScript 3 as the new ECMAScript 4. To this end, Adobe Systems released the Tamarin implementation as an open source project. However, Tamarin and ActionScript 3 were too different from established client-side scripting, and without cooperation from Microsoft, ECMAScript 4 never reached fruition.

Meanwhile, very important developments were occurring in open-source communities not affiliated with ECMA work. In 2005, Jesse James Garrett released a white paper in which he coined the term Ajax and described a set of technologies, of which JavaScript was the backbone, to create web applications where data can be loaded in the background, avoiding the need for full page reloads. This sparked a renaissance period of JavaScript, spearheaded by open-source libraries and the communities that formed around them. Many new libraries were created, including jQuery, Prototype, Dojo Toolkit, and MooTools.

Google debuted its Chrome browser in 2008, with the V8 JavaScript engine that was faster than its competition.[24][25] The key innovation was just-in-time compilation (JIT),[26] so other browser vendors needed to overhaul their engines for JIT.[27]

In July 2008, these disparate parties came together for a conference in Oslo. This led to the eventual agreement in early 2009 to combine all relevant work and drive the language forward. The result was the ECMAScript 5 standard, released in December 2009.

Reaching maturity

Ambitious work on the language continued for several years, culminating in an extensive collection of additions and refinements being formalized with the publication of ECMAScript 6 in 2015.[28]

The creation of Node.js in 2009 by Ryan Dahl sparked a significant increase in the usage of JavaScript outside of web browsers. Node combines the V8 engine, an event loop, and I/O APIs, thereby providing a stand-alone JavaScript runtime system.[29][30] As of 2018, Node had been used by millions of developers,[31] and npm had the most modules of any package manager in the world.[32]

The ECMAScript draft specification is currently maintained openly on GitHub,[33] and editions are produced via regular annual snapshots.[33] Potential revisions to the language are vetted through a comprehensive proposal process.[34][35] Now, instead of edition numbers, developers check the status of upcoming features individually.[33]

The current JavaScript ecosystem has many libraries and frameworks, established programming practices, and substantial usage of JavaScript outside of web browsers.[17] Plus, with the rise of single-page applications and other JavaScript-heavy websites, several transpilers have been created to aid the development process.[36]

Trademark

"JavaScript" is a trademark of Oracle Corporation in the United States.[37][38] The trademark was originally issued to Sun Microsystems on 6 May 1997, and was transferred to Oracle when they acquired Sun in 2009.[39][40]

A letter was circulated in September 2024, spearheaded by Ryan Dahl, calling on Oracle to free the JavaScript trademark.[41] Brendan Eich, the original creator of JavaScript, was among the over 14,000 signatories who supported the initiative.

Website client-side usage

JavaScript is the dominant client-side scripting language of the Web, with 99% of all websites using it for this purpose.[10] Scripts are embedded in or included from HTML documents and interact with the DOM.

All major web browsers have a built-in JavaScript engine that executes the code on the user's device.

Examples of scripted behavior

- Loading new web page content without reloading the page, via Ajax or a WebSocket. For example, users of social media can send and receive messages without leaving the current page.
- Web page animations, such as fading objects in and out, resizing, and moving them.
- Playing browser games.
- Controlling the playback of streaming media.
- Generating pop-up ads or alert boxes.
- Validating input values of a web form before the data is sent to a web server.
- Logging data about the user's behavior then sending it to a server. The website owner can use this data for analytics, ad tracking, and personalization.
- Redirecting a user to another page.
- Storing and retrieving data on the user's device, via the storage or IndexedDB standards.

Libraries and frameworks

Over 80% of websites use a third-party JavaScript library or web framework as part of their client-side scripting.[42]

jQuery is by far the most-used.[42] Other notable ones include Angular, Bootstrap, Lodash, Modernizr, React, Underscore, and Vue.[42] Multiple options can be used in conjunction, such as jQuery and Bootstrap.[43]

However, the term "Vanilla JS" was coined for websites not using any libraries or frameworks at all, instead relying entirely on standard JavaScript functionality.[44]

Other usage

The use of JavaScript has expanded beyond its web browser roots. JavaScript engines are now embedded in a variety of other software systems, both for server-side website deployments and non-browser applications.

Initial attempts at promoting server-side JavaScript usage were Netscape Enterprise Server and Microsoft's Internet Information Services,[45][46] but they were small niches.[47] Server-side usage eventually started to grow in the late 2000s, with the creation of Node.js and other approaches.[47]

Electron, Cordova, React Native, and other application frameworks have been used to create many applications with behavior implemented in JavaScript. Other non-browser applications include Adobe Acrobat support for scripting PDF documents[48] and GNOME Shell extensions written in JavaScript.[49]

JavaScript has been used in some embedded systems, usually by leveraging Node.js.[50][51][52]

Execution

JavaScript engine

A JavaScript engine is a software component that executes JavaScript code. The first JavaScript engines were mere interpreters, but all relevant modern engines use just-in-time compilation for improved performance.[53]

JavaScript engines are typically developed by web browser vendors, and every major browser has one. In a browser, the JavaScript engine runs in concert with the rendering engine via the Document Object Model and Web IDL bindings.[54] However, the use of JavaScript engines is not limited to browsers; for example, the V8 engine is a core component of the Node.js runtime system.[55]

Since ECMAScript is the standardized specification of JavaScript, ECMAScript engine is another name for these implementations. With the advent of WebAssembly, some engines can also execute this code in the same sandbox as regular JavaScript code.[56][55]

Runtime system

A JavaScript engine must be embedded within a runtime system (such as a web browser or a standalone system) to enable scripts to interact with the broader environment. The runtime system includes the necessary APIs for input/output operations, such as networking, storage, and graphics, and provides the ability to import scripts.

JavaScript is a single-threaded language. The runtime processes messages from a queue one at a time, and it calls a function associated with each new message, creating a call stack frame with the function's arguments and local variables. The call stack shrinks and grows based on the function's needs. When the call stack is empty upon function completion, JavaScript proceeds to the next message in the queue. This is called the event loop, described as "run to completion" because each message is fully processed before the next message is considered. However, the language's concurrency model describes the event loop as non-blocking: program I/O is performed using events and callback functions. This means, for example, that JavaScript can process a mouse click while waiting for a database query to return information.[57]

The notable standalone runtimes are Node.js, Deno, and Bun.

Features

The following features are common to all conforming ECMAScript implementations unless explicitly specified otherwise.

Imperative and structured

JavaScript supports much of the structured programming syntax from C (e.g., `if` statements, `while` loops, `switch` statements, `do while` loops, etc.). One partial exception is scoping: originally JavaScript only had function scoping with `var`; block scoping was added in ECMAScript 2015 with the keywords `let` and `const`. Like C, JavaScript makes a distinction between expressions and statements. One syntactic difference from C is automatic semicolon insertion, which allow semicolons (which terminate statements) to be omitted.[58]

Weakly typed

JavaScript is weakly typed, which means certain types are implicitly cast depending on the operation used.[59]

- The binary + operator casts both operands to a string unless both operands are numbers. This is because the addition operator doubles as a concatenation operator
- The binary `-` operator always casts both operands to a number
- Both unary operators (`+`, `-`) always cast the operand to a number. However, `+` always casts to `Number` (binary64) while `-` preserves `BigInt` (integer)[60]

Values are cast to strings like the following:[59]

- Strings are left as-is
- Numbers are converted to their string representation
- Arrays have their elements cast to strings after which they are joined by commas (`,`)

- Other objects are converted to the string `[object Object]` where `Object` is the name of the constructor of the object

Values are cast to numbers by casting to strings and then casting the strings to numbers. These processes can be modified by defining `toString` and `valueOf` functions on the prototype for string and number casting respectively.

JavaScript has received criticism for the way it implements these conversions as the complexity of the rules can be mistaken for inconsistency.[61][59] For example, when adding a number to a string, the number will be cast to a string before performing concatenation, but when subtracting a number from a string, the string is cast to a number before performing subtraction.

JavaScript type conversions

left operand	operator	right operand	result
`[]` (empty array)	+	`[]` (empty array)	`""` (empty string)
`[]` (empty array)	+	`{}` (empty object)	`"[object Object]"` (string)
`false` (boolean)	+	`[]` (empty array)	`"false"` (string)
`"123"`(string)	+	1 (number)	`"1231"` (string)
`"123"` (string)	-	1 (number)	122 (number)
`"123"` (string)	-	`"abc"` (string)	`NaN` (number)

Often also mentioned is `{} + []` resulting in `0` (number). This is misleading: the `{}` is interpreted as an empty code block instead of an empty object, and the empty array is cast to a number by the remaining unary `+` operator. If the expression is wrapped in parentheses - `({} + [])` – the curly brackets are interpreted as an empty object and the result of the expression is `"[object Object]"` as expected.[59]

Dynamic

Typing

JavaScript is dynamically typed like most other scripting languages. A type is associated with a value rather than an expression. For example, a variable initially bound to a number may be reassigned to a string.[62] JavaScript supports various ways to test the type of objects, including duck typing.[63]

Run-time evaluation

JavaScript includes an `eval` function that can execute statements provided as strings at run-time.

Object-orientation (prototype-based)

Prototypal inheritance in JavaScript is described by Douglas Crockford as:

> You make prototype objects, and then ... make new instances. Objects are mutable in JavaScript, so we can augment the new instances, giving them new fields and methods. These can then act as prototypes for even newer objects. We don't need classes to make lots of similar objects... Objects inherit from objects. What could be more object oriented than that?[64]

In JavaScript, an object is an associative array, augmented with a prototype (see below); each key provides the name for an object property, and there are two syntactical ways to specify such a name: dot notation (`obj.x = 10`) and bracket notation (`obj['x'] = 10`). A property may be added, rebound, or deleted at run-time. Most properties of an object (and any property that belongs to an object's prototype inheritance chain) can be enumerated using a `for...in` loop.

Prototypes

JavaScript uses prototypes where many other object-oriented languages use classes for inheritance.[65] It is possible to simulate many class-based features with prototypes in JavaScript.[66]

Functions as object constructors

Functions double as object constructors, along with their typical role. Prefixing a function call with *new* will create an instance of a prototype, inheriting properties and methods from the constructor (including properties from the `Object` prototype).[67] ECMAScript 5 offers the `Object.create` method, allowing explicit creation of an instance without automatically inheriting from the `Object` prototype (older environments can assign the prototype to `null`).[68] The constructor's `prototype` property determines the object used for the new object's internal prototype. New methods can be added by modifying the prototype of the function used as a constructor. JavaScript's built-in constructors, such as `Array` or `Object`, also have prototypes that can be modified. While it is possible to modify the `Object` prototype, it is generally considered bad practice because most objects in JavaScript will inherit methods and properties from the `Object` prototype, and they may not expect the prototype to be modified.[69]

Functions as methods

Unlike in many object-oriented languages, in JavaScript there is no distinction between a function definition and a method definition. Rather, the distinction occurs during function calling. When a function is called as a method of an object, the function's local *this* keyword is bound to that object for that invocation.

Functional

JavaScript functions are first-class; a function is considered to be an object.[70] As such, a function may have properties and methods, such as `.call()` and `.bind()`.[71]

Lexical closure

A *nested* function is a function defined within another function. It is created each time the outer function is invoked.

In addition, each nested function forms a lexical closure: the lexical scope of the outer function (including any constant, local variable, or argument value) becomes part of the internal state of each inner function object, even after execution of the outer function concludes.[72]

Anonymous function

JavaScript also supports anonymous functions.

Delegative

JavaScript supports implicit and explicit delegation.

Functions as roles (Traits and Mixins)

JavaScript natively supports various function-based implementations of Role[73] patterns like Traits[74][75] and Mixins.[76] Such a function defines additional behavior by at least one method bound to the `this` keyword within its `function` body. A Role then has to be delegated explicitly via `call` or `apply` to objects that need to feature additional behavior that is not shared via the prototype chain.

Object composition and inheritance

Whereas explicit function-based delegation does cover composition in JavaScript, implicit delegation already happens every time the prototype chain is walked in order to, e.g., find a method that might be related to but is not directly owned by an object. Once the method is found it gets called within this object's context. Thus inheritance in JavaScript is covered by a delegation automatism that is bound to the prototype property of constructor functions.

Miscellaneous

Zero-based numbering

JavaScript is a zero-index language.

Variadic functions

An indefinite number of parameters can be passed to a function. The function can access them through formal parameters and also through the local `arguments` object. Variadic functions can also be created by using the `bind (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function/bind)` method.

Array and object literals

Like in many scripting languages, arrays and objects (associative arrays in other languages) can each be created with a succinct shortcut syntax. In fact, these literals form the basis of the JSON data format.

Regular expressions

In a manner similar to Perl, JavaScript also supports regular expressions, which provide a concise and powerful syntax for text manipulation that is more sophisticated than the built-in string functions.[77]

Promises and Async/await

JavaScript supports promises and Async/await for handling asynchronous operations.

Promises

A built-in Promise object provides functionality for handling promises and associating handlers with an asynchronous action's eventual result. Recently, the JavaScript specification introduced combinator methods, which allow developers to combine multiple JavaScript promises and do operations based on different scenarios. The methods introduced are: Promise.race, Promise.all, Promise.allSettled and Promise.any.

Async/await

Async/await allows an asynchronous, non-blocking function to be structured in a way similar to an ordinary synchronous function. Asynchronous, non-blocking code can be written, with minimal overhead, structured similarly to traditional synchronous, blocking code.

Vendor-specific extensions

Historically, some JavaScript engines supported these non-standard features:

- conditional `catch` clauses (like Java)
- array comprehensions and generator expressions (like Python)
- concise function expressions (`function(args) expr`; this experimental syntax predated arrow functions)
- ECMAScript for XML (E4X), an extension that adds native XML support to ECMAScript (unsupported in Firefox since version 21[78])

Syntax

Variables in JavaScript can be defined using either the `var`,[79] `let`[80] or `const`[81] keywords. Variables defined without keywords will be defined at the global scope.

Arrow functions were first introduced in 6th Edition – ECMAScript 2015. They shorten the syntax for writing functions in JavaScript. Arrow functions are anonymous, so a variable is needed to refer to them in order to invoke them after their creation, unless surrounded by parenthesis and executed immediately.

Here is an example of JavaScript syntax.

```
// Declares a function-scoped variable named `x`, and implicitly assigns the
// special value `undefined` to it. Variables without value are automatically
// set to undefined.
// var is generally considered bad practice and let and const are usually preferred.
var x;

// Variables can be manually set to `undefined` like so
let x2 = undefined;
```

```
// Declares a block-scoped variable named `y`, and implicitly sets it to
// `undefined`. The `let` keyword was introduced in ECMAScript 2015.
let y;

// Declares a block-scoped, un-reassignable variable named `z`, and sets it to
// a string literal. The `const` keyword was also introduced in ECMAScript 2015,
// and must be explicitly assigned to.

// The keyword `const` means constant, hence the variable cannot be reassigned
// as the value is `constant`.
const z = "this value cannot be reassigned!";

// Declares a global-scoped variable and assigns 3.  This is generally considered
// bad practice, and will not work if strict mode is on.
t = 3;

// Declares a variable named `myNumber`, and assigns a number literal (the value
// `2`) to it.
let myNumber = 2;

// Reassigns `myNumber`, setting it to a string literal (the value `"foo"`).
// JavaScript is a dynamically-typed language, so this is legal.
myNumber = "foo";
```

Note the comments in the examples above, all of which were preceded with two forward slashes.

More examples can be found at the Wikibooks page on JavaScript syntax examples.

Security

JavaScript and the DOM provide the potential for malicious authors to deliver scripts to run on a client computer via the Web. Browser authors minimize this risk using two restrictions. First, scripts run in a sandbox in which they can only perform Web-related actions, not general-purpose programming tasks like creating files. Second, scripts are constrained by the same-origin policy: scripts from one website do not have access to information such as usernames, passwords, or cookies sent to another site. Most JavaScript-related security bugs are breaches of either the same origin policy or the sandbox.

There are subsets of general JavaScript—ADsafe, Secure ECMAScript (SES)—that provide greater levels of security, especially on code created by third parties (such as advertisements).[82][83] Closure Toolkit is another project for safe embedding and isolation of third-party JavaScript and HTML.[84]

Content Security Policy is the main intended method of ensuring that only trusted code is executed on a Web page.

Cross-site scripting

A common JavaScript-related security problem is cross-site scripting (XSS), a violation of the same-origin policy. XSS vulnerabilities occur when an attacker can cause a target Website, such as an online banking website, to include a malicious script in the webpage presented to a victim. The script in this example can then access the banking application with the privileges of the victim, potentially disclosing secret information or transferring money without the victim's authorization. One important solution to XSS vulnerabilities is HTML sanitization.

Some browsers include partial protection against *reflected* XSS attacks, in which the attacker provides a URL including malicious script. However, even users of those browsers are vulnerable to other XSS attacks, such as those where the malicious code is stored in a database. Only correct design of Web applications on the server-side can fully prevent XSS.

XSS vulnerabilities can also occur because of implementation mistakes by browser authors.[85]

Cross-site request forgery

Another cross-site vulnerability is cross-site request forgery (CSRF). In CSRF, code on an attacker's site tricks the victim's browser into taking actions the user did not intend at a target site (like transferring money at a bank). When target sites rely solely on cookies for request authentication, requests originating from code on the attacker's site can carry the same valid login credentials of the initiating user. In general, the solution to CSRF is to require an authentication value in a hidden form field, and not only in the cookies, to authenticate any request that might have lasting effects. Checking the HTTP Referrer header can also help.

"JavaScript hijacking" is a type of CSRF attack in which a `<script>` tag on an attacker's site exploits a page on the victim's site that returns private information such as JSON or JavaScript. Possible solutions include:

- requiring an authentication token in the POST and GET parameters for any response that returns private information.

Misplaced trust in the client

Developers of client-server applications must recognize that untrusted clients may be under the control of attackers. The author of an application should not assume that their JavaScript code will run as intended (or at all) because any secret embedded in the code could be extracted by a determined adversary. Some implications are:

- Website authors cannot perfectly conceal how their JavaScript operates because the raw source code must be sent to the client. The code can be obfuscated, but obfuscation can be reverse-engineered.
- JavaScript form validation only provides convenience for users, not security. If a site verifies that the user agreed to its terms of service, or filters invalid characters out of fields that should only contain numbers, it must do so on the server, not only the client.
- Scripts can be selectively disabled, so JavaScript cannot be relied on to prevent operations such as right-clicking on an image to save it.[86]
- It is considered very bad practice to embed sensitive information such as passwords in JavaScript because it can be extracted by an attacker.[87]
- Prototype pollution is a runtime vulnerability in which attackers can overwrite arbitrary properties in an object's prototype.

Misplaced trust in developers

Package management systems such as npm and Bower are popular with JavaScript developers. Such systems allow a developer to easily manage their program's dependencies upon other developers' program libraries. Developers trust that the maintainers of the libraries will keep them secure and up to date, but that is not always the case. A vulnerability has emerged because of this blind trust. Relied-upon libraries can have new releases that cause bugs or vulnerabilities to appear in all programs that rely upon the libraries. Inversely, a library can go unpatched with known vulnerabilities out in the wild. In a study done looking over a sample of 133,000 websites, researchers found 37% of the websites included a library with at least one known vulnerability.[88] "The median lag between the oldest library version used on each website and the newest available version of that library is 1,177 days in ALEXA, and development of some libraries still in active use ceased years ago."[88] Another possibility is that the maintainer of a library may remove the library entirely. This occurred in March 2016 when Azer Koçulu removed his repository from npm. This caused tens of thousands of programs and websites depending upon his libraries to break.[89][90]

Browser and plugin coding errors

JavaScript provides an interface to a wide range of browser capabilities, some of which may have flaws such as buffer overflows. These flaws can allow attackers to write scripts that would run any code they wish on the user's system. This code is not by any means limited to another JavaScript application. For example, a buffer overrun exploit can allow an attacker to gain access to the operating system's API with superuser privileges.

These flaws have affected major browsers including Firefox,[91] Internet Explorer,[92] and Safari.[93]

Plugins, such as video players, Adobe Flash, and the wide range of ActiveX controls enabled by default in Microsoft Internet Explorer, may also have flaws exploitable via JavaScript (such flaws have been exploited in the past).[94][95]

In Windows Vista, Microsoft has attempted to contain the risks of bugs such as buffer overflows by running the Internet Explorer process with limited privileges.[96] Google Chrome similarly confines its page renderers to their own "sandbox".

Sandbox implementation errors

Web browsers are capable of running JavaScript outside the sandbox, with the privileges necessary to, for example, create or delete files. Such privileges are not intended to be granted to code from the Web.

Incorrectly granting privileges to JavaScript from the Web has played a role in vulnerabilities in both Internet Explorer[97] and Firefox.[98] In Windows XP Service Pack 2, Microsoft demoted JScript's privileges in Internet Explorer.[99]

Microsoft Windows allows JavaScript source files on a computer's hard drive to be launched as general-purpose, non-sandboxed programs (see: Windows Script Host). This makes JavaScript (like VBScript) a theoretically viable vector for a Trojan horse, although JavaScript Trojan horses are uncommon in practice.[100]

Hardware vulnerabilities

In 2015, a JavaScript-based proof-of-concept implementation of a rowhammer attack was described in a paper by security researchers.[101][102][103][104]

In 2017, a JavaScript-based attack via browser was demonstrated that could bypass ASLR. It is called "ASLR⊕Cache" or AnC.[105][106]

In 2018, the paper that announced the Spectre attacks against Speculative Execution in Intel and other processors included a JavaScript implementation.[107]

Development tools

Important tools have evolved with the language.

- Every major web browser has built-in web development tools, including a JavaScript debugger.
- Static program analysis tools, such as ESLint and JSLint, scan JavaScript code for conformance to a set of standards and guidelines.
- Some browsers have built-in profilers. Stand-alone profiling libraries have also been created, such as benchmark.js and jsbench.[108][109]
- Many text editors have syntax highlighting support for JavaScript code.

Related technologies

Java

A common misconception is that JavaScript is directly related to Java. Both indeed have a C-like syntax (the C language being their most immediate common ancestor language). They are also typically sandboxed, and JavaScript was designed with Java's syntax and standard library in mind. In particular, all Java keywords were reserved in original JavaScript, JavaScript's standard library follows Java's naming conventions, and JavaScript's `Math` and `Date` objects are based on classes from Java 1.0.[110]

Both languages first appeared in 1995, but Java was developed by James Gosling of Sun Microsystems and JavaScript by Brendan Eich of Netscape Communications.

The differences between the two languages are more prominent than their similarities. Java has static typing, while JavaScript's typing is dynamic. Java is loaded from compiled bytecode, while JavaScript is loaded as human-readable source code. Java's objects are class-based, while JavaScript's are prototype-based. Finally, Java did not support functional programming until Java 8, while JavaScript has done so from the beginning, being influenced by Scheme.

JSON

JSON is a data format derived from JavaScript; hence the name JavaScript Object Notation. It is a widely used format supported by many other programming languages.

Transpilers

Many websites are JavaScript-heavy, so transpilers have been created to convert code written in other languages, which can aid the development process.[36]

TypeScript and CoffeeScript are two notable languages that transpile to JavaScript.

WebAssembly

WebAssembly is a newer language with a bytecode format designed to complement JavaScript, especially the performance-critical portions of web page scripts. All of the major JavaScript engines support WebAssembly,[111] which runs in the same sandbox as regular JavaScript code.

asm.js is a subset of JavaScript that served as the forerunner of WebAssembly.[112]

References

1. "Netscape and Sun announce JavaScript, the Open, Cross-platform Object Scripting Language for Enterprise Networks and the Internet" (https://web.archive.org/web/20070916144913/https://wp.netscape.com/newsref/pr/newsrelease67.html) (Press release). 4 December 1995. Archived from the original (https://wp.netscape.com/newsref/pr/newsrelease67.html) on 16 September 2007.
2. "ECMAScript® 2024 Language Specification" (https://262.ecma-international.org/15.0/). June 2024. Retrieved 30 August 2024.
3. "ECMAScript® 2025 Language Specification" (https://tc39.es/ecma262/). 27 March 2024. Retrieved 17 April 2024.
4. "nodejs/node-eps" (https://github.com/nodejs/node-eps/blob/master/002-es-modules.md). *GitHub*. Archived (https://web.archive.org/web/20200829024713/https://github.com/nodejs/node-eps/blob/master/002-es-modules.md) from the original on 29 August 2020. Retrieved 5 July 2018.
5. Seibel, Peter (16 September 2009). *Coders at Work: Reflections on the Craft of Programming* (https://books.google.com/books?id=nneBa6-mWfgC&q=The+immediate+concern+at+Netscape+was+it+must+look+like+Java.&pg=PA141). Apress. ISBN 978-1-4302-1948-4. Archived (https://web.archive.org/web/20201224233514/https://books.google.com/books?id=nneBa6-mWfgC&q=The+immediate+concern+at+Netscape+was+it+must+look+like+Java.&pg=PA141) from the original on 24 December 2020. Retrieved 25 December 2018. "Eich: The immediate concern at Netscape was it must look like Java."
6. "Chapter 4. How JavaScript Was Created" (https://exploringjs.com/es5/ch04.html). *speakingjs.com*. Archived (https://web.archive.org/web/20200227184037/https://speakingjs.com/es5/ch04.html) from the original on 27 February 2020. Retrieved 21 November 2017.
7. "Popularity – Brendan Eich" (https://brendaneich.com/2008/04/popularity/).
8. "Brendan Eich: An Introduction to JavaScript, JSConf 2010" (https://www.youtube.com/watch?v=1EyRscXrehw). *YouTube*. 20 January 2013. p. 22m. Archived (https://web.archive.org/web/20200829024704/https://www.youtube.com/watch?v=1EyRscXrehw) from the original on 29 August 2020. Retrieved 25 November 2019. "Eich: "function", eight letters, I was influenced by AWK."
9. Eich, Brendan (1998). "Foreword". In Goodman, Danny (ed.). *JavaScript Bible* (https://archive.org/details/javascriptbible000good) (3rd ed.). John Wiley & Sons. ISBN 0-7645-3188-3. LCCN 97078208 (https://lccn.loc.gov/97078208). OCLC 38888873 (https://search.worldcat.org/oclc/38888873). OL 712205M (https://openlibrary.org/books/OL712205M).
10. "Usage Statistics of JavaScript as Client-side Programming Language on Websites" (https://w3techs.com/technologies/details/cp-javascript). *W3Techs*. Retrieved 27 February 2024.
11. "ECMAScript 2020 Language Specification" (https://tc39.es/ecma262/#sec-overview). Archived (https://web.archive.org/web/20200508053013/https://tc39.es/ecma262/#sec-overview) from the original on 8 May 2020. Retrieved 8 May 2020.

12. "Bloomberg Game Changers: Marc Andreessen" (https://www.bloomberg.com/video/67758394). *Bloomberg*. Bloomberg. 17 March 2011. Archived (https://web.archive.org/web/20120516093712/https://www.bloomberg.com/video/67758394/) from the original on 16 May 2012. Retrieved 7 December 2011.
13. Enzer, Larry (31 August 2018). "The Evolution of the Web Browsers" (https://web.archive.org/web/20180831174847/https://www.mwdwebsites.com/nj-web-design-web-browsers.html). *Monmouth Web Developers*. Archived from the original (https://www.mwdwebsites.com/nj-web-design-web-browsers.html) on 31 August 2018. Retrieved 31 August 2018.
14. Fin JS (17 June 2016), "Brendan Eich – CEO of Brave" (https://www.youtube.com/watch?v=XOmhtfTrRxc), *YouTube*, retrieved 7 February 2018
15. "Netscape Communications Corp.", Browser enhancements. Encyclopædia Britannica 2006 Ultimate Reference Suite DVD
16. "TechVision: Innovators of the Net: Brendan Eich and JavaScript" (https://web.archive.org/web/20080208124612/https://wp.netscape.com/comprod/columns/techvision/innovators_be.html). Archived from the original (https://wp.netscape.com/comprod/columns/techvision/innovators_be.html) on 8 February 2008.
17. Han, Sheon (4 March 2024). "JavaScript Runs the World—Maybe Even Literally" (https://www.wired.com/story/javascript-runs-the-world-maybe-literally/). *Wired*. Retrieved 21 August 2024.
18. "Chapter 5. Standardization: ECMAScript" (https://web.archive.org/web/20211101184346/http://speakingjs.com/es5/ch05.html). *speakingjs.com*. Archived from the original (https://speakingjs.com/es5/ch05.html) on 1 November 2021. Retrieved 1 November 2021.
19. Champeon, Steve (6 April 2001). "JavaScript, How Did We Get Here?" (https://web.archive.org/web/20160719020828/https://archive.oreilly.com/pub/a/javascript/2001/04/06/js_history.html). *oreilly.com*. Archived from the original (https://archive.oreilly.com/pub/a/javascript/2001/04/06/js_history.html) on 19 July 2016. Retrieved 16 July 2016.
20. "Microsoft Internet Explorer 3.0 Beta Now Available" (https://news.microsoft.com/1996/05/29/microsoft-internet-explorer-3-0-beta-now-available/). *microsoft.com*. Microsoft. 29 May 1996. Archived (https://web.archive.org/web/20201124154053/https://news.microsoft.com/1996/05/29/microsoft-internet-explorer-3-0-beta-now-available/) from the original on 24 November 2020. Retrieved 16 July 2016.
21. McCracken, Harry (16 September 2010). "The Unwelcome Return of "Best Viewed with Internet Explorer" " (https://www.technologizer.com/2010/09/16/the-unwelcome-return-of-best-viewed-with-internet-explorer/). *technologizer.com*. Archived (https://web.archive.org/web/20180623192402/https://www.technologizer.com/2010/09/16/the-unwelcome-return-of-best-viewed-with-internet-explorer/) from the original on 23 June 2018. Retrieved 16 July 2016.
22. Baker, Loren (24 November 2004). "Mozilla Firefox Internet Browser Market Share Gains to 7.4%" (https://www.searchenginejournal.com/mozilla-firefox-internet-browser-market-share-gains-to-74/1082/). *Search Engine Journal*. Archived (https://web.archive.org/web/20210507013607/https://www.searchenginejournal.com/mozilla-firefox-internet-browser-market-share-gains-to-74/1082/) from the original on 7 May 2021. Retrieved 8 May 2021.
23. Weber, Tim (9 May 2005). "The assault on software giant Microsoft" (https://web.archive.org/web/20170925233936/https://news.bbc.co.uk/2/hi/business/4508897.stm). *BBC News*. Archived from the original (https://news.bbc.co.uk/2/hi/business/4508897.stm) on 25 September 2017.
24. "Big browser comparison test: Internet Explorer vs. Firefox, Opera, Safari and Chrome" (https://www.pcgameshardware.com/aid,687738/Big-browser-comparison-test-Internet-Explorer-vs-Firefox-Opera-Safari-and-Chrome-Update-Firefox-35-Final/Practice/). *PC Games Hardware*. Computec Media AG. 3 July 2009. Archived (https://web.archive.org/web/20120502043027/http://www.pcgameshardware.com/aid,687738/Big-browser-comparison-test-Internet-Explorer-vs-Firefox-Opera-Safari-and-Chrome-Update-Firefox-35-Final/Practice/) from the original on 2 May 2012. Retrieved 28 June 2010.

25. Purdy, Kevin (11 June 2009). "Lifehacker Speed Tests: Safari 4, Chrome 2" (https://lifehacker.com/lifehacker-speed-tests-safari-4-chrome-2-and-more-5286869). *Lifehacker*. Archived (https://web.archive.org/web/20210414095403/https://lifehacker.com/lifehacker-speed-tests-safari-4-chrome-2-and-more-5286869) from the original on 14 April 2021. Retrieved 8 May 2021.
26. "TraceMonkey: JavaScript Lightspeed, Brendan Eich's Blog" (https://brendaneich.com/2008/08/tracemonkey-javascript-lightspeed/). Archived (https://web.archive.org/web/20151204091540/https://brendaneich.com/2008/08/tracemonkey-javascript-lightspeed/) from the original on 4 December 2015. Retrieved 22 July 2020.
27. "Mozilla asks, 'Are we fast yet?' " (https://www.wired.com/2010/09/mozilla-asks-are-we-fast-yet/). *Wired*. Archived (https://web.archive.org/web/20180622213244/https://www.wired.com/2010/09/mozilla-asks-are-we-fast-yet/) from the original on 22 June 2018. Retrieved 18 January 2019.
28. "ECMAScript 6: New Features: Overview and Comparison" (https://web.archive.org/web/20180318064130/https://es6-features.org/). *es6-features.org*. Archived from the original on 18 March 2018. Retrieved 19 March 2018.
29. Professional Node.js: Building JavaScript Based Scalable Software (https://books.google.com/books?id=ZH6bpbcrlvYC&q=nodejs) Archived (https://web.archive.org/web/20170324021220/https://books.google.com/books?id=ZH6bpbcrlvYC&printsec=frontcover&dq=nodejs&hl=en&sa=X#v=onepage&q=nodejs&f=false) 2017-03-24 at the Wayback Machine, John Wiley & Sons, 01-Oct-2012
30. Sams Teach Yourself Node.js in 24 Hours (https://books.google.com/books?id=KGt-FxUEj48C&dq=nodejs&pg=PT24) Archived (https://web.archive.org/web/20170323192039/https://books.google.com/books?id=KGt-FxUEj48C&pg=PT24&dq=nodejs&hl=en&sa=X#v=onepage&q=nodejs&f=false) 2017-03-23 at the Wayback Machine, Sams Publishing, 05-Sep-2012
31. Lawton, George (19 July 2018). "The secret history behind the success of npm and Node" (https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/The-secret-history-behind-the-success-of-npm-and-Node). *TheServerSide*. Archived (https://web.archive.org/web/20210802165613/https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/The-secret-history-behind-the-success-of-npm-and-Node) from the original on 2 August 2021. Retrieved 2 August 2021.
32. Brown, Paul (13 January 2017). "State of the Union: npm" (https://www.linux.com/news/state-union-npm/). *Linux.com*. Archived (https://web.archive.org/web/20210802165614/https://www.linux.com/news/state-union-npm/) from the original on 2 August 2021. Retrieved 2 August 2021.
33. Branscombe, Mary (4 May 2016). "JavaScript Standard Moves to Yearly Release Schedule; Here is What's New for ES16" (https://thenewstack.io/whats-new-es2016/). *The New Stack*. Archived (https://web.archive.org/web/20210116181757/https://thenewstack.io/whats-new-es2016/) from the original on 16 January 2021. Retrieved 15 January 2021.
34. "The TC39 Process" (https://tc39.es/process-document/). *tc39.es*. Ecma International. Archived (https://web.archive.org/web/20210207105535/https://tc39.es/process-document/) from the original on 7 February 2021. Retrieved 15 January 2021.
35. "ECMAScript proposals" (https://github.com/tc39/proposals/blob/master/README.md). TC39. Archived (https://web.archive.org/web/20201204221147/https://github.com/tc39/proposals/blob/master/README.md) from the original on 4 December 2020. Retrieved 15 January 2021.
36. Ashkenas, Jeremy. "List of languages that compile to JS" (https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS). *GitHub*. Archived (https://web.archive.org/web/20200131233044/https://github.com/jashkenas/coffeescript/wiki/List-of-languages-that-compile-to-JS) from the original on 31 January 2020. Retrieved 6 February 2020.

37. "U.S. Trademark Serial No. 75026640" (https://tsdr.uspto.gov/#caseNumber=75026640&caseType=SERIAL_NO&searchType=statusSearch). *uspto.gov*. United States Patent and Trademark Office. 6 May 1997. Archived (https://web.archive.org/web/20210713022850/https://tsdr.uspto.gov/#caseNumber=75026640&caseType=SERIAL_NO&searchType=statusSearch) from the original on 13 July 2021. Retrieved 8 May 2021.
38. "Legal Notices" (https://www.oracle.com/legal/trademarks.html). *oracle.com*. Oracle Corporation. Archived (https://web.archive.org/web/20210605142505/https://www.oracle.com/legal/trademarks.html) from the original on 5 June 2021. Retrieved 8 May 2021.
39. "Oracle to buy Sun in $7.4-bn deal" (https://economictimes.indiatimes.com/tech/software/oracle-to-buy-sun-in-7-4-bn-deal/articleshow/4427747.cms). *The Economic Times*. 21 April 2009.
40. Claburn, Thomas (17 September 2024). "Oracle urged again to give up JavaScript trademark" (https://www.theregister.com/2024/09/17/oracle_urged_to_surrender_javascript_trademark/). *The Register*. Retrieved 2 February 2025.
41. "JavaScript™" (https://javascript.tm/).
42. "Usage statistics of JavaScript libraries for websites" (https://w3techs.com/technologies/overview/javascript_library). *W3Techs*. Retrieved 9 April 2021.
43. "Using jQuery with Bootstrap" (https://clouddevs.com/jquery/web-projects-with-bootstrap/). *clouddevs.com*. 10 June 2019. Retrieved 17 March 2024.
44. "Vanilla JS" (https://vanilla-js.com/). *vanilla-js.com*. 16 June 2020. Archived (https://web.archive.org/web/20200616052335/https://vanilla-js.com/) from the original on 16 June 2020. Retrieved 17 June 2020.
45. "Server-Side JavaScript Guide" (https://docs.oracle.com/cd/E19957-01/816-6411-10/contents.htm). *oracle.com*. Oracle Corporation. 11 December 1998. Archived (https://web.archive.org/web/20210311173120/https://docs.oracle.com/cd/E19957-01/816-6411-10/contents.htm) from the original on 11 March 2021. Retrieved 8 May 2021.
46. Clinick, Andrew (14 July 2000). "Introducing JScript .NET" (https://msdn.microsoft.com/en-us/library/ms974588.aspx). *Microsoft Developer Network*. Microsoft. Archived (https://web.archive.org/web/20171110201649/https://msdn.microsoft.com/en-us/library/ms974588.aspx) from the original on 10 November 2017. Retrieved 10 April 2018. "[S]ince the 1996 introduction of JScript version 1.0 ... we've been seeing a steady increase in the usage of JScript on the server—particularly in Active Server Pages (ASP)"
47. Mahemoff, Michael (17 December 2009). "Server-Side JavaScript, Back with a Vengeance" (https://readwrite.com/2009/12/17/server-side_javascript_back_with_a_vengeance/). *readwrite.com*. Archived (https://web.archive.org/web/20160617030219/https://readwrite.com/2009/12/17/server-side_javascript_back_with_a_vengeance/) from the original on 17 June 2016. Retrieved 16 July 2016.
48. "JavaScript for Acrobat" (https://www.adobe.com/devnet/acrobat/javascript.html). *adobe.com*. 7 August 2009. Archived (https://web.archive.org/web/20090807065130/https://www.adobe.com/devnet/acrobat/javascript.html) from the original on 7 August 2009. Retrieved 18 August 2009.
49. treitter (2 February 2013). "Answering the question: "How do I develop an app for GNOME?" " (https://treitter.livejournal.com/14871.html). *livejournal.com*. Archived (https://web.archive.org/web/20130211032900/https://treitter.livejournal.com/14871.html) from the original on 11 February 2013. Retrieved 7 February 2013.
50. "Tessel 2... Leverage all the libraries of Node.JS to create useful devices in minutes with Tessel" (https://tessel.io/). *tessel.io*. Archived (https://web.archive.org/web/20210526212559/https://tessel.io/) from the original on 26 May 2021. Retrieved 8 May 2021.
51. "Node.js Raspberry Pi GPIO Introduction" (https://www.w3schools.com/nodejs/nodejs_raspberrypi_gpio_intro.asp). *w3schools.com*. Archived (https://web.archive.org/web/20210813192938/https://www.w3schools.com/nodejs/nodejs_raspberrypi_gpio_intro.asp) from the original on 13 August 2021. Retrieved 3 May 2020.

52. "Espruino – JavaScript for Microcontrollers" (https://www.espruino.com/). *espruino.com*. Archived (https://web.archive.org/web/20200501010722/https://www.espruino.com/) from the original on 1 May 2020. Retrieved 3 May 2020.
53. Looper, Jen (21 September 2015). "A Guide to JavaScript Engines for Idiots" (https://web.archive.org/web/20181208123231/http://developer.telerik.com/featured/a-guide-to-javascript-engines-for-idiots/). Telerik Developer Network. Archived from the original (http://developer.telerik.com/featured/a-guide-to-javascript-engines-for-idiots/) on 8 December 2018. Retrieved 8 December 2018.
54. "How Blink Works" (https://docs.google.com/document/d/1aitSOucL0VHZa9Z2vbRJSyAIsAz24kX8LFByQ5xQnUg). Google. Retrieved 12 March 2024.
55. "Documentation · V8" (https://v8.dev/docs). Google. Retrieved 3 March 2024.
56. Nelaturu, Keerthi. "WebAssembly: What's the big deal?" (https://medium.com/coinmonks/webassembly-whats-the-big-deal-662396ff1cd6). *medium.com*. Retrieved 3 March 2024.
57. "Concurrency model and Event Loop" (https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop). *Mozilla Developer Network*. Archived (https://web.archive.org/web/20150905045241/https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop) from the original on 5 September 2015. Retrieved 28 August 2015.
58. Flanagan, David (17 August 2006). *JavaScript: The Definitive Guide: The Definitive Guide* (https://books.google.com/books?id=2weL0iAfrEMC). "O'Reilly Media, Inc.". p. 16. ISBN 978-0-596-55447-7. Archived (https://web.archive.org/web/20200801065235/https://books.google.com/books?id=2weL0iAfrEMC) from the original on 1 August 2020. Retrieved 29 March 2019.
59. Korolev, Mikhail (1 March 2019). "JavaScript quirks in one image from the Internet" (https://dev.to/mkrl/javascript-quirks-in-one-image-from-the-internet-52m7). *The DEV Community*. Archived (https://web.archive.org/web/20191028204723/https://dev.to/mkrl/javascript-quirks-in-one-image-from-the-internet-52m7) from the original on 28 October 2019. Retrieved 28 October 2019.
60. "Proposal-bigint/ADVANCED.md at master · tc39/Proposal-bigint" (https://github.com/tc39/proposal-bigint/blob/master/ADVANCED.md#dont-break-asmjs). *GitHub*.
61. Bernhardt, Gary (2012). "Wat" (https://www.destroyallsoftware.com/talks/wat). *Destroy All Software*. Archived (https://web.archive.org/web/20191028204723/https://www.destroyallsoftware.com/talks/wat) from the original on 28 October 2019. Retrieved 28 October 2019.
62. "JavaScript data types and data structures" (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures). *MDN*. 16 February 2017. Archived (https://web.archive.org/web/20170314230542/https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures) from the original on 14 March 2017. Retrieved 24 February 2017.
63. Flanagan 2006, pp. 176–178.
64. Crockford, Douglas. "Prototypal Inheritance in JavaScript" (https://javascript.crockford.com/prototypal.html). Archived (https://web.archive.org/web/20130813163035/https://javascript.crockford.com/prototypal.html) from the original on 13 August 2013. Retrieved 20 August 2013.
65. "Inheritance and the prototype chain" (https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Inheritance_and_the_prototype_chain). *Mozilla Developer Network*. Archived (https://web.archive.org/web/20130425144207/https://developer.mozilla.org/en-US/docs/JavaScript/Guide/Inheritance_and_the_prototype_chain) from the original on 25 April 2013. Retrieved 6 April 2013.
66. Herman, David (2013). *Effective JavaScript* (https://books.google.com/books?id=Nz9iAwAAQBAJ&pg=PA83). Addison-Wesley. p. 83. ISBN 978-0-321-81218-6.
67. Haverbeke, Marijn (2011). *Eloquent JavaScript* (https://books.google.com/books?id=9U5I_tskq9MC&pg=PA95). No Starch Press. pp. 95–97. ISBN 978-1-59327-282-1.

68. Katz, Yehuda (12 August 2011). "Understanding "Prototypes" in JavaScript" (https://yehudakatz.com/2011/08/12/understanding-prototypes-in-javascript/). Archived (https://web.archive.org/web/20130405154842/https://yehudakatz.com/2011/08/12/understanding-prototypes-in-javascript/) from the original on 5 April 2013. Retrieved 6 April 2013.
69. Herman, David (2013). *Effective JavaScript* (https://books.google.com/books?id=Nz9iAwAAQBAJ&pg=PA125). Addison-Wesley. pp. 125–127. ISBN 978-0-321-81218-6.
70. "Function – JavaScript" (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function). *MDN Web Docs*. Retrieved 30 October 2021.
71. "Properties of the Function Object" (https://es5.github.com/#x15.3.4-toc). Es5.github.com. Archived (https://web.archive.org/web/20130128185825/https://es5.github.com/#x15.3.4-toc) from the original on 28 January 2013. Retrieved 26 May 2013.
72. Flanagan 2006, p. 141.
73. The many talents of JavaScript for generalizing Role-Oriented Programming approaches like Traits and Mixins (https://peterseliger.blogspot.de/2014/04/the-many-talents-of-javascript.html#the-many-talents-of-javascript-for-generalizing-role-oriented-programming-approaches-like-traits-and-mixins) Archived (https://web.archive.org/web/20171005050713/https://peterseliger.blogspot.de/2014/04/the-many-talents-of-javascript.html#the-many-talents-of-javascript-for-generalizing-role-oriented-programming-approaches-like-traits-and-mixins) 2017-10-05 at the Wayback Machine, Peterseliger.blogspot.de, April 11, 2014.
74. Traits for JavaScript (https://soft.vub.ac.be/~tvcutsem/traitsjs/) Archived (https://web.archive.org/web/20140724052500/https://soft.vub.ac.be/~tvcutsem/traitsjs/) 2014-07-24 at the Wayback Machine, 2010.
75. "Home | CocktailJS" (https://cocktailjs.github.io/). *Cocktailjs.github.io*. Archived (https://web.archive.org/web/20170204083608/https://cocktailjs.github.io/) from the original on 4 February 2017. Retrieved 24 February 2017.
76. Croll, Angus (31 May 2011). "A fresh look at JavaScript Mixins" (https://javascriptweblog.wordpress.com/2011/05/31/a-fresh-look-at-javascript-mixins/). *JavaScript, JavaScript…*. Archived (https://web.archive.org/web/20200415004603/https://javascriptweblog.wordpress.com/2011/05/31/a-fresh-look-at-javascript-mixins/) from the original on 15 April 2020.
77. Haverbeke, Marijn (2011). *Eloquent JavaScript* (https://books.google.com/books?id=9U5I_tskq9MC&pg=PA139). No Starch Press. pp. 139–149. ISBN 978-1-59327-282-1.
78. "E4X – Archive of obsolete content" (https://web.archive.org/web/20140724100129/https://developer.mozilla.org/en-US/docs/Archive/Web/E4X). *Mozilla Developer Network*. Mozilla Foundation. 14 February 2014. Archived from the original (https://developer.mozilla.org/en-US/docs/Archive/Web/E4X) on 24 July 2014. Retrieved 13 July 2014.
79. "var – JavaScript" (https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Statements/var). The Mozilla Developer Network. Archived (https://web.archive.org/web/20121223162713/https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Statements/var) from the original on 23 December 2012. Retrieved 22 December 2012.
80. "let" (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let). *MDN web docs*. Mozilla. Archived (https://web.archive.org/web/20190528140803/https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let) from the original on 28 May 2019. Retrieved 27 June 2018.
81. "const" (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const). *MDN web docs*. Mozilla. Archived (https://web.archive.org/web/20180628044054/https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const) from the original on 28 June 2018. Retrieved 27 June 2018.
82. "Making JavaScript Safe for Advertising" (https://www.adsafe.org/). ADsafe. Archived (https://web.archive.org/web/20210706153324/https://www.adsafe.org/) from the original on 6 July 2021. Retrieved 8 May 2021.

83. "Secure ECMA Script (SES)" (https://code.google.com/p/es-lab/wiki/SecureEcmaScript). Archived (https://web.archive.org/web/20130515073412/https://code.google.com/p/es-lab/wiki/SecureEcmaScript) from the original on 15 May 2013. Retrieved 26 May 2013.
84. "Google Caja Project" (https://developers.google.com/caja/). *Google*. Archived (https://web.archive.org/web/20210122083321/https://developers.google.com/caja/) from the original on 22 January 2021. Retrieved 9 July 2021.
85. "Mozilla Cross-Site Scripting Vulnerability Reported and Fixed – MozillaZine Talkback" (https://www.mozillazine.org/talkback.html?article=4392). *Mozillazine.org*. Archived (https://web.archive.org/web/20110721230916/http://www.mozillazine.org/talkback.html?article=4392) from the original on 21 July 2011. Retrieved 24 February 2017.
86. Kottelin, Thor (17 June 2008). "Right-click "protection"? Forget about it" (https://web.archive.org/web/20110809195359/https://blog.anta.net/2008/06/17/right-click-%E2%80%9Cprotection%E2%80%9D-forget-about-it/). *blog.anta.net*. Archived from the original (https://blog.anta.net/2008/06/17/right-click-%E2%80%9Cprotection%E2%80%9D-forget-about-it/) on 9 August 2011. Retrieved 28 July 2022.
87. Rehorik, Jan (29 November 2016). "Why You Should Never Put Sensitive Data in Your JavaScript" (https://www.serviceobjects.com/blog/why-you-should-never-put-sensitive-data-in-your-javascript/). *ServiceObjects Blog*. ServiceObjects. Archived (https://web.archive.org/web/20190603142957/https://www.serviceobjects.com/blog/why-you-should-never-put-sensitive-data-in-your-javascript/) from the original on 3 June 2019. Retrieved 3 June 2019.
88. Lauinger, Tobias; Chaabane, Abdelberi; Arshad, Sajjad; Robertson, William; Wilson, Christo; Kirda, Engin (21 December 2016), "Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web" (https://web.archive.org/web/20170329045344/https://www.ccs.neu.edu/home/arshad/publications/ndss2017jslibs.pdf) (PDF), *Northeastern University*, arXiv:1811.00918 (https://arxiv.org/abs/1811.00918), doi:10.14722/ndss.2017.23414 (https://doi.org/10.14722%2Fndss.2017.23414), ISBN 978-1-891562-46-4, S2CID 17885720 (https://api.semanticscholar.org/CorpusID:17885720), archived from the original (https://www.ccs.neu.edu/home/arshad/publications/ndss2017jslibs.pdf) (PDF) on 29 March 2017, retrieved 28 July 2022
89. Collins, Keith (27 March 2016). "How one programmer broke the internet by deleting a tiny piece of code" (https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code/). *Quartz*. Archived (https://web.archive.org/web/20170222200836/https://qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code/) from the original on 22 February 2017. Retrieved 22 February 2017.
90. SC Magazine UK, Developer's 11 lines of deleted code 'breaks the internet' (https://www.scmagazineuk.com/developers-11-lines-of-deleted-code-breaks-the-internet/article/532050/) Archived (https://web.archive.org/web/20170223041434/https://www.scmagazineuk.com/developers-11-lines-of-deleted-code-breaks-the-internet/article/532050/) February 23, 2017, at the Wayback Machine
91. Mozilla Corporation, Buffer overflow in crypto.signText() (https://www.mozilla.org/security/announce/2006/mfsa2006-38.html) Archived (https://web.archive.org/web/20140604014705/https://www.mozilla.org/security/announce/2006/mfsa2006-38.html) 2014-06-04 at the Wayback Machine
92. Festa, Paul (19 August 1998). "Buffer-overflow bug in IE" (https://web.archive.org/web/20021225190522/https://news.com.com/2100-1001-214620.html). *CNET*. Archived from the original (https://news.com.com/2100-1001-214620.html) on 25 December 2002.
93. SecurityTracker.com, Apple Safari JavaScript Buffer Overflow Lets Remote Users Execute Arbitrary Code and HTTP Redirect Bug Lets Remote Users Access Files (https://securitytracker.com/alerts/2006/Mar/1015713.html) Archived (https://web.archive.org/web/20100218102849/https://securitytracker.com/alerts/2006/Mar/1015713.html) 2010-02-18 at the Wayback Machine

94. SecurityFocus, Microsoft WebViewFolderIcon ActiveX Control Buffer Overflow Vulnerability (https://www.securityfocus.com/bid/19030/info) Archived (https://web.archive.org/web/20111011091819/http://www.securityfocus.com/bid/19030/info) 2011-10-11 at the Wayback Machine
95. Fusion Authority, Macromedia Flash ActiveX Buffer Overflow (https://www.fusionauthority.com/security/3234-macromedia-flash-activex-buffer-overflow.htm) Archived (https://web.archive.org/web/20110813160055/https://www.fusionauthority.com/security/3234-macromedia-flash-activex-buffer-overflow.htm) August 13, 2011, at the Wayback Machine
96. "Protected Mode in Vista IE7 – IEBlog" (https://blogs.msdn.com/ie/archive/2006/02/09/528963.aspx). *Blogs.msdn.com*. 9 February 2006. Archived (https://web.archive.org/web/20100123103719/https://blogs.msdn.com/ie/archive/2006/02/09/528963.aspx) from the original on 23 January 2010. Retrieved 24 February 2017.
97. US CERT, Vulnerability Note VU#713878: Microsoft Internet Explorer does not properly validate source of redirected frame (https://www.kb.cert.org/vuls/id/713878) Archived (https://web.archive.org/web/20091030051811/https://www.kb.cert.org/vuls/id/713878/) 2009-10-30 at the Wayback Machine
98. Mozilla Foundation, Mozilla Foundation Security Advisory 2005–41: Privilege escalation via DOM property overrides (https://www.mozilla.org/security/announce/2005/mfsa2005-41.html) Archived (https://web.archive.org/web/20140604014832/https://www.mozilla.org/security/announce/2005/mfsa2005-41.html) 2014-06-04 at the Wayback Machine
99. Andersen, Starr (9 August 2004). "Part 5: Enhanced Browsing Security" (https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-xp/bb457150(v=technet.10)). TechNet. *Microsoft Docs*. Changes to Functionality in Windows XP Service Pack 2. Retrieved 20 October 2021.
100. For one example of a rare JavaScript Trojan Horse, see Symantec Corporation, JS.Seeker.K (https://www.symantec.com/security_response/writeup.jsp?docid=2003-100111-0931-99) Archived (https://web.archive.org/web/20110913210848/http://www.symantec.com/security_response/writeup.jsp?docid=2003-100111-0931-99) 2011-09-13 at the Wayback Machine
101. Gruss, Daniel; Maurice, Clémentine; Mangard, Stefan (24 July 2015). "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". arXiv:1507.06955 (https://arxiv.org/abs/1507.06955) [cs.CR (https://arxiv.org/archive/cs.CR)].
102. Jean-Pharuns, Alix (30 July 2015). "Rowhammer.js Is the Most Ingenious Hack I've Ever Seen" (https://motherboard.vice.com/en_us/article/9akpwz/rowhammerjs-is-the-most-ingenious-hack-ive-ever-seen). *Motherboard*. Vice. Archived (https://web.archive.org/web/20180127084042/https://motherboard.vice.com/en_us/article/9akpwz/rowhammerjs-is-the-most-ingenious-hack-ive-ever-seen) from the original on 27 January 2018. Retrieved 26 January 2018.
103. Goodin, Dan (4 August 2015). "DRAM 'Bitflipping' exploit for attacking PCs: Just add JavaScript" (https://arstechnica.com/information-technology/2015/08/dram-bitflipping-exploit-for-attacking-pcs-just-add-javascript/). *Ars Technica*. Archived (https://web.archive.org/web/20180127143154/https://arstechnica.com/information-technology/2015/08/dram-bitflipping-exploit-for-attacking-pcs-just-add-javascript/) from the original on 27 January 2018. Retrieved 26 January 2018.
104. Auerbach, David (28 July 2015). "Rowhammer security exploit: Why a new security attack is truly terrifying" (https://www.slate.com/articles/technology/bitwise/2015/07/rowhammer_security_exploit_why_a_new_security_attack_is_truly_terrifying.html). *slate.com*. Archived (https://web.archive.org/web/20150730004023/https://www.slate.com/articles/technology/bitwise/2015/07/rowhammer_security_exploit_why_a_new_security_attack_is_truly_terrifying.html) from the original on 30 July 2015. Retrieved 29 July 2015.
105. AnC (https://www.vusec.net/projects/anc/) Archived (https://web.archive.org/web/20170316055626/https://www.vusec.net/projects/anc/) 2017-03-16 at the Wayback Machine VUSec, 2017

106. New ASLR-busting JavaScript is about to make drive-by exploits much nastier (https://arstechnica.com/security/2017/02/new-aslr-busting-javascript-is-about-to-make-drive-by-exploits-much-nastier/) Archived (https://web.archive.org/web/20170316024419/https://arstechnica.com/security/2017/02/new-aslr-busting-javascript-is-about-to-make-drive-by-exploits-much-nastier/) 2017-03-16 at the Wayback Machine Ars Technica, 2017
107. Spectre Attack (https://spectreattack.com/spectre.pdf) Archived (https://web.archive.org/web/20180103225843/https://spectreattack.com/spectre.pdf) 2018-01-03 at the Wayback Machine Spectre Attack
108. "Benchmark.js" (https://benchmarkjs.com/). *benchmarkjs.com*. Archived (https://web.archive.org/web/20161219182724/https://benchmarkjs.com/) from the original on 19 December 2016. Retrieved 6 November 2016.
109. JSBEN.CH. "JSBEN.CH Performance Benchmarking Playground for JavaScript" (https://jsben.ch). *jsben.ch*. Archived (https://web.archive.org/web/20210227052409/https://jsben.ch/) from the original on 27 February 2021. Retrieved 13 August 2021.
110. Eich, Brendan (3 April 2008). "Popularity" (https://brendaneich.com/2008/04/popularity/). Archived (https://web.archive.org/web/20110703020955/https://brendaneich.com/2008/04/popularity/) from the original on 3 July 2011. Retrieved 19 January 2012.
111. "Edge Browser Switches WebAssembly to 'On' -- Visual Studio Magazine" (https://visualstudiomagazine.com/articles/2017/11/06/edge-webassembly.aspx). *Visual Studio Magazine*. Archived (https://web.archive.org/web/20180210002432/https://visualstudiomagazine.com/articles/2017/11/06/edge-webassembly.aspx) from the original on 10 February 2018. Retrieved 9 February 2018.
112. "frequently asked questions" (https://asmjs.org/faq.html). asm.js. Archived (https://web.archive.org/web/20140604012024/https://asmjs.org/faq.html) from the original on 4 June 2014. Retrieved 13 April 2014.

Sources

- Dere, Mohan (21 December 2017). "How to integrate create-react-app with all the libraries you need to make a great app" (https://medium.freecodecamp.org/integrating-create-react-app-redux-react-router-redux-observable-bootstrap-altogether-216db97e89a3). *freeCodeCamp*. Retrieved 14 June 2018.
- Panchal, Krunal (26 April 2022). "Angular vs React Detailed Comparison" (https://www.groovyweb.co/blog/angular-vs-react-detail-comparison). *Groovy Web*. Retrieved 5 June 2023.

Further reading

- Flanagan, David. *JavaScript: The Definitive Guide*. 7th edition. Sebastopol, California: O'Reilly, 2020. ISBN 978-1-491-95202-3.
- Haverbeke, Marijn. *Eloquent JavaScript*. 3rd edition. No Starch Press, 2018. 472 pages. ISBN 978-1593279509.*(download)* (https://eloquentjavascript.net/)
- Zakas, Nicholas. *Principles of Object-Oriented JavaScript*, 1st edition. No Starch Press, 2014. 120 pages. ISBN 978-1593275402.

External links

- The Modern JavaScript Tutorial (https://javascript.info/). A community maintained continuously updated collection of tutorials on the entirety of the language.

- "JavaScript: The First 20 Years" (https://www.pldi21.org/prerecorded_hopl.12.html). Retrieved 6 February 2022.
