Advanced Research Paper ES3

ECMAscript 3

Date**: March 28 2025**
Revision**: v8**

ECMAScript 3 or ES3:

A Comprehensive Analysis of its Definition, History, Features, and Legacy

1. Introduction: Understanding ECMAScript 3

ECMAScript, often abbreviated as ES, stands as a pivotal standard for scripting languages, with JavaScript being its most widely recognized implementation.[1] This standard, meticulously documented by Ecma International in the ECMA-262 specification, ensures a level of interoperability across different web browsers and other environments that support the language.[1] The formal journey towards creating this standard commenced in November 1996, driven by the burgeoning need for a universal scripting language to enhance the interactivity of the World Wide Web.[1] The very term "ECMAScript" was conceived as a compromise, a neutral designation agreed upon by Netscape Communications and Microsoft, whose early, yet often competitive, involvement was instrumental in shaping the language.[1] At its core, the ECMAScript specification meticulously defines the language's syntax—the precise rules governing how code is structured and written—and its semantics—the underlying meaning and behavior dictated by that code.[1] It is crucial to distinguish between "ECMAScript," the standard itself, and "JavaScript," which, in the context of web browsers, often encompasses not only the core language but also a rich collection of Web APIs, such as the Document Object Model (DOM), that are essential for creating dynamic and interactive web experiences.[3]

The standardization process of ECMAScript has evolved over time. Initially, specifications were typically published every few years and were identified by major version numbers, such as ECMAScript 3 (ES3) and ECMAScript 5 (ES5). However, with the release of ECMAScript 6 (ES6), also known as ECMAScript 2015, the naming convention shifted towards annual releases denoted by the year of publication, for example, ES2017.[3] These editions undergo a rigorous approval process by the ECMA General Assembly on a yearly basis, reflecting a commitment to continuous improvement and adaptation.[3] The ongoing maintenance and updating of the ECMAScript standard are the responsibility of dedicated specification editors, with the most current versions of the ECMA-262 document readily accessible on the TC39 (Technical Committee 39) website.[3] The initial impetus for standardizing JavaScript can be traced back to Netscape's formal submission of the language to the European Computer Manufacturers Association (ECMA) in an effort to foster a greater degree of consistency and interoperability across the increasingly diverse landscape of web browsers.[1] The selection of the name ECMAScript was a strategic move to navigate potential trademark issues associated with the widely recognized term "JavaScript".[2] The primary goal of this standardization was to establish a common and unified foundation for scripting languages, ensuring a more predictable and consistent development environment across the various browser platforms.[22] The international recognition of the ECMAScript standard is further underscored by its adoption as ISO/IEC 16262.[4] Recognizing the need for the language in resource-constrained environments, a specific subset of ES3 was also defined as the ECMAScript Compact Profile (ES-CP or ECMA-327).[14] To ensure that implementations of the ECMAScript standard adhere to the specifications, the Test262 test suite was developed as a comprehensive conformance test.[1]

The evolution of ECMAScript began with the first edition (ES1) in June 1997, followed by the second edition (ES2) in June 1998, which primarily focused on editorial revisions to achieve full alignment with the ISO standard.[6] ECMAScript 3 (ES3), the subject of this report, represents the third major iteration of the standard and was formally adopted in December 1999.[6] Building upon the foundations laid by its predecessors, ES3 was based on the implementation of JavaScript 1.2 as found in Netscape Navigator 4.0 and introduced several pivotal features, most notably powerful regular expressions and significantly enhanced string handling capabilities.[15] Following the release of ES3, there was a notable period of relative stability in the ECMAScript standard, with the next major version, ES5, not appearing until 2009, marking a decade of reliance on ES3.[6] During this interim, an ambitious project to develop ECMAScript 4 (ES4) was initiated but ultimately abandoned in 2008 due to fundamental disagreements concerning the language's complexity and scope.[6] ES5, which eventually emerged, adopted a more incremental approach, prioritizing stability and performance improvements while introducing key features such as strict mode and native support for JSON.[6] A more transformative phase in ECMAScript's history began with the release of ECMAScript 6 (ES2015), which brought about substantial changes and enhancements to the language, including the introduction of lexical block scoping through let and const, arrow functions, class declarations, modules, and promises.[6] Since the advent of ES6, ECMAScript has transitioned to an annual release cadence, with new versions being published every June, incorporating features that have successfully navigated the rigorous multi-stage proposal process managed by TC39.[1]

This report aims to provide a comprehensive and in-depth analysis of ECMAScript 3. It will delve into the definition and historical context surrounding its development, explore its key features and syntax specifications, compare and contrast it with subsequent versions, examine its common use cases and applications, investigate its limitations and drawbacks, research its current relevance in the ever-evolving landscape of web development, and finally, consider the security implications associated with its continued use.

2. The Genesis of ES3: Historical Context

The inception of JavaScript in May 1995 by Brendan Eich at Netscape Communications marked the beginning of a transformative era for the World Wide Web, with the language reportedly being developed in a mere ten days.[1] Initially codenamed Mocha, it was subsequently renamed LiveScript before settling on the name JavaScript, a strategic marketing decision intended to capitalize on the burgeoning popularity of Sun Microsystems' Java, despite the fundamental differences between the two languages.[6] The primary motivation behind JavaScript's creation was to introduce dynamic capabilities to web browsers, thereby enriching the predominantly static content of HTML and fostering more interactive user experiences.[1] However, the web development landscape became more complex with Microsoft's introduction of its own JavaScript implementation, known as JScript, for its Internet Explorer browser. This divergence in language implementations led to a period often referred to as the "browser wars," where the inconsistencies between Netscape's JavaScript and Microsoft's JScript created significant challenges for web developers striving to ensure their websites functioned correctly across different browsers.[1] During the mid-1990s, Netscape Navigator held a dominant position in the web browser market [12], but by the early 2000s, Microsoft's Internet Explorer had gained significant traction and eventually surpassed Netscape in market share.[10]

Recognizing the growing need for a standardized scripting language to ensure a more consistent web development experience, Netscape Communications took the initiative by submitting JavaScript to the European Computer Manufacturers Association (ECMA) in November 1996, thereby commencing the formal standardization process.[1] Microsoft, while also maintaining its own JScript implementation, actively participated in these standardization efforts. However, the early sessions were often marked by a degree of tension and competing priorities between the two dominant players in the browser market.[1] The name "ECMAScript" itself emerged as a direct consequence of these dynamics, serving as a politically neutral term that avoided direct association with either Netscape's or Microsoft's proprietary technologies.[1] Despite their differences, both JavaScript and JScript provided the foundational technologies and served as the primary source material upon which the ECMAScript standard was ultimately built.[1]

To facilitate the development and ongoing maintenance of the ECMAScript standard, ECMA International established Technical Committee 39 (TC39).[1] This committee operates based on the principle of consensus, ensuring that all participating member organizations have a voice in the evolution of the language specification, and it retains the authority to modify the standard as it deems necessary to meet the changing needs of the web and software development.[52] The process for introducing new features into the ECMAScript standard involves a structured, multi-stage proposal system, with each stage representing increasing levels of maturity and completeness of the proposed feature.[1] The first edition of the ECMAScript standard (ES1) was officially released in June 1997, marking the initial formal step in standardizing the language.[1] The editor credited with the creation of the first edition was Guy L. Steele Jr..[23] Subsequent editions, including the second (ES2) and the third (ES3), were edited by Mike Cowlishaw.[23]

ECMAScript 3, the focus of this analysis, was formally adopted by the ECMA General Assembly in December 1999.[14] The editorial work on the ES3 specification is primarily attributed to Mike Cowlishaw, who built upon the foundational work of the earlier ES1 and ES2 standards.[23] The content and features of the ES3 specification were largely informed by the implementation of JavaScript 1.2 as it existed in Netscape Navigator 4.0, reflecting the dominant browser technology and scripting capabilities of that era.[23] The development of the ECMA-262 standard, which encompasses all editions of ECMAScript including ES3, was a collaborative undertaking that involved numerous individuals from a diverse range of organizations within the computing and technology industries. Key contributors to this effort included prominent figures such as Brendan Eich, the original inventor of JavaScript, Mike Cowlishaw, the editor of ES3, and many other representatives from companies such as Netscape, Microsoft, IBM, Hewlett-Packard, and Sun Microsystems.[26]

3. Dissecting ES3: Key Features and Syntax

ECMAscript 3

Date**: March 28 2025**
Revision**: v8**

ECMAScript 3 laid the groundwork for a robust scripting language with a set of fundamental syntax elements. Variables were declared using the var keyword, and their scope was limited to the function in which they were defined.[1] The language supported several primitive data types, including Number for representing numeric values, String for sequences of characters, Boolean for logical true or false values, Null to indicate the intentional absence of a value, and Undefined to signify that a variable has been declared but not yet assigned a value.[3] Additionally, the Object type allowed for the creation of more complex data structures capable of holding properties and methods.[3] ES3 provided a comprehensive set of operators, including arithmetic operators for performing mathematical calculations, assignment operators for assigning values to variables, comparison operators for evaluating relationships between values, logical operators for combining boolean expressions, and bitwise operators for manipulating data at the level of individual bits.[27] Control flow within ES3 scripts was managed through a variety of statements that enabled conditional execution of code blocks using if, else, and switch statements, as well as the creation of loops for repetitive tasks using for, while, and do-while statements.[8]

A significant addition in ECMAScript 3 was the introduction of regular expressions, a powerful tool for pattern matching and text manipulation.[6] These patterns could be defined either as regular expression literals, enclosed in forward slashes (e.g., /abc/), or by using the RegExp constructor (e.g., new RegExp("abc")).[81] ES3 regular expressions supported a wide range of syntax for matching specific characters, including character sets, quantifiers to specify the number of occurrences, and anchors to define positions within the string.[27] Methods such as test(), which returned a boolean indicating if the pattern was found in a string, and exec(), which returned an array with the match details or null if no match was found, provided the means to utilize these patterns in scripts.[8]

For handling runtime errors, ES3 introduced the try...catch statement.[6] Code that might potentially throw an error could be placed within a try block. If an error occurred during the execution of this code, control would immediately pass to the catch block, which contained instructions on how to handle the error.[8] The catch block received an error object, providing information about the type and nature of the error.[60] ES3 also supported an optional finally block, which could follow the try and catch blocks. The code within the finally block was guaranteed to execute regardless of whether an error occurred or was caught, making it suitable for cleanup tasks.[60]

ECMAScript 3 provided support for object-oriented programming through a prototype-based inheritance model.[1] In this paradigm, objects could inherit properties and methods from other objects, known as prototypes, forming a chain of inheritance. The this keyword in ES3 referred to the object in which the current code was being executed, often the object that owned the method being called.[98] The specific value of this was dynamically determined based on how the function was invoked.[98] Although ES3 lacked the class syntax introduced in later versions, developers could simulate classes using constructor functions and by manipulating the prototype property of these functions.[73]

Arrays in ES3 were supported as ordered collections capable of holding values of any type.[1] Basic array operations included accessing elements by their index and determining the size of the array using the length property.[19] Functions in ES3 were treated as first-class citizens, allowing them to be assigned to variables, passed as arguments to other functions, and returned as values.[1] ES3 supported both function declarations and function expressions, providing flexibility in how functions could be defined and used.[110] The concept of closures was also part of ES3, enabling functions to retain access to variables from their outer scope even after the outer function had completed its execution.[1]

4. ES3 Compared: Evolution to ES5 and ES6

ECMAScript 5, released in 2009, introduced several key features and improvements over ES3.[6] One of the most significant additions was "strict mode," a directive that enforced stricter parsing and error handling, leading to more robust and maintainable code. ES5 also brought native support for JSON (JavaScript Object Notation), a lightweight data-interchange format, with built-in methods for parsing JSON strings and converting JavaScript objects to JSON. The capabilities for working with arrays were significantly enhanced with the introduction of methods like forEach, map, filter, reduce, every, some, indexOf, lastIndexOf, and isArray, providing more functional and concise ways to manipulate array data. ES5 also added the String.trim() method for easily removing whitespace from the beginning and end of strings. Furthermore, ES5 introduced the ability to define getters and setters for object properties, allowing for more controlled access and modification of object attributes. Finally, ES5 provided more fine-grained control over the properties of objects through new methods in the Object namespace, such as defineProperty, create, and keys.

ECMAScript 6 (ES2015), released in 2015, represented a monumental leap forward in the evolution of JavaScript.[1] It introduced block-scoped variables using let and const, which helped to mitigate the issues associated with var's function scope. Arrow functions provided a more concise syntax for writing functions and also addressed some of the complexities of the this keyword. ES6 brought a more structured approach to object-oriented programming with the introduction of classes, although JavaScript's underlying prototype-based inheritance model remained. For better code organization and modularity, ES6 introduced a native module system with the import and export keywords. Handling asynchronous operations was greatly improved with the introduction of Promises, offering a more elegant alternative to callbacks. ES6 also included template literals for easier string interpolation, destructuring for convenient value extraction from arrays and objects, default and rest parameters for functions, the spread operator for expanding iterables, iterators and the for...of loop for iterating over various data structures, generators for simplifying the creation of iterators, new collection types like Map and Set, and symbols for creating unique object properties.

ECMAscript 3

Date**: March 28 2025**
Revision**: v8**

Feature	**ES3 (1999)**	**ES5 (2009)**	**ES6 (2015)**
Strict Mode	No	Yes	Yes
JSON Support	No	Yes	Yes
Array Methods (forEach, map, filter, reduce, etc.)	No	Yes	Yes
String.trim()	No	Yes	Yes
Getters/Setters	No	Yes	Yes
Object Property Control	Limited	Yes	Yes
let and const	No	No	Yes
Arrow Functions	No	No	Yes
Classes	No	No	Yes
Modules	No	No	Yes
Promises	No	No	Yes
Template Literals	No	No	Yes

Destructuring	No	No	Yes
Maps and Sets	No	No	Yes
Symbols	No	No	Yes

5. Applications in Practice: Use Cases of ES3

ECMAScript 3 served as the foundational scripting language for client-side web development for a significant period, powering the interactivity of web pages across a wide range of browsers prevalent in the early to mid-2000s, including Internet Explorer, Netscape Navigator, and the initial versions of Firefox, Chrome, and Safari.[1] It played a crucial role in enabling Dynamic HTML (DHTML) techniques, allowing developers to dynamically modify the content and styling of web pages in response to user interactions and other events.[144] The Asynchronous JavaScript and XML (AJAX) methodology, which significantly enhanced the responsiveness of web applications, heavily relied on ES3's capabilities, particularly the XMLHttpRequest object, to facilitate background communication with servers and update specific parts of a web page without requiring a full reload.[34] ES3 was also extensively used for implementing fundamental client-side functionalities such as form validation to ensure the integrity of user input before submission, creating basic animations to improve user engagement, and handling a variety of user interactions, including mouse clicks, keyboard inputs, and form submissions.[21]

While the widespread use of JavaScript for server-side scripting gained prominence with the advent of Node.js (which utilizes more modern JavaScript engines), there were earlier server-side JavaScript environments that likely leveraged ES3 interpreters or similar technologies.[1] Notably, Google Apps Script, in its initial iterations, was based on the ECMA-262 3rd Edition, providing a specific example of ES3's application in a non-browser, server-like environment for automating tasks and extending the functionality of Google's suite of productivity applications.[149]

During the ES3 era, a number of significant web development frameworks and libraries emerged to address the challenges of cross-browser compatibility and to streamline common development tasks. Prominent examples include libraries such as jQuery, Prototype, Dojo Toolkit, and Mootools. These tools, built upon the foundation of ES3, offered developers abstractions that helped to normalize the inconsistencies found across different web browsers' JavaScript implementations, especially in areas like Document Object Model (DOM) manipulation and event handling.[6] While AngularJS gained significant traction later, its initial versions were built upon the principles and capabilities provided by ES3 and ES5, further illustrating the evolving landscape of JavaScript frameworks.[150] Moreover, Dynamic HTML (DHTML) itself can be considered an early form of "framework" or a collection of techniques that heavily utilized ES3 to create more interactive and dynamic web experiences.[144]

Beyond the realm of web development, the ECMAScript standard, including the core features of ES3, found applications in various other domains. The existence of the ECMAScript Compact Profile (ES-CP), a specification specifically designed as a strict subset of ES3 for use in resource-constrained devices, suggests its potential application in embedded systems and other environments where computational resources were limited.[14] Additionally, ActionScript, the scripting language employed by Adobe Flash for creating interactive multimedia content, and JScript, Microsoft's implementation of ECMAScript used in environments such as Windows Script Host for system scripting and automation, were both based on the ECMAScript standard, including the features defined in ES3, and were utilized in numerous non-web contexts.[1] Furthermore, ECMAScript for XML (E4X) was introduced as an extension to ECMAScript, providing specific functionalities for working with and manipulating XML documents, indicating its potential use in applications that involved XML data processing.[4]

6. The Limitations of ES3: Why Modern JavaScript Evolved

ECMAScript 3, while a significant step in standardizing JavaScript, possessed several limitations in its language features and syntax that increasingly hindered the development of more complex and sophisticated web applications. One of the primary shortcomings was the function-level scope of variables declared using the var keyword. This scoping behavior often led to issues such as variable hoisting, where variables could be accessed before their actual declaration in the code, and the unintentional creation of global variables, which could result in naming conflicts and make codebases harder to manage and debug, particularly in larger projects.[1] Furthermore, ES3 lacked the more concise and structured syntax for object creation and manipulation that was introduced in subsequent versions of the standard. Features like classes, enhanced object literals providing shorthand for method and property definitions, and destructuring for easily extracting values from arrays and objects were absent, making object-oriented programming more verbose and less intuitive.[1]

Asynchronous programming in ES3 was predominantly handled using callbacks, a pattern that could quickly lead to deeply nested and convoluted code structures, often referred to as "callback hell," especially when dealing with multiple sequential or parallel asynchronous operations. This approach made it significantly harder to manage the flow of asynchronous logic and handle errors effectively compared to the more structured and readable solutions offered by Promises and the async/await syntax introduced in later versions.[23] ES3 also did not provide built-in support for more advanced and commonly used data structures like Maps and Sets, which were introduced in ES6 and offered more efficient and convenient ways to handle key-value pairs and collections of unique values compared to the traditional use of plain JavaScript objects for these purposes.[1] Furthermore, the capabilities for string manipulation in ES3 were relatively limited compared to the more extensive set of methods and features that were added in subsequent ECMAScript standards.[6] Finally, ES3 lacked a native module system, which made it increasingly challenging to organize code into reusable and encapsulated components and to manage dependencies effectively as web applications grew in size and complexity.[1]

Despite the standardization of ES3, achieving consistent cross-browser compatibility remained a notable challenge for web developers. Subtle variations in how different browser engines implemented the ES3 specification could lead to inconsistencies in the behavior and rendering of web applications across various browsers. This often required developers to resort to writing browser-specific code or relying on external JavaScript libraries, such as jQuery, that aimed to abstract away these inconsistencies and provide a more uniform development experience.[6] During the "browser wars" era, the intense competition between Netscape and Microsoft sometimes led to browser vendors prioritizing proprietary features over strict adherence to web standards, including ECMAScript, which further complicated the issue of cross-browser compatibility.[10]

JavaScript engines have undergone significant evolution since the ES3 era, with modern engines incorporating sophisticated performance optimizations like Just-In-Time (JIT) compilation.[3] These advancements were either not as prevalent or as refined in the engines that primarily supported ES3. Moreover, ES3 lacked certain language features that contribute to improved performance in modern JavaScript, such as Typed Arrays for efficient handling of binary data.[1]

Finally, ES3 lacked native support for many of the contemporary web development paradigms and APIs that are now considered standard practice. For instance, component-based architectures, popularized by frameworks like React, Angular, and Vue.js, heavily leverage features introduced in ES6 and later, such as classes, modules, and template literals.[32] The absence of built-in support for numerous modern Web

APIs further limited the capabilities of ES3 for creating rich and interactive web experiences.

7. ES3 in the Present: Relevance and Usage Today

Despite the significant advancements in ECMAScript since its release, ES3 still maintains a degree of relevance in specific contexts. Large enterprise-level applications, particularly those with long development cycles and strict requirements for backward compatibility, often with older web browsers like Internet Explorer 8, may still have substantial portions of their codebase written in ES3.[141] Additionally, certain specialized environments, such as some embedded systems or proprietary software platforms with limited computational resources, might still rely on JavaScript interpreters that primarily support the ES3 standard or its compact profile.[4] Furthermore, legacy web applications that have not undergone significant modernization efforts might still contain considerable amounts of ES3 code.[141]

In scenarios where maintaining compatibility with extremely outdated web browsers, such as Internet Explorer 6, 7, or 8, is a critical requirement, developers might still find it necessary to target ES3 or to transpile modern JavaScript code to an ES3-compatible level.[112] Similarly, specific embedded systems or legacy platforms with JavaScript engines that only support the ES3 standard might necessitate writing code that adheres to its specifications.[4]

When working with existing ES3 code or when the need arises to target ES3 environments, several strategies can be employed. Modern JavaScript code can be transpiled to ES3 using tools like Babel, which converts newer syntax and features into equivalent ES3 code.[1] Polyfills, which are code snippets that provide implementations of newer JavaScript features in older environments, can also be used to bring functionalities from ES5 and later to ES3, although this might come with performance considerations.[1] In some cases, developers might opt to use JavaScript libraries that were specifically designed to be compatible with ES3 to ensure broader support.[141] For long-term maintainability and to leverage the advantages of more modern JavaScript features, gradually migrating ES3 codebases to newer ECMAScript versions is often the most recommended approach.[58]

8. Navigating the Past: Security Considerations with ES3

Given its age, JavaScript engines that primarily support ES3 may contain known security vulnerabilities that have been addressed in more recent versions.[6] ES3 inherently lacks some of the security features and browser security mechanisms introduced in later ECMAScript standards to mitigate common web security threats. Furthermore, codebases that still rely on ES3 may also depend on older versions of JavaScript libraries and frameworks, which themselves might contain security vulnerabilities that have since been discovered and patched in newer releases.

When maintaining or interacting with ES3 code, it is crucial to keep the JavaScript engine (browser or runtime environment) as up-to-date as possible within the system's constraints, as newer versions often include patches for known vulnerabilities. Any third-party libraries or frameworks used in the ES3 codebase should be thoroughly vetted for known security flaws and updated to the latest available versions if feasible. Implementing robust server-side security measures is also essential to compensate for potential client-side vulnerabilities that might exist in ES3 code. In certain scenarios, isolating ES3 code within sandboxed environments can help limit the potential impact of security breaches.

For applications where security is a paramount concern, the most effective long-term strategy is to prioritize the modernization or complete replacement of ES3 codebases with versions that adhere to more recent ECMAScript standards, such as ES5 or later. These newer standards often include important security enhancements and are supported by actively maintained JavaScript engines that receive regular security updates. If a full upgrade is not immediately feasible, rewriting critical, security-sensitive portions of the code in a more modern JavaScript version can be a viable interim step.

9. Conclusion: Reflecting on ES3's Legacy

ECMAScript 3, adopted in 1999, represents a significant milestone in the standardization of JavaScript, providing core language features that powered the early interactive web. Its introduction of regular expressions and structured error handling marked a substantial advancement over previous versions. While ES3 served as a stable and widely used standard for nearly a decade, the evolution of web development and the increasing complexity of web applications necessitated further advancements in the language. Subsequent versions, ES5 and ES6, introduced a wealth of new features and syntax improvements that addressed many of the limitations of ES3, leading to more robust, efficient, and developer-friendly code.

ES3's legacy persists today primarily in older systems, particularly in enterprise environments with strict backward compatibility requirements, and in specific resource-constrained or embedded systems. Working with ES3 in the present often involves strategies like transpiling and polyfilling to bridge the gap with modern JavaScript. However, the age of the standard also raises security considerations, highlighting the importance of careful maintenance and a potential need for modernization.

In conclusion, ECMAScript 3 played a crucial role in the history of the web, providing a foundational scripting language that enabled early web interactivity. While modern JavaScript has evolved significantly, understanding ES3 offers valuable context into the language's origins and the driving forces behind its continued development.

Works cited

1. ECMAScript - Wikipedia, accessed May 2, 2025, https://en.wikipedia.org/wiki/ECMAScript
2. What is ECMAScript? - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/4269150/what-is-ecmascript
3. JavaScript technologies overview - MDN Web Docs, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/JavaScript_technologies_overview
4. ECMAScript Language (ECMA-262), including JavaScript - Library of Congress, accessed May 2, 2025, https://www.loc.gov/preservation/digital/formats/fdd/fdd000443.shtml
5. Ecmascript Evolution - priyanka chaudhari's, accessed May 2, 2025, https://priyankachaudhari.hashnode.dev/history-ecmascriptjavascript
6. History of ECMAScript | Ben Ilegbodu, accessed May 2, 2025, https://www.benmvp.com/blog/learning-es6-history-of-ecmascript/
7. 5 History and evolution of JavaScript - Exploring JS, accessed May 2, 2025, https://exploringjs.com/js/book/ch_history.html
8. A Brief History of ECMAScript Versions in JavaScript - Web Reference, accessed May 2, 2025, https://webreference.com/javascript/basics/versions/
9. ECMAScript - QPan, accessed May 2, 2025, https://www.quanpan302.com/sd/ECMAScript/
10. What is ECMAScript? What is it to do with JavaScript? - ExplainThis, accessed May 2, 2025, https://www.explainthis.io/en/swe/ECMAScript
11. Chapter 5. Standardization: ECMAScript - Exploring JS, accessed May 2, 2025, https://exploringjs.com/es5/ch05.html
12. ECMAScript, TC39, and the History of JavaScript - ui.dev, accessed May 2, 2025, https://ui.dev/ecmascript
13. The History of JavaScript: A Journey from Netscape to Frameworks and Libraries, accessed May 2, 2025, https://www.techaheadcorp.com/knowledge-center/history-of-javascript/
14. Standard ECMA-327 ECMAScript 3 Edition Compact Profile, accessed May 2, 2025, https://www.ecma-international.org/wp-content/uploads/ECMA-327_1st_edition_june_2001.pdf
15. ECMAScript® 2017 Language Specification - TC39, accessed May 2, 2025,

https://tc39.es/ecma262/2017/
16. Ecma 262 - ECMAScript® 2026 Language Specification, accessed May 2, 2025, https://tc39.es/ecma262/multipage/
17. ECMAScript® 2016 Language Specification - TC39, accessed May 2, 2025, https://tc39.es/ecma262/2016/
18. Understanding the ECMAScript spec, part 3 - V8.Dev, accessed May 2, 2025, https://v8.dev/blog/understanding-ecmascript-part-3
19. The Evolution of JavaScript: From Netscape to ECMAScript - Codedamn, accessed May 2, 2025, https://codedamn.com/news/javascript/evolution-of-javascript-netscape-ecmascript
20. A Brief History of JavaScript - DEV Community, accessed May 2, 2025, https://dev.to/dboatengx/history-of-javascript-how-it-all-began-92a
21. History of JavaScript - read our article to find out! - SoftTeco, accessed May 2, 2025, https://softteco.com/blog/history-of-javascript
22. The Evolution of JavaScript: A Journey Through Its History - MVJ College of Engineering, accessed May 2, 2025, https://mvjce.edu.in/blog/evolution-of-javascript/
23. ECMAScript version history - Wikipedia, accessed May 2, 2025, https://en.wikipedia.org/wiki/ECMAScript_version_history
24. From Research Prototypes to Continuous Integration: Guiding the Design and Implementation of JavaScript | SIGPLAN Blog, accessed May 2, 2025, https://blog.sigplan.org/2023/01/12/from-research-prototypes-to-continuous-integration-guiding-the-design-and-implementation-of-javascript/
25. ECMA-262 - Ecma International, accessed May 2, 2025, https://ecma-international.org/publications-and-standards/standards/ecma-262/
26. Edition 3 Final ECMAScript Language Specification - Mozilla, accessed May 2, 2025, https://www-archive.mozilla.org/js/language/e262-3.pdf
27. ECMA-262, 3rd edition, December 1999, accessed May 2, 2025, https://ecma-international.org/wp-content/uploads/ECMA-262_3rd_edition_december_1999.pdf
28. Brief History of JavaScript - roadmap.sh, accessed May 2, 2025, https://roadmap.sh/guides/history-of-javascript
29. The Evolutionary Journey: A Brief History of ECMAScript, accessed May 2, 2025,

https://www.mehregansmart.com/en/content30.html
30. JavaScript / ECMAScript Versions… - DEV Community, accessed May 2, 2025, https://dev.to/avinashtechlvr/javascript-ecmascript-versions-m9n
31. 1995: The Birth of JavaScript | Cybercultural, accessed May 2, 2025, https://cybercultural.com/p/1995-the-birth-of-javascript/
32. A Brief History of JavaScript Frameworks - Primal Skill Programming, accessed May 2, 2025, https://primalskill.blog/a-brief-history-of-javascript-frameworks
33. The History of JavaScript: A Journey Through Time - By SW Habitation, accessed May 2, 2025, https://www.swhabitation.com/story/history-of-javascript
34. History of JavaScript on a Timeline - RisingStack Engineering, accessed May 2, 2025, https://blog.risingstack.com/history-of-javascript-on-a-timeline/
35. A Brief History of JavaScript - Auth0, accessed May 2, 2025, https://auth0.com/blog/a-brief-history-of-javascript/
36. The Weird History of JavaScript - DEV Community, accessed May 2, 2025, https://dev.to/codediodeio/the-weird-history-of-javascript-2bnb
37. 25 Years of JavaScript and Java! | Okta Developer, accessed May 2, 2025, https://developer.okta.com/blog/2020/12/04/25-years-javascript-java
38. Brendan Eich - Wikipedia, accessed May 2, 2025, https://en.wikipedia.org/wiki/Brendan_Eich
39. The History of JavaScript | Fireship.io, accessed May 2, 2025, https://fireship.io/courses/javascript/intro-history/
40. Brendan Eich: The TRUE History Of The Javascript Programming Language - YouTube, accessed May 2, 2025, https://www.youtube.com/watch?v=_UjYWgM8guY
41. Animation: The Rise and Fall of Popular Web Browsers Since 1994 - Visual Capitalist, accessed May 2, 2025, https://www.visualcapitalist.com/cp/the-rise-and-fall-of-popular-web-browsers-since-1994/
42. Animation: Internet Browser Market Share (1996-2019) - Visual Capitalist, accessed May 2, 2025, https://www.visualcapitalist.com/internet-browser-market-share/
43. History of the web browser - Wikipedia, accessed May 2, 2025, https://en.wikipedia.org/wiki/History_of_the_web_browser
44. The Secret Web Browser Monopoly | Fractional CISO, accessed May 2, 2025,

https://fractionalciso.com/the-secret-web-browser-monopoly/
45. Web Browser Market Share In 2025: 85+ Browser Usage Statistics - Backlinko, accessed May 2, 2025, https://backlinko.com/browser-market-share
46. Web Browser Market Share (1996-2019) : r/chromeos - Reddit, accessed May 2, 2025, https://www.reddit.com/r/chromeos/comments/d3o08c/web_browser_market_share_19962019/
47. Back to the Bad Old Days of the Web - Jorge Arango, accessed May 2, 2025, https://jarango.com/2021/07/02/back-to-the-bad-old-days-of-the-web/
48. Usage share of web browsers - Wikipedia, accessed May 2, 2025, https://en.wikipedia.org/wiki/Usage_share_of_web_browsers
49. Desktop Browser Market Share (2000 - 2021) #Shorts - YouTube, accessed May 2, 2025, https://www.youtube.com/watch?v=k1Pdu4kAU2c
50. Web browsers market share for the last 28 years : r/Infographics - Reddit, accessed May 2, 2025, https://www.reddit.com/r/Infographics/comments/vb5e8l/web_browsers_market_share_for_the_last_28_years/
51. 1997: Netscape Crossware vs the Windows Web | Cybercultural, accessed May 2, 2025, https://cybercultural.com/p/1997-netscape-crossware-vs-the-windows-web/
52. The TC39 Process, accessed May 2, 2025, https://tc39.es/process-document/
53. ECMAScript Development Archive - Ecma International, accessed May 2, 2025, https://ecma-international.org/ecmascript-development-archive/
54. TC39 - Ecma International, accessed May 2, 2025, https://ecma-international.org/technical-committees/tc39/
55. tc39/proposals: Tracking ECMAScript Proposals - GitHub, accessed May 2, 2025, https://github.com/tc39/proposals
56. GitHub - sudheerj/ECMAScript-features, accessed May 2, 2025, https://github.com/sudheerj/ECMAScript-features
57. Stages - ECMAScript Proposals, accessed May 2, 2025, https://www.proposals.es/stages
58. Start with ES3: A Key to Easing JavaScript Learning for Beginners - DEV Community, accessed May 2, 2025, https://dev.to/owens_akpede/start-with-es3-a-key-to-easing-javascript-learning-

for-beginners-41m
59. Control flow | web.dev, accessed May 2, 2025, https://web.dev/learn/javascript/control-flow
60. Control flow and error handling - JavaScript - MDN Web Docs, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling
61. var - JavaScript - MDN Web Docs - Mozilla, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var
62. Six Types of Scope in JavaScript: A Deep Dive for Developers - DEV Community, accessed May 2, 2025, https://dev.to/yugjadvani/five-types-of-scope-in-javascript-a-deep-dive-for-developers-285a
63. Variable Scope - JavaScript: The Definitive Guide, Fourth Edition [Book] - O'Reilly Media, accessed May 2, 2025, https://www.oreilly.com/library/view/javascript-the-definitive/0596000480/ch04s03.html
64. 9. Variables and scoping - Exploring JS, accessed May 2, 2025, https://exploringjs.com/es6/ch_variables.html
65. What is the scope of a function in Javascript/ECMAScript? - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/235360/what-is-the-scope-of-a-function-in-javascript-ecmascript
66. The History of JavaScript | Ample Blog, accessed May 2, 2025, https://www.ample.co/blog/javascript-history
67. JavaScript Versions | GeeksforGeeks, accessed May 2, 2025, https://www.geeksforgeeks.org/javascript-versions/
68. JavaScript data types and data structures - JavaScript - MDN Web Docs - Mozilla, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Data_structures
69. JavaScript data types | Modern JS, accessed May 2, 2025, https://www.modernjs.com/data-types.html
70. ECMAScript data types - EduTech Wiki, accessed May 2, 2025,

https://edutechwiki.unige.ch/en/ECMAScript_data_types
71. 6 ECMAScript Data Types and Values - TC39, accessed May 2, 2025, https://tc39.es/ecma262/multipage/ecmascript-data-types-and-values.html
72. What is an ECMAScript "native object"? - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/8052578/what-is-an-ecmascript-native-object
73. Re: [Q] OOP in ExtendScript / ECMAscript 3 - Adobe Community - 14368315, accessed May 2, 2025, https://community.adobe.com/t5/illustrator-discussions/q-oop-in-extendscript-ecmascript-3/m-p/14372645
74. 20 Fundamental Objects - ECMAScript® 2026 Language Specification, accessed May 2, 2025, https://tc39.es/ecma262/multipage/fundamental-objects.html
75. Class-based and Object-based languages comparison (ECMAScript Specification), accessed May 2, 2025, https://stackoverflow.com/questions/34010495/class-based-and-object-based-languages-comparison-ecmascript-specification
76. ECMAScript Operators - NetIQ Identity Manager - Administrator's Guide to Designing the Identity Applications, accessed May 2, 2025, https://www.netiq.com/documentation/identity-manager-49/identity_apps_design/data/ecmascript-operators.html
77. 13 ECMAScript Language: Expressions - TC39, accessed May 2, 2025, https://tc39.es/ecma262/multipage/ecmascript-language-expressions.html
78. Master ECMAScript: A comprehensive list with Real-World Examples - ServiceNow, accessed May 2, 2025, https://www.servicenow.com/community/developer-articles/master-ecmascript-a-comprehensive-list-with-real-world-examples/ta-p/2420283
79. How javascript(ECMAScript) assignment operator works - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/31311284/how-javascriptecmascript-assignment-operator-works
80. 4. Control Flow - Learning JavaScript, 3rd Edition [Book] - O'Reilly, accessed May 2, 2025, https://www.oreilly.com/library/view/learning-javascript-3rd/9781491914892/ch04.html

81. Regular expressions - JavaScript - MDN Web Docs, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_expressions
82. RegExp - JavaScript - MDN Web Docs - Mozilla, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp
83. Modified ECMAScript regular expression grammar - cppreference.com - C++ Reference, accessed May 2, 2025, https://en.cppreference.com/w/cpp/regex/ecmascript
84. ECMAScript regular expressions are getting better! - Mathias Bynens, accessed May 2, 2025, https://mathiasbynens.be/notes/es-regexp-proposals
85. try...catch - JavaScript - MDN Web Docs, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch
86. Error handling, "try...catch" - The Modern JavaScript Tutorial, accessed May 2, 2025, https://javascript.info/try-catch
87. JavaScript Exceptions - try...catch...finally - Codeguage, accessed May 2, 2025, https://www.codeguage.com/courses/js/exceptions-basics
88. Javascript error handling with try .. catch .. finally - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/286297/javascript-error-handling-with-try-catch-finally
89. Object prototypes - Learn web development | MDN, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Advanced_JavaScript_objects/Object_prototypes
90. Inheritance and the prototype chain - JavaScript - MDN Web Docs, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Inheritance_and_the_prototype_chain
91. ECMA-262-3 in detail. Chapter 7.1. OOP: The general theory. - Dmitry Soshnikov, accessed May 2, 2025, http://dmitrysoshnikov.com/ecmascript/chapter-7-1-oop-general-theory/
92. Understanding JavaScript Prototypes: A Key to Mastering OOP - DEV Community, accessed May 2, 2025,

https://dev.to/jps27cse/understanding-javascript-prototypes-a-key-to-mastering-oop-1j1b
93. Is the following explanation of `prototypes` in Javascript valid? - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/70549067/is-the-following-explanation-of-prototypes-in-javascript-valid
94. ECMAScript® 2026 Language Specification - TC39, accessed May 2, 2025, https://tc39.es/ecma262/
95. ECMAScript Language Specification - ECMA-262 Edition 5.1, accessed May 2, 2025, https://262.ecma-international.org/5.1/
96. ECMAScript 6 Features - GitHub Pages, accessed May 2, 2025, https://drstearns.github.io/tutorials/es6/
97. A gist which shows the difference between the ES5 and ES6 classes with a simple example using inheritance! : r/javascript - Reddit, accessed May 2, 2025, https://www.reddit.com/r/javascript/comments/cuyuqf/a_gist_which_shows_the_difference_between_the_es5/
98. this - JavaScript | MDN - MDN Web Docs - Mozilla, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/this
99. JavaScript this Keyword | GeeksforGeeks, accessed May 2, 2025, https://www.geeksforgeeks.org/javascript-this-keyword/
100. ECMA-262-3 in detail. Chapter 3. This. - Dmitry Soshnikov, accessed May 2, 2025, http://dmitrysoshnikov.com/ecmascript/chapter-3-this/
101. javascript - How does the "this" keyword work, and when should it be used? - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/3127429/how-does-the-this-keyword-work-and-when-should-it-be-used
102. Cannot understand "this" keyword : r/learnjavascript - Reddit, accessed May 2, 2025, https://www.reddit.com/r/learnjavascript/comments/zoxb19/cannot_understand_this_keyword/
103. js: one confusing point about 'this' keyword usage between ES5 and ES6 - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/43670878/js-one-confusing-point-about-thi

s-keyword-usage-between-es5-and-es6
104. JavaScript this Keyword Explained in 3 Minutes : r/learnjavascript - Reddit, accessed May 2, 2025, https://www.reddit.com/r/learnjavascript/comments/edppa1/javascript_this_keyword_explained_in_3_minutes/
105. The History of Simulated Classes in JavaScript - WebReflection, accessed May 2, 2025, https://www.webreflection.co.uk/blog/2015/11/07/the-history-of-simulated-classes-in-javascript
106. Array Methods - JavaScript: The Definitive Guide, 6th Edition [Book] - O'Reilly Media, accessed May 2, 2025, https://www.oreilly.com/library/view/javascript-the-definitive/9781449393854/ch07s08.html
107. A small utility to create ECMAScript `Array`s with members of a single type. - Reddit, accessed May 2, 2025, https://www.reddit.com/r/javascript/comments/1i6sqg7/a_small_utility_to_create_ecmascript_arrays_with/
108. Array - JavaScript - MDN Web Docs, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array
109. JavaScript New Arrays Methods: ECMAScript 2023 - DEV Community, accessed May 2, 2025, https://dev.to/abidullah786/javascript-new-arrays-methods-ecmascript-2023-1c31
110. Functions - JavaScript - MDN Web Docs, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions
111. ECMA-262-3 in detail. Chapter 5. Functions. - Dmitry Soshnikov, accessed May 2, 2025, http://dmitrysoshnikov.com/ecmascript/chapter-5-functions/
112. How to figure out what Javascript methods/features/etc are available in ECMAscript 3.0, accessed May 2, 2025, https://stackoverflow.com/questions/65961661/how-to-figure-out-what-javascript-methods-features-etc-are-available-in-ecmascri
113. Closures - JavaScript - MDN Web Docs, accessed May 2, 2025, https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Closures

114. ECMA-262-3 in detail. Chapter 6. Closures. - Dmitry Soshnikov, accessed May 2, 2025, http://dmitrysoshnikov.com/ecmascript/chapter-6-closures/
115. ECMASCRIPT Closures - What is Evaluation block in JavaScript? - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/75247311/ecmascript-closures-what-is-evaluation-block-in-javascript
116. Mastering Closures in JavaScript: A Comprehensive Guide - DEV Community, accessed May 2, 2025, https://dev.to/imranabdulmalik/mastering-closures-in-javascript-a-comprehensive-guide-4ja8
117. JavaScript closures in 17 minutes. With code and practical examples. : r/learnjavascript - Reddit, accessed May 2, 2025, https://www.reddit.com/r/learnjavascript/comments/qzhpcy/javascript_closures_in_17_minutes_with_code_and/
118. [AskJS] What is your opinion on using Closures instead of Object Prototype/Classes for OOP? : r/javascript - Reddit, accessed May 2, 2025, https://www.reddit.com/r/javascript/comments/dr58q3/askjs_what_is_your_opinion_on_using_closures/
119. JavaScript Closures - Using the ECMA Spec, please explain how the closure is created and maintained - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/15117687/javascript-closures-using-the-ecma-spec-please-explain-how-the-closure-is-cre
120. ES5, ES6, ES7, ES8, ES9: What's new in each Version of JavaScript - OdinSchool, accessed May 2, 2025, https://www.odinschool.com/blog/programming/java-script-versions
121. JavaScript Versions: How JavaScript has changed over the years - Educative.io, accessed May 2, 2025, https://www.educative.io/blog/javascript-versions-history
122. Difference between ES5 and ES6 - GeeksforGeeks, accessed May 2, 2025, https://www.geeksforgeeks.org/difference-between-es5-and-es6/
123. JavaScript ES5 vs ES6 Major Updates and Code Examples - AST Consulting, accessed May 2, 2025, https://astconsulting.in/java-script/javascript-es5-vs-es6
124. JavaScript ES5 Features - Tutorialspoint, accessed May 2, 2025, https://www.tutorialspoint.com/javascript/javascript_es5.htm

125. New features in ES5 - Manh Phan, accessed May 2, 2025, https://ducmanhphan.github.io/2019-02-28-New-features-in-ES5/
126. JavaScript ES5 (JS 2009) - GeeksforGeeks, accessed May 2, 2025, https://www.geeksforgeeks.org/javascript-es5-js-2009/
127. The Evolution of JavaScript: ES5 to ES2022 - DEV Community, accessed May 2, 2025, https://dev.to/rowsanali/the-evolution-of-javascript-es5-to-es2022-49p9
128. ES5 vs ES6 in JavaScript. 14 big changes⚠️ - DEV Community, accessed May 2, 2025, https://dev.to/diwakarkashyap/es5-vs-es6-in-javascript-14-big-changes-ple
129. How ReactJS ES6 syntax is different compared to ES5 - GeeksforGeeks, accessed May 2, 2025, https://www.geeksforgeeks.org/how-reactjs-es6-syntax-is-different-compared-to-es5/
130. Differences between ES5 and ES6 - DEV Community, accessed May 2, 2025, https://dev.to/dhanushaperera07/differences-between-es5-and-es6-24k0
131. ES5 vs ES6 - Front-End Engineering Curriculum - Turing School of Software and Design, accessed May 2, 2025, https://frontend.turing.edu/lessons/module-2/es5-vs-es6.html
132. Introduction to ES6 | GeeksforGeeks, accessed May 2, 2025, https://www.geeksforgeeks.org/introduction-to-es6/
133. ES6 to ES15 - Features list | JS - DEV Community, accessed May 2, 2025, https://dev.to/shubhamtiwari909/es6-to-es14-features-list-57am
134. Top 10 Features of ES6: A Comprehensive Guide to Modern JavaScript - Board Infinity, accessed May 2, 2025, https://www.boardinfinity.com/blog/top-10-features-of-es6/
135. lukehoban/es6features: Overview of ECMAScript 6 features - GitHub, accessed May 2, 2025, https://github.com/lukehoban/es6features
136. A Guide to Exploring JavaScript ES6 Features - Scribbler.live, accessed May 2, 2025, https://scribbler.live/2024/04/12/ES6-Features-with-Examples.html
137. Introduction to JavaScript | GeeksforGeeks, accessed May 2, 2025, https://www.geeksforgeeks.org/introduction-to-javascript/
138. How is Javascript used in web development? : r/webdev - Reddit, accessed May 2, 2025, https://www.reddit.com/r/webdev/comments/3gkrf4/how_is_javascript_used_in_

web_development/
139. What is the role of JavaScript in modern web development? - Nucamp Coding Bootcamp, accessed May 2, 2025, https://www.nucamp.co/blog/coding-bootcamp-full-stack-web-and-mobile-development-what-is-the-role-of-javascript-in-modern-web-development
140. What kinda things can you do with Js? : r/learnjavascript - Reddit, accessed May 2, 2025, https://www.reddit.com/r/learnjavascript/comments/ejw5zn/what_kinda_things_can_you_do_with_js/
141. Using JavaScript Next Features in an ES3 Enterprise World - Telerik Blogs, accessed May 2, 2025, https://www.telerik.com/blogs/using-javascript-next-features-es3-enterprise-world
142. [AskJS] I asked ChatGPT if I can still code in ES3 (ECMA Script 1) : r/javascript - Reddit, accessed May 2, 2025, https://www.reddit.com/r/javascript/comments/1ftppdl/askjs_i_asked_chatgpt_if_i_can_still_code_in_es3/
143. javascript - How Client Side Programming Works - Software Engineering Stack Exchange, accessed May 2, 2025, https://softwareengineering.stackexchange.com/questions/205851/how-client-side-programming-works
144. Re-Evolution of JavaScript | Simply Technologies, accessed May 2, 2025, https://www.simplytechnologies.net/reevolution-of-javascript
145. History of Javascript | ecmascript - YouTube, accessed May 2, 2025, https://www.youtube.com/watch?v=NQUjEo9iqfo
146. Node.js server-side javascript process consuming too much memory, accessed May 2, 2025, https://developercommunity.visualstudio.com/content/problem/27033/nodejs-server-side-javascript-process-consuming-to.html
147. Rendering javascript at the server side level. A good or bad idea? - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/2847176/rendering-javascript-at-the-server-side-level-a-good-or-bad-idea
148. How to list object properties on server side scripting? - ServiceNow, accessed

May 2, 2025, https://www.servicenow.com/community/service-management-forum/how-to-list-object-properties-on-server-side-scripting/m-p/336412
149. Which Edition of ECMA-262 Does Google Apps Script Support? - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/17252409/which-edition-of-ecma-262-does-google-apps-script-support
150. A Brief History and Evolution of JavaScript - Zipy.ai, accessed May 2, 2025, https://www.zipy.ai/blog/brief-history-and-evolution-of-javascript
151. HTML to Svelte 4: The Evolution of Web Development Frameworks - Codesphere, accessed May 2, 2025, https://codesphere.com/articles/the-evolution-of-web-development-frameworks
152. The Complete ECMAScript 2015-2019 Guide - Flavio Copes, accessed May 2, 2025, https://flaviocopes.com/ecmascript/
153. 3. ES5, ES6 and their features | Javascript tutorial for beginners - YouTube, accessed May 2, 2025, https://www.youtube.com/watch?v=iEZdKWHI5bA
154. A Brief History of JavaScript - Brendan Eich, accessed May 2, 2025, https://brendaneich.com/2010/07/a-brief-history-of-javascript/
155. A On the design of the ECMAScript Reflection API - Google Research, accessed May 2, 2025, https://research.google.com/pubs/archive/37741.pdf
156. Development and Operating Platforms - ES3, accessed May 2, 2025, https://www.es3inc.com/mars-solver/development-and-operating-platforms/
157. ES3, accessed May 2, 2025, https://www.es3inc.com/
158. ELI5: The meaning and differences of es5 and es6 in Javascript. : r/webdev - Reddit, accessed May 2, 2025, https://www.reddit.com/r/webdev/comments/106ipom/eli5_the_meaning_and_differences_of_es5_and_es6/
159. Using ES6 features with all browsers support - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/34205167/using-es6-features-with-all-browsers-support
160. Target different Javascript versions with TypeScript - Stack Overflow, accessed May 2, 2025,

https://stackoverflow.com/questions/38951661/target-different-javascript-versions-with-typescript

161. Which ECMA version should I aim for? : r/learnjavascript - Reddit, accessed May 2, 2025, https://www.reddit.com/r/learnjavascript/comments/1huwwqf/which_ecma_version_should_i_aim_for/
162. ES7, ES8, ES9, ES10, ES11 Browser support - Stack Overflow, accessed May 2, 2025, https://stackoverflow.com/questions/61835971/es7-es8-es9-es10-es11-browser-support
163. Exploring Advanced JavaScript Features: A Deep Dive into ECMAScript 2023 Updates, accessed May 2, 2025, https://blog.greenroots.info/exploring-advanced-javascript-features-ecmascript-2023