JavaScript

From Wikipedia, the free encyclopedia

In computing, **JavaScript** (/ˈdʒævəˌskrɪpt/[5]) is a high-level, dynamic, untyped, and interpreted programming language.[6] It has been standardized in the ECMAScript language specification.[7] Alongside HTML and CSS, JavaScript is one of the three core technologies of World Wide Web content production; the majority of websites employ it, and all modern Web browsers supported it without the need for plug-ins.[6] JavaScript is prototype-based with first-class functions, making it a multi-paradigm language, supporting object-oriented,[8] imperative, and functional programming styles.[6] It has an API for working with text, arrays, dates and regular expressions, but does not include any I/O, such as networking, storage, or graphics facilities, relying for these upon the host environment in which it is embedded.[7]

Although there are strong outward similarities between JavaScript and Java, including language name, syntax, and respective standard libraries, the two are distinct languages and differ greatly in their design. JavaScript was influenced by programming languages such as Self and Scheme.[9]

JavaScript is also used in environments that are not Web-based, such as PDF documents, site-specific browsers, and desktop widgets. Newer and faster JavaScript virtual machines (VMs) and platforms built upon them have also increased the popularity of JavaScript for server-side Web applications. On the client side, developers have traditionally implemented JavaScript as an interpreted language, but more recent browsers perform just-in-time compilation. Programmers also use JavaScript in video-game development, in crafting desktop and mobile applications, and in server-side network programming with run-time environments such as Node.js.

JavaScript	
Paradigm	Multi-paradigm: scripting, object-oriented (prototype-based), imperative, functional, event-driven[1]
Designed by	Brendan Eich
Developer	Netscape Communications Corporation, Mozilla Foundation, Ecma International
First appeared	May 23, 1995
Stable release	ECMAScript 2016[2] / June 17, 2016
Typing discipline	dynamic, duck
Major implementations	
V8, JavaScriptCore, SpiderMonkey, Chakra	
Influenced by	
Lua, Scheme, Perl, Self, Java, C, Python, AWK, HyperTalk	
Influenced	
ActionScript, AtScript, CoffeeScript, Dart, JScript .NET, LiveScript, Objective-J, Opa, Perl 6, QML, TypeScript	
〰 JavaScript at Wikibooks	

JavaScript	
Filename extension	`.js`
Internet media type	`application/javascript` · `text/javascript` (obsolete)[3]
Uniform Type Identifier (UTI)	com.netscape.javascript-source [4]
Type of format	Scripting language

Contents

History

Beginnings at Netscape

In 1993, the National Center for Supercomputing Applications (NCSA), a unit of the University of Illinois at Urbana-Champaign, released NCSA Mosaic, the first popular graphical Web browser, which played an important part in expanding the growth of the nascent World Wide Web. In 1994, a company called Mosaic Communications was founded in Mountain View, California and employed many of the original NCSA Mosaic authors to create Mosaic Netscape. However, it intentionally shared no code with NCSA Mosaic. The internal codename for the company's browser was Mozilla, which stood for "Mosaic killer", as the company's goal was to displace NCSA Mosaic as the world's number one web browser. The first version of the Web browser, Mosaic Netscape 0.9, was released in late 1994. Within four months it had already taken three-quarters of the browser market and became the main browser for Internet in the 1990s. To avoid trademark ownership problems with the NCSA, the browser was subsequently renamed Netscape Navigator in the same year, and the company took the name Netscape Communications.

Netscape Communications realized that the Web needed to become more dynamic. Marc Andreessen, the founder of the company believed that HTML needed a "glue language" that was easy to use by Web designers and part-time programmers to assemble components such as images and plugins, where the code could be written directly in the Web page markup. In 1995, the company recruited Brendan Eich with the goal of embedding the Scheme programming language into its Netscape Navigator. Before he could get started, Netscape Communications collaborated with Sun Microsystems to include in Netscape Navigator Sun's more static programming language Java, in order to compete with Microsoft for user adoption of Web technologies and platforms.[10] Netscape Communications then decided that the scripting language they wanted to create would complement Java and should have a similar syntax, which excluded adopting other languages such as Perl, Python, TCL, or Scheme. To defend the idea of JavaScript against competing proposals, the company needed a prototype. Eich wrote one in 10 days, in May 1995.

Although it was developed under the name **Mocha**, the language was officially called **LiveScript** when it first shipped in beta releases of Netscape Navigator 2.0 in September 1995, but it was renamed **JavaScript**[11] when it was deployed in the Netscape Navigator 2.0 beta 3 in December.[12] The final choice of name caused confusion, giving the impression that the language was a spin-off of the Java programming language, and the choice has been characterized as a marketing ploy by Netscape to give JavaScript the cachet of what was then the hot new Web programming language.

There is a common misconception that JavaScript was influenced by an earlier Web page scripting language developed by Nombas named C-- (not to be confused with the later C-- created in 1997).[13][14] Brendan Eich, however, had never heard of C-- before he created LiveScript.[15] Nombas did pitch their embedded Web page scripting to Netscape, though Web page scripting was not a new concept, as shown by the ViolaWWW Web browser.[16] Nombas later switched to offering JavaScript instead of C-- in their ScriptEase product and was part of the TC39 group that standardized ECMAScript.[17]

Server-side JavaScript

Netscape introduced an implementation of the language for server-side scripting with Netscape Enterprise Server in December 1995, soon after releasing JavaScript for browsers.[18] Since the mid-2000s, there has been a resurgence of server-side JavaScript implementations, such as Node.js[19] and MarkLogic.[20]

Adoption by Microsoft

Microsoft script technologies including VBScript and JScript were released in 1996. JScript, a reverse-engineered implementation of Netscape's JavaScript, was part of Internet Explorer 3 as well as being available server-side in Internet Information Server. Internet Explorer 3 also included Microsoft's first support for CSS and various extensions to HTML, but in each case the implementation was noticeably different to that found in Netscape Navigator at the time.[21][22] These differences made it difficult for designers and programmers to make a single website work well in both browsers, leading to the use of "best viewed in Netscape" and "best viewed in Internet Explorer" logos that characterized these early years of the browser wars.[23] JavaScript began to acquire a reputation for being one of the roadblocks to a cross-platform and standards-driven Web. Some developers took on the difficult task of trying to make their sites work in both major browsers, but many could not afford the time.[21] With the release of Internet Explorer 4, Microsoft introduced the concept of Dynamic HTML, but the differences in language implementations and the different and proprietary Document Object Models remained and were obstacles to widespread take-up of JavaScript on the Web.[21]

Standardization

In November 1996, Netscape submitted JavaScript to Ecma International to carve out a standard specification, which other browser vendors could then implement based on the work done at Netscape. This led to the official release of the language specification ECMAScript published in the first edition of the ECMA-262 standard in June 1997, with JavaScript being the most well known of the implementations. ActionScript and JScript are other well-known implementations of ECMAScript, with extensions.

The standards process continued in cycles, with the release of ECMAScript 2 in June 1998, which brings some modifications to conform to the ISO/IEC 16262 international standard. The release of ECMAScript 3 followed in December 1999, which is the baseline for modern day JavaScript. The original ECMAScript 4 work led by Waldemar Horwat (then at Netscape, now at Google) started in 2000 and at first, Microsoft seemed to participate and even implemented some of the proposals in their JScript .NET language.

Over time it was clear though that Microsoft had no intention of cooperating or implementing proper JavaScript in Internet Explorer, even though they had no competing proposal and they had a partial (and diverged at this point) implementation on the .NET server side. So by 2003, the original ECMAScript 4 work was mothballed.

The next major event was in 2005, with two major happenings in JavaScript's history. First, Brendan Eich and Mozilla rejoined Ecma International as a not-for-profit member and work started on ECMAScript for XML (E4X), the ECMA-357 standard, which came from ex-Microsoft employees at BEA Systems (originally acquired as Crossgain). This led to working jointly with Macromedia (later acquired by Adobe Systems), who were implementing E4X in ActionScript 3 (ActionScript 3 was a fork of original ECMAScript 4).

So, along with Macromedia, work restarted on ECMAScript 4 with the goal of standardizing what was in ActionScript 3. To this end, Adobe Systems released the ActionScript Virtual Machine 2, code named Tamarin, as an open source project. But Tamarin and ActionScript 3 were too different from web JavaScript to converge, as was realized by the parties in 2007 and 2008.

Alas, there was still turmoil between the various players; Douglas Crockford—then at Yahoo!—joined forces with Microsoft in 2007 to oppose ECMAScript 4, which led to the ECMAScript 3.1 effort. The development of ECMAScript 4 was never completed, but that work influenced subsequent versions.[24]

While all of this was happening, the open source and developer communities set to work to revolutionize what could be done with JavaScript. This community effort was sparked in 2005 when 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, used to create web applications where data can be loaded in the background, avoiding the need for full page reloads and leading to more dynamic applications. This resulted in a renaissance period of JavaScript usage spearheaded by open source libraries and the communities that formed around them, with libraries such as Prototype, jQuery, Dojo Toolkit, MooTools and others being released.

In July 2008, the disparate parties on either side came together in Oslo. This led to the eventual agreement in early 2009 to rename ECMAScript 3.1 to ECMAScript 5 and drive the language forward using an agenda that is known as Harmony. ECMAScript 5 was finally released in December 2009.

In June 2011, ECMAScript 5.1 was released to fully align with the third edition of the ISO/IEC 16262 international standard. ECMAScript 2015 was released in June 2015. The current version is ECMAScript 2016, released in June 2016.[25]

Later developments

JavaScript has become one of the most popular programming languages on the Web. Initially, however, many professional programmers denigrated the language because, among other reasons, its target audience consisted of Web authors and other such "amateurs".[26] The advent of Ajax returned JavaScript to the spotlight and brought more professional programming attention. The result was a proliferation of comprehensive frameworks and libraries, improved JavaScript programming practices, and increased usage of JavaScript outside Web browsers, as seen by the proliferation of server-side JavaScript platforms.

In January 2009, the CommonJS project was founded with the goal of specifying a common standard library mainly for JavaScript development outside the browser.[27]

With the rise of single-page applications and JavaScript-heavy sites, it is increasingly being used as a compile target for source-to-source compilers from both dynamic languages and static languages.

Trademark

"JavaScript" is a trademark of Oracle Corporation.[28] It is used under license for technology invented and implemented by Netscape Communications and current entities such as the Mozilla Foundation.[29]

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: JavaScript originally had only function scoping with `var`. ECMAScript 2015 added a `let` keyword for block scoping, meaning JavaScript now has both function and block scoping. Like C, JavaScript makes a distinction between expressions and statements. One syntactic difference from C is automatic semicolon insertion, which allows the semicolons that would normally terminate statements to be omitted.[30]

Dynamic

Typing
> As with most scripting languages, JavaScript is dynamically typed; a type is associated with each value, rather than just with each expression. For example, a variable that is at one time bound to a number may later be re-bound to a string.[31] JavaScript supports various ways to test the type of an object, including duck typing.[32]

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

Prototype-based (Object-oriented)

JavaScript is almost entirely object-based. In JavaScript, an object is an associative array, augmented with a prototype (see below); each string 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.

JavaScript has a small number of built-in objects, including `Function` and `Date`.

Prototypes
JavaScript uses prototypes where many other object-oriented languages use classes for inheritance.[33] It is possible to simulate many class-based features with prototypes in JavaScript.[34]

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).[35] 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`).[36] 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.[37]

Functions as methods
Unlike many object-oriented languages, 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

A function is first-class; a function is considered to be an object. As such, a function may have properties and methods, such as `.call()` and `.bind()`.[38] 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.[39] 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[40] patterns like Traits[41][42] and Mixins.[43] 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

Run-time environment

JavaScript typically relies on a run-time environment (e.g., a Web browser) to provide objects and methods by which scripts can interact with the environment (e.g., a webpage DOM). It also relies on the run-time environment to provide the ability to include/import scripts (e.g., HTML `<script>` elements). This is not a language feature per se, but it is common in most JavaScript implementations.

JavaScript processes messages from a queue one at a time. Upon loading a new message, JavaScript calls a function associated with that message, which creates a call stack frame (the function's arguments and local variables). The call stack shrinks and grows based on the function's needs. Upon function completion, when the stack is empty, 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 input/output is performed using events and callback functions. This means, for instance, that JavaScript can process a mouse click while waiting for a database query to return information.[44]

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_Obj ects/Function/bind)` method.

Array and object literals

Like 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

JavaScript also supports regular expressions in a manner similar to Perl, which provide a concise and powerful syntax for text manipulation that is more sophisticated than the built-in string functions.[45]

Vendor-specific extensions

JavaScript is officially managed by Mozilla Foundation, and new language features are added periodically. However, only some JavaScript engines support these new features:

- property getter and setter functions (supported by WebKit, Gecko, Opera,[46] ActionScript, and Rhino)[47]
- conditional `catch` clauses
- iterator protocol (adopted from Python)
- shallow generators-coroutines (adopted from Python)
- array comprehensions and generator expressions (adopted from Python)
- proper block scope via the `let` keyword
- array and object destructuring (limited form of pattern matching)
- concise function expressions (`function(args) expr`)
- ECMAScript for XML (E4X), an extension that adds native XML support to ECMAScript (unsupported in Firefox since version 21[48])

Syntax

Simple examples

Variables in JavaScript can be defined using the `var` keyword:[49]

```javascript
var x; // defines the variable x and assigns to it the special value "undefined" (not to be confused with an undefin
var y = 2; // defines the variable y and assigns to it the value 2
```

Note the comments in the example above, both of which were preceded with two forward slashes.

There is no built-in I/O functionality in JavaScript; the run-time environment provides that. The ECMAScript specification in edition 5.1 mentions:[50]

> … indeed, there are no provisions in this specification for input of external data or output of computed results.

However, most runtime environments have a `console` object [51] that can be used to print output. Here is a minimalist Hello World program:

```javascript
console.log("Hello World!");
```

A simple recursive function:

```javascript
function factorial(n) {
    if (n == 0) {
        return 1;
    }

    return n * factorial(n - 1);
}
```

Multiplication of two numbers:

```html
<html>
<body>
    <h1>Multiplication</h1>
    Enter Number 1:<input type="text" id="a1"><br>
    Enter Number 2:<input type="text" id="a2"><br>
    <button onclick="myFunction()">Product</button>
    <p id="a3"></p>
    <script>
        function myFunction(){
            var a=parseInt(document.getElementById("a1").value);
            var b=parseInt(document.getElementById("a2").value);
            var c=a*b;
            document.getElementById("a3").innerHTML=c;

        }
    </script>
</body>
</html>
```

An anonymous function (or lambda):

```javascript
function counter() {
    var count = 0;
    return function () {
        return ++count;
    };
};
```

```
var closure = counter();
closure(); // returns 1
closure(); // returns 2
closure(); // returns 3
```

This example shows that in JavaScript, function closures captures their non-local variables *by reference*.

Variadic function demonstration (`arguments` is a special variable):[52]

```
function sum() {
    var x = 0;

    for (var i = 0; i < arguments.length; ++i) {
        x += arguments[i];
    }

    return x;
}

sum(1, 2); // returns 3
sum(1, 2, 3); // returns 6
```

Immediately-invoked function expressions are often used to create modules, as before ECMAScript 2015 there was not built-in construct in the language. Modules allow gathering properties and methods in a namespace and making some of them private:

```
var counter = (function () {
    var i = 0; // private property

    return {      // public methods
        get: function () {
            alert(i);
        },
        set: function (value) {
            i = value;
        },
        increment: function () {
            alert(++i);
        }
    };
})(); // module

counter.get();       // shows 0
counter.set(6);
counter.increment(); // shows 7
counter.increment(); // shows 8
```

More advanced example

This sample code displays various JavaScript features.

```
/* Finds the lowest common multiple (LCM) of two numbers */
function LCMCalculator(x, y) { // constructor function
    var checkInt = function (x) { // inner function
        if (x % 1 !== 0) {
            throw new TypeError(x + " is not an integer"); // throw an exception
        }
        return x;
    };
    this.a = checkInt(x)
    //   semicolons   ^^^^  are optional, a newline is enough
    this.b = checkInt(y);
```

```javascript
}
// The prototype of object instances created by a constructor is
// that constructor's "prototype" property.
LCMCalculator.prototype = { // object literal
    constructor: LCMCalculator, // when reassigning a prototype, set the constructor property appropriately
    gcd: function () { // method that calculates the greatest common divisor
        // Euclidean algorithm:
        var a = Math.abs(this.a), b = Math.abs(this.b), t;
        if (a < b) {
            // swap variables
            t = b;
            b = a;
            a = t;
        }
        while (b !== 0) {
            t = b;
            b = a % b;
            a = t;
        }
        // Only need to calculate GCD once, so "redefine" this method.
        // (Actually not redefinition—it's defined on the instance itself,
        // so that this.gcd refers to this "redefinition" instead of LCMCalculator.prototype.gcd.
        // Note that this leads to a wrong result if the LCMCalculator object members "a" and/or "b" are altered aft
        // Also, 'gcd' === "gcd", this['gcd'] === this.gcd
        this['gcd'] = function () {
            return a;
        };
        return a;
    },
    // Object property names can be specified by strings delimited by double (") or single (') quotes.
    lcm : function () {
        // Variable names don't collide with object properties, e.g., |lcm| is not |this.lcm|.
        // not using |this.a*this.b| to avoid FP precision issues
        var lcm = this.a/this.gcd()*this.b;
        // Only need to calculate lcm once, so "redefine" this method.
        this.lcm = function () {
            return lcm;
        };
        return lcm;
    },
    toString: function () {
        return "LCMCalculator: a = " + this.a + ", b = " + this.b;
    }
};

// Define generic output function; this implementation only works for Web browsers
function output(x) {
    document.body.appendChild(document.createTextNode(x));
    document.body.appendChild(document.createElement('br'));
}

// Note: Array's map() and forEach() are defined in JavaScript 1.6.
// They are used here to demonstrate JavaScript's inherent functional nature.
[[25, 55], [21, 56], [22, 58], [28, 56]].map(function (pair) { // array literal + mapping function
    return new LCMCalculator(pair[0], pair[1]);
}).sort(function (a, b) { // sort with this comparative function
    return a.lcm() - b.lcm();
}).forEach(function (obj) {
    output(obj + ", gcd = " + obj.gcd() + ", lcm = " + obj.lcm());
});
```

The following output should be displayed in the browser window.

```
LCMCalculator: a = 28, b = 56, gcd = 28, lcm = 56
LCMCalculator: a = 21, b = 56, gcd = 7, lcm = 168
LCMCalculator: a = 25, b = 55, gcd = 5, lcm = 275
LCMCalculator: a = 22, b = 58, gcd = 2, lcm = 638
```

Use in Web pages

The most common use of JavaScript is to add client-side behavior to HTML pages, also known as Dynamic HTML (DHTML). Scripts are embedded in or included from HTML pages and interact with the Document Object Model (DOM) of the page. Some simple examples of this usage are:

- Loading new page content or submitting data to the server via Ajax without reloading the page (for example, a social network might allow the user to post status updates without leaving the page).
- Animation of page elements, fading them in and out, resizing them, moving them, etc.
- Interactive content, for example games, and playing audio and video.
- Validating input values of a Web form to make sure that they are acceptable before being submitted to the server.
- Transmitting information about the user's reading habits and browsing activities to various websites. Web pages frequently do this for Web analytics, ad tracking, personalization or other purposes.[53]

Because JavaScript code can run locally in a user's browser (rather than on a remote server), the browser can respond to user actions quickly, making an application more responsive. Furthermore, JavaScript code can detect user actions that HTML alone cannot, such as individual keystrokes. Applications such as Gmail take advantage of this: much of the user-interface logic is written in JavaScript, and JavaScript dispatches requests for information (such as the content of an e-mail message) to the server. The wider trend of Ajax programming similarly exploits this strength.

A JavaScript engine (also known as JavaScript interpreter or JavaScript implementation) is an interpreter that interprets JavaScript source code and executes the script accordingly. The first JavaScript engine was created by Brendan Eich at Netscape, for the Netscape Navigator Web browser. The engine, code-named SpiderMonkey, is implemented in C. It has since been updated (in JavaScript 1.5) to conform to ECMAScript 3. The Rhino engine, created primarily by Norris Boyd (formerly at Netscape, now at Google) is a JavaScript implementation in Java. Rhino, like SpiderMonkey, is ECMAScript 3 compliant.

A Web browser is by far the most common host environment for JavaScript. Web browsers typically create "host objects" to represent the DOM in JavaScript. The Web server is another common host environment. A JavaScript Web server would typically expose host objects representing HTTP request and response objects, which a JavaScript program could then interrogate and manipulate to dynamically generate Web pages.

Because JavaScript is the only language that the most popular browsers share support for, it has become a target language for many frameworks in other languages, even though JavaScript was never intended to be such a language.[54] Despite the performance limitations inherent to its dynamic nature, the increasing speed of JavaScript engines has made the language a surprisingly feasible compilation target.

Example script

Below is a minimal example of a standards-conforming Web page containing JavaScript (using HTML 5 syntax) and the DOM:

```
<!DOCTYPE html>
<html><head><title>Minimal Example</title><script>window.onload = function() {document.body.appendChild(document.cre
            var h1 = document.getElementById('header');
            h1 = document.getElementsByTagName('h1')[0];}</script></head><body><h1 id="header">This is JavaScri
```

Compatibility considerations

Because JavaScript runs in widely varying environments, an important part of testing and debugging is to test and verify that the JavaScript works across multiple browsers.

The DOM interfaces for manipulating Web pages are not part of the ECMAScript standard, or of JavaScript itself. Officially, the DOM interfaces are defined by a separate standardization effort by the W3C; in practice, browser implementations differ from the standards and from each other, and not all browsers execute JavaScript.

To deal with these differences, JavaScript authors can attempt to write standards-compliant code that will also be executed correctly by most browsers; failing that, they can write code that checks for the presence of certain browser features and behaves differently if they are not available.[55] In some cases, two browsers may both implement a feature but with different behavior, and authors may find it practical to detect what browser is running and change their script's behavior to match.[56][57] Programmers may also use libraries or toolkits that take browser differences into account.

Furthermore, scripts may not work for some users. For example, a user may:

- use an old or rare browser with incomplete or unusual DOM support;
- use a PDA or mobile phone browser that cannot execute JavaScript;
- have JavaScript execution disabled as a security precaution;
- use a speech browser due to, for example, a visual disability.

To support these users, Web authors can try to create pages that degrade gracefully on user agents (browsers) that do not support the page's JavaScript. In particular, the page should remain usable albeit without the extra features that the JavaScript would have added. An alternative approach that many find preferable is to first author content using basic technologies that work in all browsers, then enhance the content for users that have JavaScript enabled. This is known as progressive enhancement.

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 contain 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 Web site 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 level of security, especially on code created by third parties (such as advertisements).[58][59] Caja is another project for safe embedding and isolation of third-party JavaScript and HTML.

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

Cross-site vulnerabilities

A common JavaScript-related security problem is cross-site scripting (XSS), a violation of the same-origin policy. XSS vulnerabilities occur when an attacker is able to cause a target Web site, 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. A solution to XSS vulnerabilities is to use *HTML escaping* whenever displaying untrusted data.

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.[60]

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 didn't intend at a target site (like transferring money at a bank). It works because, if the target site relies only on cookies to authenticate requests, then requests initiated by code on the attacker's site will carry the same legitimate login credentials as requests initiated by the 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 application author cannot assume that his 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:

- Web site 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 can't be relied on to prevent operations such as right-clicking on an image to save it.[61]
- It is extremely bad practice to embed sensitive information such as passwords in JavaScript because it can be extracted by an attacker.

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,[62] Internet Explorer,[63] and Safari.[64]

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).[65][66]

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.[67] 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. Of course, such privileges aren't meant 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[68] and Firefox.[69] In Windows XP Service Pack 2, Microsoft demoted JScript's privileges in Internet Explorer.[70]

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.[71]

Uses outside Web pages

In addition to Web browsers and servers, JavaScript interpreters are embedded in a number of tools. Each of these applications provides its own object model that provides access to the host environment. The core JavaScript language remains mostly the same in each application.

Embedded scripting language

- Google's Chrome extensions, Opera's extensions, Apple's Safari 5 extensions, Apple's Dashboard Widgets, Microsoft's Gadgets, Yahoo! Widgets, Google Desktop Gadgets, and Serence Klipfolio are implemented using JavaScript.
- The MongoDB database accepts queries written in JavaScript. MongoDB and NodeJS are the core components of MEAN: a solution stack for creating Web applications using just JavaScript.
- The Clusterpoint database accept queries written in JS/SQL, which is a combination of SQL and JavaScript. Clusterpoint has built-in computing engine that allows execution of JavaScript code right inside the distributed database.
- Adobe's Acrobat and Adobe Reader support JavaScript in PDF files.[72]
- Tools in the Adobe Creative Suite, including Photoshop, Illustrator, Dreamweaver, and InDesign, allow scripting through JavaScript.
- OpenOffice.org, an office application suite, as well as its popular fork LibreOffice, allows JavaScript to be used as a scripting language.
- The visual programming language Max, released by Cycling '74, offers a JavaScript model of its environment for use by developers. It allows users to reduce visual clutter by using a object for a task rather than many.
- Apple's Logic Pro X digital audio workstation (DAW) software can create custom MIDI effects plugins using JavaScript.
- ECMAScript was included in the VRML97 standard for scripting nodes of VRML scene description files.
- The Unity game engine supports a modified version of JavaScript for scripting via Mono.[73]
- DX Studio (3D engine) uses the SpiderMonkey implementation of JavaScript for game and simulation logic.[74]
- Maxwell Render (rendering software) provides an ECMA standard based scripting engine for tasks automation.[75]
- Google Apps Script in Google Spreadsheets and Google Sites allows users to create custom formulas, automate repetitive tasks and also interact with other Google products such as Gmail.[76]
- Many IRC clients, like ChatZilla or XChat, use JavaScript for their scripting abilities.[77][78]
- RPG Maker MV uses Javascript as its scripting language.[79]

Scripting engine

- Microsoft's Active Scripting technology supports JScript as a scripting language.[80]
- Java introduced the `javax.script` package in version 6 that includes a JavaScript implementation based on Mozilla Rhino. Thus, Java applications can host scripts that access the application's variables and objects, much like Web browsers host scripts that access a webpage's Document Object Model (DOM).[81][82]
- The Qt C++ toolkit includes a `QtScript` module to interpret JavaScript, analogous to Java's `javax.script` package.[83]
- OS X Yosemite introduced JavaScript for Automation (JXA), which is built upon JavaScriptCore and the Open Scripting Architecture. It features an Objective-C bridge that enables entire Cocoa applications to be programmed in JavaScript.
- Late Night Software's JavaScript OSA (also known as JavaScript for OSA, or JSOSA) is a freeware alternative to AppleScript for OS X. It is based on the Mozilla JavaScript 1.5 implementation, with the addition of a `MacOS` object for interaction with the operating system and third-party applications.[84]

Application platform

- ActionScript, the programming language used in Adobe Flash, is another implementation of the ECMAScript standard.
- Adobe Integrated Runtime is a JavaScript runtime that allows developers to create desktop applications.
- Atom, the open-source text editor developed by Github, was implemented using Javascript, and has a special API on Javascript for packages that are developed for it.
- CA, Inc.'s AutoShell cross-application scripting environment is built on the SpiderMonkey JavaScript engine. It contains preprocessor-like extensions for command definition, as well as custom classes for various system-related tasks like file I/O, operation system command invocation and redirection, and COM scripting.
- GNOME Shell, the shell for the GNOME 3 desktop environment,[85] made JavaScript its default programming language in 2013.[86]
- The Mozilla platform, which underlies Firefox, Thunderbird, and some other Web browsers, uses JavaScript to implement the graphical user interface (GUI) of its various products.
- Qt Quick's markup language (available since Qt 4.7) uses JavaScript for its application logic. Its declarative syntax is also similar to JavaScript.
- TypeScript is a programming language based on JavaScript that adds support for optional type annotations and some other language extensions such as classes, interfaces and modules. A TS-script compiles into plain JavaScript and can be executed in any JS host supporting ECMAScript 3 or higher. The compiler is itself written in TypeScript.
- Ubuntu Touch provides a JavaScript API for its unified usability interface.
- webOS uses the WebKit implementation of JavaScript in its SDK to allow developers to create stand-alone applications solely in JavaScript.
- WinJS provides a special Windows Library for JavaScript functionality in Windows 8 that enables the development of Modern style (formerly *Metro style*) applications in HTML5 and JavaScript.

Development tools

Within JavaScript, access to a debugger becomes invaluable when developing large, non-trivial programs. Because there can be implementation differences between the various browsers (particularly within the DOM), it is useful to have access to a debugger for each of the browsers that a Web application targets.[87]

Script debuggers are integrated within Internet Explorer, Firefox, Safari, Google Chrome, Opera and Node.js.[88][89][90]

In addition to the native Internet Explorer Developer Tools, three debuggers are available for Internet Explorer: Microsoft Visual Studio is the richest of the three, closely followed by Microsoft Script Editor (a component of Microsoft Office),[91] and finally the free Microsoft Script Debugger that is far more basic than the other two. The free Microsoft Visual Web Developer Express provides a limited version of the JavaScript debugging functionality in Microsoft Visual Studio. Internet Explorer has included developer tools since version 8.

In comparison to Internet Explorer, Firefox has a more comprehensive set of developer tools, which include a debugger as well. Old versions of Firefox without these tools used a Firefox addon called Firebug, or the older Venkman debugger. Also, WebKit's Web Inspector includes a JavaScript debugger,[92] which is used in Safari. A modified version called Blink DevTools is used in Google Chrome. Node.js has Node Inspector, an interactive debugger that integrates with the Blink DevTools, available in Google Chrome. Opera includes a set of tools called Dragonfly.[93]

In addition to the native computer software, there are online JavaScript IDEs, debugging aids that are themselves written in JavaScript and built to run on the Web. An example is the program JSLint, developed by Douglas Crockford who has written extensively on the language. JSLint scans JavaScript code for conformance to a set of standards and guidelines. Many libraries for JavaScript, such as three.js, provide links to demonstration code that can be edited by users. They are also used as a pedagogical tool by institutions such as Khan Academy[94] to allow students to experience writing code in an environment where they can see the output of their programs, without needing any setup beyond a Web browser.

Benchmark tools for developers

Since JavaScript is getting more important for web development (frontend overtakes many aspects which were done in backend before), there is also more consideration done about performance. Especially mobile devices could have problems with rendering and processing unoptimized complex logic.

A library for doing benchmarks is benchmark.js (https://benchmarkjs.com/). A benchmarking library that supports high-resolution timers and returns statistically significant results.

Another tool is jsben.ch (http://jsben.ch). An online JavaScript benchmarking tool, where code snippets can be tested against each other.

Version history

JavaScript was initially developed in 1996 for use in the Netscape Navigator Web browser. In the same year Microsoft released an implementation for Internet Explorer. This implementation was called JScript due to trademark issues. In 1997 the first standardized version of the language was released under the name ECMAScript in the first edition of the ECMA-252 standard. The explicit versioning and opt-in of language features was Mozilla-specific and has been removed. Firefox 4 was the last version which referred to a JavaScript version (1.8.5). With new editions of the ECMA-262 standard, JavaScript language features are now often mentioned with their initial definition in the ECMA-262 editions.

The following table is based on information from multiple sources.[95][96][97]

Version	Release date	Equivalent to	Netscape Navigator	Mozilla Firefox	Internet Explorer	Opera	Safari	Google Chrome
1.0	March 1996		2.0		3.0			
1.1	August 1996		3.0					
1.2	June 1997		4.0-4.05			3[98]		
1.3	October 1998	ECMA-262 1st + 2nd edition	4.06-4.7x		4.0	5[99]		
1.4			Netscape Server			6		
1.5	November 2000	ECMA-262 3rd edition	6.0	1.0	5.5 (JScript 5.5), 6 (JScript 5.6), 7 (JScript 5.7), 8 (JScript 5.8)	7.0	3.0-5	1.0-10.0.666
1.6	November 2005	1.5 + array extras + array and string generics + E4X		1.5				
1.7	October 2006	1.6 + Pythonic generators (https://developer.mozilla.org/en-US/docs/JavaScript/New_in_JavaScript/1.7?redirectlocale=en-US&redirectslug=New_in_JavaScript_1.7#Generators) + iterators + let		2.0				28.0.1500.95
1.8	June 2008	1.7 + generator expressions + expression closures		3.0		11.50		
1.8.1		1.8 + native JSON support + minor updates		3.5				
1.8.2	June 22, 2009	1.8.1 + minor updates		3.6				
1.8.5	July 27, 2010	1.8.2 + new features for ECMA-262 5th edition compliance		4.0				

Related languages and features

JSON, or JavaScript Object Notation, is a general-purpose data interchange format that is defined as a subset of JavaScript's object literal syntax. Like much of JavaScript (regexps and anonymous functions as 1st class elements, closures, flexible classes, 'use strict'), JSON, except for replacing Perl's key-value operator '=>' by an RFC 822[100] inspired ':', is syntactically pure Perl.

jQuery is a popular JavaScript library designed to simplify DOM-oriented client-side HTML scripting along with offering cross-browser compatibility because various browsers respond differently to certain vanilla JavaScript code.

Underscore.js is a utility JavaScript library for data manipulation that is used in both client-side and server-side network applications.

AngularJS is a web application framework to use for developing single-page applications and also cross-platform mobile apps.

React (JavaScript library) is an open-source JavaScript library providing a views that is rendered using components specified as custom HTML tags.

Mozilla browsers currently support LiveConnect, a feature that allows JavaScript and Java to intercommunicate on the Web. However, Mozilla-specific support for LiveConnect is scheduled to be phased out in the future in favor of passing on the LiveConnect handling via NPAPI to the Java 1.6+ plug-in (not yet supported on the Mac as of March 2010).[101] Most browser inspection tools, such as Firebug in Firefox, include JavaScript interpreters that can act on the visible page's DOM.

asm.js is a subset of JavaScript that can be run in any JavaScript engine or run faster in an ahead-of-time (AOT) compiling engine.[102]

JSFuck is an esoteric programming language. Programs are written using only six different characters, but are still valid JavaScript code.

p5.js[103] is an object oriented JavaScript library designed for artists and designers. It is based on the ideas of the Processing project but is for the web.

jsben.ch (http://jsben.ch) is an online JavaScript benchmarking tool, where different code snippets can be tested against each other.

Use as an intermediate language

As JavaScript is the most widely supported client-side language that can run within a Web browser, it has become an intermediate language for other languages to target. This has included both newly created languages and ports of existing languages. Some of these include:

- OberonScript, a full implementation of the Oberon programming language that compiles to high-level JavaScript.[104]
- Objective-J, a superset of JavaScript that compiles to standard JavaScript. It adds traditional inheritance and Smalltalk/Objective-C style dynamic dispatch and optional pseudo-static typing to JavaScript.
- Processing.js, a JavaScript port of the Processing programming language designed to write visualizations, images, and interactive content. It allows Web browsers to display animations, visual applications, games and other graphical rich content without the need for a Java applet or Flash plugin.
- CoffeeScript, an alternate syntax for JavaScript intended to be more concise and readable. It adds features like array comprehensions (also available in JavaScript since version 1.7)[105] and pattern matching. Like Objective-J, it compiles to JavaScript. Ruby and Python have been cited as influential on CoffeeScript syntax.
- Google Web Toolkit translates a subset of Java to JavaScript.
- Scala, an object-oriented and functional programming language, has a Scala-to-JavaScript compiler.[106]
- Pyjamas, a port of Google Web Toolkit to Python translates a subset of Python to JavaScript.
- Dart, an open-source programming language developed by Google, can be compiled to JavaScript.

- Whalesong,[107] a Racket-to-JavaScript compiler.
- Emscripten, a LLVM-backend for porting native libraries to JavaScript.
- Fantom a programming language that runs on JVM, .NET and JavaScript.
- TypeScript, a free and open-source programming language developed by Microsoft. It is a superset of JavaScript, and essentially adds optional static typing and class-based object-oriented programming to the language.
- Haxe, an open-source high-level multiplatform programming language and compiler that can produce applications and source code for many different platforms including JavaScript.
- ClojureScript,[108] a compiler for Clojure that targets JavaScript. It is designed to emit JavaScript code that is compatible with the advanced compilation mode of the Google Closure optimizing compiler.
- Kotlin, a statically-typed language that also compiles to Java byte code.
- SqueakJS (https://bertfreudenberg.github.io/SqueakJS/), a virtual machine and DOM environment for the open-source Squeak implementation of the Smalltalk programming language.

As JavaScript has unusual limitations – such as no separate integer type, using floating point – languages that compile to JavaScript commonly have slightly different behavior than in other environments.

JavaScript and Java

A common misconception is that JavaScript is similar or closely related to Java. It is true that both have a C-like syntax (the C language being their most immediate common ancestor language). They also are both typically sandboxed (when used inside a browser), 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,[109] but the similarities end there.

Java and JavaScript both first appeared on 23 May 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.

References

1. Flanagan 2011, pp. 1–2.
2. "Standard ECMA-262". Ecma International. 2015-06-17.
3. "RFC 4329". Apps.ietf.org. Archived from the original on 2014-03-16. Retrieved 16 February 2016.
4. "System-Declared Uniform Type Identifiers". *Mac OS X Reference Library*. Apple Inc. Retrieved 2010-03-05.
5. "JavaScript". *Collins English Dictionary – Complete & Unabridged 2012 Digital Edition*. William Collins Sons & Co. 2012. Retrieved 21 August 2015.
6. Flanagan 2011, p. 1.
7. Flanagan 2011, p. 2.
8. "ECMAScript Language Specification" (PDF).
9. "ECMAScript Language Overview" (PDF). 2007-10-23. p. 4. Retrieved 2009-05-03.
10. Severance, Charles (February 2012). "JavaScript: Designing a Language in 10 Days". *Computer*. IEEE Computer Society. **45** (2): 7–8. doi:10.1109/MC.2012.57. Retrieved 23 March 2013.
11. Press release announcing JavaScript (https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html), "Netscape and Sun announce JavaScript", PR Newswire, December 4, 1995
12. "TechVision: Innovators of the Net: Brendan Eich and JavaScript". web.archive.org. Archived from the original on 2008-02-08.

13. "The History of Programming Languages". *oreilly.com*. O'Reilly Media. 2004. Retrieved 16 July 2016.
14. "What Is JavaScript?" (PDF). *wiley.com*. Wiley. Retrieved 16 July 2016.
15. Noorda, Brent (21 September 2013). "Brent Noorda's Answers on ECMA". *quora.com*. Retrieved 16 July 2016.
16. Noorda, Brent (24 June 2010). "History of Nombas". *brent-noorda.com*. Retrieved 16 July 2016.
17. Eich, Brendan (21 June 2011). "New JavaScript Engine Module Owner". *brendaneich.com*. Retrieved 16 July 2016.
18. "Server-Side JavaScript Guide". *oracle.com*. Netscape Communications Corporation. 11 December 1998. Retrieved 16 July 2016.
19. "Server-Side JavaScript, Back with a Vengeance". *readwrite.com*. 17 December 2009. Retrieved 16 July 2016.
20. "Server-side Javascript for Developers". *marklogic.com*. 1 January 2015. Retrieved 9 August 2016.
21. Champeon, Steve (6 April 2001). "JavaScript, How Did We Get Here?". *oreilly.com*. Retrieved 16 July 2016.
22. "Microsoft Internet Explorer 3.0 Beta Now Available". *microsoft.com*. Microsoft. 29 May 1996. Retrieved 16 July 2016.
23. McCracken, Harry (16 September 2010). "The Unwelcome Return of "Best Viewed with Internet Explorer" ". *technologizer.com*. Retrieved 16 July 2016.
24. "Documentation". *ecmascript.org*. Retrieved 16 July 2016.
25. "Standard ECMA-262, ECMAScript® 2016 Language Specification". *ecma-international.org*. June 2016. Retrieved 16 July 2016.
26. Crockford, Douglas (2001). "JavaScript, The World's Most Misunderstood Programming Language". *crockford.com*. Retrieved 16 July 2016.
27. Kowal, Kris (1 December 2009). "CommonJS Effort Sets JavaScript on Path for World Domination". *arstechnica.com*. Retrieved 16 July 2016.
28. "USPTO Copyright entry #75026640". USPTO.
29. "Sun Trademarks". Sun Microsystems. Archived from the original on 28 May 2010. Retrieved 2007-11-08.
30. Flanagan 2006, p. 16.
31. JavaScript data types and data structures (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures)
32. Flanagan 2006, pp. 176–178.
33. "Inheritance and the prototype chain". *Mozilla Developer Network*. Mozilla. Retrieved 6 April 2013.
34. Herman, David (2013). *Effective JavaScript*. Addison-Wesley. p. 83. ISBN 978-0-321-81218-6.
35. Haverbeke, Marijn (2011). *Eloquent JavaScript*. No Starch Press. pp. 95–97. ISBN 978-1-59327-282-1.
36. Katz, Yehuda. "Understanding "Prototypes" in JavaScript". Retrieved 6 April 2013.
37. Herman, David (2013). *Effective JavaScript*. Addison-Wesley. pp. 125–127. ISBN 978-0-321-81218-6.
38. "Properties of the Function Object". Es5.github.com. Retrieved 2013-05-26.
39. Flanagan 2006, p. 141.
40. The many talents of JavaScript for generalizing Role-Oriented Programming approaches like Traits and Mixins (http://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), April 11, 2014.
41. Traits for JavaScript (http://soft.vub.ac.be/~tvcutsem/traitsjs/), 2010.
42. CocktailJS – ANNOTATIONS. TRAITS. TALENTS. (http://cocktailjs.github.io/), April 2013.
43. Angus Croll, A fresh look at JavaScript Mixins (http://javascriptweblog.wordpress.com/2011/05/31/a-fresh-look-at-javascript-mixins/), published May 31, 2011.
44. "Concurrency model and Event Loop". *Mozilla Developer Network*. Retrieved 2015-08-28.
45. Haverbeke, Marijn (2011). *Eloquent JavaScript*. No Starch Press. pp. 139–149. ISBN 978-1-59327-282-1.
46. Robert Nyman, Getters And Setters With JavaScript – Code Samples And Demos (http://robertnyman.com/2009/05/28/getters-and-setters-with-javascript-code-samples-and-demos/), published 29 May 2009, accessed 2 January 2010.
47. John Resig, JavaScript Getters and Setters (http://ejohn.org/blog/javascript-getters-and-setters/), 18 July 2007, accessed 2 January 2010
48. "E4X – Archive of obsolete content | MDN". *Mozilla Developer Network*. Mozilla Foundation. Feb 14, 2014. Retrieved 13 July 2014.
49. "var – JavaScript – MDN". The Mozilla Developer Network. Retrieved 22 December 2012.
50. "ECMAScript Language Specification – ECMA-262 Edition 5.1". Ecma International. Retrieved 22 December 2012.
51. "console". *Mozilla Developer Network*. Mozilla. Retrieved 6 April 2013.
52. "arguments". *Mozilla Developer Network*. Mozilla. Retrieved 6 April 2013.
53. "JavaScript tracking – Piwik". Piwik. Retrieved 31 March 2012.
54. Hamilton, Naomi (2008-07-31). "The A-Z of Programming Languages: JavaScript". computerworld.com.au.
55. Peter-Paul Koch, Object detection (http://www.quirksmode.org/js/support.html)
56. Peter-Paul Koch, Mission Impossible – mouse position (http://www.evolt.org/node/23335)
57. Peter-Paul Koch, Browser detect (http://www.quirksmode.org/js/detect.html)
58. "Making JavaScript Safe for Advertising". ADsafe. Retrieved 2013-05-26.

59. "Secure ECMA Script (SES)". Code.google.com. Retrieved 2013-05-26.
60. MozillaZine, Mozilla Cross-Site Scripting Vulnerability Reported and Fixed (http://www.mozillazine.org/talkback.html?article=4392)
61. "Right-click "protection"? Forget about it". 2008-06-17. ISSN 1797-1993. Retrieved 2008-06-17.
62. Mozilla Corporation, Buffer overflow in crypto.signText() (http://www.mozilla.org/security/announce/2006/mfsa2006-38.html)
63. Festa, Paul (August 19, 1998). "Buffer-overflow bug in IE". *CNET*.
64. SecurityTracker.com, Apple Safari JavaScript Buffer Overflow Lets Remote Users Execute Arbitrary Code and HTTP Redirect Bug Lets Remote Users Access Files (http://securitytracker.com/alerts/2006/Mar/1015713.html)
65. SecurityFocus, Microsoft WebViewFolderIcon ActiveX Control Buffer Overflow Vulnerability (http://www.securityfocus.com/bid/19030/info)
66. Fusion Authority, Macromedia Flash ActiveX Buffer Overflow (http://www.fusionauthority.com/security/3234-macromedia-flash-activex-buffer-overflow.htm)
67. Mike Friedman, Protected Mode in Vista IE7 (http://blogs.msdn.com/ie/archive/2006/02/09/528963.aspx)
68. 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)
69. Mozilla Foundation, Mozilla Foundation Security Advisory 2005–41: Privilege escalation via DOM property overrides (http://www.mozilla.org/security/announce/2005/mfsa2005-41.html)
70. Microsoft Corporation, Changes to Functionality in Microsoft Windows XP Service Pack 2: Part 5: Enhanced Browsing Security (http://technet.microsoft.com/en-us/library/bb457150.aspx#EHAA)
71. For one example of a rare JavaScript Trojan Horse, see Symantec Corporation, JS.Seeker.K (http://www.symantec.com/security_response/writeup.jsp?docid=2003-100111-0931-99)
72. "JavaScript for Acrobat". Retrieved 2009-08-18.
73. "Unity Scripting". unity3d.com. Retrieved 2013-01-29.
74. "Technical Specification". dxstudio.com. Retrieved 2009-10-20.
75. THINK! The Maxwell Render Resourcer Center, Scripting References (http://think.maxwellrender.com/scripting_references-269.html)
76. Google Apps Script, Google Apps Script (https://www.google.com/script/start/)
77. "ChatZilla! Frequently Asked Questions – 4.5. How do I write scripts?". Chatzilla.hacksrus.com. Retrieved 11 February 2011.
78. "Xcdscript". Retrieved 11 February 2011.
79. "RPG Maker MV | RPG Maker | Make Your Own Games!". Retrieved 28 August 2015.
80. "Version Information (JavaScript)". Msdn.microsoft.com. Retrieved 2013-05-26.
81. "javax.script release notes". Java.sun.com. Retrieved 2009-05-19.
82. Flanagan 2006, pp. 214 et seq.
83. Nokia Corporation, QtScript Module (http://doc.qt.nokia.com/4.6/qtscript.html)
84. Open Scripting Architecture
85. "Behind the Scenes with Owen Taylor". The GNOME Journal. Archived from the original on 2012-12-21. Retrieved 2010-01-23.
86. "Answering the question: "How do I develop an app for GNOME?" ".
87. "Advanced Debugging With JavaScript". alistapart.com. 2009-02-03. Retrieved 2010-05-28.
88. "The JavaScript Debugging Console". javascript.about.com. 2010-05-28. Retrieved 2010-05-28.
89. "SplineTech JavaScript Debugger – an independent standalone JavaScript Debugger". javascript-debugger.com. 2013-08-26. Retrieved 2013-08-26.
90. "Debugging with Node Inspector". docs.strongloop.com. Retrieved 2014-05-07.
91. JScript development in Microsoft Office 11 (http://msdn2.microsoft.com/en-us/library/aa202668(office.11).aspx) (MS InfoPath 2003)
92. "Introducing Drosera – Surfin' Safari". Webkit.org. 2006-06-28. Retrieved 2009-05-19.
93. "Opera DragonFly". Opera Software.
94. "Khan Academy Computer Science". Retrieved 28 Sep 2012.
95. "New in JavaScript". developer.mozilla.org. 2014. Retrieved 2016-07-16.
96. "JavaScript – JScript – ECMAScript version history". Webmasterworld.com. Retrieved 2009-12-17.
97. John Resig. "Versions of JavaScript". Ejohn.org. Retrieved 2009-05-19.
98. History of the Opera web browser#Version 3
99. "What Version of JavaScript".
100. "RFC 822".

101. Release Notes for the Next-Generation Java™ Plug-In Technology (introduced in Java SE 6 update 10) (http://java.sun.com/javase/6/webnotes/6u10/plugin2/liveconnect/). Java.sun.com. Retrieved on 2013-06-13.
102. "frequently asked questions". asm.js. Retrieved 2014-04-13.
103. "p5.js".
104. Ralph Sommerer. "Oberon Script. A Lightweight Compiler and Runtime System for the Web". research.microsoft.com. Retrieved 2015-12-18.
105. "New in JavaScript 1.7". Developer.mozilla.org. 2012-12-05. Retrieved 2013-05-26.
106. Sébastien Doeraene. "Scala.js". Lampwww.epfl.ch. Retrieved 2014-04-13.
107. "Whalesong: a Racket to JavaScript compiler". Hashcollision.org. Retrieved 2014-04-13.
108. "clojure/clojurescript · GitHub". Github.com. Retrieved 2014-04-13.
109. Brendan Eich (3 April 2008). "Popularity". Retrieved 2012-01-19.

Further reading

- Bhangal, Sham; Jankowski, Tomasz (2003). *Foundation Web Design: Essential HTML, JavaScript, CSS, PhotoShop, Fireworks, and Flash*. APress L. P. ISBN 1-59059-152-6.
- Burns, Joe; Growney, Andree S. (2001). *JavaScript Goodies*. Pearson Education. ISBN 0-7897-2612-2.
- Duffy, Scott (2003). *How to do Everything with JavaScript*. Osborne. ISBN 0-07-222887-3.
- Zakas, Nicholas C. (2012). *Professional JavaScript for Web Developers* (3rd ed.). Wrox. ISBN 978-1-118-02669-4.
- Flanagan, David (2006). *JavaScript: The Definitive Guide* (5th ed.). O'Reilly & Associates. ISBN 0-596-10199-6.
- Flanagan, David (2011). *JavaScript: The Definitive Guide* (6th ed.). O'Reilly & Associates. ISBN 978-0-596-80552-4.
- Goodman, Danny; Eich, Brendan (2001). *JavaScript Bible*. John Wiley & Sons. ISBN 0-7645-3342-8.
- Goodman, Danny; Markel, Scott (2003). *JavaScript and DHTML Cookbook*. O'Reilly & Associates. ISBN 0-596-00467-2.
- Harris, Andy (2001). *JavaScript Programming for the Absolute Beginner*. Premier Press. ISBN 0-7615-3410-5.
- Haverbeke, Marijn (2011). *Eloquent JavaScript*. No Starch Press. ISBN 978-1-59327-282-1.
- Heinle, Nick; Koman, Richard (1997). *Designing with JavaScript*. O'Reilly & Associates. ISBN 1-56592-300-6.
- Husted, Robert; Kuslich, JJ (1999). *Server-Side JavaScript: Developing Integrated Web Applications* (1st ed.). Addison-Wesley. ISBN 0-201-43329-X.
- McDuffie, Tina Spain (2003). *JavaScript Concepts & Techniques: Programming Interactive Web Sites*. Franklin, Beedle & Associates. ISBN 1-887902-69-4.
- McFarlane, Nigel (2003). *Rapid Application Development with Mozilla*. Prentice Hall Professional Technical References. ISBN 0-13-142343-6.
- Powell, Thomas A.; Schneider, Fritz (2001). *JavaScript: The Complete Reference*. McGraw-Hill Companies. ISBN 0-07-219127-9.
- Shelly, Gary B.; Cashman, Thomas J.; Dorin, William J.; Quasney, Jeffrey J. (2000). *JavaScript: Complete Concepts and Techniques*. Cambridge: Course Technology. ISBN 0-7895-6233-2.
- Vander Veer, Emily A. (2004). *JavaScript For Dummies* (4th ed.). Wiley Pub. ISBN 0-7645-7659-3.
- Watt, Andrew H.; Watt, Jonathan A.; Simon, Jinjer L. (2002). *Teach Yourself JavaScript in 21 Days*. Pearson Education. ISBN 0-672-32297-8.

External links

- Douglas Crockford's video lectures on JavaScript (https://www.youtube.com/playlist?list=PL62E185BB8577B63D)
- Douglas Crockford's A Survey of the JavaScript Programming Language (http://javascript.crockford.com/survey.html)
- JavaScript (https://www.dmoz.org/Computers/Programming/Languages/JavaScript/) at DMOZ
- List of languages that compile to JS (https://github.com/jashkenas/coffee-script/wiki/List-of-languages-that-compile-to-JS/)

- This page was last modified on 8 November 2016, at 05:32.