JavaScript: Here's What You Need To "Learn JavaScript Deeply"

Sharing is caring!

Last Updated on January 15, 2020

If you were to ask a developer back in the early 2000s what the most important programming language for web development was going to be, the last thing they would have said is JavaScript. Flash, PHP, and Java were all strong contenders for dominance. JavaScript, they might say, is a lightweight language, a toy language for doing silly little browser effects and (at best) some basic form data validation.

But today, JavaScript is perhaps the most important web development language. Even though most server-side development is done with one of the more "conventional" programming languages, just about every website uses some amount of JavaScript for "front-end" functionality. Moreover, the evolution of the language itself and the standardization of browser technology has given rise to in-browser web apps powered almost entirely on JavaScript. JavaScript frameworks have thrown that trend to hyperdive.

Additionally, JavaScript is no longer just a front-end language. Thanks to the Node.js project, it is now possible to easily use JavaScript to build applications server-side as well. And with the rise of technologies such as MongoDB that use JSON as a data serialization language, it is now possible to use JavaScript for an entire web application, from data storage to server-side processing to front-end UI.

Whatever other programming languages you use or are interested in learning, it is increasingly clear that JavaScript is the *sine qua non* of modern web development. It is time for all of us to "learn JavaScript deeply."

Contents [hide]

About the language

History

JavaScript was originally developed at Netscape, for use in their browser Netscape Navigator. This was in the mid-1990s, during the height of the browser wars, when each browser developer was attempting to win market share by creating unique features unavailable in other browsers. There was also a line of thinking during that period that browsers would be, in many ways, an extension of the operating system and general user experience. There seems to have been some thought that Netscape's browser would be part of a distributed operating system running a portable version of the Java environment. This would be complemented by a lightweight scripting language, which was officially called *LiveScript* and developed under the project code name *Mocha*.

It isn't clear whether it was that plan, or a simple marketing ploy, but this "lightweight interpreted language" eventually got name "JavaScript," even though the language had no real connection to Java.

JavaScript was added to the Netscape Navigator browser in 1996, and Microsoft quickly reverse-engineered a JavaScript implementation for their Internet Explorer browser, which they called JScript. Unfortunately, the implementations were quite different. Different approaches to HTML and what became CSS complicated matters even more, as JavaScript exists primarily to manipulate objects within the document — different document models produced different results. It was common during this period to see websites instruct their users that the site "Works Best on Netscape" or "Works Best on Internet Explorer." (AOL, which eventually bought Netscape, also had an in-app browser. But nothing worked the best on it.)

Netscape submitted JavaScript to Ecma, as a potential international, industry-wide standard. In 1997, the standard version, called "ECMAScript" was released. There have been a number of subsequent version releases since then. Ecma, and not Netscape, now determines the course of the language.

Thanks to a general trend toward browser standardization that began in the early 2000s, and was largely complete by the middle of the 2010s, most browsers implement ECMAScript (and the HTML Document Object Model) in largely similar ways. Today, you can be fairly confident that a JavaScript-based application written according to widely-recognized best practices will work on all major browsers.

Language features

Object Orientation

JavaScript is object-oriented. It's a little weird in this way, though; there are objects, but no classes. So many "classical" object-oriented programmers often find it somewhat deficient, or would say it isn't *actually* object-oriented at all. But it is.

object. New objects are then cloned from there, but new methods and properties can be added to the new objects as needed.

There are some advantages and disadvantages to this approach. One (possible) advantage is that if you just need a single instance of an object, you don't have the overhead of writing an entire class *and then* writing the code that instantiates it. Given JavaScripts primary domain (web documents) this makes a lot of sense.

The main disadvantage is basically philosophical in nature. In a complex application, it will certainly be the case that you would write a number of prototype objects that are note *used*, but only cloned into new objects — essentially re-creating a class-based paradigm. Except the original objects are still there, as objects. There is no essential difference between them, even though they are being used in fundamentally different ways. The class-object dichotomy represents a particular ("classical," even) view of reality — abstract ideas about categories and types of things which are instantiated into real objects. JavaScript presents a messier (but, probably, more realistic) view of reality in which there are no abstract, Platonic notions; there is only this object, and this one, and this one, and so on.

```
function car(make, model, year, color) {
    this.make = make;
    this.model = model;
    this.year = year;
    this.color = color;
}

var codersCar = new car("Volkswagen", "Golf", "2008", "Black")
```

Structure

JavaScript, like most modern programming languages, follows the **structured programming model**. The syntax is somewhat similar to C, with `if`, `else`, `for`, and `while` all behaving as may be expected.

Unlike C, JavaScript *mostly* only **supports function-level variable scope**. Block level scoping was added somewhat recently but (as of this writing), it is still much safer to code as if this were not present.

Speaking of variables, **variables in JavaScript are dynamically typed**. This means you don't have to declare a variable and its data type (string, number, etc.) ahead of actually assigning the value to the variable. This is quite different from C, but is pretty common among other interpreted scripting languages.

Functions in JavaScript are first-class, which means that they can be passed as arguments to other functions or stored as variables. This allows (among other things) functions to be defined inside of other functions. This is used frequently in in-browser applications to cause functions to get defined at particular points. For example, in the following jQuery code, the inner function adjusts the `target` property of all on-page links so that they open in a new tab if they link to another domain. This function is created inside a function that is only called when the document is ready (fully loaded). This is a minor example (which happens to be used on this website), but illustrates something done frequently in JavaScript.

```
jQuery( document ).ready( function ( e ) {

    // Open outbound links in a new window.
    jQuery(document.links)
        .filter(function() {
            return this.hostname != window.location.hostname;
        })
        .attr('target', '_blank');
```

The Event Loop

JavaScript relies on an **event loop** which listens for messages (such as mouse clicks, key strokes, or messages created by other object), and queues those messages for processing. Functions can be created that listen for certain messages, and are then triggered when those events happen (such as the example above, which triggers the inner function when the `document ready` event occurs).

This *event driven architecture* makes perfect sense for JavaScript's original (and still primary) purpose — website user interaction. It allows the application to listen for user-created, unprompted inputs. More recently, this feature has been exploited in server-side environments like Node.js. Having a built-in event-listening system makes it (relatively) easy to build real-time applications in domains such as chat, gaming, collaboration, and dispatch.

JavaScript Events

JavaScript provides a way of interacting with, and manipulating, the content of a web page. This can be done through code stored in a linked `.js` file, in an on-page `<script>` code block, or by accessing JavaScript event triggers within the document's elements.

An event is like a signal or message that something has happened. For example, every time your mouse clicks on an element of a page, the `onclick` event for that element is fired. Under normal circumstances, this doesn't cause anything else to happen. But you can use those events to trigger other code to run. You can write code that says (in essence): *Everytime this thing happens, do this other thing.*

You can imagine the possibilities here:

- Every time this picture is clicked… load the next picture.
- Every time the mouse enters this area… change the color.
- Every time the mouse approaches the navigation bar … open a modal window with a special offer.

JavaScript, being a language designed for user-interation, is "event-driven". There are a lot of events in JavaScript, and bunch of them are tied to HTML elements.

Events as HTML Attributes

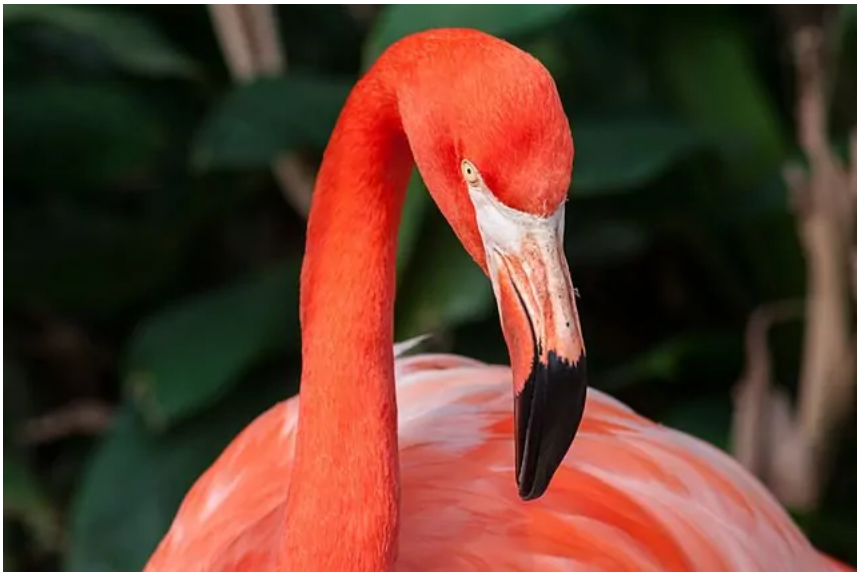
All HTML elements on a webpage have a set of built-in events which fire under certain circumstances. You can access these events as if they were attributes of the element:

```
<!-- Explanation of markup. -->
<img src="example.jpg" onclick="{ some JS code here ">


<!-- Working Example -->
<style>
#flamingo-picture-1 {
 border: 15px solid red;
}
</style>

<img id="flamingo-picture-1" src="/wp-content/uploads/flamingo.jpg" onclick="
document.getElementById('flamingo-picture-1').style.border='15px groove blue'">
```

```
border: 15px solid red;
}
```



If you are doing much more than one single line, it can get a bit too much to try to keep all of it inside the `onclick` attribute. In that case, you might just want to call a function from the event.

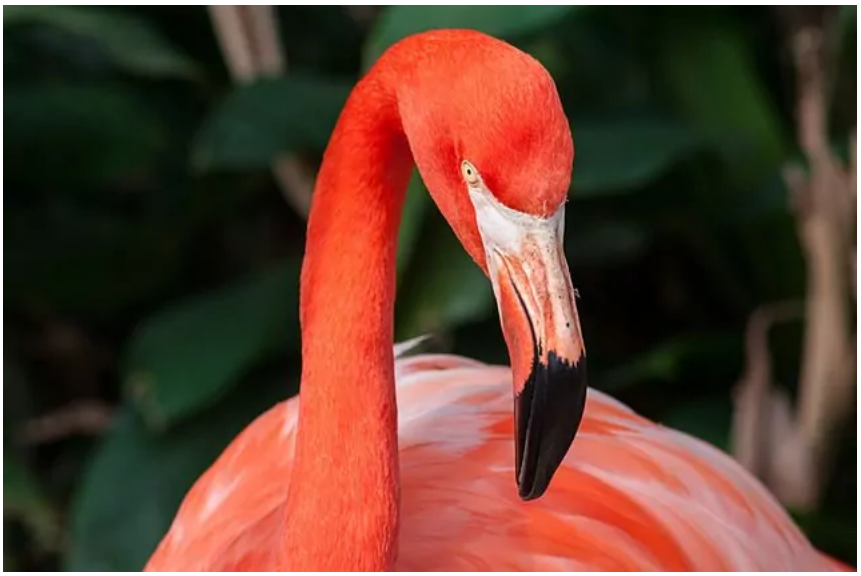
```
<style>
#flamingo-picture-2 {
 border-width: 15px;
 border-style: solid;
 border-color: red;
}
</style>

<script>
var changeBorderColor = function(){
 img = document.getElementById('flamingo-picture-2');
 if ( img.style.border-color == 'red' ) {
  img.style.border-color = 'blue';
 } else {
  img.style.border-color = 'red';
}
</script>

<img id="flamingo-picture-2" src="/wp-content/uploads/flamingo.jpg" onclick="changeBorderColor">
```

```
#flamingo-picture-2{
border-width: 15px;
border-style: solid;
border-color: red;
}
```

```
if (img.style.borderColor === 'red') {
img.style.borderColor = 'blue';
} else {
img.style.borderColor = 'red';
}
}
```



List of JavaScript Events on HTML Elements

Most of the names describe exactly when they fire. A few of them are only applicable to a few kinds of elements. See JavaScript Elements for code samples and demonstrations.

onclick
When the user clicks on the element. (Mouse button is pressed and released immediately.)

oncontextmenu
When the context menu ("right-click") is opened (or when the user tries to open it — this can be used to disable the context menu).

ondblclick
When the element is double-clicked.

onmousedown
When the user presses the mouse button down while inside (or over) the element. (Mouse button is pressed and held.)

onmouseenter
When the mouse enters the element. (Sometimes called "hover," but that is not the name of the event.)

onmouseleave
When the mouse leaves an element.

onmousemove
When the mouse moves while inside an element.

onmouseout
When the mouse exits an element, or one of its children.

onmouseup
When the user releases the mouse button, while the mouse is inside the element.

onerror
When an error occurs while loading an external file (most often used on media files).

onload
When the elements completes loading (mostly used on media files, but can be used also for `<body>`).

onscroll
When the content is scrolled.

Programming Notes

The Document Object Model

HTML is used for your page's content. CSS controls how everything looks. How do you use JavaScript to manipulate what is on the screen and what it looks like? To you use JavaScript to manipulate the HTML directly?

The answer is the DOM, or the Document Object Model. And no, you do not manipulate the HTML directly. You manipulate the DOM.

The DOM is the browser's own representation of a web page, based on the HTML and CSS. But the DOM is not a document, is the current, live state of page *as it is displayed in the browser currently*. This means that the DOM knows things like element size, playback position and volume, which element is being clicked on or hovered over. Also, the DOM updates as it is being manipulated. So if you change the content of an element, that change is made in the DOM and reflected in the view (the browser's window).

Ajax

JavaScript is the *J* in *Ajax*, which originally stood for *Asynchronous JavaScript and XML*. (XML, or Extended Markup Language, is a data serialization format. It has now largely been replaced by JSON — JavaScript Object Notation — but the name sticks because "Ajax" sounds catchy.)

Ajax is something you probably use everyday. It is an application pattern wherein data is sent to or fetched from the server without the web page being refreshed. (Requests and response are not out of sync with browser refreshes or changed in URL.) This is how many web apps, like Gmail for example, work. The client-side (browser) JS application sends requests to the server and receives all the needed data or content as a JSON object (or XML document) and then uses that data to update the view.

jQuery, the unofficial standard library

Most programming languages have a *standard library*, a set of classes and/or functions that help programmers do common, low-level programming tasks. Typically, these standard libraries come packaged with the language and are documented along with them.

For a number of historical reasons, JavaScript does not have a standard library. This means that in a "pure" JavaScript implementation, many very common tasks are needlessly verbose and complicated.

JQuery is one of several JavaScript libraries which provide a wide range of basic functionality, making it easy to do common programming tasks like traversing the DOM, manipulating HTML objects and their properties, communicating with a server, and interfacing with the browser.

Not everyone is happy with jQuery being considered the standard library, and a number of developers and teams have tried to create an alternative. Some are promising, some have gotten traction in particular communities, but none come anywhere close to jQuery's market share.

- Prototype
- Zepto
- Cash
- Minified

Resources

Learning JavaScript

Tutorials and Online Courses

- JavaScript Novice to Ninja at Sitepoint
- Codecademy
- Treehouse
- Udemy
- Stanford University's CS101

The Stanford online course is probably your best bet if you're really serious about becoming a developer. It covers serious Computer Science and programming techniques, using JavaScript.

There are more and more JavaScript tutorials coming online everyday. Almost all of them cover fairly similar ground work, and none of them really prepare you to be an actual JavaScript developer.

Don't fall into the trap of going from tutorial to tutorial, thinking the next one will be the one that helps you finally "learn JavaScript." Once you understand basic principles and syntax, you should go deeper by delving into material written for developers, such as JavaScript documentation, developer guides for the more popular frameworks, and tutorials on advanced techniques and methodologies.

JavaScript Documentation and Online References

- Mozilla JavaScript Guide
- Mozilla JavaScript Reference
- ECMAScript 2015 Language Specification
- JS the Right Way

Books

If you are just starting, you probably don't want to learn JavaScript from a book. One of the online tutorials above will get you going in the right direction. But once you get the basics down, and want to move into intermediate and advanced programming techniques, you'll want to consult some of these books.

- JavaScript: The Good Parts, by Douglas Crocker, considered by many to be the most important writer and advocate on all things JavaScript.
- Eloquent JavaScript: A Modern Introduction to Programming, by Marijn Haverbeke. This book, which is also available for free online, is one of the most read and respected books on JavaScript development available.
- Learning JavaScript Design Patterns, by Addy Osmani is another excellent book, and is also available for free online.

unstructured conversation about the language and its broader ecosystem. Keeping up with new developments, recent releases, and vutting-edge practices will transform you from a JavaScript coder to a JavaScript expert.

- JavaScript Playground
- David Walsh Blog
- Adventures in JavaScript

Also see this list of 33 JS developers to follow.

JavaScript Tools

There are a *huge* number of JS libraries and toolsets, and more are coming available everyday. We couldn't hope to cover all of them even if we wanted to. And you will never know or use most of them.

But, it's important to realize that being a good JavaScript developer doesn't just mean knowing the language — it means knowing the most popular (or most useful) tools. With that in mind, the lists below only cover the most important JS tools, libraries, frameworks, and extensions.

If you are looking for completeness over relevance, check out this community driven catalog of JavaScript libraries.

Libraries

- General Libraries — Generalized toolsets that accelerate development. Many of these are essentially incompatible with each other, providing competing wyas of accomplishing similar things.
  - jQuery — As mentioned above, this is the most common JS library. Many other systems, toolkits, and frameworks rely on it.
  - Dojo Toolkit
  - mootools
  - Underscore.js
  - Midori

- UI Libraries — Many UI libraries include CSS and are quite opinionated about how HTML is written. These should be considered early in the design process, not as a late add-on.
  - Bootstrap — Twitter's incredibly popular front-end framework makes it relatively easy to build form-based application UIs.
  - jQuery Mobile — A UI framework, somewhat similar in scope (though very different in implementation) to Twitter Bootstrap. JQuery Mobile is both optimized for a small screen, and designed to emulate the look and feel of a mobile application.
  - jQueryUI — A library of UI widgets.
  - Modernizr — Detects the ability of a browser to implement modern (HTML5 and CSS3) UI features, and uses JavaScript to either fill the gap or provide graceful fallbacks.

- Templating
  - Mustache
  - Handlebars

- Graphics and Data Visualization
  - D3
  - ProcessingJS

- Testing
  - Mocha

Frameworks

- Angular — Perhaps the most important JS development framework. Provides a Model-View-Controller architecture, and a fairly opinionated approach to rapid development. Developed by Google.
- Backbone — The main competitor to Backbone. The biggest difference is that Backbone is much less opinionated. Additionally, Backbone was originally abstracted from a Ruby on Rails application, so there are some conveniences in using the two frameworks together.
- socket.io — A real-time application library that includes a client-side library and a server-side node.js component.
- Ember
- React

Advanced Tools

This is a list of relatively advanced tools which drastically extend what JavaScript is capable of, or which dramatically alter how JavaScript is used by a developer.

- Closure — This is set of JS tools created by and for Google developers. The fundamental core of Closure is a compiler that compiles "from JavaScript to better JavaScript."
- Caja — Another tool from Google. Caja compiles third-party JS (and HTML and CSS) for secure embedding on site.
- CoffeeScript — CoffeeScript is a language that compiles to JavaScript, allowing developers to write simpler, less verbose code.
- Dart — A class-based, object-oriented language which compiles to JS.
- Uglify — Compresses, beautifies, auto-indents, removes whitespace, renames variables, and does other code manipulations.
- Narcissus — a "meta-circular JavaScript interpreter." Among other uses, this and similar tools are used to build JavaScript compilers.
- ContextJS — "a context-oriented programming language extension to JavaScript."
- asmjs — a low-level subset of JavaScript. This is a specification to be used as a target language by other JS compilers.

See also this very long list of JavaScript compilation tools. (Did you know you can compile almost every commonly used language to JavaScript? Incredible…)

Conclusion

HTML is the language of web content, but JavaScript is the language of web functionality. It is the most important language for web development, and one of the fastest evolving languages, in terms of practices, tooling, and ecosystem. It's an incredibly exciting language to be using right now.

And it's also a relatively easy language to get into. Everything you need to get started is already on your computer — your browser is your run-time environment, and you can see you initial results right away. There is no shortage of tutorials and online courses to help you learn, and a severe shortage of skilled developers who know the language well. If you are looking for a career in web development, you need to start learning JavaScript.

Adam Wood
Adam is a technical writer who specializes in developer documentation and tutorials.

Tutorials and Resources

Popup Windows Made Easy: Here's The JavaScript Code To Copy And Paste