**What methods can we use?**

We showed how you can use the following methods to find an element or elements in your webpage:

* [document.getElementById(id)](https://developer.mozilla.org/en-US/docs/Web/API/document.getElementById)
* [document.getElementsByClassName(className)](https://developer.mozilla.org/en-US/docs/Web/API/document.getElementsByClassName)
* [document.getElementsByTagName(tagName)](https://developer.mozilla.org/en-US/docs/Web/API/document.getElementsByTagName)
* [document.querySelector(cssSelector)](https://developer.mozilla.org/en-US/docs/Web/API/document.querySelector)
* [document.querySelectorAll(cssSelector)](https://developer.mozilla.org/en-US/docs/Web/API/document.querySelectorAll)

**What do they return?**

There are two methods that return a single [Element](https://developer.mozilla.org/en-US/docs/Web/API/Element) object, getElementById and querySelector:

var salsMotto = document.getElementById("salsMotto");

salsMotto.innerHTML = "Math is cool";

The methods getElementsByClassName and getElementsByTagName return an [HTMLCollection](https://developer.mozilla.org/en-US/docs/Web/API/HTMLCollection) object that acts like an array. That collection is "live", which means the collection is updated if additional elements with tag name or class name are added to the document.

var teamMembers = document.getElementsByClassName("team-member");

for (var i = 0; i < teamMembers.length; i++) {

console.log(teamMembers[i].innerHTML);

}

The method querySelectorAll() returns a [NodeList](https://developer.mozilla.org/en-US/docs/Web/API/NodeList), which also acts like an array. That list is "static", which means that the list won't update even if new matching elements are added to the page. Most likely, you won't run into the difference between the two return data types when you're using these methods, but it's good to keep in mind.

var teamMembers = document.querySelectorAll(".team-member");

for (var i = 0; i < teamMembers.length; i++) {

console.log(teamMembers[i].innerHTML);

}

**Accessing with sub-queries**

Once you've found an element, you can do subqueries on it using the methods we've just shown. For example:

// find the element with that ID

var salsMotto = document.getElementById("salsMotto");

// find the spans inside that element:

var mottoWords = salsMotto.getElementsByTagName("span");

// log out how many there are

console.log(mottoWords.length);

**Traversing the DOM**

Another way to access elements is to "traverse" the DOM tree. Each element has properties that point to elements related to it:

* firstElementChild
* lastElementChild
* nextElementChild/nextElementSibling
* previousElementChild/previousElementSibling
* childNodes
* childElementCount

For example:

var salsMotto = document.getElementById("salsMotto");

for (var i = 0; i < salsMotto.childNodes.length; i++) {

console.log(salsMotto.childNodes[i]);

}

These properties are **not** available on Text nodes, only on Element nodes. To make sure you can traverse an element, you can check its nodeType/nodeValue properties. You likely will not need or want to use DOM traversal, but it is another option available to you.

## Modifying an existing element

We covered various ways that you can modify aspects of an existing element:

## Modifying attributes

You can set an attribute on an element by setting the property of the same name. For example, to change the src of an <img>:

imgEl.src = "http://www.dogs.com/dog.gif";

In addition, you can also use the [setAttribute](https://developer.mozilla.org/en-US/docs/Web/API/Element.setAttribute) method, like so:

imgEl.setAttribute("src", "http://www.dogs.com/dog.gif");

If you want to remove an attribute, you should do that with [removeAttribute](https://developer.mozilla.org/en-US/docs/Web/API/Element.removeAttribute) - like to remove the disabled attribute off a button, effectively enabling it:

imgEl.removeAttribute("disabled");

### Modifying styles

You can change styles just like how you change attributes, by accessing the style property of the element, and setting the corresponding property. For example, to change the color:

headingEl.style.color = "hotpink";

Remember to "camelCase" the multi-word CSS property names, since hyphens are not valid JS property names:

headingEl.style.backgroundColor = "salmon";

### Modifying class names

To add a class to an element, you can set the className property:

mainEl.className = "warning";

That will override any existing classes, so you should do this instead if you just want to add to the list of classes:

mainEl.className += " warning";

In [newer browsers](http://caniuse.com/#search=classList), you can use the [classList](https://developer.mozilla.org/en-US/docs/Web/API/Element.classList) functionality instead:

mainEl.classList.add("warning");

### Modifying inner HTML / text

To completely replace the contents of an element with an arbitrary string of HTML, use [innerHTML](https://developer.mozilla.org/en-US/docs/Web/API/Element.innerHTML):

mainEl.innerHTML = "cats are the <strong>cutest</strong>";

If you don't need to pass in HTML tags, you should use [textContent](https://developer.mozilla.org/en-US/docs/Web/API/Node.textContent) instead:

mainEl.textContent = "cats are the cutest";

Generally, you should be careful when using either of these 2 properties, because they will also remove event listeners (which we teach in the next tutorial).

## Creating elements from scratch

There is a whole set of functions that you can use to create entirely new elements and add them to the page.

To create a new element, use the aptly named [createElement](https://developer.mozilla.org/en-US/docs/Web/API/document.createElement):

var imgEl = document.createElement("img");

To append it to the page, call [appendChild](https://developer.mozilla.org/en-US/docs/Web/API/Node.appendChild) on the target parent element:

document.body.appendChild(imgEl);

Similarly, you can also use [insertBefore](https://developer.mozilla.org/en-US/docs/Web/API/Node.insertBefore), [replaceChild](https://developer.mozilla.org/en-US/docs/Web/API/Node/replaceChild), [removeChild](https://developer.mozilla.org/en-US/docs/Web/API/Node/removeChild), and [insertAdjacentHTML](https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentHTML).

# DOM event types

The browser triggers many events. A full list is available in [MDN](https://developer.mozilla.org/en-US/docs/Web/Events), but here are some of the most common event types and event names:

* **mouse events ([MouseEvent](https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent" \t "_blank))**: mousedown, mouseup, click, dblclick, mousemove, mouseover, mousewheel, mouseout, contextmenu
* **touch events ([TouchEvent](https://developer.mozilla.org/en-US/docs/Web/API/TouchEvent" \t "_blank))**: touchstart, touchmove, touchend, touchcancel
* **keyboard events ([KeyboardEvent](https://developer.mozilla.org/en-US/docs/Web/API/KeyboardEvent" \t "_blank))**: keydown, keypress, keyup
* **form events**: focus, blur, change, submit
* **window events**: scroll, resize, hashchange, load, unload

You might be wondering when to use touch events versus mouse events, since they're so similar.

Touch events are only triggered on touch-enabled devices like smartphones and touch-screen laptops. Mouse events like click and mousemove are triggered on the majority of browsers and devices. However, in most smartphones, the mouseover event isn't triggered at all, because they can't detect a finger hovering over the phone. Some smartphones are adding sensors for that though, so more smartphones will detect mouseover in the future.

In most cases, you'll want to listen to mouse events instead of touch events, because those are the most universal.

# Summary: DOM events

## Adding event listeners

To make sure that the browser calls a particular function when an event happens on an element, you must use [document.addEventListener](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget.addEventListener):

var buttonEl = document.getElementById("clicker");

var onButtonClick = function() {

console.log("Oh golly gosh, you clicked me");

};

buttonEl.addEventListener("click", onButtonClick);

You can pass many valid strings as the first argument, see the event types article.

If you want information about the event that happened, you can look at the event object that the browser passes to your callback function:

var faceEl = document.getElementById("face");

var onFaceClick = function(e) {

console.log("You clicked " + e.clientX + " , " + e.clientY);

};

faceEl.addEventListener("click", onFaceClick);

There are many properties on the event object, you can see [a full list here](https://developer.mozilla.org/en-US/docs/Web/API/Event).

If you are overriding click behavior on a link or submit behavior on a form, you may want to call event.preventDefault() to prevent the browser's default behavior.

## Removing event listeners

If you no longer need a particular event listener, you can remove it using [removeEventListener](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/removeEventListener):

var faceEl = document.getElementById("face");

var onFaceClick = function(e) {

console.log("You clicked " + e.clientX + " , " + e.clientY);

};

faceEl.addEventListener("click", onFaceClick);

// later...

faceEl.removeEventListener("click", onFaceClick);

# What DOM animation technique should you use?

You've now seen three techniques that let you animate parts of your webpage: [window.setInterval](https://developer.mozilla.org/en-US/docs/Web/API/WindowTimers.setInterval)/[setTimeout](https://developer.mozilla.org/en-US/docs/Web/API/WindowTimers.setTimeout" \t "_blank), [window.requestAnimationFrame](https://developer.mozilla.org/en-US/docs/Web/API/window.requestAnimationFrame), and CSS animations/transitions.

When you're considering what technique to use, you should consider these questions:

* Can it actually do what I want to do?
* How well does this technique perform? (Does it slow down the browser/computer?)
* How accurate does my timing need to be?
* Does it work in all the browsers that I want my webpage to work in?

When making webpages, we're very concerned about performance, so we like to consider that as the most important criteria. However, browser performance does change when new browsers come out, and mobile browsers can perform very differently from desktop browsers, so the technique that performs best today may not be the technique that performs best tomorrow. Currently, CSS animations/transitions are the most performant, then requestAnimationFrame, then setInterval.

However, CSS can't do everything. For example, to draw pixels and shapes in a <canvas> tag, you have to call functions like fillRect(), you can't use CSS. You would need to use requestAnimationFrame or setInterval to call those functions periodicially instead. In fact, that's what we do here on Khan Academy, in our ProcessingJS environment. ProcessingJS is a JS library that writes out to a <canvas> tag, and if you define a draw() function in your code, then ProcessingJS uses setInterval to call that draw() function repeatedly based on the frameRate.

Sometimes, you want to call JavaScript functions periodically, but not because you want to animate something on the page. You might be polling a server for updates, like Twitter does when it updates its realtime feed. In that case, you can just use setInterval, and it doesn't matter that the timing isn't accurate, because you're only calling it every minute or so. That's what we do on Khan Academy on the help requests page, to continuously check for new help requests every 2 minutes.

Of course, you should keep browser compatibility in mind. If you're writing code that needs to work in IE9, then you can't use requestAnimationFrame or CSS animations. You need to use a combination of techniques that work across the browsers, or find a library that does that for you, like [Velocity.js(Opens in a new window)(Opens in a new window)](http://julian.com/research/velocity/).

There is a lot more to learn about all of these techniques. Follow these links to learn more:

* [CreativeJS: requestAnimationFrame](http://creativejs.com/resources/requestanimationframe/)
* [CSS Tricks: Using requestAnimationFrame](http://css-tricks.com/using-requestanimationframe/)
* [MDN: Guide to using CSS animations](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Using_CSS_animations/)
* [TutsPlus: Beginners intro to CSS animation](http://webdesign.tutsplus.com/tutorials/a-beginners-introduction-to-css-animation--cms-21068/)
* [The art of web: CSS animation](http://www.the-art-of-web.com/css/css-animation/)
* [CSS Animation: An interactive guide](https://itunes.apple.com/us/book/css-animation-interactive/id799975886?mt=11)

Bonus for Doctor Who fans: one of my favorite examples of the power of CSS3 animations is this [animated TARDIS(Opens in a new window)(Opens in a new window)](http://joshnetherton.com/presentations/ConvergeSE2013/slides.html#slide63).

# What's a JS library?

There are a billion webpages, and many of them are interactive in some way - in fact, many of them are interactive in the same ways. How many websites have you seen that use a slideshow? I've seen hundreds, myself!

For example, here's one from [National Geographic](http://news.yahoo.com/photos/125-years-of-national-geographic-stunning-photography-since-1888-1380118227-slideshow/jou-jou-captive-chimpanzee-reaches-out-it%C3%A2s-hand-to-dr-jane-goodall-in-brazzaville-zoo-brazzaville-photo-1380113767228.html):



Screenshot of a slideshow from National Geographic

Here's one from [Instagram](https://www.instagram.com/p/B508V6TH6_b/):



Photo of a slideshow of waterfall photos on Instagram

One more, from [The Onion](http://www.theonion.com/slideshow/6-dogs-who-know-how-to-have-fun-35356#6):



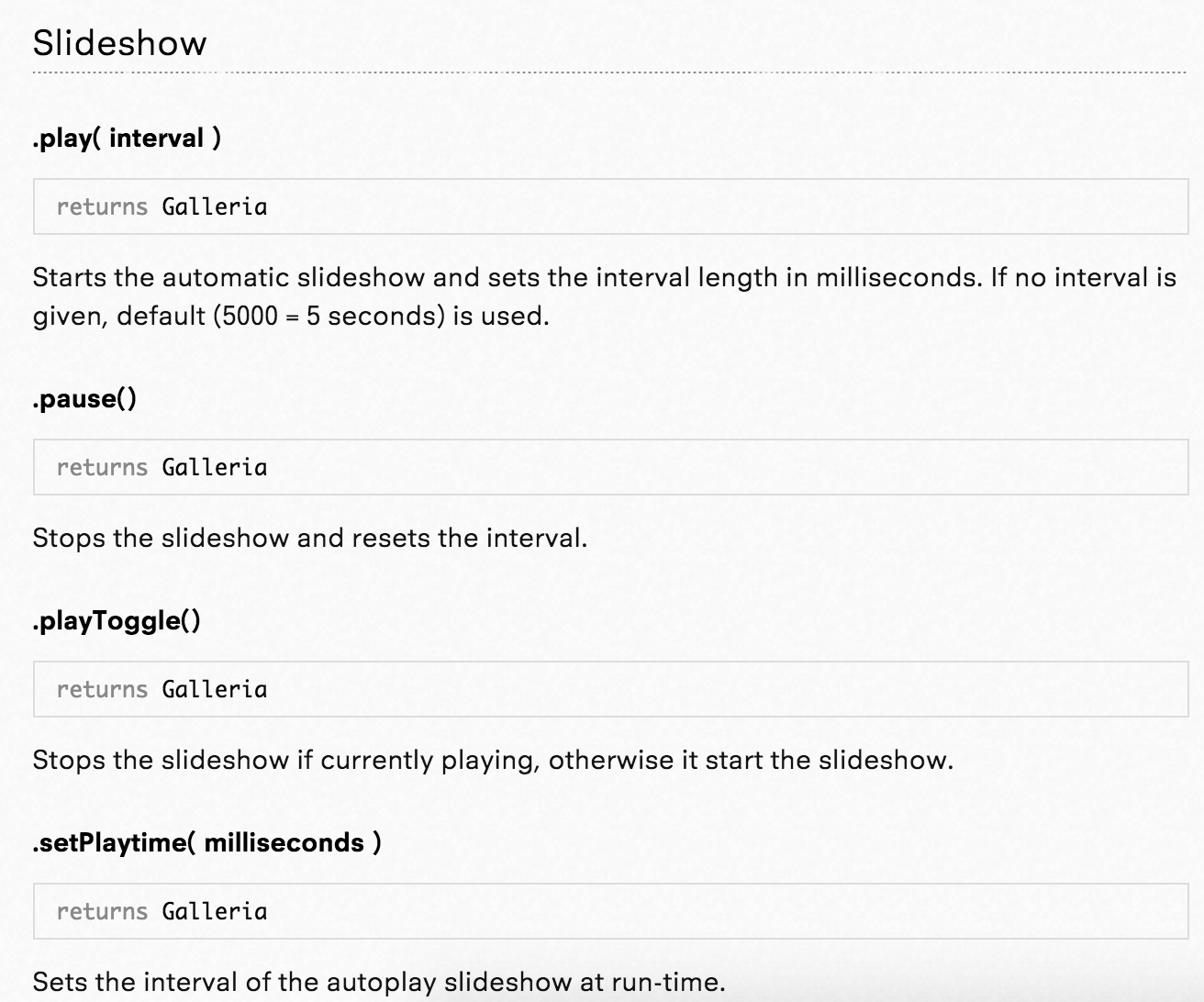
Screenshot of a slideshow from The Onion

Now, it's possible that every one of those web developers wrote their own JavaScript code to make a slideshow, but a lot of them probably re-used the same code. As programmers, we should re-use existing code when we can, so that we don't waste our time writing code that another programmer has already written.

In JavaScript, the way we do that is by using a **library**. A library is a JavaScript file that contains a bunch of functions, and those functions accomplish some useful task for your webpage.

How do we know what functions we can use? We could look at the JavaScript file, if it's short, or better, we could look at the **documentation**. Most libraries have documentation with a list of available functions or a real-world example.

For example, here's a snippet of the [documentation for Galleria](https://galleria.io/docs/api/methods.html), a popular JS library for making slideshows:



Screenshot of the documentation for Galleria

When a programmer creates a JS library and puts it out into the world, they're purposefully deciding to share it with the world - so that often means they put in the effort to come up with great documentation and examples. Perhaps one day you will decide to create a library of some functionality that you find really useful and share it with the world.

# Where are JS libraries hosted?

When we included the slideshow JS library in our webpage in the previous talk-through, we typed this URL:

<https://cdn.jsdelivr.net/gh/pamelafox/ka-slideshow-example@master/slideshow.js>

That URL is an **absolute** URL, which means that it includes the protocol and domain. When we include JS libraries in the Khan Academy environment, we must use an absolute URL so that the environment knows the full path to the library.

If you were working on a webpage on your own computer and you had all the files downloaded, then you could use a **relative** URL instead, like simply "slideshow.js" or "lib/slideshow.js" if it was inside a folder named "lib". When you're working locally, open your browser's network panel to make sure it found all the local resources like your JS files and CSS files. If the browser couldn't find a resource, you'll see a 404 and you can debug the URL and file location.

Now, back to this URL:

<https://cdn.jsdelivr.net/gh/pamelafox/ka-slideshow-example@master/slideshow.js>

The protocol "https" means that it’s a **secure** URL. We only allow you to bring in secure resources in Khan Academy webpages, and that's a best practice in web development. Not all servers have HTTPS enabled yet, however, so you may sometimes have to start URLs with "http" in your projects.

The server "cdn.jsdelivr.net" is an example of a **content delivery network (CDN)**. CDNs are optimized for serving static files like JS libraries and serving them very quickly. There are a few big CDNs that host multiple JS libraries (like [Google's CDN](https://developers.google.com/speed/libraries/) and [cdnjs](https://cdnjs.com/)), and there are some libraries that have their own dedicated CDN.

When you bring a JS library into your webpage from a CDN—or any server that's not your own—you should not trust that server. A malicious server could replace the JS library with code that stole your user's data and sent it somewhere, and you certainly don't want that to happen!

On Khan Academy, we use a security mechanism called [CSP(Opens in a new window)(Opens in a new window)](http://www.html5rocks.com/en/tutorials/security/content-security-policy/) in our webpage environment to make sure that you can only bring in resources from servers we trust: [bootstrapcdn.com](http://getbootstrap.com/), [googleapis.com](https://developers.google.com/speed/libraries/), [jsdelivr.net](http://www.jsdelivr.com/), and [cdnjs.com](https://cdnjs.com/).

On your own websites, you'll need to make the decision whether to serve the JS library from your own server or include it from an external CDN. Most big websites decide to host libraries on their own server, because they can have more control and make optimizations like combine multiple libraries into a single JS file for better loading performance. It's up to you—just keep in mind everything we talked about here!

# The world of JS libraries

There are thousands of JavaScript libraries that you could bring into your webpage, and many aspects of your webpage that they can help you with:

* DOM manipulation
* DOM Events
* AJAX / Data retrieval
* Effects & animation
* HTML templating
* Page layout
* UI widgets
* Graphics & charts
* Data modeling
* Routing & navigation
* Accessibility
* Multi-browser support
* Mobile support

Here's another way I like to think of it - a webpage is made up of the User Interface (HTML & CSS), Interactivity (JS + DOM), and Data (which we often bring in through JS). You can make your data-driven interactive UI without a library, or you can bring in different libraries to help out with different parts. Some libraries will do a lot, some will do just one part:

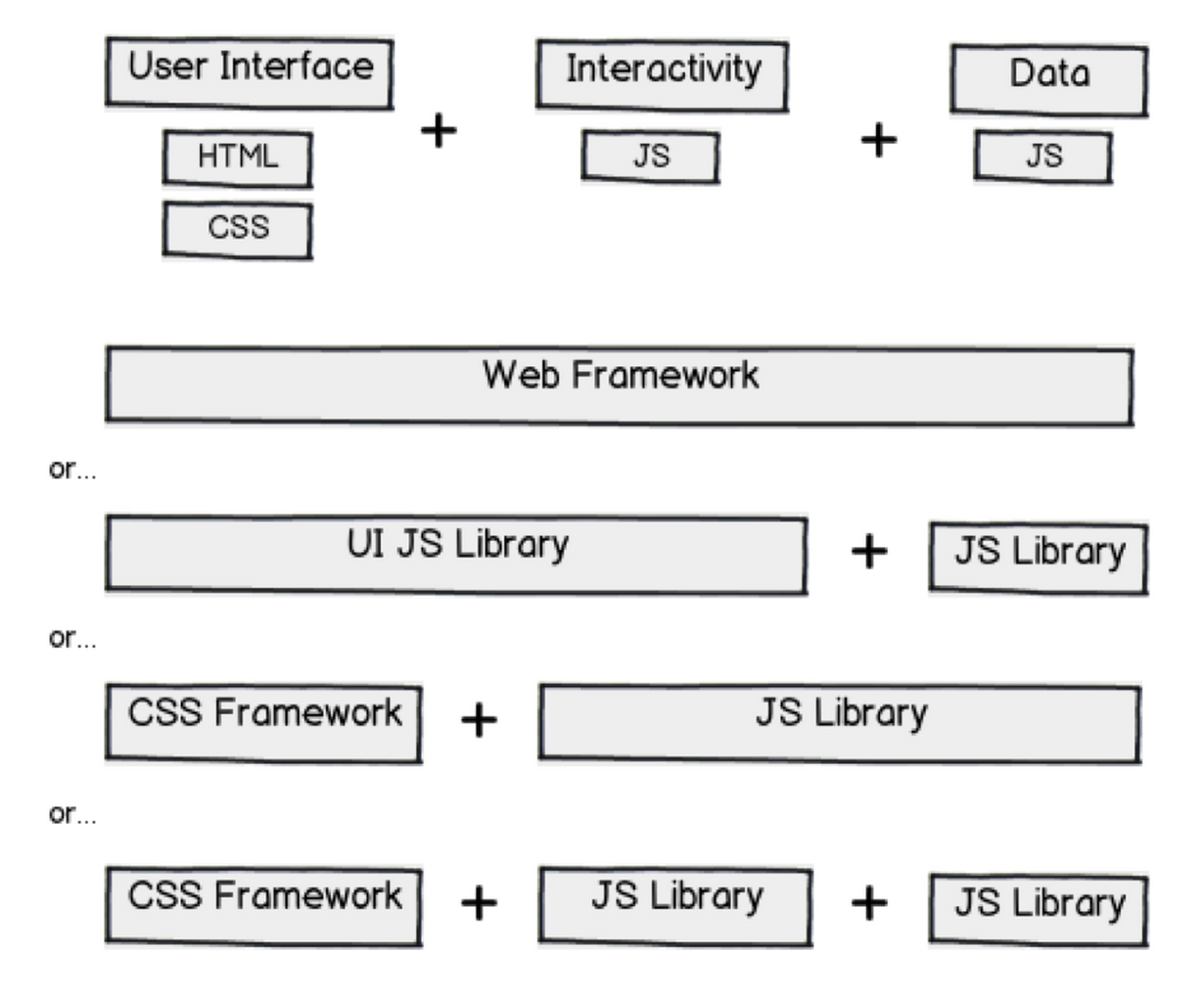


Diagram showing different ways of breaking a webapp into UI, interactivity, and data.

One of the hardest parts of web development is deciding what libraries to use, because you'll often be writing code that builds on top of the libraries, and it's hard to change your decision later. There's no right answer - know your options, then make an educated guess.

## Web frameworks

There are some libraries that "do it all" - everything from data retrieval to DOM manipulation to fancy UI widgets. If you use one of these frameworks, you'll typically be bringing a lot of JavaScript into your page, so you'll want to make sure you really need most of what the framework is offering you. For some developers, these frameworks do too much, and they find it better to use smaller libraries with more focused tasks.

* [jQuery UI](https://jqueryui.com/)
* [jQueryMobile](https://jquerymobile.com/)
* [Dojo](https://dojotoolkit.org/)
* [Sencha](https://www.sencha.com/)
* [QooxDoo](http://qooxdoo.org/)
* [Closure](https://developers.google.com/closure/)
* [KendoUI](http://www.telerik.com/kendo-ui)

## CSS frameworks

A CSS library is a collection of CSS style rules, and documentation that shows you what HTML tags and classes to use. There are many popular CSS libraries that will make it easier for you to make a beautifully styled website. They sometimes include optional JavaScript widgets, but developers often choose to use other JS libraries with a CSS library.

* [Twitter Bootstrap](http://getbootstrap.com/) (See [example on KA](https://www.khanacademy.org/computer-programming/css-library-example-bootstrap/4697699761913856))
* [ZURB Foundation](http://foundation.zurb.com/) (See [example on KA](https://www.khanacademy.org/computer-programming/example-css-library-zurb/6648586516430848))
* [Pure CSS](http://purecss.io/)
* [Topcoat](http://topcoat.io/)
* [Skeleton](http://getskeleton.com/)

## JS browser libraries

These are libraries that will make it easier for you to use browser functionality - like the DOM API, the window methods - the topics we discussed in this course. They often make it possible to accomplish the same thing in less code, and they'll take care of cross-browser bugs.

* [jQuery](https://jquery.com/) (The most popular library, by far! - See [example on KA](https://www.khanacademy.org/computer-programming/crocodile-order-form/5090752502824960))
* [ZeptoJS](http://zeptojs.com/)
* [MooTools](http://mootools.net/)
* [dollardom](https://github.com/julienw/dollardom)
* [QuoJS](https://github.com/soyjavi/QuoJS)

## JS app architecture libraries

These libraries help you with your web apps as they grow larger. Your code might be manageable now, at a few hundred lines of code, but once you get to thousands of lines of code, your code might become "spaghetti code" - an unmanageable mess. These libraries help you separate your "model" (data) from your "view" (presentation), and often use object-oriented programming principles to organize your code.

* [Facebook React](https://facebook.github.io/react/) + [Facebook Flux](https://facebook.github.io/flux/)
* [Backbone](http://backbonejs.org/)
* [AngularJS](https://angularjs.org/)
* [EmberJS](http://emberjs.com/)

## Mini libraries & microlibraries

You can also use small libraries that developers have written for a specific task - when they're really small, they're sometimes called "microlibraries". Here's a smattering:

* DOM: [tinyDOM](https://github.com/ctult/TinyDOM)
* Events: [Events.js](https://github.com/kbjr/Events.js)
* AJAX: [MicroAjax](http://code.google.com/p/microajax/)
* Animation: [Morpheus](https://github.com/ded/morpheus), [Viper](https://github.com/alpha123/Viper)
* Data Modeling: [Spine](https://github.com/spine), [Knockout](http://knockoutjs.com/)
* Templates: [Handlebars](http://handlebarsjs.com/), [Mustache](https://github.com/janl/mustache.js)
* Routing: [PathJS](https://github.com/mtrpcic/pathjs)
* Object-Orientation: [Classy](http://www.pocoo.org/projects/classy/), [Klass](https://github.com/ded/klass)
* Graphics: [Processing.JS](http://processingjs.org/) (See [example on KA](https://www.khanacademy.org/computer-programming/processingjs-inside-webpages-template/5157014494511104)), [RaphaelJS](http://code.tutsplus.com/tutorials/an-introduction-to-the-raphael-js-library--net-7186" \t "_blank) (See [example on KA](https://www.khanacademy.org/computer-programming/example-js-library-raphaeljs/6507593544630272)), [D3](http://d3js.org/), [Three.JS](http://threejs.org/) (See [example on KA](https://www.khanacademy.org/computer-programming/example-js-library-threejs/6158072058871808)), [Paper.js](http://paperjs.org/)
* Charts: [gRaphael](http://www.javascriptoo.com/graphael), [High Charts](http://www.highcharts.com/)
* Date/time: [moment.js](http://momentjs.com/)

You can find more at microjs.com or just by searching the internet for "javascript library [functionality]" for whatever functionality you're looking for.

Have you tried out a JS library in your webpages here on Khan Academy? Share it in the comments below.

# Which JS library should you use?

There are a huge number of libraries out there, and for any given bit of functionality, there are likely multiple libraries that accomplish that bit of functionality. For example, there are so many date-picker libraries out there, that there are articles like ["Top 15 jQuery DatePickers"](http://www.smashingapps.com/2012/01/19/15-jquery-calendar-date-picker-plugins.html) to try to help developers pick from them.

But too many choices can turn into decision paralysis for us web developers. How do we know which one is best? What if we make the wrong choice?

There's often not a single "best choice" in web development. But there are often better choices than others, and thinking through the considerations below can help you make the better choice.

Since a JS library is often used when developing a user-facing product, these considerations should satisfy two audiences: the developers that must code and maintain the code that uses the library (like you!), and the users that will interact with it.

### Will it be a good developer experience?

* **Well documented**: It should be easy to find a reference of function signatures, demos of actual usage, and a more narrative how-to-use guide. If a library has no documentation, it's usually a sign that they are not the most developer-friendly.
* **Flexible**: The demos in the documentation might look great - but might want to use a library in a slightly or completely different way than what the demos show. Look for signs of flexibility - Is it easy to send in configuration options? Is there a documented plugin architecture? Does it trigger many events that you could hook your code into?
* **Actively maintained**: Browsers change frequently. Libraries that once worked can suddenly stop working, because they relied on some quirk of the browser that changed. This is specially true of [HTML5 shims and polyfills](https://github.com/Modernizr/Modernizr/wiki/HTML5-Cross-Browser-Polyfills), because browsers are frequently releasing new versions with evolving implementations of the HTML5 elements. You can figure out how recently the library was updated by checking the date in their [changelog](https://en.wikipedia.org/wiki/Changelog). If there's no changelog and the library is hosted in an open source repository like Github, you can check the date of the last commit.
* **Future thinking**: If you're looking for an HTML5 "shim", prefer a "polyfill" - a shim that mimics the API. That way, theoretically, when all your users were using browsers that supported the technology, you'd be able to stop using the library entirely, with no change to your code at all. For example, if you're using a library to use video in your webpage, use a polyfill that will let you use the HTML5 video tag, and it will replace it with a fallback technology like Flash in older browsers.
* **Tested**: All good libraries should include tests that make sure their functionality works as expected. When a library is tested, then we can have confidence there will be some degree of backwards compatibility in new versions of the library.
* **Clean code**: We could treat open-source libraries as black boxes, and refuse to look inside of them, but sometimes, you may need to dig inside of the library code to debug an issue or add a new bit of functionality. Take a quick look at the code and see how easy it is to read, and if it has any red flags, like big chunks of commented-out lines of code.
* **Responsive community**: You will have questions. You will encounter bugs. Ideally, you'll be able to figure them out with developers, whether that's the maintainers or users.

If the library is hosted on a version control site like Github, you can look at:

* **Number of forks**: Lots of forks (or stars) means there are at least a lot of developers that cared enough to fork the library. That doesn't mean they'll help you, but it's a start! Large libraries often have thousands of forks, more niche libraries have 100s or 10s of forks.
* **Number of issues**: Are there many open issues? That might be a sign that there's not a community effort around responding and closing issues. It can also mean it's just a very popular project with a lot of ideas for improvement, so continue on to the next point.
* **Vibe on issues**: Read through a few issues and pull requests. Are the maintainers receptive to feedback? Do they answer usage questions? Do you get a positive or negative vibe from the conversations on them?
* **External community**: Are questions about the library answered on StackOverflow? Are there libraries that build on top of the library? Many smaller libraries won't be big enough to have a visible external community, but bigger ones like Modernizr or Backbone have significant ones, and that's a big motivation for using them. You can do a search on the internet for the library name to see what kind of results you find.

### Will it be a good user experience?

If the JS library does not create a UI component, then only the first few of these matter.

* **File size**: How much will it contribute to how much JS your users have to download? For context, jQuery gzipped and minified is 18k and Select2 is 7K.
* **Performance**: Besides size, other aspects of a JS library can affect its performance, like if it does heavy DOM manipulation, graphics rendering, computation, synchronous storage calls, etc. Look for promises of great performance on the documentation, and of course, try it out yourself.
* **Browser support**: Check that it supports all your desired browsers. Many libraries these days purposefully don't support older browsers (which your webpage may need to support), because they're designed to be lightweight and only for mobile browsers.
* **Accessibility**: Many libraries for UI components look great, but they are not accessible (they do not work well for users with visual disabilities). For a quick check, you can run [WAVE](http://wave.webaim.org/) on the library's demos page.
* **Responsive**: If your users will ever use the UI component from a library on a mobile browser, then it should work well for them there. Are the buttons big enough? Does it use touch events? Does it scale to small screen sizes?

If you've considered all that criteria, and still can't decide between a handful of libraries, then you might try the call-a-friend approach: ask colleagues or developer friends what library they use. You might just find a crowd favorite.

Remember: there isn't one right answer, there isn't one best choice. Also, you don't have to comprehensively review every JS library you are thinking about using, especially if you're working on projects for your own. You can just pick a library and see what you like about it while you use it. You'll start to build a list in your head of your favorite libraries to use, and your own criteria for libraries, and that will help you in your future decisions.

# What to learn next

Congratulations on learning how to manipulate your webpages with JavaScript!

## Using new HTML features

HTML, CSS, and JavaScript are all evolving technologies. They include many features that we didn't cover in this course, and they continue to change to meet the needs of web developers.

Here are some examples of new features to explore:

* [Video and audio](https://developer.mozilla.org/en-US/docs/Learn/HTML/Multimedia_and_embedding/Video_and_audio_content)
* [Web storage API](https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API)
* [WebGL](https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API)

Browse more [web technologies](https://developer.mozilla.org/en-US/docs/Web/API) or search the web for the feature you're interested in—it might actually exist!

## Bringing in data to your page

Many webpages let the users interact with dynamic data, like from a database or service. If you have your webpage hosted on your own server, you can bring it in using [AJAX](https://developer.mozilla.org/en-US/docs/Web/Guide/AJAX). If you want to bring the data from an external server and that server makes that data available to others, you can bring it in using [their public API](https://www.programmableweb.com/category/all/apis).

Note that our webpages environment only allows you to bring in data from [certain servers](https://khanacademy.zendesk.com/hc/articles/205175520-What-resources-images-scripts-etc-can-I-bring-into-webpages-that-I-make-on-Khan-Academy-), and those servers must be called via SSL (https://). Check your developer console to see if you're trying to bring in data from a disallowed server, and use a development environment other than Khan Academy if needed.

## Keeping up to date

Browsers release improvements often and the web development community is an active one. One of our favorite ways to keep up-to-date with what's new is to subscribe to [Frontend Focus](http://html5weekly.com/) and [JavaScript Weekly](http://javascriptweekly.com/), newsletters with links to articles and demos. Of course, whenever you learn something new, you should try it out to really understand how it works.

Practice, practice, practice