Introduction to the DOM

[(https://github.com/learn-co-curriculum/phase-0-the-dom-introduction)](https://github.com/learn-co-curriculum/phase-0-the-dom-introduction) [(https://github.com/learn-co-curriculum/phase-0-the-dom-introduction/issues/new)](https://github.com/learn-co-curriculum/phase-0-the-dom-introduction/issues/new)

Learning Goals

- Identify the Document Object Model (DOM)
- Explain how the DOM is created
- Identify the DOM as accessed by JavaScript objects
- Introduce the Console and Chrome DevTools
- Learn how to open HTML files in the browser

Introduction

We have learned how to write HTML and style it with CSS. We have also built our JavaScript programming skills. With this knowledge, we're now ready to learn **Document Object Model (DOM) programming**.

DOM programming consists of using JavaScript to:

1. Ask the DOM to find or select an HTML element or elements in the rendered page
2. Remove and/or insert one or more elements
3. Adjust a property of selected element(s)

In other words, we can use DOM programming to create and modify content that users see in their browsers and add interactivity to our web pages.

Identify the Document Object Model

Let's start with a biology metaphor. Your DNA represents a code-based version of *you*. The DOM represents a code-based version of *a web page*. If something edits your DNA, changes will be made in your body (perhaps giving you mutant powers). Similarly, when you change something in the DOM, you change what's displayed in the browser.

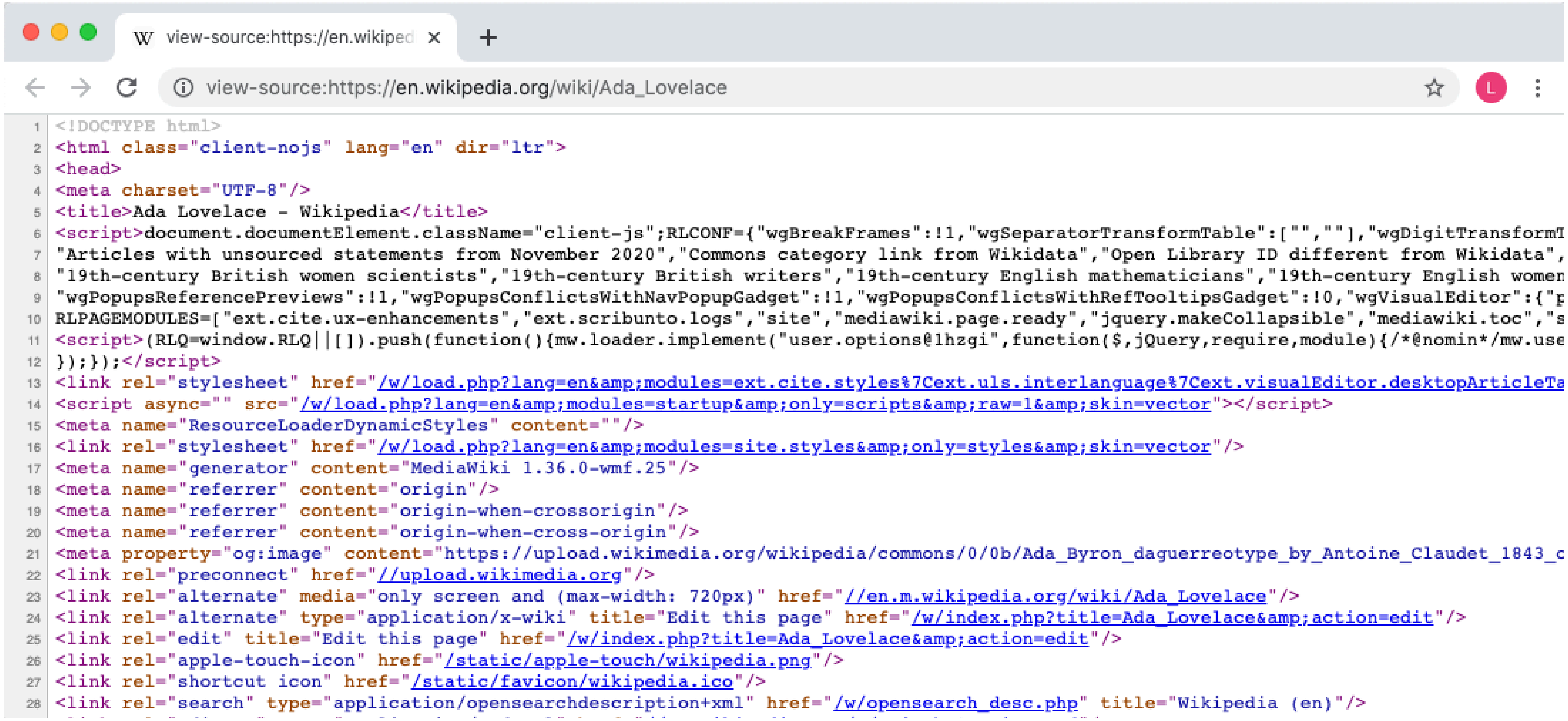
But what exactly ***is*** the DOM? You can think of it as a "middle layer" between the user and the underlying HTML, CSS, and JavaScript that makes up the page. What the user is actually seeing on the page is the DOM. When the page initially loads, the DOM represents the underlying HTML, CSS and JavaScript. When we use JavaScript and DOM programming to modify the DOM and change what the user sees, the underlying code is not modified: if we refresh the page, it goes back to its original state.

Explain How the DOM Is Created

The DOM is created when the page loads from the HTML/CSS/JavaScript that the web server provides to the browser. Let's examine this process step-by-step:

> **NOTE**: To ensure that instructions and screenshots match up with your experience, be sure to use the **Google Chrome (https://www.google.com/chrome/browser/desktop/index.html)** browser.

1. In Google Chrome, open a tab and navigate to the **Wikipedia page for Ada Lovelace (https://en.wikipedia.org/wiki/Ada_Lovelace)**.
2. To see the HTML of this page, add `view-source:` to the front of the URL in the URL bar. Using the `view-source` URL prefix will display all the page's source HTML. It will look something like this:


view-source:https://en.wikiped
view-source:https://en.wikipedia.org/wiki/Ada_Lovelace

```
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Ada Lovelace - Wikipedia</title>
<script>document.documentElement.className="client-js";RLCONF={"wgBreakFrames":!1,"wgSeparatorTransformTable":["",""],"wgDigitTransformT
"Articles with unsourced statements from November 2020","Commons category link from Wikidata","Open Library ID different from Wikidata",
"19th-century British women scientists","19th-century British writers","19th-century English mathematicians","19th-century English women
"wgPopupsReferencePreviews":!1,"wgPopupsConflictsWithNavPopupGadget":!1,"wgPopupsConflictsWithRefTooltipsGadget":!0,"wgVisualEditor":{"p
RLPAGEMODULES=["ext.cite.ux-enhancements","ext.scribunto.logs","site","mediawiki.page.ready","jquery.makeCollapsible","mediawiki.toc","s
<script>(RLQ=window.RLQ||[]).push(function(){mw.loader.implement("user.options@1hzgi",function($,jQuery,require,module){/*@nomin*/mw.use
});});</script>
<link rel="stylesheet" href="/w/load.php?lang=en&amp;modules=ext.cite.styles%7Cext.uls.interlanguage%7Cext.visualEditor.desktopArticleTa
<script async="" src="/w/load.php?lang=en&amp;modules=startup&amp;only=scripts&amp;raw=1&amp;skin=vector"></script>
<meta name="ResourceLoaderDynamicStyles" content=""/>
<link rel="stylesheet" href="/w/load.php?lang=en&amp;modules=site.styles&amp;only=styles&amp;skin=vector"/>
<meta name="generator" content="MediaWiki 1.36.0-wmf.25"/>
<meta name="referrer" content="origin"/>
<meta name="referrer" content="origin-when-crossorigin"/>
<meta name="referrer" content="origin-when-cross-origin"/>
<meta property="og:image" content="https://upload.wikimedia.org/wikipedia/commons/0/0b/Ada_Byron_daguerreotype_by_Antoine_Claudet_1843_c
<link rel="preconnect" href="//upload.wikimedia.org"/>
<link rel="alternate" media="only screen and (max-width: 720px)" href="//en.m.wikipedia.org/wiki/Ada_Lovelace"/>
<link rel="alternate" type="application/x-wiki" title="Edit this page" href="/w/index.php?title=Ada_Lovelace&amp;action=edit"/>
<link rel="edit" title="Edit this page" href="/w/index.php?title=Ada_Lovelace&amp;action=edit"/>
<link rel="apple-touch-icon" href="/static/apple-touch/wikipedia.png"/>
<link rel="shortcut icon" href="/static/favicon/wikipedia.ico"/>
<link rel="search" type="application/opensearchdescription+xml" href="/w/opensearch_desc.php" title="Wikipedia (en)"/>
```

3. The browser reads this HTML, along with CSS and JavaScript defined in `<script>` or `<link>` tags, to create the DOM inside the browser. At this point, nothing is displayed on the screen. This time when nothing is displayed is very brief so our human eyes never really catch it.
4. The browser then uses the DOM object to create the rendered page. While we often learn that browsers "display HTML," that's not exactly accurate. Browsers use the HTML to create a "middleman" that they, in turn, use to display the structured and styled content.

Identify the DOM as Accessed by JavaScript Objects

We can access the DOM, using JavaScript and DOM programming, through two *variables*: `window` and `document` .

The `window` variable points to an *object* that represents Chrome's information about the browser, well, "window." It has many functions, but the main one is "it's a place where everything is." Not to be Zen here, but a browser without a `window` is like the universe before the Big Bang;

there's just... *nothing*.

Like all objects, the `window` has properties and methods. For example, we can access operating system browser information like:

```
window.innerHeight;
// returns the inner height of the browser window.
```

For the most part, we won't interact with `window` : we don't want to mess with the container of everything or with operating system stuff. We want, rather, to change content. To do that, we'll focus on an object called `document` .

As an *object*, `document` has *properties*:

```
document.URL;
//=> https://en.wikipedia.org/wiki/Ada_Lovelace
```

As an *object*, `document` also has *methods*:

```
document.querySelector("h1");
//=> Returns the element on the page with an id attribute equal to "firstHeading"
```

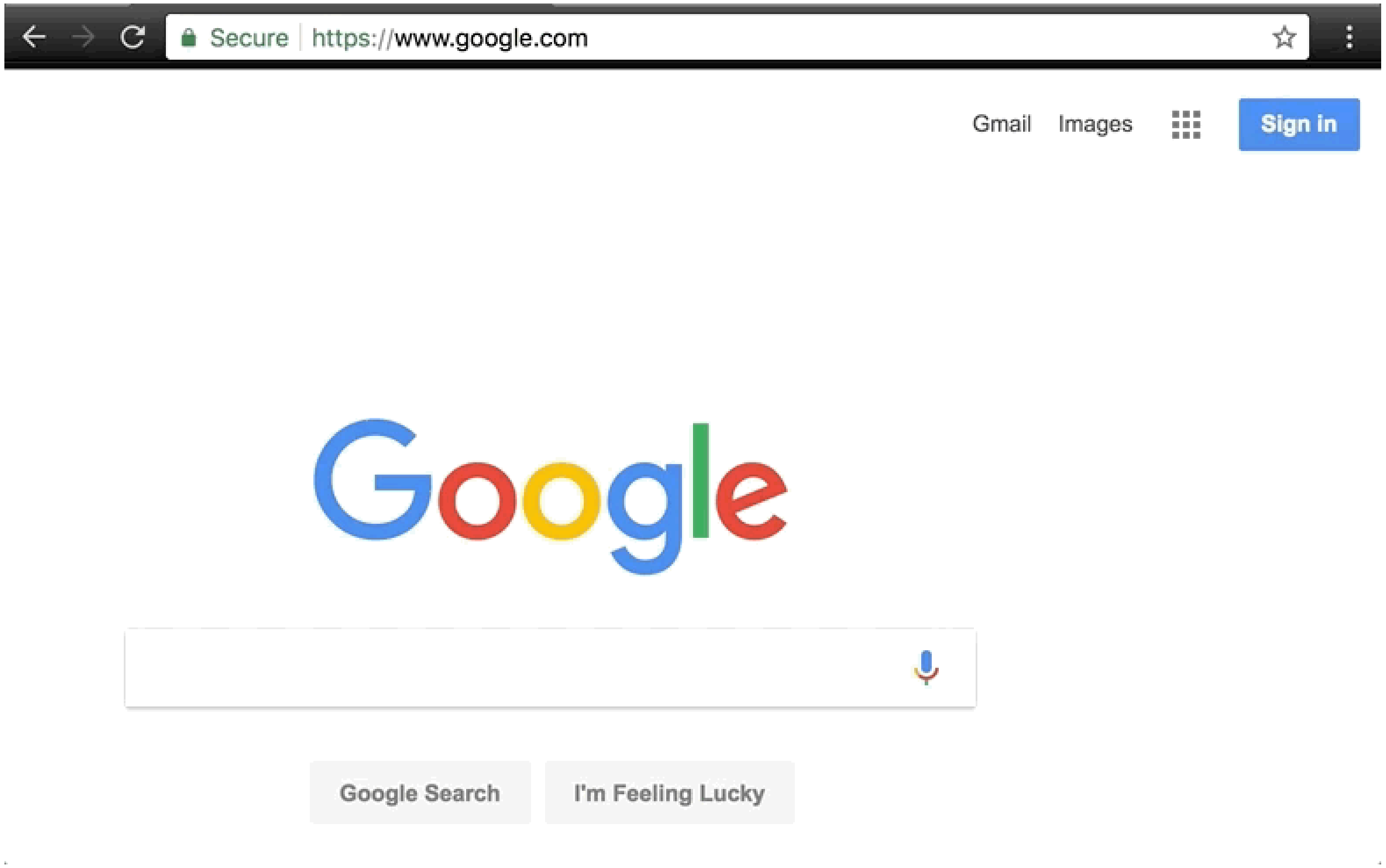
The *methods* and *properties* that the DOM provides via its objects is called the DOM's "Application Programming Interface," or "API." It's just a programming word that you're likely to see online. But it just means "the things that these objects know how to do."

The Browser Console and Chrome Developer Tools

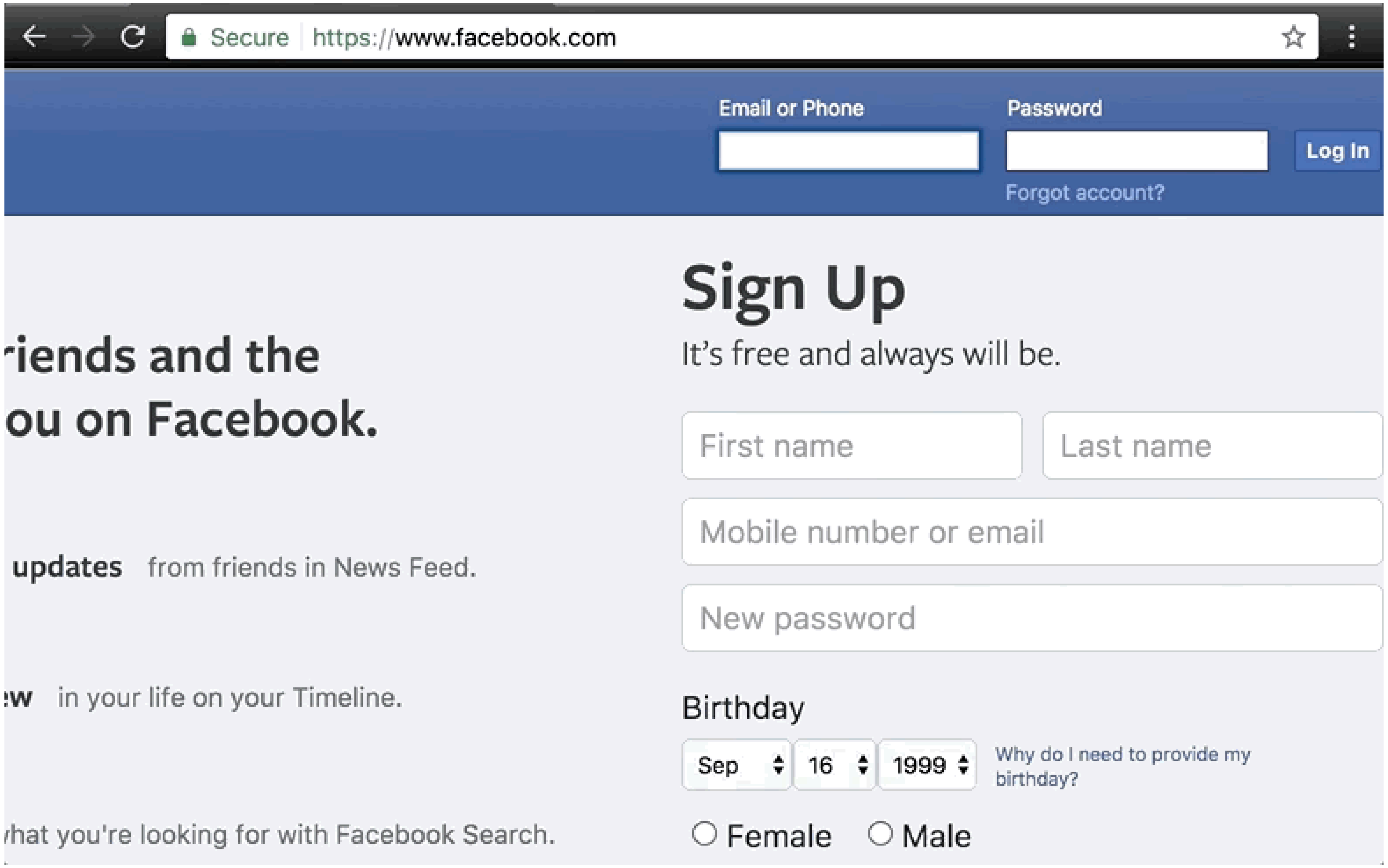
Every major browser comes with a built-in set of developer tools that you can use to inspect, modify, and debug the content of a web page. To **open the dev tools in Chrome (https://developers.google.com/web/tools/chrome-devtools/console/#open_as_panel)**, press `Ctrl+Shift+J` (Windows / Linux) or `Cmd+Opt+J` (Mac). Chrome ships with a whole suite of useful developer tools, but the only one we care about for now is the JavaScript console.

The console is an environment in the browser where we can type and run JavaScript code in the context of the current browser window. The console is *sandboxed*, meaning the only resources it has access to are those loaded on the current page. Once we start declaring variables and functions in separate JavaScript files, we'll be able to access and play around with them in the console. The console is the single best tool for debugging JavaScript in the browser, so start familiarizing yourself with it now.

The `Ctrl+Shift+J` / `Cmd+Opt+J` command should open up straight into the console. If for whatever reason, it doesn't, you can always click on `Console` in the dropdown (when the DevTools are collapsed) or in the list of tabs:



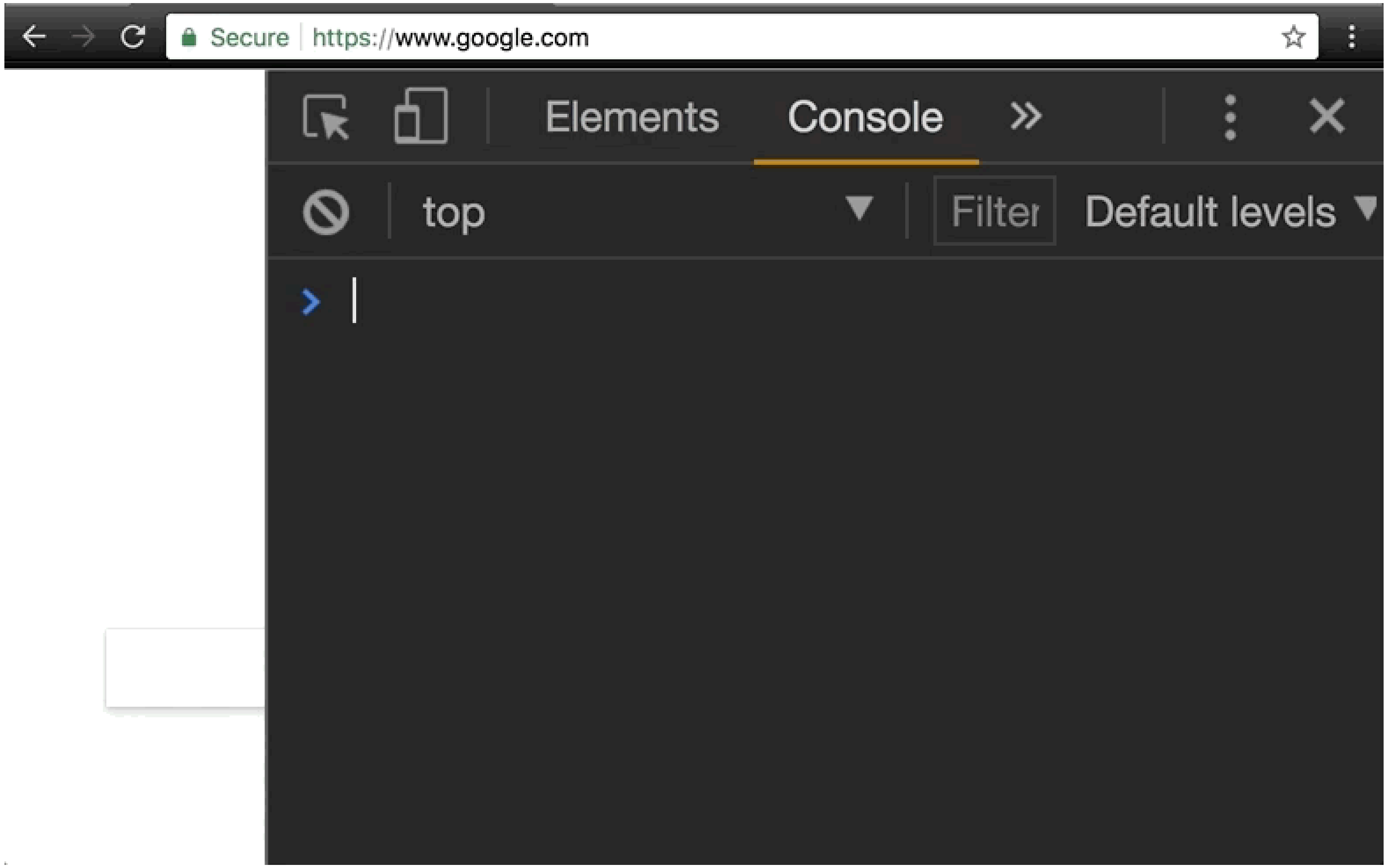
If at any point the console becomes cluttered with errors, warnings, or anything else, click the `Clear console` button:


Secure https://www.facebook.com
Email or Phone
Password
Log In
Forgot account?
Sign Up
It's free and always will be.
riends and the
ou on Facebook.
updates from friends in News Feed.
ew in your life on your Timeline.
hat you're looking for with Facebook Search.
First name
Last name
Mobile number or email
New password
Birthday
Sep
16
1999
Why do I need to provide my birthday?
Female
Male

Okay, okay, enough background and setup. Let's write some code!

Coding in the Console

You can write and test out JavaScript code in the console. We'll start off with some simple math. In the console, type `1 + 1` and press enter. You should see the number `2` appear.



Try out some other mathematical expressions and see what they return.

Next up, let's write some text. To make sure the JavaScript engine knows that we're trying to write some literal text, we need to wrap it in quotation marks, like so:

```
"This is some literal text in JavaScript!";
```

Go ahead and type that classic phrase, `"Hello, world!"` , into the console and press enter. It returned `"Hello, world!"` right back to us. Try typing some more literal text into the console, such as your name. Don't forget the quotation marks!

We can go far beyond simple literal expressions: we can create variables, loops, or if statements in the console. We can even define and run functions!

> **Note:** It's impossible to overstate how important practice is when you're learning a new programming language. As you continue moving through the JavaScript curriculum, you should almost always have a browser console open. Code along with every example. Get used to the syntax and familiarize yourself with the errors that arise when you mistype something. Clear the console or simply refresh the page whenever you need a clean slate. Code, code, code, **code**, ***code***.

Opening Files in the Browser

In this section, you will begin working with HTML files (most often, `index.html`). In order to view the results of the coding you'll be doing, you'll need to open the file in the browser. Instructions for each programming environment are as follows:

- **Local environment on Mac**: Run `open index.html` in the terminal.
- **Local environment using WSL/Ubuntu**: Run `explorer.exe index.html` in the terminal.

Keep these instructions handy — you will be doing this often.

> **Note:** In order for these instructions to work, you will need to have Google Chrome set as your default browser. You can find **instructions for Mac and Windows here** (https://support.google.com/chrome/answer/95417?hl=en&co=GENIE.Platform=Desktop) .

Conclusion

In this lesson we learned about the DOM, which is a "middle layer" that presents the HTML, CSS and JavaScript loaded by the browser when we visit a page. We normally interact with it through the `document` object. Because it is the "source of truth" for what browsers display, changes to the DOM create changes in the browser screen. We also learned how to access the Chrome Developer Tools and use the Console to try out code as we're building our programs.

Resources

- **CSS Tricks - What is the DOM? (https://css-tricks.com/dom/)**
- **MDN - The DOM (https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction)**