Lesson | Monday

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / How Web Browsers Work

Text | Cheat sheet

There are many standard structures in place that define how the internet works and how websites get loaded into our browsers. Throughout the program, we'll incrementally learn about these subjects, and this lesson is one of our first opportunities.

The goal of this lesson is to build context around how web browsers are structured so that we are prepared to build interactive websites with JavaScript and built-in web browser structures and tools. As such, this lesson is really meant to be a conceptual introduction to tools we will apply in future lessons. If any new concept in this lesson doesn't make sense to you, know that we will be revisiting all of them.

Browsers Manage All of Our Interactions on the Web

It's helpful to start out by reminding ourselves of something we probably take for granted: web browsers handle the entire process of displaying websites to us. What's in this process? There are a lot of steps:

- **Providing a way for us to navigate to websites**: a browser window with an address bar to input URLs so we can request to view a website.
- **Fetching website resources**: getting the source code and other files for the website we are navigating to.
- **Processing website resources**: parsing the source code and combining it into the cohesive website that we see in our browser.
- **Displaying the website on our computer screen**: the technical process of actually rendering a website in our browser window.

This process of populating a webpage in our browser is fascinating and quite complex! The above steps are generalized and can each be broken down into more steps that each draw on various web technologies.

The process of populating a web page is also a standardized process, which makes it so that everyone — different browser companies, web developers, and people — can access the internet and its websites by following the same standard process, no matter which browser or internet provider you are using.

We don't need to worry about understanding this process in detail, but what we do need to recognize is that web browsers manage all web interactivity. So, if we want to build a website that is interactive, we need to learn how to access and use web browser structures and tools.

Web APIs

The web browser tools and structures that are made available to developers are just a collection of many different objects that are built-in to the browser. These tools are collectively called Web Application Programming Interfaces, or **Web APIs**, which we'll learn more about in an upcoming lesson.

Notably, Web APIs are not a part of the JavaScript programming language. Instead, they sit on top of JavaScript; this means that they look like JavaScript and we can use JavaScript to access and use these tools. However, Web APIs are not a part of the core JavaScript programming language.

Differentiating between Web APIs and JavaScript can be confusing at first, but we have a series of lessons in this and future course sections that work on differentiating the two by exploring where all of these tools are located within Mozilla Developer Network (MDN) documentation (https://developer.mozilla.org/en-US/). So remember that this is a conceptual introduction to web browser tools. If you find anything confusing or unclear in this lesson, know that we'll be revisiting all of this information in upcoming lessons.

Up next in this lesson, we'll learn about 3 important browser structures that enable developers to interact with the browser window, our HTML document, and events (like a mouse click). In a future course section, we'll discuss the browser structure that enables us to interact with our CSS — so don't worry about that now!

Our Web Browser Is Accessed Through the `window` Object

A browser window or tab is represented by **the `window` object**. Every browser automatically creates a new `window` object every time we open a new window or tab. In other words, the `window`

object represents our current browsing context and lets us access it with JavaScript. So, if we navigate to a new tab, there will be a separate `window` object for that tab.

Note that `window` is a global variable that's built-in to our browsers and always available. For example, you could open up your DevTools console now and input `window` and see the details of the current browsing context.

Just like with JavaScript objects, the `window` object has properties that describe what a browser `window` is and does. For example, we can access `window` properties to find this information:

- The URL of the website we are on.
- The height and width of our browser window in pixels.
- The language that our browser is using.

And we can call on `window` methods to interact with our browser window:

- Go backwards and forwards in our browser history.
- Refresh the page.
- Open a new browser window.

This is a very short list of examples. The `window` object offers a lot of functionality, and we'll learn more soon. For now, understand that the `window` object is a representation of a browser window or tab, and all of the information related to our current browsing context.

Our HTML Document Is Accessed Through the HTML Document Object Model (DOM)

When the browser displays our website's HTML, it takes the HTML source code and turns into a HTML **Document Object Model (DOM)**. In fact when we see our HTML in the browser, what we're actually looking at is the HTML Document Object Model, and not our source code.

At its most basic, we can understand the DOM as a series of related objects. All of these objects in the DOM are organized hierarchically so they accurately represent the structure, content, and markup of our HTML source code. In other words, anything that's in our HTML source code — an element, an attribute of an element, text — is turned into an object in the DOM.

The browser creates a Document Object Model representation of our HTML so that our HTML can be manipulated with a scripting language like JavaScript. For example, we can use JavaScript to access the DOM to add or remove HTML elements. Manipulating the DOM doesn't change the HTML source code — it only changes the DOM that's displayed in the browser window. Without the DOM, JavaScript would have no way of knowing about or interacting with our websites!

Similar to the browser's `window` object, the DOM has an object called `document` that represents, well, the DOM! If we want to change anything in the DOM, we'd do so by accessing the `document` object. Note that `document` is also a global variable that's built-in to our browsers and always available.

Just like with JavaScript objects, the `document` object has properties that describe what a `document` is and does. Since we know that the `document` object represents the HTML DOM, we'll be able to access `document` properties to do all sort of things related to the DOM:

- Change, add, or remove an HTML element's attribute.
- Get all HTML elements that match a search query, like all `<p>` tags.
- Add and remove HTML elements.
- Show and hide HTML elements.

This is just a short list of examples! We can manipulate our website's HTML in many ways, thanks to the DOM.

Events

In computer programming an **event** is any action or occurrence that software can recognize. An event can be triggered by a human, like a mouse click, or by something that our program creates, like an error. When we write our programs to be interactive, we write our programs to react to events!

So far we've created **static** webpages with HTML and CSS. The most interactive feature we've used are HTML anchor tags to create hyperlinks that take us to other webpages. It's interactive, because we can click it and the webpage reacts by navigating to another webpage. However that's pretty limited — our webpages haven't included buttons that we can click, forms that we can submit, or other interactive features. Because of this, our webpages are considered static.

When we write code that reacts to events, our webpages go from static to interactive. Consider an online website that hosts a game like Tic Tac Toe (https://www.mathsisfun.com/games/tic-tac-toe.html). We play the game with mouse clicks: every time a player is ready to place their X or O on the game board, they must click on a square on the webpage. To make this Tic Tac Toe game functional, we need to have code that is able to recognize the click event and do something in response.

We call this process **event handling**. The click is the event and how we handle the click event corresponds to how our code is set up to react to the event. In this course, we'll learn how to use JavaScript and web browser tools to handle events. Using Tic Tac Toe as an example, we can write code that listens for a click event on the game board that will trigger a JavaScript function to update the board with an X or an O.

In web development, events arise from user interaction with specific browser structures — like scrolling in a browser window or clicking on an HTML element. Each event has a name, like 'click' or

'scroll', and belongs to a specific type of object, like `document` (our website's HTML) or `window` (our browser window or tab). This enables developers to easily write code that targets specific events.

We'll learn more about event handling soon, for now just keep in mind that events are occurrences that our websites can recognize and we can write JavaScript to react to.

Summary

We covered a lot of new concepts in this lesson! By now, we should have a basic conceptual understanding about how the web browser handles creating both the `window` and `document` objects when we navigate to a webpage. Developers write the source code (the HTML, CSS, and JavaScript), and the browser handles parsing it (including creating the `window` and `document` objects) and displaying it to users.

If anything we've covered is unclear, know that this lesson is only meant to be a conceptual introduction to these concepts and tools. We will be reviewing and applying all of these concepts in upcoming lessons, so you will come to understand them all soon.

These are the main points that we covered in this lesson:

- Web browsers manage interactivity on the web — they provide the tools that web developers can learn to use to make their own websites interactive.
- These browser tools are called Web APIs; they are separate from the JavaScript language, but we do use JavaScript to interact with them.
- We can access and manipulate the browser window (or browsing context) via the `window` object.
- We can access and manipulate our HTML document via the DOM, which is accessed via the `document` object.
- Just like with JavaScript objects, the `window` and `document` objects have properties that describe what they are and can

do.

- The `window` and `document` variables are global, which means they are always available.
- Making an interactive website means writing code that reacts to events.
- Events in computer programming are any occurrence that a program can recognize and react to, like a mouse click or a form submission.
- Events belong to specific browser structures, like the `window` object and `document` object. In other words, there are browser window events, and there are HTML document events.

Previous (/introduction-to-programming/javascript-and-web-browsers/practice-writing-functions)
Next (/introduction-to-programming/javascript-and-web-browsers/accessing-window-properties)

Lesson 28 of 75
Last updated March 25, 2023

disable dark mode

(http://www.epicodus.com)