Lesson Tuesday

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / Understanding Web APIs: The DOM

Text Cheat sheet

It's time to move beyond `window` methods to get a user response. Remember in the "Business and User Interface" lesson when we learned this?

> However, `window.prompt()` and `window.alert()` really shouldn't be in our code at all — very few users like to be prompted or alerted (and both now have strong associations with hacky sites and malware). We've been using these `window` methods because they are the easiest way for beginners to get a user response, and we will continue to use them for a little while longer because they are so easy to use — but be aware they should generally be avoided.

Well, it's very true. Now that we have a taste of writing code that interacts with users, it's time to learn the conventional tools used to handle user interaction: forms, buttons, and event handling.

But before we can dive into learning these tools, we need to spend time learning how to access and manipulate the DOM. Why? The DOM houses all of our website's HTML. So, if we create a form in our webpage that we want the user to fill out and submit (an event), we'll need to know how to access the form through the DOM so we can set up our event handling.

We're not going to worry about understanding the exact process of setting up event handling now. In this lesson, we will take time to review what the DOM is in more depth. Then in the next lesson, we'll learn how to access DOM elements.

The 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.

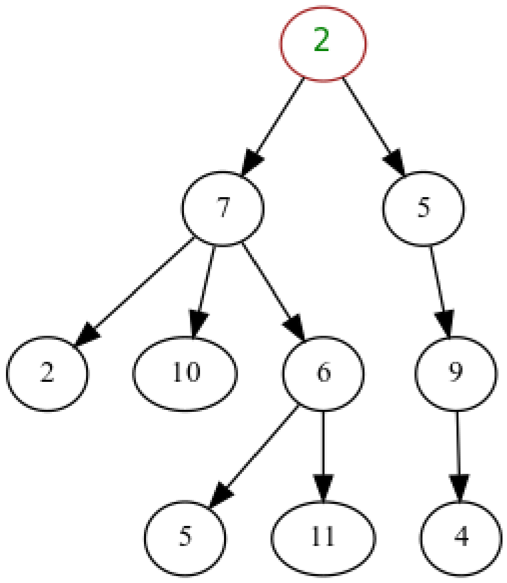
At its most basic, we can understand the DOM as a series of related objects. All of these objects in the DOM are organized hierarchically so they accurately represent the structure, content, and markup of our HTML source code. In other words, anything that's in our HTML source code — an element, an attribute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

The DOM is a Tree

In technical terms, the DOM is our HTML represented as a hierarchical "tree" made up of "nodes". A **tree** is a data structure in computer programming, and there are many types of trees, but we won't spend time learning about all of them in this program.

At its most basic, a tree consists of a root node that branches to child nodes, which in turn branch off into more child nodes. By default, nodes don't have any value — they can be of any data type. A node is just a data point along the tree, and it is always defined as in relation to other nodes. The following image from Wikipedia (https://en.wikipedia.org/wiki/Tree_(data_structure)) shows a tree with a root node of `2`, and a series connected and branching nodes, going downwards:

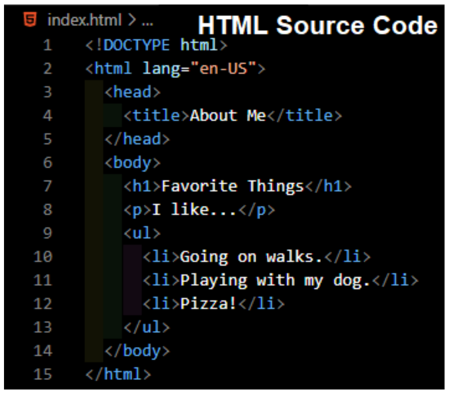


The tree data structure is similar to how trees in nature start at the trunk (the root node) and extend into a canopy, moving from limbs to branches to twigs (or, moving from node to node to node). We also use trees outside of computer programming to represent hierarchical relationships, like in a family tree or a species classification tree. And this is the main goal of trees — to represent hierarchical relationships between data.

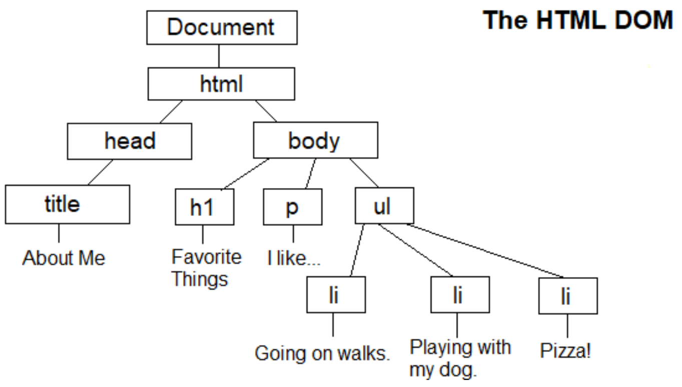
The way we describe the relationships between each node in a tree is to identify whether it is a parent, child, or sibling. Using the image above, we can say:

- The node that has no parent is the root node, and it has a value of `2`.
- The parent of node `7` is the root node with a value of `2`.
- The nodes with values `7` and `5` are sibling nodes.
- The node with `9` as its value is a child of node `5`, and a parent to node `4`.

Let's look at another example of a tree, this time of an HTML document.



```html
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <title>About Me</title>
  </head>
  <body>
    <h1>Favorite Things</h1>
    <p>I like...</p>
    <ul>
      <li>Going on walks.</li>
      <li>Playing with my dog.</li>
      <li>Pizza!</li>
    </ul>
  </body>
</html>
```

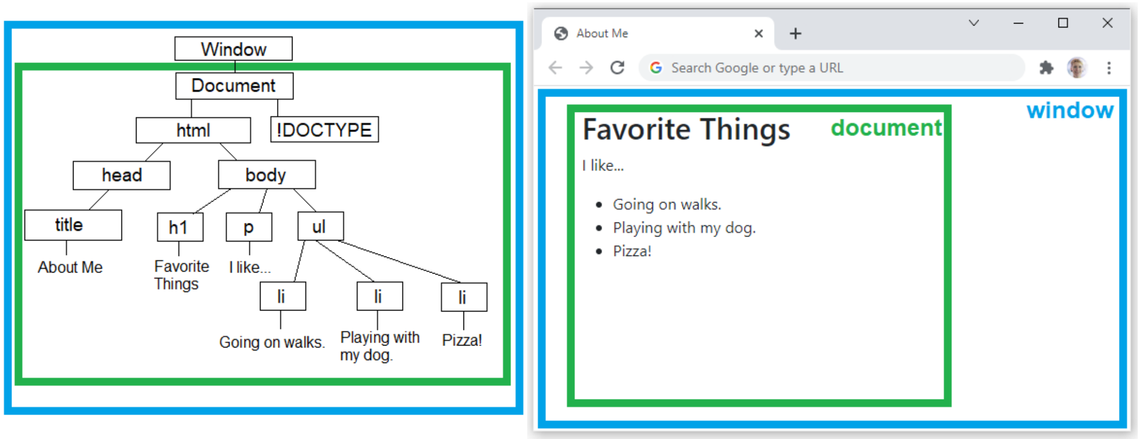


This image shows one HTML document: on the left it's represented as source code, and on the right it's represented as a Document Object Model. In the DOM tree (the right in the image), the root node is the `document` object, which is a built-in object provided by the browser, just like the `window` object. Every node that branches from the `document` object in the DOM is an object that represents either an HTML element, text, or attribute. **The `document` object** is the entry point to accessing all of the DOM, including HTML elements, attributes, and text.

It's also important to note that every node in the DOM is an object. In terms of data structures, the DOM is a tree made up of nodes. However, since each node's value is an object, the HTML DOM is effectively a series of nested objects that stem from the `document` object.

A Hierarchy of Browser Objects — `document` Belongs to the `window` Object

The `document` and `window` objects we have learned about so far are not separate objects. **In fact, by definition a `window` object represents our current browsing context with an HTML DOM inside of it**. This means that the `document` object belongs to the `window` object. Check out this image that visualizes the relationship:



The image shows the DOM tree next to the DOM in the browser. In the DOM tree, we can see that we've added `window` above `document`. Because trees show hierarchical relationships, what this means is that the `document` object is nested inside of the `window` object. Or, in other words, the `document` object is a property of the `window` object.

Also in the image, we can see two colored squares that show what the `window` and the `document` include, both in the DOM tree and in the browser. The blue square represents the `window` object — our browser window or tab. The green square represents the `document` object, or the HTML DOM, and it is nested inside of the blue `window` square. Hopefully this relationship is intuitive — if `window` is our browser window, and our browser window contains our HTML, then the HTML `document` should be located within the `window`. (If you are wondering about our website's CSS and JavaScript, this also gets added to the `window` object. We'll learn about each topic a bit more in upcoming lessons, but we won't be learning about either topic in depth.)

The DOM is a Web API

Just a reminder that the DOM is a Web API!

- The DOM is a Web API specification. A **specification** describes what the purpose of the API is, how it is structured, what it can do, and what other technology it interacts with.
- An interface belonging to the DOM is the `document` object, which is also the entry point to the DOM. An **interface** is an object of a specific type that contains a specific set of functionality. We interact with a Web API through its interface(s).
    - Remember: the `document` object is a global variable that's built-in to web browsers.

Summary

We reviewed information about the DOM that we already learned in previous lessons:

- The DOM is a Web API specification.
- The `document` object is the entry point to the DOM.
- The `document` object is a Web API interface.

We also learned new information about the DOM:

- The DOM is a tree made up of nodes, organized hierarchically. Each node is an object, so the HTML DOM is effectively a series of nested objects that stem from the `document` object.
- Just as we view our website within the browser window, the `document` object is nested inside of the `window` object. By

definition a `window` object represents our current browsing context with an HTML DOM inside of it.

Previous (/introduction-to-programming/javascript-and-web-browsers/practice-more-function-writing)
Next (/introduction-to-programming/javascript-and-web-browsers/accessing-the-dom)
Lesson 43 of 75

Last updated more than 3 months ago.

disable dark mode

Epicodus (http://www.epicodus.com)