Web Programming for Beginners: A Tutorial

Clarissa Littler

August 1, 2016

1 The Document Object Model

In this section, we'll be introducing the document object model and explain how to accomplish basic tasks with it.

1.1 What is the DOM?

In short, the Document Object Model (DOM) is an interface that the browser provides a JavaScript programmer so that JavaScript code can access the elements of the site and modify them.

This ability to programmatically modify and create websites is really the key technology that separates modern websites from how websites looked in the early '90s.

As the name might imply, this interface is in the form of giving a representation of a web page and its elements *as objects*.

1.2 First things first: making sure your page is loaded

The very first thing that we need to do is make sure that the web page is *fully loaded* before we start trying to manipulate the DOM, or else you'll find yourself in a position of trying to change elements before they've actually been made yet.

This is necessary because there's no guarantee that your JavaScript code will run *after* the page has been loaded.

Instead, you need to wrap your code up in the following way

```
window.onload = function () {
    (your code goes here)
};
```

1.3 Creating elements

The first thing we're going to explain is how to *create* HTML elements in code. The key function here is `document.createElement`. Most of the functions you'll need for changing a website with code are going to be *methods* of the `document` object, and this one is no exception. You can make an element corresponding to any tag by calling this function like `document.createElement("div")` or `document.createElement("ol")`. These functions *create* an element, but that element isn't a part of the site until it has been **added** to the site. To do that, we need two things: a place to put the elements and a way to add them.

The first, a place to put these new elements, is going to be `document.body` which, as the name implies, is the **body** of the webpage. The next is a way to attach it to the place of our choosing, which is going to be the `appendChild` method that *every* DOM element has.

Finally, we need one more piece to make something interesting: a way to make text. Text is not made in the same way as other elements, instead it's created with the `document.createTextNode` function which takes a string and returns a DOM element corresponding to raw text.

It's time now for our first example, which creates a very simple website just in code.

```
<!doctype html>
<html>
  <head>
    <script>
      window.onload = function () {
         var newHeader = document.createElement("h1");
         newHeader.appendChild(document.createTextNode("This is a header!"));
         document.body.appendChild(newHeader);
      };
    </script>
  </head>
  <body>
  </body>
</html>
```

and if you put this code in a file, say `createElement.html`, and visit the file in your browser then you'll see a page that has a single header.

Exercise: Take the above code and modify it so that below the header there is an ordered list with three list items inside them.

1.4 Retrieving elements

Being able to create items is all well and good, but much of the time we're not going to be creating an entire page from scratch and are, instead, going to modify elements that already exist.

The easiest way to *find* an element in the web page is to the use the function `document.getElementById`. For example, the following site will demonstrate using `document.getElementById` to get the *second*, not the *first*, ordered list.

```
<!doctype html>
<html>
  <head>
    <script>
      window.onload = function () {
        var newItem = document.createElement("li");
        newItem.appendChild(document.createTextNode("item in the second list"));
        var secondList = document.getElementById("list2");
        secondList.appendChild(newItem);
      };
    </script>
  </head>
  <body>
    <ol id="list1">
      <li>This is a list</li>
    </ol>
    <ol id="list2">
```

```
      <li>This is our second list</li>
    </ol>
  </body>
</html>
```

Beyond `getElementById` a useful way to retrieve elements is by their *type*. The function `document.getElementsByTagName` allows you to retrieve *all* elements with a particular tag type. The function returns a "NodeList", which while not actually an array can be treated like one for the most part. So, for example, in our previous example we could add an element to *each* of the two ordered lists as follows

```
window.onload = function () {
    var lists = document.getElementsByTagName("ol");

    for(var i = 0; i < lists.length; i++){
        var list = lists[i];
        var newItem = document.createElement("li");
        newItem.appendChild(document.createTextNode("new element"));
        list.appendChild(newItem);
    }
};
```

There's also a variation of `getElementsByTagName` where you can select all the items that are children of a *particular element*. For example, in the following code we select all of the line elements in the list and add some text to them. This requires two other pieces we haven't seen before, which are retrieving modifying the text within the text node *and* getting the text node as the child of the list item.

The code looks as follows:

```
<!doctype html>
<html>
  <head>
    <script>
      window.onload = function () {
        var list = document.getElementById("list");
        var items = list.getElementsByTagName("li");
        for(var i=0; i < items.length; i = i+1) {
          items[i].firstChild.nodeValue = "changed!";
        }
      };
    </script>
  </head>
  <body>
    <ol id="list">
      <li> This </li>
      <li> is </li>
      <li> a </li>
      <li> list </li>
    </ol>
  </body>
</html>
```

Here we used the propert `firstChild` to access the text node inside the list item and `nodeValue` in order to access the actual text-within-the-text-node. We can assign a different value to `nodeValue` like with any other property.

Exercise: Take the above web site and change the code so that it appends the index of the element to the end of the text.

1.5 Modifying CSS

Another thing we can do, programmatically, is change the styles and CSS class associated with our data. First, we can access the style data for an element by using the `style` property.

For example, we can change the font color of a header using code as follows

```
<!doctype html>
<html>
  <head>
    <script>
      window.onload = function () {
        var h = document.getElementById("header");
        h.style.color = "red";
      }
    </script>
  </head>
  <body>
    <h1 id="header">This is a header!</h1>
  </body>
</html>
```

As you can see from the example, the properties we're used to setting in CSS become properties of `style` object within in the element.

We can *also* change the CSS class of an element, using the property `classList`, which itself will allow us to modify the set of classes that apply to an element. For example, let's redo our example above with classes instead of modifying style directly

```
<!doctype html>
<html>
  <head>
    <style>
      .reddish {
        color: red;
      }
    </style>
    <script>
      window.onload = function () {
         var h = document.getElementById("header");
         h.classList.add("reddish");
      };
    </script>
  </head>
  <body>
    <h1 id="header">This is a header</h1>
  </body>
</html>
```

Similarly to the above example, you can *remove* a class with `classList.remove`.

1.6 Events

The entirety of this tutorial so-far we've been looking at changing things, in some sense, before anything you really start using the page. There's been nothing *interactive* yet. In order to add interactivety into our sites we need to learn about *events* in JavaScript.

Events are the connection between the user-interface and the code. Every time you click, type, or move your mouse the browser registers it as an *event*. Normally, nothing happens when these events fire, but you can add code that *listens* for an event and then runs some function.

Our first example will be the `mouseover` event. We're going to have a webpage with just a simple header that changes color when you mouse over it. To do this, we're going to *add* an event listener to the header element for the `mouseover` event that will change the color to red and an event listener for the `mouseleave` event that will turn it back to black when you move the mouse away.

```
<!doctype html>

<html>
  <head>
    <script>
      window.onload = function () {
         var h = document.getElementById("header");

         h.addEventListener("mouseover", function () {
            this.style.color = "red";
         });

         h.addEventListener("mouseleave", function () {
            this.style.color = "black";
         });
      };
    </script>
  </head>
  <body>
    <h1 id="header">This is our header!</h1>
  </body>
</html>
```

There are a number of other events in JavaScript. There's a general reference available through Mozilla, but it's not the most beginner friendly list of possible events. Most of the ones you'll need are things like, `mouseover`, `mouseleave`, `click` and their ilk.

We'll look at one more quick example, this one that will create a collapsing list. We'll wrap our content in a `div` and attach the event handlers to the `div` so that the mouse events are tracked over the total size of the visible content.

```
<!doctype html>

<html>
  <head>
    <script>
      window.onload = function () {
        var list = document.getElementById("list");
        var div = document.getElementById("content");
        div.addEventListener("mouseover", function () {
            list.style.display = "block";
        });
        div.addEventListener("mouseleave", function () {
            list.style.display = "none";
        });
      };
    </script>
  </head>
  <body>
    <div id="content">
      <h3>Our list is below here</h3>
      <ol id="list">
        <li>First item</li>
        <li>Second item</li>
        <li>Third item</li>
        <li>Fourth item</li>
      </ol>
    </div>
  </body>
</html>
```

1.7 Using inputs and buttons

Our final section in this beginner's tutorial is how to use buttons and inputs in your interactive code.

First, buttons are rather simple: mostly you'll use them by attaching a `click` event to them and running code accordingly. Second, we'll be specifically considering text inputs but this holds true for all inputs: you use the `value` property to retrieve the data that's inside the input. For a simple text input, this means the string that's currently inside the input.

To demonstrate all these pieces, we'll create a very very simplified to-do list page where you can add items to a list from the text input. [1]

```
<html>
  <head>
    <script>
      window.onload = function () {
         var inputElement = document.getElementById("input");
         var todoList = document.getElementById("list");
         var addButton = document.getElementById("add");

         addButton.addEventListener("click", function () {
            var itemText = document.createTextNode(inputElement.value);
            var newItem = document.createElement("li");
            newItem.appendChild(itemText);
            todoList.appendChild(newItem);
            inputElement.value = "";
         });

         inputElement.addEventListener("focus", function () {
            inputElement.style.fontWeight = "bold";
         });

         inputElement.addEventListener("blur", function () {
            inputElement.style.fontWeight = "normal";
         });
      };
```

[1] In later sections we'll expand this to be a **real** application that allows you to delete items and synchronize them with a server

```
    </script>
  </head>
  <body>
    <h1>Welcome to your to-do list</h1>
    <ol id="list">
    </ol>
    <input id="input" type="text"></input>
    <button id="add">Add element</button>
  </body>
</html>
```

The new events we've introduced are

- click
- focus, which is when the element gains focus
- blur, which is when an element loses focus

2 On Servers

These sections are To Be Written

2.1 The role of a server

2.2 Installing Node

2.3 Asynchronous vs. Synchronous Execution

2.4 The Simplest Node Server