Introduction to JavaScript for Web Development

Web Design and Development Bootcamp

Version 1.0

Table of Contents

Chapter 1: Introduction to JavaScript

Topic: The JavaScript Runtime

Every programming language requires a runtime that translates its instructions into a native operating system language.

- Java has the JRE (Java Runtime Environment)
- C# as the .NET CLR (Common Language Runtime)
- Python uses the Python Runtime
- JavaScript uses the browser's JavaScript runtime environment

JavaScript was developed to run in a browser. It was designed to give users a richer experience when interacting with web pages.

A web page has three distinct parts: HTML, CSS, and JavaScript.

Image: A web page consists of HTML, CSS and JavaScript

The HTML file links those parts together. This example HTML file could be named `index.html`:

```
<html>
    <head>
            <title>Page Name</title>

            <link rel="stylesheet" href="css/main.css" />
            <script src="js/app.js"></script>
    </head>
    <body>
    </body>
</html>
```

When the browser receives the index.html file, its contents are loaded into memory. The browser parses the HTML file, locates the CSS and JavaScript dependencies, and loads the additional files into memory.

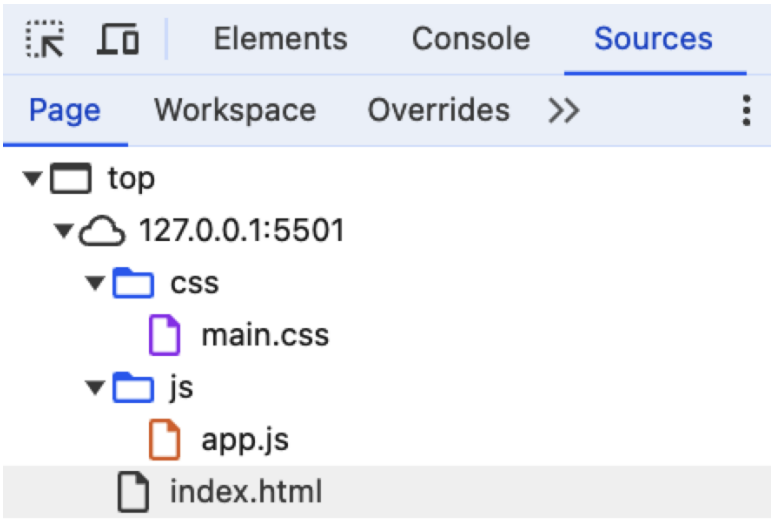

Image: The developer tools sources tab shows three distinct files

The web browser is the container for a web application. Since a web page has three parts, the browser must manage three distinct responsibilities.

1. It parses the HTML file to define the structure of the web page - this is called the DOM (Document Object Model)
2. It ***paints*** the canvas of the page using CSS

3. It provides a robust runtime in which your JavaScript application can execute

In this chapter, we will focus on the JavaScript runtime to learn both how JavaScript is like Java and also how it differs. In other words, we will focus on the JavaScript language, its rules, and syntax.

Beginning in the next chapter, we will examine how to use JavaScript to interact with a page's HTML and CSS.

Topic: Java vs JavaScript

Java and JavaScript have a few things in common, but don't let the name fool you. JavaScript is NOT *Java light*. It is easy to think that because the languages share similar names that JavaScript was developed to be a light version of Java that runs in a browser. This is not the case.

In 1995, Brendan Eich, a developer of Netscape, developed a scripting language named **Mocha** to enhance the browser's capability to do more than just display static pages.

By this time, Java had emerged as a popular programming language. Netscape and Sun Microsystems partnered and re-branded Mocha with the new name of JavaScript. The two languages, however, are distinct from each other and serve different use cases.

In 1996, with the desire to standardize JavaScript across browsers, Netscape submitted JavaScript to the European Computer Manufacturers Association (ECMA).

In 1997, the first edition of the ECMAScript standard was published. While the popular name of the language is still JavaScript, the official name is ECMAScript, and the official versions are named using ECMAScript. For example the following 3 names all refer to the same version of "JavaScript": **ECMAScript 2015** (for the year it was released), **ECMAScript 6** (for the version number) or **ES6** (for a short abbreviation).

Static vs Dynamic Typing

Java is a strongly typed language, whereas JavaScript is a dynamically or loosely typed language.

In Java, when a variable is declared, you must define its data type. The data type of that variable can never change.

```
int age = 25;
String name = "Larson";
Person person = new Person("Larson", 25);
```

In JavaScript, however, we cannot specify the data type in the declaration; instead, the JavaScript runtime infers the type of the variable based on its current value.

```
// age is a number because 25 is inferred to be a number
let age = 25;

// name is a string because of the quote''
let name = 'Larson';

// person is a dynamic object that may contain any attributes we define
let person = {
                    name: 'Larson',
                    age: 25
             };
```

In JavaScript, variables can be re-defined to hold different data types than when they were first initialized. For example, a variable can go from storing a number to storing a string or from storing an object to storing a string.

```
// age can be redefined a string
age = 'twenty-five';

// person can be re-defined as well
person = 'Larson';
```

Declaring Variables

JavaScript provides three keywords to declare variables: **var**, **let**, and **const**.

`var` is the original keyword used for variable declaration. It is used to declare global variables and, therefore, comes with unintended side effects.

In 2015, ECMAScript 6 was released and introduced 2 new variable declaration keywords. These keywords allow greater control over variable scope.

You should **avoid using** `var` in new code. It is only still available for backwards compatibility with existing code.

`let` declares a variable whose value can be changed later.

`const` declares a variable whose value (and therefore datatype) CANNOT be changed. This is the preferred variable declaration method and should be your default declaration. You can change a declaration to `let` if you determine that there is a need.

```
// declares a global variable named age that can be changed
```

```
var age;

// declares a scoped variable that can be changed
let price;

// declares a scoped variable that cannot be changed
// this should be your default declaration type
const name = 'Jose';
```

Working with Functions

JavaScript functions work like Java functions but they are declared differently.

```
// this function adds two numbers and returns the value
function add(first, second)
{
    return first + second;
}

// this function DOES NOT return a value
function displayMessage(message)
{
    console.log(message);
}
```

Note that JavaScript function signature DO NOT define a return type. Additionally, input parameters DO NOT specify the data type of variable being supplied. This is because JavaScript is a dynamically typed language. You can choose to pass in any data type you would like or to pass no parameters at all, and JavaScript allows it.

Arrays

Arrays in JavaScript function more like ArrayLists in Java. They are flexible and easy to use.

```
// declares an array with values
const names = ['Emma', 'Liam', 'Olivia', 'Noah', 'Ava'];

// declare an empty array
const numbers = [];

// push() adds a value to the END of the array
numbers.push(7);
numbers.push(21);
numbers.push(3);

// pop() removes a value from the END of the array
// lastNumber has the value 3
const lastNumber = numbers.pop()

// shift() adds a value to the BEGINNING of the array
numbers.shift(35);

// unshift() removes a value from the BEGINNING of the array
// firstNumber has the value 35
const firstNumber = numbers.unshift();
```

Arrow Functions

JavaScript arrays include several ***arrow functions*** that work like Java Stream functions.

filter()

`filter()` allows you to find items that match your criteria.

```
const numbers = [1, 2, 3, 4];
// this returns an array of all even numbers: [2, 4]
const even = numbers.filter(n => n % 2 === 0);
```

map()

`map()` changes the **shape** of the items in the array.

```
const numbers = [1, 2, 3];
```

```
// create a new array with each number doubled: [2, 4, 6]
const doubled = numbers.map(n => n * 2);
```

reduce()

reduce() aggregates the values in your array. Notice that the initial value is defined as the **second** argument instead of the first as it is in Java.

```
const numbers = [1, 2, 3, 4];
// aggregate all numbers by summing them: 10
const sum = numbers.reduce(
    (temp, current) => temp + current
    , 0 // initial value is the 2nd argument
);
```

Summary

JavaScript and Java share some similarities, but it is best to view JavaScript as a new programming language so that you don't confuse the nuances of each language with each other.

JavaScript was initially designed to be used in browsers, but it has evolved to a robust full stack development language that can also run as stand-alone applications on desktops and web servers. We will focus on the browser implementations of web development throughout this bootcamp.

The official JavaScript developers guide can be found at:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
- Data Structures
- Numbers and Dates
- String functions

Additional tutorials can be found at

- https://www.w3schools.com/js/
- https://www.w3schools.com/js/js_exercises.asp
- https://www.w3schools.com/js/js_quiz.asp

Chapter 2: Working with the DOM

In the last chapter, our focus was strictly on the JavaScript language and syntax. In this chapter, we will learn how to use JavaScript to manipulate the DOM. But first, what is the DOM?

Topic: Introducing the DOM

The DOM is an acronym for the **D**ocument **O**bject **M**odel. When your browser loads an HTML page into memory, it parses the file and creates a hierarchical tree of nodes in memory. Each node is an object that has properties (or attributes) and can be accessed through JavaScript. This hierarchical node tree is the DOM.

```
<html>
<head></head>
<body>
  <nav>
    <div>
      <img src="images/logo.png" />
      <ul>
        <li></li>
        <li></li>
        <li></li>
      </ul>
    </div>
  </nav>
  <main>
    <header>content</header>
    <form>
      <input type="text" id="priceInput" />
      <input type="text" id="quantity" />
      <button class="btn">Submit</button>
    </form>
  </main>
  <footer>Copyright ACME Corp 2024</footer>
</body>
</html>
```

The previous HTML would generate the following DOM.

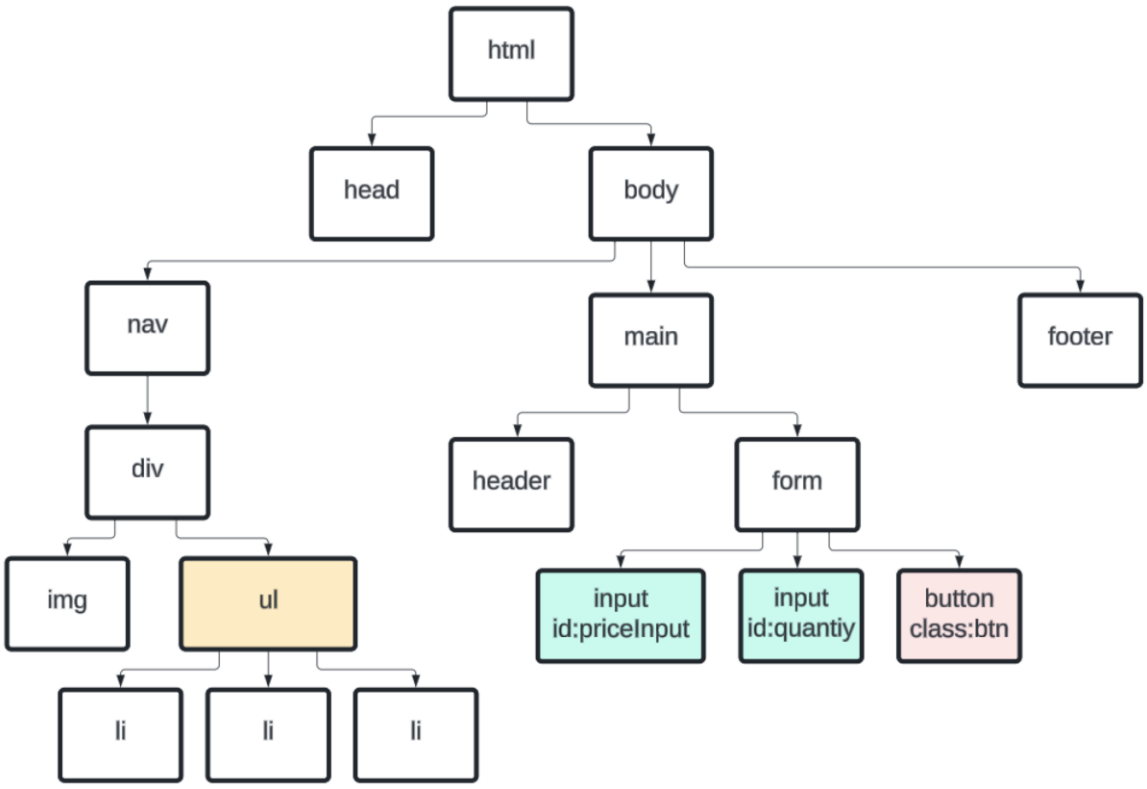

Image: The DOM is an in-memory tree of HTML nodes

Locating DOM elements

Using the DOM, the browser can efficiently find and access any element on the page. It can find elements by searching for an object by its `ID`, CSS `class name`, or `element name`. It is most common to search by `ID`. This is the most efficient type of search because an object's `ID` should be unique within the page and is indexed by the DOM.

The DOM is loaded into the browser's `document` object, which means all searches begin with the `document`. We can search for individual nodes.

```
// get a reference to the priceInput form element
const priceInput = document.getElementById("priceInput");

// search for and return the first element by class name
// note that we search for class names the same as with CSS
const button = document.querySelector(".btn");

// search for the first element by html element name
// this returns the first li in the nav bar's ul list
const listItem = document.querySelector("nav ul li");
```

We can also search for collections of nodes using `document.querySelectorAll()`.

```
// find all form input elements
const inputs = document.querySelectorAll("input");

// find all hyperlinks on a page
const links = document.querySelectorAll("a");

// find all elements with the class name priority-high
const divs = document.querySelectorAll(".priority-high");
```

Topic: Working with Elements

Once we have found the element we are looking for, we can interact with it programmatically. For example, we could read the value of a textbox or select list; we could add or remove a CSS class from a div, or replace the text in a paragraph; or we could create a new DOM element and insert it as a child of a specific node.

Common DOM Interactions

Changing Text

Change the text of a paragraph.

```
const element = document.getElementById("target");
element.textContent = "New random quote of the day";
```

Add HTML

Replace the contents of an element with new HTML.

```
const element = document.getElementById("target");
element.innerHTML = "<strong>New bold text</strong>";
```

Change Inline Styles

Dynamically add or modify styles inline.

```
const element = document.getElementById("target");
element.style.color = "maroon";
element.style.backgroundColor = "khaki";
```

Add/Remove CSS Classes

Dynamically add or remove CSS classes.

```
const element = document.getElementById("target");

// add a new class
element.classList.add("new-class");

// remove a class
element.classList.remove("old-class");
```

```
// toggle adding/removing a class
element.classList.toggle("active");
```

Manipulating the DOM Structure

We can also dynamically create new content and add it to the DOM.

Creating New Elements

We use the document object to create a new element.

```
// this creates a new div
const newDiv = document.createElement("div");
newDiv.textContent = "Lorem Ipsum...";
```

Note that creating a new element does not add it to the DOM, we still need to insert it.

Adding Elements to the DOM

```
// creates the new div
const newDiv = document.createElement("div");
newDiv.textContent = "Lorem Ipsum...";

// locate the parent element
const parent = document.getElementById("parent");

// insert the new div as a child of the parent element
parent.appendChild(newDiv);
```

Removing Elements from the DOM

```
// locate the element
const element = document.getElementById("target");
element.remove();
```

Summary

The DOM is an in-memory representation of the HTML page. Each HTML element is an object in the DOM.

Using JavaScript, you can locate and manipulate any element in the DOM. This allows you to create a rich user experience by responding to user actions.

To learn more about working with the DOM please see:
https://www.w3schools.com/JS/js_htmldom.asp

Chapter 3: Event Driven Programming

In Java console applications the flow of the application is controlled by the developer. As a developer, you prompt the user to select from a list of pre-defined options. Additionally, the user can only select from those options when they are presented. In other words, as a developer, you have complete control of when and how methods are executed.

For instance, when prompting a user for input, you prompt them one piece of information at a time. It is easy to validate user input in this model.

Creating rich and engaging user interfaces turns some control of application flow over to the user. A web page "listens" for user actions like button clicks, key presses, mouse movements and more. These are called events. As a developer you can respond to these events. They allow developers to create a much more interactive experience for the user.

An event driven application also allows a user to complete forms, click buttons, or links in any order. They may not complete an entire form before clicking the submit button. As a developer, you now need to include logic that accounts for the flexibility given to a user. Dealing with user input includes validating their input before submitting forms.

You need to think about how a user will interact with your site, and write your JavaScript to respond accordingly.

Topic: Responding to Events

The browser hosting your web page runs a continuous event loop which "listens" for web page and user events. When an event is fired the browser notifies the pages JavaScript runtime of the event.

You can subscribe to these events by creating an event handler function. Once you subscribe to an event your function will be executed every time the event is fired.

There are two ways to subscribe to an event, through declarative and imperative programming styles.

Declarative

Declarative programming means using semantic HTML to describe what you want to happen. The following HTML registers a buttons click event to be handled by a JavaScript function named `handleSubmit()`.

```
function handleSubmit()
{
    console.log("submit button clicked");
}
```

```
<button id="submitButton" onclick="handleSubmit()">Submit</button>
```

Imperative

Imperative programming means using code to explicitly define how each event should be handled. This approach separates all JavaScript and events from the HTML.

```
function handleSubmit()
{
    console.log("submit button clicked");
}

const button = document.getElementById("submitButton");

button.addEventHandler("click", () => handleSubmit);
```

```
<button id="submitButton">Submit</button>
```

When using Vanilla JavaScript it is most common to use the imperative programming style to attach event handlers to HTML elements.

Topic: Common Events

The events that a developer can react to are dependent on the type of HTML element a user interacts with. For example form input elements expose events that relate to user input, while display elements expose mostly mouse events.

Non-Form Events

HTML elements emit events to which developers can respond, such as `click`, `hover`, `mouseover`, `mousedown`, etc.

These events allow us to do things like convert common display elements into objects with which a user can interact.

click

The `click` event is not just for buttons, but can be attached to any HTML element, such as a `div`, `tr`, `image` or any other element.

A `click` event can often be used to navigate to another page programmatically, rather than using a pre-defined `<a href="">` element.

```
<div id="productDetail">Product Name</div>
```

```
function displayProductInformation(id)
{
    const url = `http://mysite.com/products?id=${id}`;
    window.navigate(url);
}

const div = document.getElementById("productDetail");

div.addEventHandler("click", () => displayProductInformation(5));
```

mousedown

The `mousedown` event is fired when a mouse button is pressed. The `mousedown` event has information about which mouse button was clicked, i.e. the left or right mouse button.

```
function mouseClick(event)
{
  if (event.button === 0)
  {
    console.log('Left button clicked');
  }
  else if (event.button === 1)
  {
    console.log('Middle button clicked');
  }
  else if (event.button === 2)
  {
    console.log('Right button clicked');
  }
}
```

Form Events

change

The `change` event fires when the value of an input has changed. Note: an input type text `change` event does not fire until after the textbox loses focus.

For a `checkbox` or `radio button` the `change` event fires immediately when the checked state changes.

keyup

The `keyup` event fires after the keyboard key is released.

keydown

The `keydown` event fires immediately when a keyboard key is pressed. If a key is held down this event fires repeatedly.

focus

The `focus` event fires when a form control receives focus.

blur

The `blur` event fires when a form control loses focus.

submit

A form's `submit` event fires when the embedded submit button is pressed.

Cancelling Events

Form events can be cancelled programmatically. For example, if a form fails validation, you may want to prevent the form from being submitted.

To cancel an `event`, your event handler must accept the `event` object as an input parameter. The event loop passes the event object to the handler function when the `event` is raised.

```
function submitForm(event)
{
    event.preventDefault();
}

const form = document.getElementById("registrationForm");

form.addEventHandler("submit", submitForm);
```

Summary

JavaScript applications are event driven applications. A rich user experience means creating an application that gives users intuitive options for interacting with the web page.

When a user triggers an event a JavaScript event handler function is executed.

Additional Event Documentation:

- MDN Introduction to Events
- W3Schools DOM Events List
- Client Side Form Validation
- Bootstrap Validation