Front-End Technologies - Module 2

Noroff Guide

Noroff Education

Table of Contents

Module overview

Introduction

Welcome to Module 2 of Front-End Technologies.

If anything is unclear, check your progression plan and/or contact a tutor on Discord.

Module structure	**Estimated workload**
2.1. Lesson and task HTML Document Object Model (part 1)	8 hours
2.2. Lesson and task HTML Document Object Model (part 2)	8 hours
2.3. Lesson and task JS Browser Object Model (part 1)	8 hours
2.4. Lesson and task JS Browser Object Model (part 2)	8 hours
2.5. Lesson and task Self-study	8 hours

Learning outcomes

In this module, we are covering the following knowledge learning outcomes:

→ The candidate has knowledge of the concepts, processes, and tools used to build static web pages.

- The candidate has knowledge of concepts, processes and tools in JavaScript language development.
- The candidate has knowledge of concepts of the Document Object Model (DOM).
- The candidate has knowledge of the concepts, processes, and tools used to interact with RESTful services.
- The candidate has knowledge of concepts, processes, and tools of git version control software.
- The candidate has insight into the HTML/CSS and ECMAScript specifications.
- The candidate can update their vocational knowledge in HTML/CSS.
- The candidate can update their vocational knowledge in JavaScript language.

In this module, we are covering the following skill learning outcomes:

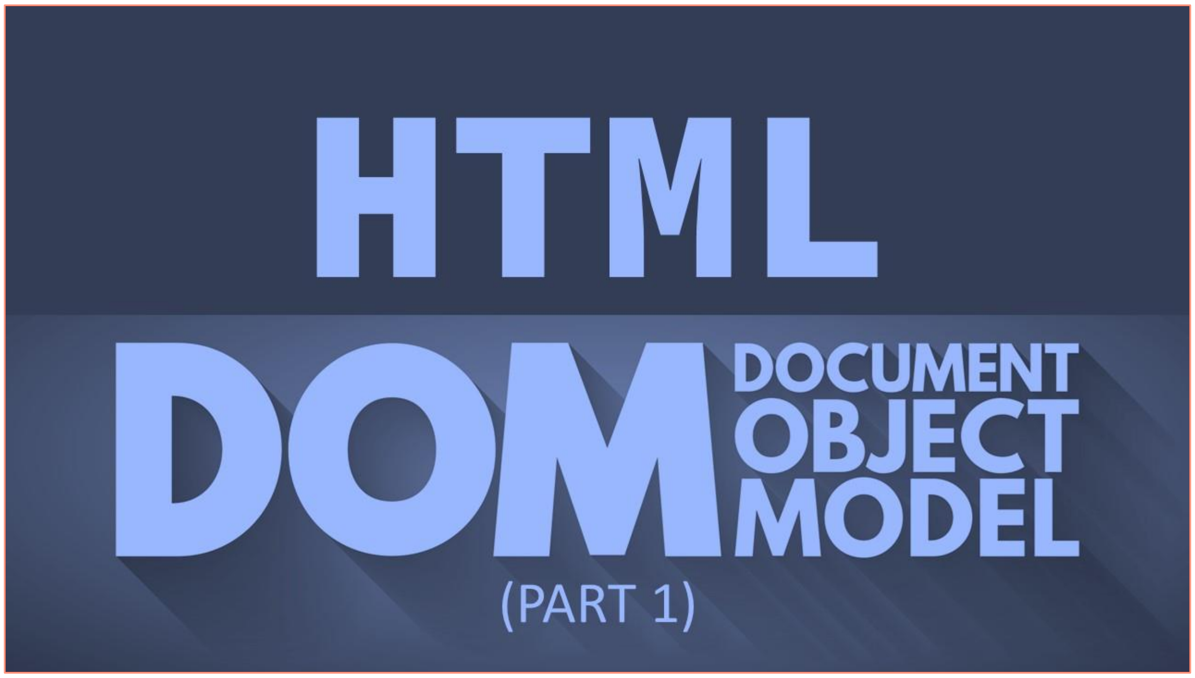
- The candidate can apply vocational knowledge of JavaScript to manipulate the Document Object Model (DOM) and retrieve and display data from existing REST services in static web pages.
- The candidate masters relevant vocational tools, materials, and techniques to create static web pages using HTML and CSS.
- The candidate masters JavaScript tools, materials, techniques, and styles used to enhance functionality in static web pages both as inline scripting and as external script files.
- The candidate masters git source control tools, materials, techniques, and styles used to create and work with a version-controlled project.
- The candidate can find information and material relevant to extending JavaScript functionality in static web pages through third-party libraries.

In this module, we are covering the following general competence learning outcomes:

- The candidate understands the ethical principles when creating JavaScript web solutions for public use.
- The candidate can carry out work based on the needs of a junior JavaScript developer.

2.1. Lesson - HTML Document Object Model (part 1)

Introduction



We already have a solid understanding of HTML, CSS and JavaScript. In this lesson, we'll learn how to connect and change our elements dynamically. We can do this thanks to the HTML Document Object Model (DOM)

We'll learn about the general idea and schema of HTML DOM. After this lesson, we'll know how to find the elements we're looking for with DOM and change their values. We'll also add new values and remove the old ones.

The last thing presented in this lesson is about changing the content of the HTML elements and their attributes. We can present our content in HTML dynamically, meaning the content will be determined when the program is run. Otherwise, our website content is predefined and static.

Learning outcomes

In this lesson, we are covering the following knowledge learning outcome:

→ The candidate has knowledge of concepts of the Document Object Model (DOM).

In this lesson, we are covering the following skill learning outcome:

→ The candidate can apply vocational knowledge of JavaScript to manipulate the Document Object Model (DOM) and retrieve and display data from existing REST services in static web pages.

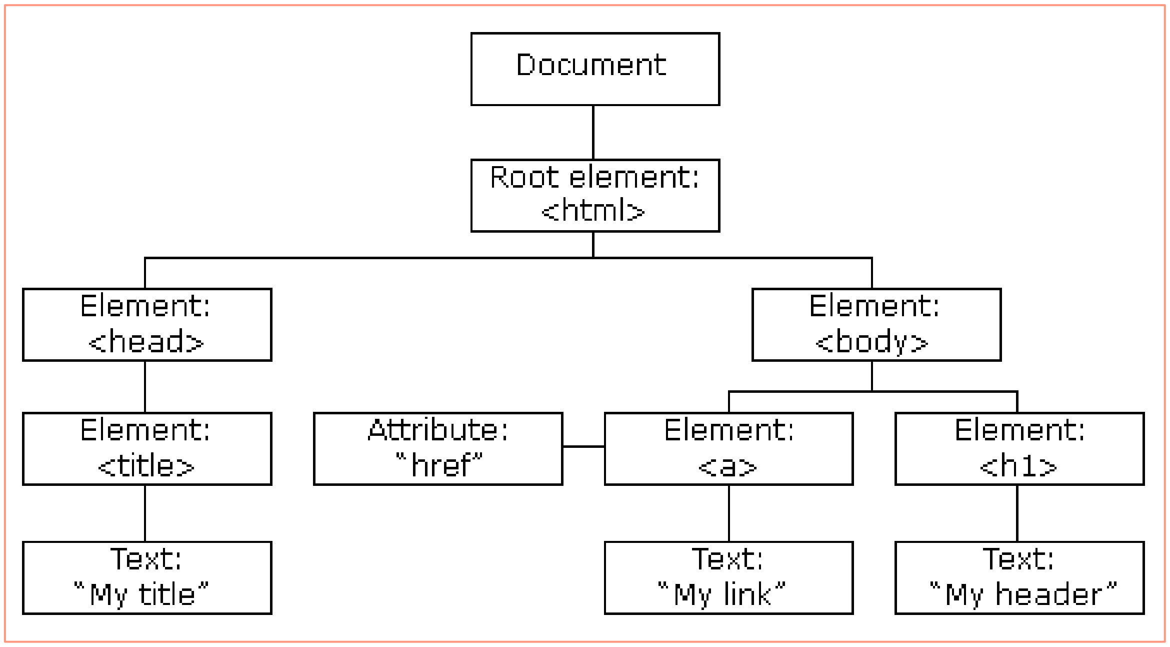
In this lesson, we are covering the following general competence learning outcome:

→ The candidate can carry out work based on the needs of a junior JavaScript developer.

Introduction to HTML DOM

HTML DOM enables JavaScript to access and change all of our HTML elements in the HTML document. This is a programming interface for HTML - a standard for how to get, change, add or delete HTML elements. After loading, the browser creates a Document Object Model of our page.

This is constructed as a tree of objects:



After getting such a model, JavaScript can create dynamic HTML, for example:

- change HTML elements on the page
- change HTML attributes on the page
- change all CSS styles on the page
- remove old HTML elements or attributes
- add new HTML elements or attributes
- react to existing HTML events and create new ones.

	READ Page: Introduction to the DOM by MDN Web Docs.

	WATCH Video: What is the DOM? (5m 18s) by Christian H. on LinkedIn Learning.

DOM methods

HTML DOM provides us with multiple methods and properties. Methods are actions we can perform on HTML elements. Properties are values of HTML elements, which we can set or change.

Let’s look at an example of changing a paragraph’s value within JavaScript:

dom.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>App</title>
    </head>
    <body>
        <p id="para1">Some random content</p>
        <script src="dom.js"></script>
    </body>
</html>
```

dom.js:

```
document.getElementById("para1").innerHTML = "It's the first
paragraph!";
```

We can see that the text “Some random content” was replaced by “It’s the first paragraph!”

The getElementById() is the most common method to access HTML element. ID is always unique and is required to find an element. This method will always only find one element.

The innerHTML is a property and the easiest way to get or set the element’s content. It can be used to get or change any HTML element, <html> and <body> too!

READ

Page: JavaScript Dom Methods by Tutorials Tonight.

WATCH

Video: The Document methods (10m 27s) by Christian H. on LinkedIn Learning.

ACTIVITY

Once again, change the paragraph’s value in JavaScript of the first example to “It’s the second paragraph”. Note which content is visible.

Answer: Click here to reveal.

DOM document

After loading into a web browser, our HTML document becomes a document object. It stores all other objects on our web page. If we want to access any element in an HTML page, we should always start with accessing the document itself.

We can use document objects to:

- find HTML elements
- change HTML elements
- add or delete HTML elements
- add event handlers
- find HTML objects.

Below we can see an example of accessing our website URL and storing it within the variable using a document object:

```
let url1 = window.document.URL;
let url2 = document.URL;
```

Since the document object is a window object property, both ways of accessing it above are correct.

The document object is the source of most information we can get using HTML DOM. We'll learn more about the details later in this course.

READ

Page: Document by MDN Web Docs.

WATCH

Video: The Document object (12m 8s) by Christian H. on LinkedIn Learning.

ACTIVITY

Open any website on the Internet and access and print the website’s URL using the browser’s console.

Finding DOM elements

DOM enables us to manipulate HTML elements, but to do this, we need to find them first. There are several ways to do this:

- by ID
- by tag name
- by class name
- by attribute name
- by CSS selectors
- by object collections.

The easiest way is by the element’s ID. IDs are always unique, so we’ll always find only one. As mentioned in earlier sections, we can do this using the getElementById() method. If we find the element, it’s returned as an object. Otherwise, null is returned.

To get all elements of a certain tag, we need to use the getElementsByTagName() method. This returns the array of the elements of this tag. With the getElementById() method, we can only select elements within another element, not necessarily in the entire document:

```
<!DOCTYPE html>
<html>
    <body>
    <div id="main">
        <p>Finding HTML Elements by Tag Name</p>
        <p>This example demonstrates the
<b>getElementsByTagName</b> method.</p>
    </div>
    <p id="demo"></p>
    <script>
        const x = document.getElementById("main");
        const y = x.getElementsByTagName("p");
        document.getElementById("demo").innerHTML =
        'The first paragraph (index 0) inside "main" is: ' +
y[0].innerHTML;
    </script>
    </body>
</html>
```

The code above presents where we only check the elements from the element of id= “main”. We use this method on the element instead of the entire document.

To find the element by class name, we should use the getElemenByClassName(). As with finding by tags, this method returns the array of elements. As with the parameter, we should pass the class name.

To use the CSS selector, we should use the querySelectorAll() method and pass the selector as the string argument:

```
<!DOCTYPE html>
<html>
    <body>
        <p>Add a background color to the first p element with
class="example".</p>
        <p>Paragraph 1</p>
        <p class="example">Paragraph 2</p>
        <p class="example">Paragraph 3</p>
        <script>
```

```
            document.querySelector("p.example").style.backgro
undColor = "green";
        </script>
    </body>
</html>
```

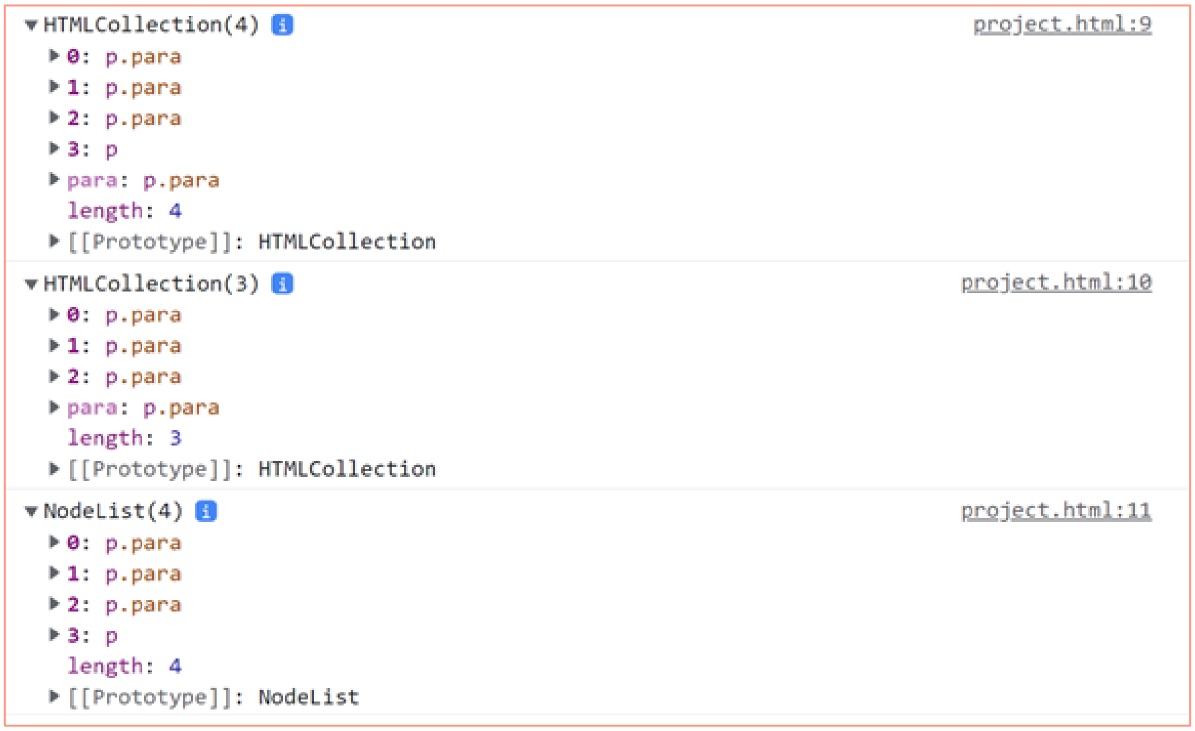
We’ll learn more about CSS selectors in the lesson about jQuery – the framework to help with HTML DOM operations.

HTML elements can be grouped into object collections, such as forms or images. We can access it by typing document._collection_name_, like in the example below where we print the number of images:

```
<!DOCTYPE html>
<html>
    <body>
    <img id="pikachu1"
src="https://i.pinimg.com/originals/c9/40/a3/c940a3dae3900d10
0990e5b697ef196c.jpg" alt="Pika Pika">
    <img id="pikachu2"
src="https://i.pinimg.com/originals/c9/40/a3/c940a3dae3900d10
0990e5b697ef196c.jpg" alt="Pika Pika">
    <p id="demo"></p>
    <script>
    document.getElementById("demo").innerHTML =
    "Number of images: " + document.images.length;
    </script>
    </body>
</html>
```

Let’s look at one more example of selecting multiple elements in other ways than using ID:

```
<!DOCTYPE html>
<html>
    <body>
        <p name="para" class="para">The first paragraph</p>
        <p name="para" class="para">The second paragraph</p>
        <p name="para" class="para">The third paragraph</p>
        <p name="para">The fourth paragraph</p>
        <script>
            console.log(document.getElementsByTagName("p"));
            console.log(document.getElementsByClassName("para
"));
            console.log(document.getElementsByName("para"));
        </script>
    </body>
</html>
```



```
▼HTMLCollection(4) ℹ                                   project.html:9
  ▶0: p.para
  ▶1: p.para
  ▶2: p.para
  ▶3: p
  ▶para: p.para
   length: 4
  ▶[[Prototype]]: HTMLCollection
▼HTMLCollection(3) ℹ                                   project.html:10
  ▶0: p.para
  ▶1: p.para
  ▶2: p.para
  ▶para: p.para
   length: 3
  ▶[[Prototype]]: HTMLCollection
▼NodeList(4) ℹ                                         project.html:11
  ▶0: p.para
  ▶1: p.para
  ▶2: p.para
  ▶3: p
   length: 4
  ▶[[Prototype]]: NodeList
```

We successfully selected all elements we wanted to. The last paragraph had no class attribute specified. Because of this, the second log shows only three paragraphs. We also see the collection isn’t a simple array but an HTMLCollection or NodeList. For now, we can use it like a simple array. We’ll learn about the differences later in this course.

WATCH

1. Video: Selecting elements with getElementById (4m 10s) by Ray Villalobos on LinkedIn Learning.
2. Video: Choosing elements by HTML tag (3m 20s) by Ray Villalobos on LinkedIn Learning.
3. Video: Isolating elements by class name (3m 12s) by Ray Villalobos on LinkedIn Learning.

ACTIVITY

Create a website containing three different links and show their number using DOM.

Source code: Click here to reveal.

Changing HTML

With the use of DOM, we can change the content of our HTML elements. To do this, we need to select them (as in the previous section) and modify the content with the innerHTML property, like in the example below:

```
<html>
    <body>
    <p id="p1">Hello World!</p>
    <script>
        document.getElementById("p1").innerHTML = "New
text!";
    </script>
    </body>
</html>
```

To change the attribute, we have to access a particular attribute as the element’s property, like in the example below:

```
<html>
    <body>
    <p id="p1">Hello World!</p>
    <script>
        document.getElementById("p1").style.backgroundColor =
"red";
    </script>
    </body>
</html>
```

We first need to access the style object to change the CSS property. We can then set the necessary property to its new value.

There are many other ways to change our element’s properties. For example, we can change its class:

```
<!DOCTYPE html>
<html>
    <body>
        <p id="para1" class="para">The first paragraph</p>
        <p class="para">The second paragraph</p>
        <p class="para">The third paragraph</p>
        <p>The fourth paragraph</p>
        <script>
            var para = document.getElementById("para1");
            para.className = "para2";
        </script>
    </body>
</html>
```

After executing the JavaScript code, the first paragraph will no longer have the para class but the para2 class.

We can change many element properties this way, but we won’t learn about all of them. However, you can always find the entire list of properties and examples of doing it in the official documentation linked in the READ section.

	READ 1. Page: Element by MDN Web Docs. 2. Page: Element.innerHTML by MDN Web Docs.

	WATCH Video: Changing HTML attributes (5m 25s) by Ray Villalobos on LinkedIn Learning.

	ACTIVITY Change the size of the paragraph's font to 20. **Source code:** Click here to reveal.

Dynamic HTML

JavaScript can create dynamic HTML content. Let's look at the example below:

```
<!DOCTYPE html>
<html>
    <body>
        <p id="demo"></p>
    <script>
        document.getElementById("demo").innerHTML = "Date : "
```

```
+ Date();
    </script>
    </body>
</html>
```

With every refresh, the displayed date is different. We can even go one step further. With intervals, we can refresh every five seconds:

```
<!DOCTYPE html>
<html>
    <body>
        <p id=”demo”></p>
    <script>
        setInterval(() =>
document.getElementById(“demo”).innerHTML = “Date : “ +
Date(), 5000)
    </script>
    </body>
</html>
```

We can also use the document.writeln() function we already know well from the beginning of the tutorial. This function is used to write directly to the HTML output stream. However, we shouldn’t use it after our document is loaded because it will override it.

ACTIVITY

Create a web page with one paragraph. Set the background colour to either red or blue randomly with an equal chance every five seconds.

Source code: Click here to reveal.

What did I learn in this lesson?

This lesson provided the following insights:

- HTML DOM and how to use it.
- Finding elements with the use of DOM.
- Modifying our page’s content with the use of DOM.

References

MDN Web Docs (2022) *Web technology for developers*. Available at: https://developer.mozilla.org/en-US/docs/Web

2.1. Lesson task - HTML Document Object Model (part 1)

The task

This lesson taught standard HTML DOM basics, such as document structure and finding DOM elements. You need to finish two tasks connected to what you learned in this lesson.

Part 1

In this task, you need to create a website that contains three paragraphs:

- Every five seconds, the font size of the first paragraph can either become twice smaller or larger, with an equal chance.
- The second paragraph shows the current date and refreshes it every 10 seconds.
- The background colour of the third paragraph can become either yellow, magenta or cyan every three seconds (with an equal chance).

You can use the

```
window.getComputedStyle(document.getElementById("para1"),
null).getPropertyValue('font-size');
```

code to get the font size after site initialisation. In this case:

```
document.getElementById("para1").style.fontSize
```

would return an empty string at the very beginning.

Source code: Click here to reveal.

Part 2

Having the following HTML template:

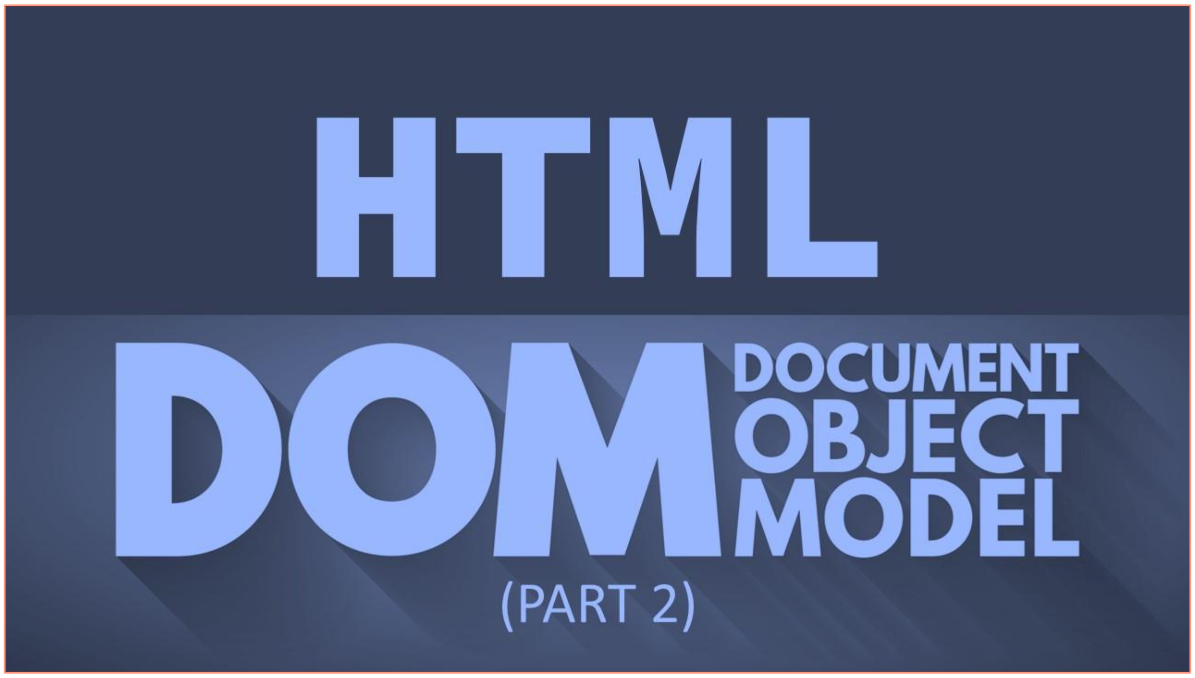
```
<!DOCTYPE html>
<html>
    <body>
        <img
src="https://i.pinimg.com/originals/c9/40/a3/c940a3dae3900d10
0990e5b697ef196c.jpg" alt="Pika Pika">
        <img
src="https://i.pinimg.com/originals/c9/40/a3/c940a3dae3900d10
0990e5b697ef196c.jpg" alt="Pika Pika">
        <p id="demo"></p>
        <button onclick="showDetails()">Click me!</button>
    </body>
</html>
```

Create JavaScript code that:

- Shows the number of currently displayed images as the text of a paragraph with the "demo" ID.
- After clicking the "Click me button", show the first image's width, height and source link in an alert. Get information directly from the image. The solution should work even if details were changed dynamically.

2.2. Lesson - HTML Document Object Model (part 2)

Introduction



We already know what HTML DOM is and how to use it. In this lesson, we'll learn more details about it. We'll learn about forms, events and nodes while improving our knowledge about other DOM uses, like collections.

Forms enable us to group controls and validate their values. We'll learn more about data validation, why it's crucial, and the concept of constraint validation. We'll also create animations with JavaScript, not with CSS (transitions, animations) but 'manually'.

We'll then focus on events and event listeners and the possibilities they provide. We'll conclude with nodes and collections, which will allow us to insert or remove the elements at the selected places.

Learning outcomes

In this lesson, we are covering the following knowledge learning outcome:

→ The candidate has knowledge of concepts of the Document Object Model (DOM).

In this lesson, we are covering the following skill learning outcome:

→ The candidate can apply vocational knowledge of JavaScript to manipulate the Document Object Model (DOM) and retrieve and display data from existing REST services in static web pages.

In this lesson, we are covering the following general competence learning outcome:

→ The candidate can carry out work based on the needs of a junior JavaScript developer.

DOM events

Thanks to DOM, JavaScript can react to HTML events. When an event occurs, JavaScript code is executed. We define the code to be executed in a function which is later bound to the event. There are various possible events, for example:

- when a user clicks the mouse
- when a web page is loaded
- when an image is loaded
- when the mouse hovers on an element
- when the input field is changed
- when the form is submitted
- when a user strokes a key.

Let’s look at a sample event in a simple example:

```
<!DOCTYPE html>
<html>
    <body>
    <h1 onclick="changeText(this)">Click on this text!</h1>
    <script>
        function changeText(id) {
        id.innerHTML = "Ooops!";
        }
    </script>
    </body>
</html>
```

As we can see, clicking on the text causes its value to change.

However, binding a function as the attribute isn’t the only way to do this. We can also use the HTML DOM.

Let’s look at an example:

```
<!DOCTYPE html>
<html>
    <body>
    <h1 id="myHeader">Click on this text!</h1>
    <script>
        function changeText() {
            document.getElementById("myHeader").innerHTML =
"Ooops!";
        }
        document.getElementById("myHeader").onclick =
changeText;
    </script>
    </body>
</html>
```

Other useful events are:

- onload and onunload – triggered when a user enters/leaves the page.
- onchange – triggered when a content value changes.

- onmouseover and onmouseout – triggered when we hover on an element and take the cursor out of it.
- onmousedown, onmouseup and onclick – when we press the mouse button, an onmousedown event triggers. When we release it, an onmouseup is triggered. When we finish the entire click, the onclick event triggers.
- onfocus – changes the background colour of the focused input.

You can view more events and examples of the listed events in the READ section.

Event listeners provide us with another way of applying events. Instead of binding a simple function, we can bind as many functions as possible to any DOM elements using DOM and event listeners. To do this, we need to use the addEventListener() method.

Let's look at a simple example of showing a Hello World alert after clicking the text:

```
<!DOCTYPE html>
<html>
    <body>
    <h1 id="myHeader">Click on this text!</h1>
    <script>
        document.getElementById("myHeader").addEventListener(
"click", myFunction);
        function myFunction() {
        alert ("Hello World!");
        }
    </script>
    </body>
</html>
```

We can add as many listeners to one event as we want. We merely need to invoke the addEventListener() method multiple times with the same event and on the same element:

```
<!DOCTYPE html>
<html>
    <body>
    <h1 id="myHeader">Click on this text!</h1>
    <script>
        document.getElementById("myHeader").addEventListener(
"click", myFunction);
        document.getElementById("myHeader").addEventListener(
"click", myFunction2);
        function myFunction() {
            alert ("Hello World!");
        }
        function myFunction2() {
            console.log("Hello world!");
        }
    </script>
    </body>
</html>
```

We can even create an event listener to our window object. Let’s show this alert on the window’s resize:

```
<!DOCTYPE html>
<html>
    <body>
    <h1 id="myHeader">Click on this text!</h1>
    <script>
        window.addEventListener("resize", myFunction);
        function myFunction() {
        alert ("Hello World!");
        }
    </script>
    </body>
</html>
```

We can also use an arrow function that calls another one with the parameters inside:

```
<!DOCTYPE html>
<html>
    <body>
    <h1 id="myHeader">Click on this text!</h1>
    <script>
        window.addEventListener("resize", () =>
myFunction("Matt"));
        function myFunction(name) {
        alert ("Hello " + name + " !");
        }
    </script>
    </body>
</html>
```

Event propagation is a way of defining the element order when an event occurs.

There are two ways of propagating events in the HTML DOM: bubbling and capturing.

Bubbling is about the inner elements being handled first and the outer last.

Capturing is about capturing the outer elements first and then the inner ones. By default, events are propagated by bubbling. However, we can specify which to use by passing the third parameter: useCapture.

If this parameter is true, capturing will be used:

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            #myDiv1, #myDiv2 {
                background-color: yellow;
                padding: 50px;
            }
            #para1, #para2 {
                background-color: blue;
                font-size: 20px;
                border: 1px solid;
                padding: 20px;
            }
        </style>
    </head>
    <body>
```

```
        <div id=”myDiv1”>
        <h2>Bubbling:</h2>
        <p id=”para1”>Click here!</p>
        </div><br>

        <div id=”myDiv2”>
        <h2>Capturing:</h2>
        <p id=”para2”>Click here!</p>
        </div>
        <script>
        document.getElementById(“para1”).addEventListener(“cl
ick”, function() {
        alert(“You clicked the blue element!”);
        }, false);

        document.getElementById(“myDiv1”).addEventListener(“c
lick”, function() {
        alert(“You clicked the yellow element!”);
        }, false);

        document.getElementById(“para2”).addEventListener(“cl
ick”, function() {
        alert(“You clicked the blue element!”);
        }, true);
        document.getElementById(“myDiv2”).addEventListener(“c
lick”, function() {
        alert(“You clicked the yellow element!”);
        }, true);
        </script>
    </body>
</html>
```

READ

Page: Introduction to events by MDN Web Docs.

WATCH

Video: Event objects (11m 54s) by Christian H. on LinkedIn Learning.

ACTIVITY

Create a website containing one button. After a button click, we should get a “Hello World” alert, and the button should change its text to “Clicked.”

Source code: Click here to reveal.

DOM CSS

We can modify all CSS properties using DOM and its style object. To do this, we need to modify the proper CSS property in the style object. Moreover, we can do this at the events too. Let’s look at an example of changing the heading’s background on the button click:

```
<!DOCTYPE html>
<html>
    <body>
    <h1 id="id1">My Heading 1</h1>
    <button type="button"
onclick="document.getElementById('id1').style.color = 'red'">
        Click Me!
    </button>
    </body>
</html>
```

CSS tricks dynamically! Let's see another example of changing the text's visibility on the button click:

```
<!DOCTYPE html>
<html>
    <body>
        <p id="p1">
        This is a text.
        This is a text.
        This is a text.
        This is a text.
        </p>
        <input type="button" value="Hide text"
onclick="document.getElementById('p1').style.visibility='hidd
en'">
        <input type="button" value="Show text"
onclick="document.getElementById('p1').style.visibility='visi
ble'">
    </body>
</html>
```

We can read much more about the style object in the READ section.

READ

Page: HTMLElement.style by MDN Web Docs.

WATCH

Video: CSS properties and the DOM (13m 50s) by Christian H. on LinkedIn Learning.

ACTIVITY

Create a website containing one button. After a button click, we should get a “Hello World” alert, and the button should change its background to green.

Source code: Click here to reveal.

DOM animations

We have already learned how to create various animations with transforms and transitions, even though they aren’t always displayed correctly in older versions of the browsers. We can create animations more ‘manually’ using HTML DOM.

Let’s look at an example of a simple animation:

```
<!DOCTYPE html>
    <html>
        <style>
            #container {
                width: 400px;
                height: 400px;
                position: relative;
                background: yellow;
            }
            #animate {
                width: 50px;
                height: 50px;
                position: absolute;
                background-color: red;
            }
        </style>
    <body>
        <p><button onclick="myMove()">Click Me</button></p>
        <div id ="container">
            <div id ="animate"></div>
        </div>
        <script>
            function myMove() {
                let id = null;
                const elem = 
document.getElementById("animate");
                let pos = 0;
                clearInterval(id);
                id = setInterval(frame, 5000);
                function frame() {
                    if (pos == 350) {
                        clearInterval(id);
                    } else {
                        pos++;
                        elem.style.top = pos + "px";
                        elem.style.left = pos + "px";
                    }
            }
        }
        </script>
    </body>
</html>
```

To set our element's coordinates, we need our container to have a relative position, while our animation has an absolute one. We find our animate element with DOM and then refresh the frame every five seconds. Every new frame is one pixel further from the previous one.

	WATCH Video section: 2. JavaScript Animation Foundations (18m 40s) by Joseph Labrecque on LinkedIn Learning.

DOM form validation

The form is the set of controls for the <form> element's children. Clicking the form's submit button triggers the submit event. Usually, we want this event to trigger our validation functions. Validation is the process of checking whether our data is correct. For example, if necessary, the fields are filled, and their values are in the proper format.

Let's look at a simple example of the form and its validation:

```
<!DOCTYPE html>
<html>
    <head>
        <script>
        function validateForm() {
            let x = document.forms["myForm"]["fname"].value;
            if (x == "") {
                alert("Name must be filled out");
                return false;
            }
            return true;
        }
        </script>
    </head>
    <body>
        <h2>JavaScript Validation</h2>
        <form name="myForm" onsubmit="return validateForm()"
method="post">
            Name: <input type="text" name="fname">
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

We can access our form and controls from the DOM and its forms collection. In this case, we check whether our field named ‘fname’ is empty or not. If it is, we’ll show the alert about the data not being filled instead of submitting the form.

Typical validation tasks are, for example:

- Has the user filled in all the required fields?
- Is the date the user provided valid?
- Has the user entered a text in a numeric field?

There are two types of data validation:

1. Client-side - as presented, performed by a web browser before sending data to the server.
2. Server side - performed after submitting the form and sending the data by a web server.

HTML5 introduces a new client-side validation concept called constraint validation. This is based on:

- HTML Input Attributes
- CSS Pseudo Classes
- DOM Properties and Methods

You can read more about these in the READ section.

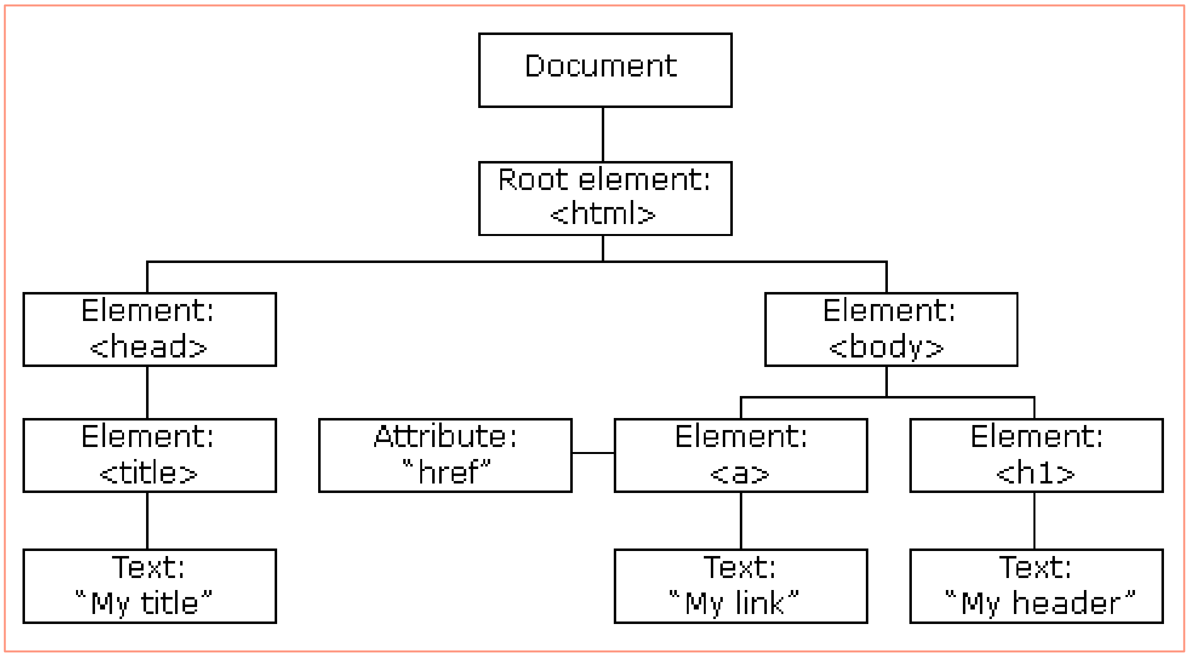
	READ 1. Page: Document.forms by MDN Web Docs. 2. Page: <input>: The Input (Form Input) element by MDN Web Docs.

	WATCH Video: Working with named form elements (3m 39s) by Ray Villalobos on LinkedIn Learning.

	ACTIVITY Check whether the value of the control in our example is longer than 10 characters. If it is, show a proper alert. **Source code:** Click here to reveal.

DOM nodes

Let's remind ourselves of the tree structure of the DOM:



We can see that each element and its attribute is a node within this tree. Knowing this structure, we can insert new elements at the selected position.

Firstly, we need to select a node and then decide whether we want a new element as the parent or the child of the selected one. We need to create a new element, for example, using a document.createElement() method.

If we want to insert it as a child, we should use the selectedElement.appendChild(newElement) method.

If we want to add it as the child of a specified parent, use selectedElement.insertBefore(newElement, childElement).

If we do not specify the childElement then newElement is inserted at the end of selectedElement's child nodes.

For example: selectedElement.insertBefore(newElement).

Let’s look at this example:

```
<!DOCTYPE html>
    <html>
    <body>
        <h2>JavaScript HTML DOM</h2>
        <p>Add a new HTML Element.</p>
        <div id="div1">
            <p id="p1">This is a paragraph.</p>
            <p id="p2">This is another paragraph.</p>
        </div>

        <script>
            const para = document.createElement("p");
            const node = document.createTextNode("This is
new.");
            para.appendChild(node);

            const element = document.getElementById("div1");
            const child = document.getElementById("p1");
            element.insertBefore(para,child);
        </script>
    </body>
</html>
```

We create a new paragraph and append a text of it as a child (because the text is always inside the paragraph, not the other way around). We then use the insertBefore method to add our new paragraph before the first child (p1) of our parent div (div1).

However, we can also use DOM nodes to remove or replace existing elements. This is less complicated because we already know the place we want to remove from or replace.

To do this, we need to use the parent.removeChild(child) method to remove a child or the parent.replaceChild(newElement, child) to replace a child with the new element.

READ

Page: Node by MDN Web Docs.

WATCH

Video: The Node interface (10m 50s) by Christian H. on LinkedIn Learning.

DOM collections

So far, we saw that methods like getElementsByTagName() return the array of all elements, but this isn't necessarily the truth. It's not the array but an array-like list (collection) of HTML elements. We can't use array methods like valueOf(), pop(), push() or join() on it.

Depending on the method and the browser's version, there are two possible collections: HTMLCollection and NodeList. They are pretty much the same thing, but HTMLCollections store elements, while NodeList - nodes. HTMLCollection items can be accessed by their name, ID or index number. NodeList items can only be accessed by index number.

Moreover, HTML collection is a live collection, while NodeList is a static one. If we add elements in the DOM, HTMLCollection will change, but NodeList won't.

READ

1. Page: HTMLCollection by MDN Web Docs.
2. Page: NodeList by MDN Web Docs.

WATCH

Video: Access elements using older methods (3m 45s) by Morten Rand-Hendriksen on LinkedIn Learning.

What did I learn in this lesson?

This lesson provided the following insights:

- Events are and how to use them.
- How to create and validate a form.
- What DOM nodes are, how to edit them, and why they are different from elements.

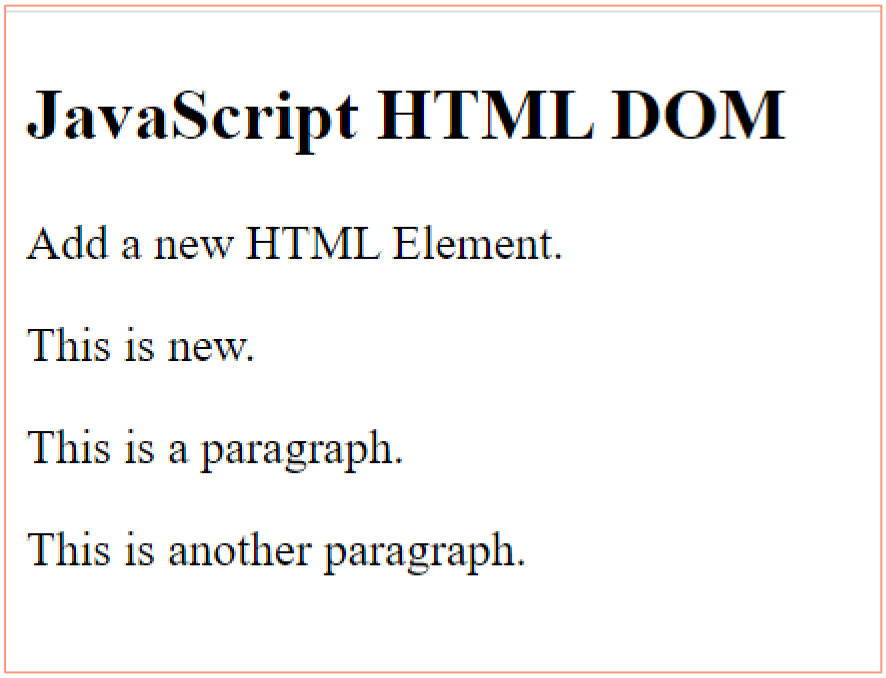
References

MDN Web Docs (2022) *Web technology for developers*. Available at: https://developer.mozilla.org/en-US/docs/Web

2.2. Lesson task - HTML Document Object Model (part 2)

The task

In this lesson, we learnt about more HTML DOM elements, such as CSS, animations, form validation and collections. In this task, you need to prepare a website similar to this one:



JavaScript HTML DOM

Add a new HTML Element.

This is new.

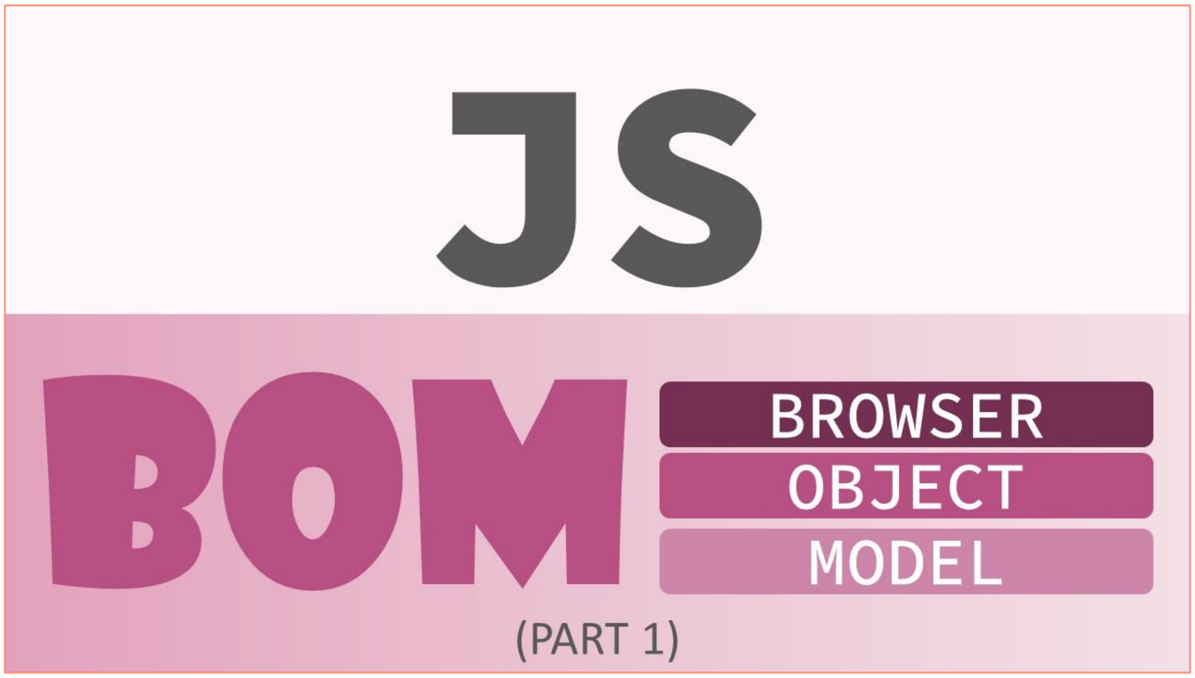
This is a paragraph.

This is another paragraph.

This is the same web page as generated in the nodes section. The difference is that you need to insert all HTML elements as nodes. In this task, you can't use any HTML code in the body element besides the script element. You should achieve this site by only adding HTML elements as nodes in the correct order.

2.3. Lesson - JS Browser Object Model (part 1)

Introduction



We already know how to interact with our HTML code when using HTML DOM. In this lesson, we'll learn about a new possibility: interacting with our browser. The Browser Object Model (BOM) provides us with the objects that expose the web browser's functionality.

In this lesson, we'll focus on the window, screen, location, history and navigator objects. We'll see how to create various interactions with the user and gather fascinating user data.

Learning outcomes

In this lesson, we are covering the following knowledge learning outcome:

→ The candidate has knowledge of concepts of the Document Object Model (DOM).

In this lesson, we are covering the following skill learning outcome:

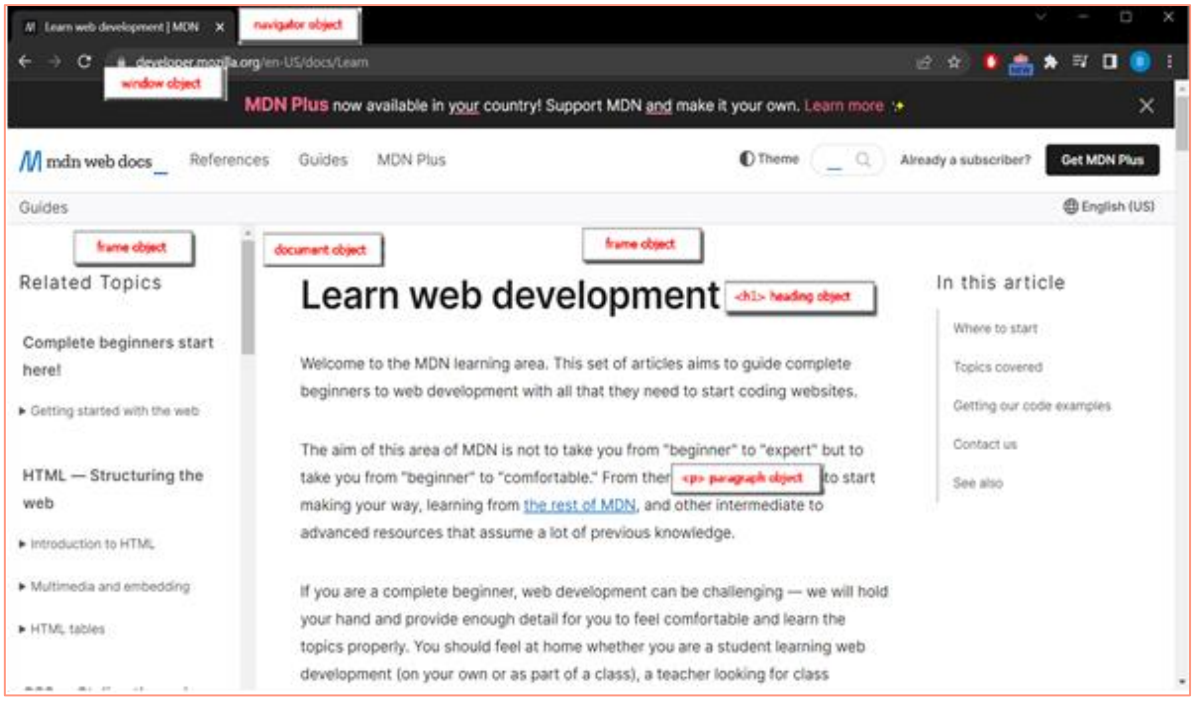
→ The candidate can apply vocational knowledge of JavaScript to manipulate the Document Object Model (DOM) and retrieve and display data from existing REST services in static web pages.

In this lesson, we are covering the following general competence learning outcome:

→ The candidate can carry out work based on the needs of a junior JavaScript developer.

Introduction to Browser Object Model

The Browser Object Model (BOM) is a hierarchy of browser objects that can be used to manipulate the methods and properties associated with the web browser itself. We can see its hierarchy below:



MDN Plus now available in your country! Support MDN and make it your own. Learn more

Learn web development

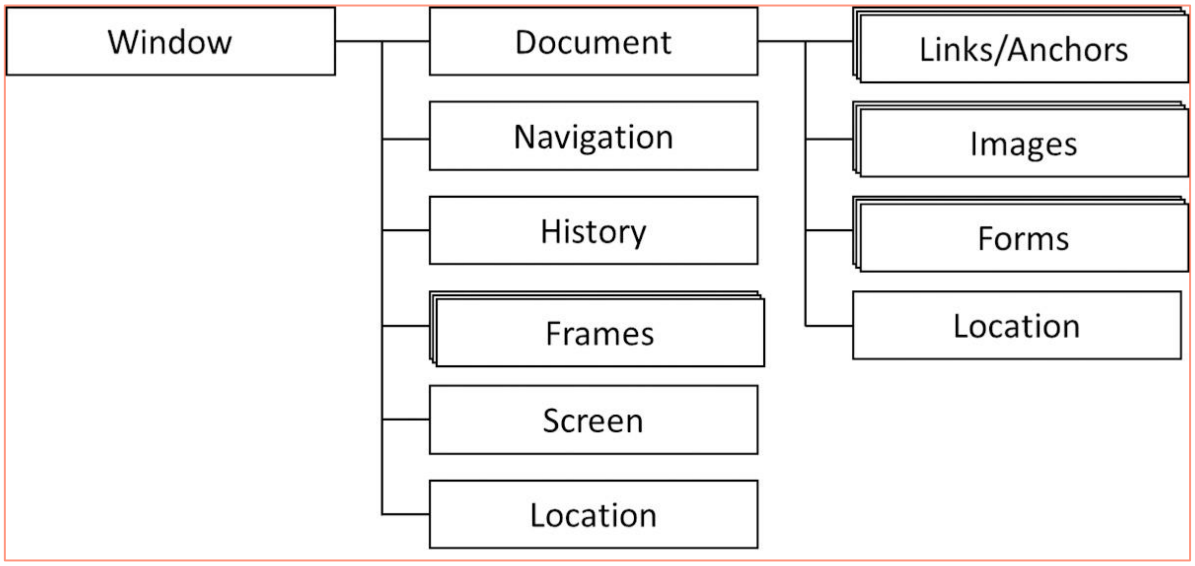
Welcome to the MDN learning area. This set of articles aims to guide complete beginners to web development with all that they need to start coding websites.

The aim of this area of MDN is not to take you from "beginner" to "expert" but to take you from "beginner" to "comfortable." From ther to start making your way, learning from the rest of MDN, and other intermediate to advanced resources that assume a lot of previous knowledge.

If you are a complete beginner, web development can be challenging — we will hold your hand and provide enough detail for you to feel comfortable and learn the topics properly. You should feel at home whether you are a student learning web development (on your own or as part of a class), a teacher looking for class

The window is the main object and represents the browser's window. All other objects are placed within it. The document object is the web page displayed in the browser. We already know of this object from the DOM lessons. The other objects contain information about the user's screen and data.

The diagram below is a representation of the hierarchy of browser objects.



Window

The window object is the main object of BOM and represents the browser's window. All browsers support this. All global JavaScript objects, functions and variables are members of the window object by default. It has two properties determining size: innerHeight and innerWidth.

Both values are in pixels and don't include toolbars and scrollbars existing on our website. We should treat these properties as read-only. Changing them doesn't affect our browser's size. To change this size, we need to use a window.resizeTo() method.

Let's look at an example of creating a new window (with the window.open() method) and resizing it to a specific value:

```
<!DOCTYPE html>
<html>
    <body>
        <h1>The Window Object</h1>
        <h2>The resizeBy() Method</h2>
        <p>Open a new window, and resize it to 300 x 300:</p>
        <button onclick="openWin()">Create window</button>
        <button onclick="resizeWin()">Resize window</button>
        <script>
            let myWindow;
            function openWin() {
                myWindow = window.open("", "", "width=200,
height=100");
            }
            function resizeWin() {
                myWindow.resizeTo(300, 300);
            }
        </script>
    </body>
</html>
```

The open method creates a new window and has the following syntax:

```
window.open(URL, name, specs, replace)
```

These parameters stand for:

- URL – optional parameter, the URL of the page to open. If not specified, a blank window/tab opens.
- Name – optional parameter, can have the following values:
  1. _blank – URL is loaded in a new tab/window, the default value.
  2. _parent – URL is loaded into the parent frame.
  3. _self – URL replaces the current page.
  4. _top – URL replaces any frameset that may be loaded.
  5. *name* – the name of the window.
- specs – optional, comma-separated list of window details, such as width, height.
- replace – optional and deprecated Boolean used to decide whether to replace the current entry in the history list.

We can set the new window's URL, name and attributes such as its size.

It returns a new window object. We can save it to a variable and close it with the close() method. This way, we can close the windows generated by JavaScript.

We can also move created windows with the moveTo() method, which sets the window's top left coordinates. There are also analogous methods, moveBy() and resizeBy(), which modify the current ones instead of setting new coordinates.

Let's look at a simple example of the moveTo and moveBy method:

```
<!DOCTYPE html>
<html>
    <body>
        <h1>The Window Object</h1>
        <h2>The moveTo() and moveBy() Methods</h2>
        <p><button onclick="openWin()">Create a
window</button></p>
        <p><button onclick="moveWinTo()">Move the window to
150 150</button></p>
        <p><button onclick="moveWinBy()">Move the window by
75 75</button></p>
        <script>
```

```
            let myWindow;
            function openWin() {
                myWindow = window.open("", "", "width=400,
height=400");
            }
            function moveWinTo() {
                myWindow.moveTo(1, 999);
            }
            function moveWinBy() {
                myWindow.moveBy(75, 75);
            }
        </script>
    </body>
</html>
```

READ

Page: Window by MDN Web Docs.

WATCH

Video: Window object (6m 56s) by Christian H. on LinkedIn Learning.

ACTIVITY

With the use of BOM, create a new window. After five seconds, resize it. After another five seconds, move it to the other place. After another five seconds, close it.

Source code: Click here to reveal.

Screen

The window.screen object contains information about the user's screen. It has the following properties:

- width
- height
- availWidth
- availHeight
- colorDepth
- pixelDepth

All the properties above are read-only. Changing them in code won't affect the user's screen.

The width and height properties return the current size of the screen. They are read-only. Attempting to change them won't affect the screen size.

The availWidth and availHeight properties return the amount of space in pixels available to the window.

The pixelDepth and colorDepth properties return the pixel and colour depth of the screen.

READ

Page: Screen by MDN Web Docs.

Location

We can use the window.location object to access the current page's address (URL) or redirect the browser to the new page.

The location.href property can either get or set our URL to the new one.

The location.hostname returns the name of the Internet host of the current page.

The location.pathname returns the path name of the current page.

The location.protocol returns the web protocol of the page.

The location.port returns the number of the Internet host port of the current page.

The location.assign() method loads a new document.

We can see the use of all of these in an example below:

```
<!DOCTYPE html>
<html>
    <body>
        <h2>JavaScript</h2>
        <h3>The window.location object</h3>
        <p id="p1"></p>
        <p id="p2"></p>
        <p id="p3"></p>
        <p id="p4"></p>
        <p id="p5"></p>
        <input type="button" value="Load new document"
onclick="newDoc()">
        <script>
```

```
            document.getElementById("p1").innerHTML =
            "Page location is " + window.location.href;
            document.getElementById("p2").innerHTML =
            "Page hostname is " + window.location.hostname;
            document.getElementById("p3").innerHTML =
            "Page path is " + window.location.pathname;
            document.getElementById("p4").innerHTML =
            "Page protocol is " + window.location.protocol;
            document.getElementById("p5").innerHTML =
            "Port number is " + window.location.port;
            function newDoc() {
                window.location.assign("https://www.google.co
m")
            }
        </script>
    </body>
</html>
```

Since our websites aren't hosted on any servers; hostname, protocol and port are empty.

	READ Page: Location by MDN Web Docs.

	ACTIVITY Create the button that sets the current URL to https://google.com. **Source code:** Click here to reveal.

History

The history object stores the information about the user’s browser history. However, to protect the users’ privacy, we only have access to some properties and methods:

- back() method – will go back one page earlier in the user’s history.
- forward() method – will go forward one page in the user’s history.
- length property – number of pages in the user’s history.
- go() method – navigates forward or backward for a specified number of pages. A negative number means navigating backwards.

Let’s look at the code example of these:

```
<!DOCTYPE html>
<html>
    <body>
        <h1>The Window History Object</h1>
        <button onclick="history.go(-2)">Go 2 pages
back</button>
        <button onclick="history.back()">Go back</button>
        <button onclick="history.forward()">Go next</button>
    </body>
</html>
```

We have three buttons that demonstrate how these methods work. Note that we could use history.go(-1) instead of history.back().

READ

Page: History by MDN Web Docs.

	ACTIVITY Create a button that navigates the user to five sites backwards in their history. If there are less than five sites, navigate by max possible amount. **Source code:** Click here to reveal.

Navigator

The window.navigator object stores information about the user's browser. Remember the notifications to accept cookies before using a website? Websites access this information in precisely the same way as from this object.

Navigator has the following properties:

- navigator.cookieEnabled – returns the Boolean, true is cookies are enabled.
- navigator.appName – returns the browser name.
- navigator.appCodeName – returns the browser code name.
- navigator.product – returns the browser engine name.
- navigator.appVersion – returns the browser version.
- navigator.userAgent – returns browser user-agent header.
- navigator.platform – returns browser platform.
- navigator.language – returns browser language.
- navigator.online – returns the Boolean. True if the browser is online.
- navigator.geolocation – Returns the geo-location object of the user's location.

And methods:

- navigator.javaEnabled() – returns the Boolean. True if Java is enabled.

Although it might look like we have a lot of information, this isn't always true, especially when detecting the browser's version. Different browsers can use the same name. The browser owner can change the navigator data. Some browsers misidentify themselves to bypass site tests. Also, browsers can't report new operating systems released later than the browser.

READ

Page: Window.navigator by MDN Web Docs.

What did I learn in this lesson?

This lesson provided the following insights:

- The Browser Object Model (BOM) is and how to use it.
- How to get information about users' windows.
- How to access the user's data.

References

MDN Web Docs (2022) *Web technology for developers*. Available at: https://developer.mozilla.org/en-US/docs/Web

2.3. Lesson task - JS Browser Object Model (part 1)

The task

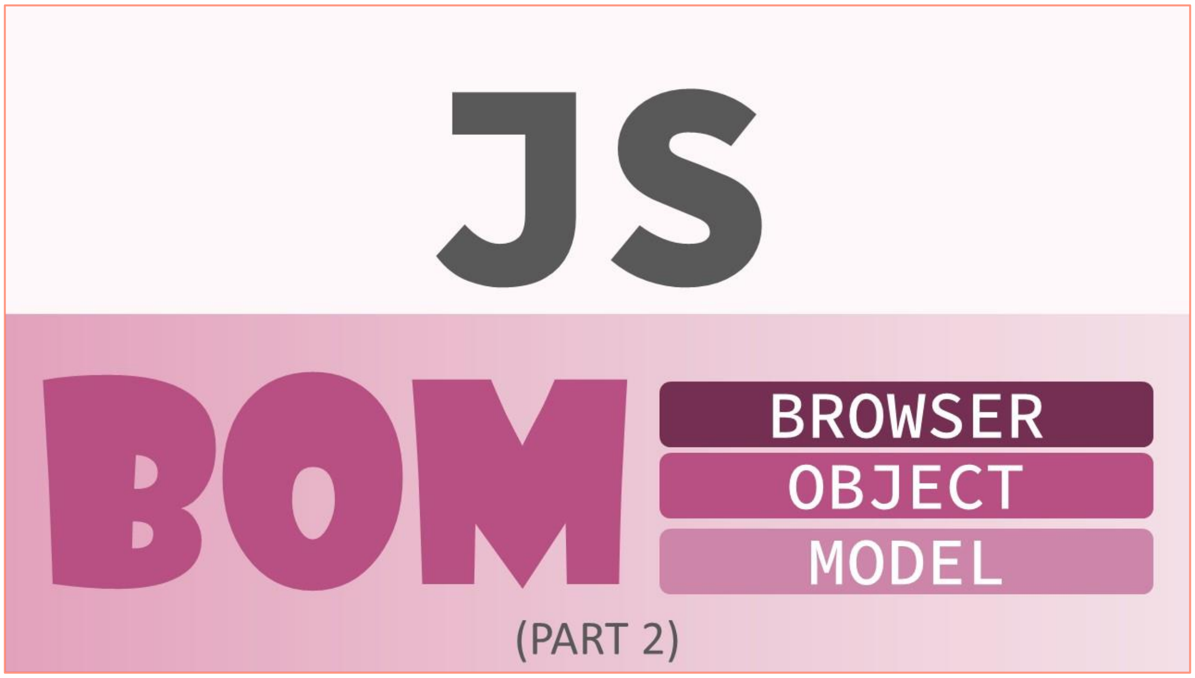
In this lesson, we learned about the basics of the JS Browser Object Model, such as window, screen, location or history. In this task, you need to create a website containing:

- A button that creates a new window of width and height equal to 800px.
- A button that makes the last window created two times smaller.
- A button that shows an alert with the current URL after five seconds of clicking.
- Back and next buttons (moving to previous/next card from the history).

Note that when trying to display a URL, nothing will be displayed. This is because your page is local. If you paste this code into another browser's console that has a live web page open, it will work correctly and display the URL of that web page.

2.4. Lesson - JS Browser Object Model (part 2)

Introduction



We already know the majority of BOM elements. In this lesson, we'll recap alerts and timers. There's also a brand-new topic – cookies.

We learned a bit about alerts and timing and then used them as the black box – we learned what command we should call to create a black box but didn't know anything about the structure behind it. With the knowledge about HTML DOM and BOM, we now understand how they work.

The last topic is cookies. Cookies are the data stored in small text files within the user's computer. Usage of them allows the website to gain legible information about the user.

Learning outcomes

In this lesson, we are covering the following knowledge learning outcome:

→ The candidate has knowledge of concepts of the Document Object Model (DOM).

In this lesson, we are covering the following skill learning outcome:

→ The candidate can apply vocational knowledge of JavaScript to manipulate the Document Object Model (DOM) and retrieve and display data from existing REST services in static web pages.

In this lesson, we are covering the following general competence learning outcome:

→ The candidate can carry out work based on the needs of a junior JavaScript developer.

Alerts

The window object provides us with three popups: alerts, prompts, and confirm boxes. We can use these to show essential data or notifications, for example, after some events or changes in data. They can also be used as a clear way to display other HTML DOM or BOM elements.

Let's look at an example of displaying the website's hostname within the alert:

```
<!DOCTYPE html>
<html>
    <body>
        <h1>The Window Object</h1>
        <h2>The alert() Method</h2>
        <p>Click the button to alert the hostname of the 
current URL.</p>
        <button onclick="myFunction()">Try it</button>
        <script>
            function myFunction() {
              alert(location.hostname);
            }
        </script>
    </body>
</html>
```

Alerts are also an excellent tool to show information from cookies. We'll learn about this later. On the other hand, we shouldn't overuse alerts. They prevent users from accessing other parts of the page until the alert box is closed, which might result in a bad user experience.

Default alerts are helpful, but we can get more attractive and eye-catching results with external libraries. One of the most popular ones is sweetalert2.

To use this library, you need to download a file from: https://cdn.jsdelivr.net/npm/sweetalert2@11 and put it in our folder at the same level as other JS files.

To download this file, click the link and open the webpage. You will be shown a mass of unreadable code. This is the code for the library.

Right-click -> Save As -> Save your JS library file in your project folder.

Don't forget to link to this JS Library in your <script> tag, in your HTML file.

You can now use this library but access it through the <script> tag in your HTML file.

Instead of the alert() command, we need to use the Swal.fire() command.

Let's look at a simple example. We can find more complex examples in the documentation:

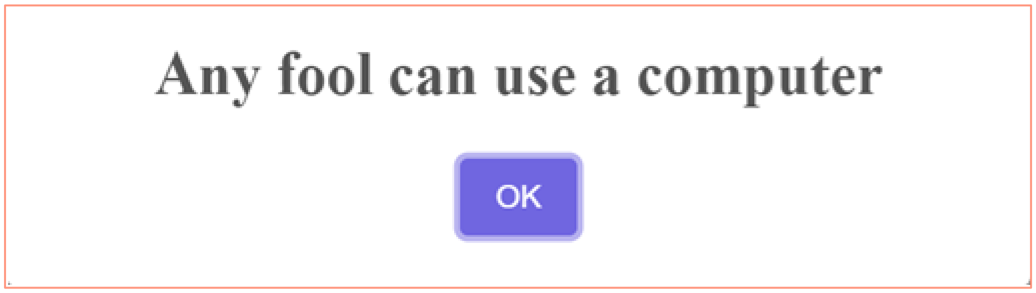
HTML code:

```
<!DOCTYPE html>
<html>
  <head>
        <title>App</title>
        <script src="l1m1.js"></script>
        <script src="sweetalert2@11.js"></script>
    </head>
    <body>
        <button onclick="alertMe()">Update details</button>
    </body>
</html>
```

JS code:

```
function alertMe() {
    Swal.fire('Any fool can use a computer')
}
```

After clicking the button, we get the alert:


Any fool can use a computer

Note that our new alert has a much different style than the standard one. Due to this, it's more visible, and we pay more attention to it.

READ

1. Page: Window.alert() by MDN Web Docs.
2. Page: Window.prompt() by MDN Web Docs.
3. Website: https://sweetalert2.github.io/.

ACTIVITY

Open any website within a browser and, using the browser’s console, show an alert with the website’s hostname.

Source code: Click here to reveal.

Confirm box

A confirm box is an alert box with one more button – the cancel option. It’s used more or less in the same situation, but this time, we need some input from the user. Let’s recap a simple example of it:

```
<!DOCTYPE html>
<html>
    <body>
        <h2>JavaScript Confirm Box</h2>
        <button onclick="myFunction()">Try it</button>
        <p id="demo"></p>
        <script>
            function myFunction() {
                var txt;
                if (confirm("Press a button!")) {
                    txt = "You pressed OK!";
                } else {
                    txt = "You pressed Cancel!";
                }
                document.getElementById("demo").innerHTML =
txt;
            }
        </script>
    </body>
</html>
```

Confirm returns a Boolean, so we end up in a proper code block of the user's choice. There are also prompts used to get data from the user, just like in the example below:

```
<!DOCTYPE html>
<html>
    <body>
        <h2>JavaScript Prompt</h2>
        <button onclick="myFunction()">Try it</button>
        <p id="demo"></p>
        <script>
            function myFunction() {
                let text;
                let person = prompt("Please enter your
name:", "Harry Potter");
                if (person == null || person == "") {
                    text = "User cancelled the prompt.";
                } else {
                    text = "Hello " + person + "! How are you
today?";
                }
                document.getElementById("demo").innerHTML =
text;
            }
        </script>
    </body>
</html>
```

READ

Page: Window.confirm() by MDN Web Docs.

Timing

We need to treat the timers as a special event – an event that is executed after the specified time or every provided period. We can execute our code at specified time intervals with the window object.

There are two key methods to use with JavaScript:

- setTimeout() – executes a function after a specified time.
- setInterval() – same as setTimeout, but the function's execution repeats continuously.

We can stop this with the window.clearTimeout() or window.clearInterval() methods.

Let's look at an example of timers by building a clock that refreshes every second:

```
<!DOCTYPE html>
<html>
    <body onload="startTime()">
    <h2>JavaScript Clock</h2>
    <div id="txt"></div>
        <script>
            function startTime() {
                const today = new Date();
                let h = today.getHours();
                let m = today.getMinutes();
                let s = today.getSeconds();
                m = checkTime(m);
                s = checkTime(s);
                document.getElementById('txt').innerHTML =  h
+ ":" + m + ":" + s;
                setTimeout(startTime, 1000);
            }
            function checkTime(i) {
                if (i < 10) {i = "0" + i};
                return i;
            }
        </script>
    </body>
</html>
```

	READ 1. Page: setTimeout() by MDN Web Docs. 2. Page: setInterval() by MDN Web Docs.

	WATCH Video: Incorporate setInterval (3m 11s) by Alex Banks on LinkedIn Learning.

Introduction to cookies

Cookies are small text files in the user's computer storing information about them. When the web server sends a web page to a browser, the connection is shut down and no longer has access to any user data. Cookies are the solution to this problem.

When a user visits a web page for the first time, their name could be stored in a cookie. Next time they visit the site, the cookie will remember their name. We can create a cookie in a similar way to the below:

```
document.cookie = "username=John Doe; expires=Thu, 18 Dec
2013 12:00:00 UTC; path=/";
```

We need to set the document.cookie property. All cookies are stored in one text variable. All cookies need to have their names set. In this case, our cookie has 'username', but we could have cookies with other names. Our cookie is set to 'John Doe'.

To delete the cookie, we just need to set its expiration date to a date that has already passed.

The document.cookie property looks like a simple string but isn't. If we set a new cookie, older cookies aren't overwritten. The new cookie is added to the document.cookie object. We need the proper JavaScript functions to get and set cookies that search for the cookie in the cookie string.

Note that all cookies need to have their domain. We can't save them on our local examples. We'll learn how to use them on our websites later in the course. For now, you need to open any significant website and paste this code into the browser's console.

READ

Page: Document.cookie by MDN Web Docs.

WATCH

Video: Understanding cookies and sessions (3m 54s) by Daniel Khan on LinkedIn Learning.

ACTIVITY

Create a cookie with your name (or dummy data).

Source code: Click here to reveal.

Getting cookies

We can use the function similar to the below to access the cookies:

```
function getCookie(cname) {
    let name = cname + "=";
    let decodedCookie = decodeURIComponent(document.cookie);
    let ca = decodedCookie.split(';');
    for(let i = 0; i <ca.length; i++) {
      let c = ca[i];
      while (c.charAt(0) == ' ') {
        c = c.substring(1);
      }
      if (c.indexOf(name) == 0) {
        return c.substring(name.length, c.length);
      }
    }
    return "";
}
```

String operations are beneficial while working on cookies.

Sometimes, our cookies contain no English characters. For example, if we ask a Russian for their username, it will probably be provided in Cyrillic. Because of this, we decode our cookies at the beginning of the function.

Special characters in cookies are coded like Uniform Resource Identifier (URI) components. The question mark is coded as ‘%3F’, for example. You can check the entire list of how special characters are coded in URI or how to code the particular character in the READ section.

Using the string methods, we can then split our cookie by semicolons to get our data. We then iterate on it and return the result.

READ

1. Page: decodeURIComponent() by MDN Web Docs.
2. Page: HTML URL-encoding Reference by ESO.
3. Page: URL Encode and Decode Tool by Dan's Tools.

Setting cookies

We need a function that stores the name of our cookie in a cookie variable:

```
function setCookie(cname, cvalue, exdays) {
    const d = new Date();
    d.setTime(d.getTime() + (exdays*24*60*60*1000));
    let expires = "expires="+ d.toUTCString();
    document.cookie = cname + "=" + cvalue + ";" + expires +
";path=/";
}
```

Cname is the cookie name, cvalue is the value of the cookie, and exdays is the expiration date. Our function sets the cookie by merging all these values into one string. We then add our cookies to all cookies by assigning them to document.cookie.

We can see this in the final example of creating a cookie on the first-page loading and access this data with every next loading:

```
<!DOCTYPE html>
<html>
    <head>
        <script>
            function setCookie(cname,cvalue,exdays) {
                    const d = new Date();
                    d.setTime(d.getTime() +
(exdays*24*60*60*1000));
                    let expires = "expires=" +
d.toUTCString();
                    document.cookie = cname + "=" + cvalue +
";" + expires + ";path=/";
                }
                function getCookie(cname) {
                    let name = cname + "=";
                    let decodedCookie =
decodeURIComponent(document.cookie);
                    let ca = decodedCookie.split(';');
                    for(let i = 0; i < ca.length; i++) {
                        let c = ca[i];
                        while (c.charAt(0) == ' ') {
                            c = c.substring(1);
                        }
                        if (c.indexOf(name) == 0) {
                            return c.substring(name.length,
c.length);
                        }
                    }
                    return "";
                }

                function checkCookie() {
                    let user = getCookie("username");
                    if (user != "") {
                        alert("Welcome again " + user);
                    } else {
                        user = prompt("Please enter your
name:","");
                        if (user != "" && user != null) {
                            setCookie("username", user, 30);
                        }
                    }
                }
        </script>
    </head>
    <body onload="checkCookie()"></body>
</html>
```

READ

1. Page: Cookies by QuirksMode.
2. Page: Cookies, document.cookie by javascript.info.

ACTIVITY

Open any significant website and open the browser's console. Copy all three functions and paste them into the console. Run the checkCookie() method twice and then log the document.cookie object.

What did I learn in this lesson?

This lesson provided the following insights:

- Pop-ups and timings and how they are connected to BOM.
- How and why to create cookies.
- How to get and set cookies.

References

MDN Web Docs (2022) *Web technology for developers*. Available at: https://developer.mozilla.org/en-US/docs/Web

2.4. Lesson task - JS Browser Object Model (part 2)

The task

In this lesson, we learnt about pop-ups, timings and cookies. You must now create a program that will create a cookie and change its value to a random number from 0 to 10 every 10 seconds.

Remember, our local programs can't use cookies. You can check whether the code works properly by entering any website using cookies (for example Mozilla Developer - MDN Web Docs) and pasting your code into the browser's console.

2.5. Lesson - Self-study

Introduction

This is a self-study lesson which consolidates knowledge of the second module. You'll have the chance to recap HTML DOM and solve some exciting exercises connected to it.

Materials

	WATCH 1. Video: What is the Document Object Model (DOM)? (3m 2s) by Ray Villalobos on LinkedIn Learning. 2. Video: Selecting elements with getElementById (4m 10s) by Ray Villalobos on LinkedIn Learning. 3. Video: Developer tools (7m 12s) by Christian H. on LinkedIn Learning. 4. Video: The Document methods (10m 27s) by Christian H. on LinkedIn Learning. 5. Video: Select elements with getElementsByClassName (9m 41s) by Christian H. on LinkedIn Learning.

2.5. Lesson task - Self-study

The task

In this task, you need to complete multiple simple JavaScript DOM tasks.

1. Create a text paragraph. Create a button that changes the paragraph's font size to 16px, font family to Comic Sans MS and font colour to blue.

   Answer: Click here to reveal.

2. Create two text paragraphs. Without giving them either ID or class, create a button that changes the font colour of the first one to red and the second one to green.

   Answer: Click here to reveal.

3. Create a link and set its Href, target and type attributes. Create a button showing the value of these attributes in an alert.

   Answer: Click here to reveal.

4. Create a table containing two columns: Name and Surname. Create a button that adds a new row at the beginning, with values ‘New Name’ as the name and ‘New Surname’ as the surname.

   Answer: Click here to reveal the HTML code.

   Answer: Click here to reveal the JS code.

5. Create a table containing two columns: Name and Surname, and three rows of data. Create a button that changes the text of the chosen cell after a click. The user should specify the text and cell index via prompt.

   Answer: Click here to reveal HTML code.

   Answer: Click here to reveal the JS code.

6. Create a button that creates the table of the provided size after a click. Get information about the size from the user via prompts. After creating the table, hide the button.

   Answer: Click here to reveal the HTML code.

   Answer: Click here to reveal the JS code.

7. Create a dropdown list and add a button that removes the selected element.

   Answer: Click here to reveal.

8. After loading, print in the current window size in the console. Also, print it whenever the size changes.

   Answer: Click here to reveal.

9. Create an image of Pikachu. Create two buttons. The first one will change the currently displayed image to Bulbasaur. The second will change it back to Pikachu. The Pikachu image should have width=240px

and height=200px, while the Bulbasaur image width=235px and height=220px.

Answer: Click here to reveal the HTML code.

Answer: Click here to reveal the JS code.

10. Create a text paragraph and make some words italic inside of it. Create an image of Pikachu. Highlight all the italic words with blue when the mouse cursor is on Pikachu. Remove the highlight when it isn't. Height=220px.

Answer: Click here to reveal the HTML code.

Answer: Click here to reveal the JS code.

Noroff
Education