LAB 13 (WEEK 46)

JavaScript

DOM

Table of Contents

Introduction

In this lab we are going to play with the Document Object Model. The examples are focused on tasks that can be reused (with some modifications) in the 3rd coursework. Remember that JavaScript is a programming language[1] used to handle the dynamic nature of contents as well as the interaction between the browser and the user.
With the Document Object Model, JavaScript gets all the power it needs to create dynamic HTML. JavaScript can:

- Change all the HTML elements in the page.
- Change all the HTML attributes in the page.
- Change all the CSS styles in the page.
- Remove existing HTML elements and attributes.
- Add new HTML elements and attributes.
- React to all existing HTML events in the page.
- Create new HTML events in the page.

We will start with a small tutorial with a very simple example that manipulates the DOM to add new contents into the page. Let's go!

> ***STEP1: Create an HTML5 file (index.html). Add a title, a heading (h1, a paragraph identified by "myParagraph" and link an external ".js" file named "script.js".***

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Playing with the DOM</title>
</head>
```

[1] A programming language is a formal language that specifies a set of instructions that can be used to produce various kinds of output. Programming languages generally consist of instructions for a computer. Programming languages can be used to create programs that implement specific algorithms.

```
<body>
    <h1>My first DOM manipulation</h1>
    <p id="myParagraph"></p>
    <script src="script.js"></script>
</body>
</html>
```

Now let's add some Javascript code in our "script.js" file. It's a good moment to check this resource:
https://www.w3schools.com/jsref/met_document_getelementbyid.asp.

> ***STEP2: Use JavaScript to manipulate the DOM so that you change the content of the paragraph identified by "myParagraph" with the text "This text has been added using JavaScript".***

```
document.getElementById("myParagraph").innerHTML = "This text has been added using javascript";
```

So far, we have manipulated the DOM to change an HTML tag but, in the DOM, events are also propagated and with javascript we can handle this events. Take a look to these resources:

- https://www.w3schools.com/tags/tag_button.asp
- https://www.w3schools.com/js/js_htmldom_events.asp

> ***STEP3: Modify your html to add a button with the text "Increment". Add also another paragraph identified by "counter".***

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Playing with the DOM</title>
</head>
<body>
    <h1>My first DOM manipulation</h1>
    <p id="myParagraph"></p>
    <p id="counter"></p>
```

```
    <button type="button">Increment</button>

    <!-- (Best practice) Javascript files are loaded at the end of the body -->
    <script src="script.js"></script>
</body>
</html>
```

Notice that the new paragraph identified by “counter” is empty. Moreover, when you click on the button, nothing happens. Remember that JavaScript can react to all existing HTML events in the page. For example, when the user clicks on the button, an event is fired, and that event is propagated through the DOM.

> ***STEP4: create a function in your javascript file named “incrementCounter”. That function must increment the counter and change the text of the paragraph identified by “counter”. The text will be: “You have clicked the button X times”, where X represents the number of times the button has been pressed (Notice that this value is stored in the counter variable.***

```
document.getElementById("myParagraph").innerHTML = "This text has been added using javascript";

var counter = 0;
var counterTextPrefix = "You have clicked the button "
var counterTextSufix = " times."

function incrementCounter() {
    counter++;
    document.getElementById("counter").innerHTML = counterTextPrefix + counter + counterTextSufix;

}
```

> ***STEP5: Add the onclick function to the html button and call the incrementCounter() function.***

```
button type="button" onclick="incrementCounter()">Increment</button>
```

Now it’s your turn. Do the following exercises. But, first, check the following resources:

- Javascript length property: https://www.w3schools.com/jsref/jsref_length_string.asp
- Arrays: https://www.w3schools.com/js/js_arrays.asp
- Get element: https://www.w3schools.com/jsref/met_document_getelementbyid.asp
- InnerHtml: https://www.w3schools.com/jsref/prop_html_innerhtml.asp
- Create element: https://www.w3schools.com/jsref/met_document_createelement.asp
- Class name property: https://www.w3schools.com/jsref/prop_html_classname.asp
- Append element: https://www.w3schools.com/jsref/met_node_appendchild.asp
- For loop: https://www.w3schools.com/js/js_loop_for.asp

Exercise 1

Use JavaScript to print all the ice cream categories. ex1-show-categories.html and show-categories.js files. In this example, I haShowve created a <p> per category. So, the generated HTML code using JS looks like this:

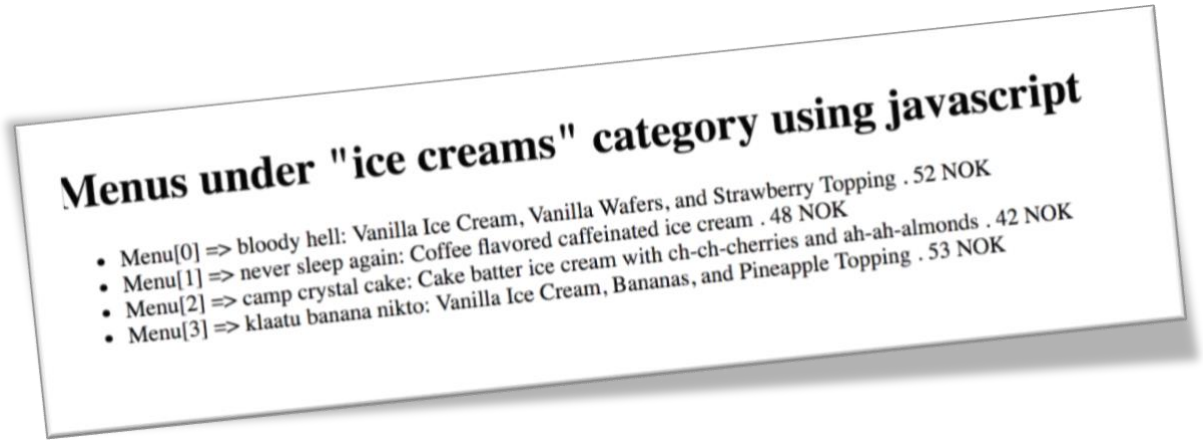
```
<h1>Categories from javascript</h1>
<p>ice creams</p>
<p>sundaes</p>
<p>mousses</p>
<p>smoothies</p>
```

And the page like this:

Exercise 2

Use JavaScript to print all the ice cream under the “ice cream” category. ex2-show-ice-creams.html and show-ice-creams.js files.

In this example, the elements are displayed using an <ul>. The final result looks like this:



Exercise 3

Now, use JS to print the first element of the list of ice creams under the "ice creams" category. This is, `iceCreams["ice creams"]]0].`

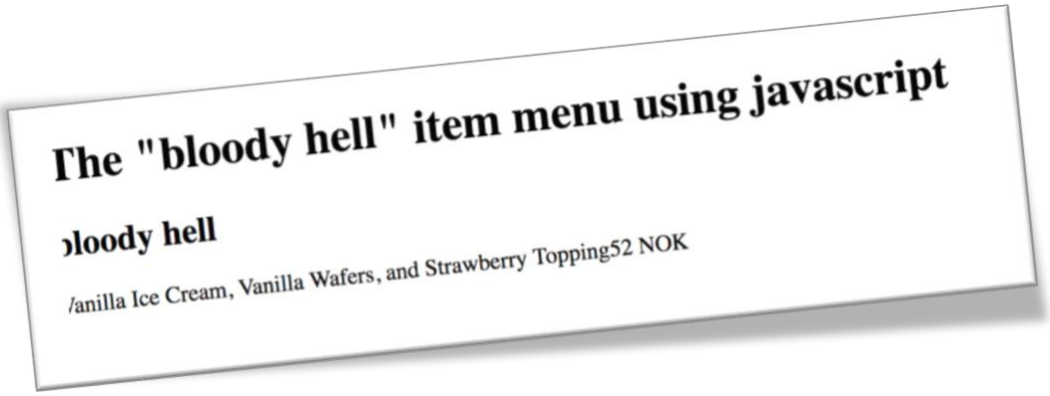
Let's make this more interesting. Using JS create the following structure

```
<div class="menu-item">
      <h2>title goes here…</h2>
      <p>
         <span>description goes here…</span>
         <span class="price">XX NOK</span>
      </p>
</div>
```

Notice that what we want to do is to create a div container with the class "menu-item". Sitting inside this container, there is going to be 2 elements, a heading (<h2> for this example) and a paragraph (<p>). The heading will contain the title of the ice cream menu (this is, "bloddy hell"). The paragraph will contain 2 span elements (). The first span will display the description of the ice cream (this is, Vanilla Ice Cream, Vanilla Wafers, and Strawberry Topping) and the second span the price of the ice cream (this is, 52 NOK). Pay attention to the class of the second span (class="price").
To attach a class to an element using the DOM you can modify the attribute "className" of the element that you are targeting. For example, the following code creates a paragraph and attach the class "custom-p" to it.

```
var p = document.createElement("p");
p.className = "custom-p";
```

The final result looks like this:



Pretty boring, right? Create a "css" file and add some rules to make the "menu-item" container looks better. Just as an example:

Exercise 4

As you can see, creating the previous elements requires a lot of JavaScript code. In fact, some of the code is duplicated again and again. For that reason, I have created for you the "helpers.js" file. The same file that you can use in your coursework 3. This file contains a function "getHTMLMenuFromTitleDescriptionPrice" that receives as input parameters a title, a description and a price and as a result returns a div element with the same structure presented in Exercise 3. Read the code, which is plenty of comments, and try to understand how it works.
After that, use the helpers to display all the ice creams under the "ice cream" category. Also use the "css" file created in the previous exercise.

The result will be similar to this:

Exercise 5

The last exercise consists in combining all the previous examples in one. Use JavaScript for go through all the categories and print all the desserts.

Notice that the function “getHTMLMenuFromTitleDescriptionPrice” implemented in the helpers, returns a div container where the title of the dessert is marked up with a h2. However, in this example, the desserts are printed right after printing their category and, the category is also using a <h2>. As you can see, this is an example of content not very well structured. Among other solutions, a way of fixing this problem could be reimplementing the code of the helpers, function “getTitleElement” (line 68) so that instead of creating a <h2> elements, the function creates a <h3>.