Web APIs summary and exercises

Web APIs are a set of browser features that we can reach from JavaScript and we can enrich our web application through them, for example we can use the Fetch API to send HTTP requests. As Web APIs are constantly extending we can always explore new features of it, but we have to be careful about what we use in our code. The main problem is that Web APIs only defines a set of features, but browsers have to implement these features in order to be able to use them in our JavaScript code. Naturally there will be differences between browsers (and most importantly between browser versions).
We can find a table at the bottom of MDN pages where they mark the browsers which can support that feature (or alternatively we can use caniuse.com).

List of available Web APIs: https://developer.mozilla.org/en-US/docs/Web/API

We will explore two Web APIs in this course the LocalStorage and the DOM API, you can find some information about them in this document.

Note that you can try these examples and solve these exercises either in a local IDE or in an online sandbox (eg.: StackBlitz — JavaScript blank project)

LocalStorage API

LocalStorage is a storage solution inside the browser. It's designed to store **key—value** pairs, where both the key and the value are strings. The advantage of this solution (compared to some variables in JavaScript) is that even if the user closes the tab (or the browser) the content of the LocalStorage will be persisted. It will be stored there as long as the user doesn't clear the browser history (or in some cases the browser can decide about the storage eviction, but that's a topic for another day).

For example if you want to welcome you users on your page you can ask their name in an input then store it in the LocalStorage. Next time he visits the page, you can read his name from the storage so you don't have to ask it again.

It's important to know that every **origin** — protocol + domain + port -> (like: https://google.com:443)— has a separate LocalStorage object, so for example if you save the username there, you don't have to worry about that another site would reach that value.

Most important storage methods: https://developer.mozilla.org/en-US/docs/Web/API/Storage#methods

- getItem - retrieve a value with a key
- setItem - store a key, value pair
- removeItem - remove the item (with the specified key) from the store
- clear - remove every item from the LocalStorage (remember: only for your origin)

Exercise: Create a webpage with an input and two buttons on it. The user can give it's name in the input, then save it into LocalStorage with a **Save** button click. The other button should be called as

Clear and it should remove the user's name from the storage. On page load it should read the name of the user from the storage (if there was any) and display it in the input field.

Example solution: https://github.com/Webuni-JavaScript-courses/localstorage-example

DOM API

DOM API's purpose is to make it possible to reach the DOM tree from JavaScript. It's one of the most basic Web APIs because it's critical in order to make the webpage interactive. We couldn't even add a click listener to a button without this API.

In the following section we will go through some basic DOM API use case.

DOMContentLoaded event

DOM tree is reachable through the `document` variable in JavaScript.
Since the DOM API is mostly event based it's not unusual that we need to add some event listener to a DOM tree element. But this is only possible once the desired element is rendered on the page, so first we need to wait for the browser to finish the document rendering (or in other words: we have to wait to for the browser to build the DOM tree).
That's the main purpose of the DOMContentLoaded event, so it fires once the browser is done with the job.

Exercise: Create an HTML webpage with a button on it. Then register an event listener on the button for the **click** event.
First we should try our solution without waiting for the DOMContentLoaded event, and let's see what happens.
Then we should improve our implementation and add an event listener for DOMContentLoaded event and register the button's click event listener inside the handler of this event.

Solution:
HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <button id="button">Click me</button>
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 document
   .getElementById('button')
   .addEventListener('click', () => console.log('clicked'));
});
```

Note: If you are working on StackBlitz you won't see an error if you forget to add the DOMContentLoaded event listener because the platform will catch and absorb the error (although it won't work).

Access an HTML element by ID

There are multiple ways to achieve this, the most commonly used methods are

- getElementById
- querySelector — it expects a CSS selector (it can be any selector, not only for id) (like: #<id>)

We use the ID based access if we want to get only and exactly one element from the page (since the id attribute has to be unique for the whole page). It has the advantage that usually ids don't change often (opposed to for example a CSS class) so it will be easier to maintain queries like that.

Exercise: Insert a new input into an HTML webpage and assign an ID to it (with the **id** attribute) then register an event handler to this input.

Solution:
HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <input id="my-input" />
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 document
   .querySelector('#my-input') // or getElementById('my-input')
   .addEventListener('input', (event) => console.log(event));
});
```

In general we can say that if we register an event handler function then this function can expect an **event** object as a parameter and the browser will provide the event there. Note that this event will differ for different types of HTML elements and different types of events (click, scroll, etc.).

There's a common and important attribute, the **target** contains the element which was the target of this event.

Inputs

HTML input elements are really important, because we can receive inputs from the user through them (which is usually a necessity if we are working with users). There are multiple subtypes of input (text, number, password, etc.) — you can find the whole list here.
Note that maybe a bit confusing but **inputs** are not the only input type elements, we also have select and textarea elements.

It's general for these inputs that they have a **value** attribute where we can read / write the value of the input. Note that this value will be a string in every case (even if the input type is number) and if we are writing a value into the input then it will automatically convert the value to string.

The easiest way to observe the changes of an input is adding an event listener for **input** or **change** events (**input** will be fired on every key stroke, while the **change** will fire only after the user finish the typing — like clicking out of the input field)

Exercise: Insert a new text input into an HTML webpage and add two event listeners (one for the input's **input** event and one for the **change** event).
Observe their firing sequence with a console log inside their handler function.
Set a new value for the input from JavaScript (eg.: Cubix).

Solution:

HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <input id="my-input" />
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 const input = document.querySelector('#my-input');

 input.value = 'Cubix';

 input.addEventListener('input', (e) =>
   console.log('input event', e.target.value)
 ); // Access the inputs actual value 1.: e.target.value
 input.addEventListener('change', (e) =>
   console.log('change event', input.value)
 ); // Access the inputs actual value 2.: input.value
});
```

Alter the content of an HTML element

It's a common use case that we want to alter the DOM tree from our JavaScript code. For example if we want to display some dynamic content for the user (eg.: some data downloaded from a web server) then this content won't be available at the time we write our HTML code, so we need to insert this content into the HTML after we have downloaded it from our JavaScript code.
In a case like this we can put a placeholder element into our HTML code, and then we can fill it up with some content from our JavaScript code later on. We can modify the content of an HTML element with two attributes:

- innerHTML — if we want to add an HTML string as content
- textContent — if we want to add plain text as content

Exercise: Insert a new number input (type=number) into an HTML webpage where the user can set up how many button does he like to display on the page.
Then create an empty div element on the page and insert the defined number of buttons (from JavaScript) inside this div.

Solution:

HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <input id="my-input" type="number" />
   <div id="buttons"></div>
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 document.querySelector('#my-input').addEventListener('change', (e) => {
   const value = e.target.value;

   const div = document.getElementById('buttons');
   div.innerHTML = ''; // Clear the div to make sure it's empty
   for (i = 0; i < value; i++) { // For loop can be handy if you need to do an
operation X times
     div.innerHTML = div.innerHTML + `<button>${i}</button>`; // We concatenate a
new button in every iteration
   }
 });
});
```

Access HTML elements by tag name

It's possible that we want to operate on multiple HTML element (eg.: add an event listener for all of them) and in these cases it would be painful to reach them one-by-one based on their ID. But DOM API has the ability to query multiple elements at a time (we can do that in many ways actually), for example it can query elements by their tag name (eg.: input, div, etc.):

- getElementsByTagName
- querySelectorAll — like: div

Both of these functions return all matching HTML elements, but the advantage of the **querySelectorAll** is that we can use the array operations on the result (eg.: forEach method).

Exercise: Insert three button into an HTML webpage and register (from JavaScript) the same click event handler for all of them.

Solution:

HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <button>1</button>
   <button>2</button>
   <button>3</button>
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 document.querySelectorAll('button').forEach((button, index) => {
   button.addEventListener('click', () =>
     console.log(`You clicked the ${index + 1}. button`)
   );
 });
});
```

Access HTML elements by CSS class

The tag name based selection usually targets too much elements on an average webpage, so we can use another approach in these cases. We can append a custom common CSS class for those elements that we want to select by our selector and then query the HTML elements based on that CSS class. We can use the following methods for this:

- getElementsByCssClassName
- querySelectorAll — like: .green

Both of these functions return all matching HTML elements, but the advantage of the **querySelectorAll** is that we can use the array operations on the result (eg.: forEach method).

Exercise: Insert four button into an HTML webpage and register (from JavaScript) the same click event handler for the first three of them.

Solution:

HTML:

```
<html>
 <head>
   <meta charset="UTF-8" />
   <script src="script.js"></script>
 </head>
 <body>
   <h1>Hello there!</h1>
   <button class="selected">1</button>
   <button class="selected">2</button>
   <button class="selected">3</button>
   <button>4</button>
 </body>
</html>
```

JS:

```
document.addEventListener('DOMContentLoaded', () => {
 document.querySelectorAll('.selected').forEach((button, index) => {
   button.addEventListener('click', () =>
     console.log(`You clicked the ${index + 1}. button`)
   );
 });
});
```