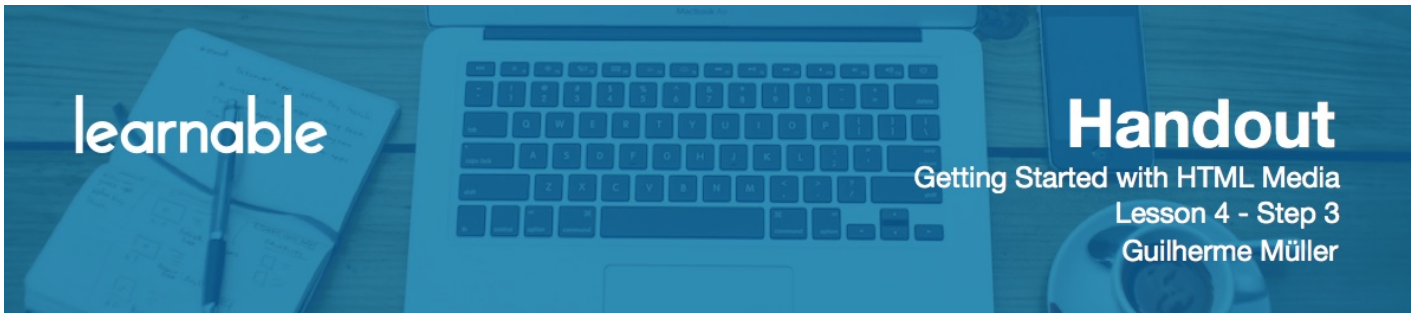

Introduction

In this lesson, we are going to develop the JavaScript part of our media player. I think that this is a good opportunity to use plain JavaScript and not something like jQuery. We become too used to JavaScript libraries and as a result, we can become dependent on them. Libraries are great, but at least every once in a while, it's good to return to the roots and use just JavaScript.

JavaScript for the Media Player

Create a *js* folder and a file named *mediaplayer.js* inside. Then, let's link it with the HTML file just before the closing body tag:

```
<script src="js/mediaplayer.js"></script>
```

The media player assignment should be encapsulated in a function so that we have better organization and portability of the code:

```
function mediaPlayer(playerid) {
}
```

To make things easier, we are going to select the media player by ID. That means we need to assign an ID to each media player we are using:

```
<div id="mediaplayer1" class="media-player" data-target="video-element"></div>
[...]
<div id="mediaplayer2" class="media-player" data-target="video-element"></div>
```

Then, after the function declaration, let's execute the function:

```
mediaPlayer('mediaplayer1');
mediaPlayer('mediaplayer2');
```

Now assign the media player `div` to a variable and select the controls inside it:

```
if(!playerid) {
    return false;
}

var mediaPlayer = document.getElementById(playerid);

if(!mediaPlayer) {
    return false;
}

var playPause = mediaPlayer.querySelector('.controls-play-pause');
var seek = mediaPlayer.querySelector('.controls-seek');
var mute = mediaPlayer.querySelector('.controls-mute');
var volume = mediaPlayer.querySelector('.controls-volume');
var fullscreen = mediaPlayer.querySelector('.controls-fullscreen');
var controls = mediaPlayer.querySelector('.media-player-controls');
```

If the `playerid` is not specified or `mediaPlayer` is not found, we end the function by returning `false`.

Now, we should also assign the media and the media type to a variable. To make things easier, let's use another hook in the HTML, so that our JavaScript will know which player will play which media:

```
<video id="video-element" controls poster="media/poster.png" preload="auto">[...]</video>
[...]
<audio id="audio-element" controls preload="auto">[...]</audio>
```

To link the media player with each tag, let's use a **data attribute**. Data attributes are HTML attributes that have no meaning in HTML and that have the purpose of providing data to JavaScript:

```
<div id="mediaplayer1" class="media-player" data-target="video-element">[...]</div>
[...]
<div id="mediaplayer2" class="media-player" data-target="audio-element">[...]</div>
```

JavaScript Data Elements

Now retrieve the value stored in the `data-target` attribute and find the target:

```
var target = mediaPlayer.dataset.target;
var media = document.getElementById(target);
```

It's also a good idea to store its tag name so we know if we are working with video or audio:

```
var mediaType = media.tagName.toLowerCase();
```

Assign the current media volume to a variable:

```
var currentVolume = 1;
```

The next step is to run some initialization procedures. It's a good practice to turn on the default controls on the video and audio tags. This way if there's some problem with our script, at least the default controls appear. We already have `controls` present in the HTML file, so just deactivate them in JavaScript:

```
media.controls = false;
```

Set the default values for the sliders:

```
seek.value = 0;
volume.value = 100;
```

As we know which media type we are going to be working with, we can hide the fullscreen button if audio is being played:

```
if(mediaType == 'audio') {
    fullscreen.style.display = "none";
    seek.style.width = "79%";
}
```

The `style.display` property accesses the `display` CSS property of the selected element, in this case, the fullscreen button. Hiding this button causes the layouts to be a bit off, so we stretch the seek bar a little more to compensate.