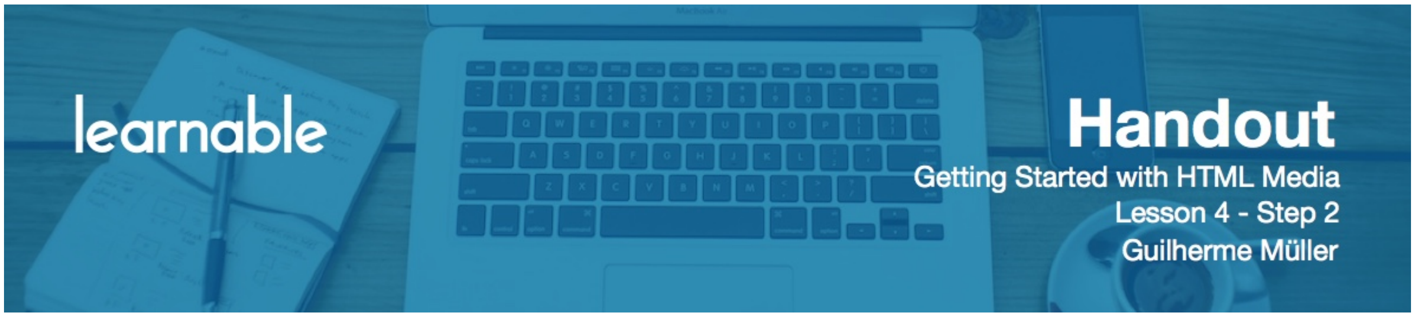
learnable

Handout

Getting Started with HTML Media
Lesson 4 - Step 2
Guilherme Müller

Introduction

In this step, we are going to continue the development of our media player. This time it's the CSS for the custom controls.

Styling the Media Player

Create a *mediaplayer.css* file inside the *css* folder of the project. Then, link it to the HTML file:

```
<link rel="stylesheet" href="css/mediaplayer.css">
```

Before we start styling, it's useful to have a reference to the design we would like to execute. The *mediaplayer_result.png* file is a screenshot of the final media player.

First, lets apply some styles to the buttons:

```
.media-player .button {
    border: 0;
    background: transparent;
    text-align: center;
}
```

This removes the default border and background that comes with the buttons and ensures that the text is centered.

Next, based on the design, the background of the controls should be black:

```
.media-player-controls {
    background: #000;
}
```

With that, we already see a few problems. The bar is too large compared to the video and the text is the same color as the background.

Go back to the `button` class:

```
.media-player .button {
    border: 0;
    color: #FFF;
    background: transparent;
    text-align: center;
}
```

Then there's something we can do about the media being smaller than the player:

```
.media-player video {
    width: 100%;
}
```

Now, `video` inside the `media-player` class will always stretch with its container, along with the controls.

The controls itself aren't stretching yet, so let's do that now. For this to happen, we have to build a grid system. To ensure precise control over the width and height of every button and slider, let's create a rule floating everything to the left and changing the `display` property to `block`:

```
.media-player .controls {
    float: left;
    height: 2.5rem;
    display: block;
    cursor: pointer;
}
```

The `height` here is using a different unit. The `em` unit scales with the `font-size` and it's very useful when designing responsive websites. However, if I change the font size on one of these buttons, the size of the `em` will also change, which can break our layout. That's why there's a relatively new unit called `rem` (root em). The root em takes its value from the root of the document. That means that its value will be a constant, only changing when we change the `font-size` for the entire document. Therefore using `2.5rem` means that the height of each control will be 40 pixels by default (`font-size` is set to `16px` by default).

clearfix and Pseudo Elements

If you check the result in the browser, you'll notice that the layout broke. That's the effect of floating all elements inside of a container. We will introduce the `clearfix` class to clear the floats right before the end of the tag:

```
.clearfix:before, .clearfix:after {
    content: " ";
    display: table;
}

.clearfix:after {
    clear: both;
}
```

The `:before` and `:after` are pseudo-elements that are added right after the starting tag and right before the ending tag of the affected element.

Use this class in the HTML file:

```
<div class="media-player-controls clearfix">[...]</div>
```

Now you will see that the layout has come back to normal.

Width in Percentages

Now we have to calculate the grid so that the controls occupy all the horizontal space available. It's best to work with the percentages here, already thinking of the variety of situations this player could be used on.

We have 100% of width to occupy here. Of that 100%, we have the width of three buttons, two sliders and a little margin between all elements. So let's start with the margin:

```
.media-player .controls {
    float: left;
    height: 2.5rem;
    margin-right: 1%;
    display: block;
    cursor: pointer;
}
```

Now every control has a margin on its right side. The fullscreen button is included in that,so let's add another rule to remove that extra margin:

```
.media-player .controls-fullscreen {
    margin-right: 0;
}
```

With that, we have a total of 4% of space used by margins. That leaves 96% of space to be used by our controls.

Standardize the width of the buttons:

```
.media-player .button {
    border: 0;
    color: #FFF;
    background: transparent;
    text-align: center;
    width: 4%;
}
```

Now we have a total of 12% of space being used by the three buttons and there's 84% left to use. The volume slider should be small, and the seek slider takes up most of the space:

```
.media-player .controls-seek {
    width: 75%;
}

.media-player .controls-volume {
    width: 9%;
}
```

Now the controls are scaled in percent and can stretch with any size of the player!

Styling an Icon Font

The next task is to substitute the text on the buttons with the icons present in the design.

Images could be used, or a CSS sprite, but I think that this is a great opportunity to use an icon font. Icon fonts scale easily and also display very well in higher resolution screens like Apple's Retina display.

To generate this font, I've used a web site called Fontello. We need to embed this font in our CSS file. The generated font comes with the default code in the CSS folder, in a file named after what you chose. Open the *mediaplayer.css* file to find the code to embed the font. Paste it here at the beginning of the CSS file:

```
@font-face {
    font-family: 'mediaplayer';
    src: url('../font/mediaplayer.eot?88461241');
    src: url('../font/mediaplayer.eot?88461241#iefix') format('embedded-opentype'),
         url('../font/mediaplayer.woff?88461241') format('woff'),
         url('../font/mediaplayer.ttf?88461241') format('truetype'),
         url('../font/mediaplayer.svg?88461241#mediaplayer') format('svg');
    font-weight: normal;
    font-style: normal;
}
```

Please ensure that the file path is right for all formats, or else you will not see the font applied correctly.

Next, we have to define the classes that we use with each of the fonts. I could use the default code generated by Fontello, but we are going to create our own simpler code here and then copy the codes of the icons that will be needed. The icons should be placed with the `:before` pseudo-element, so that they have less interference in the HTML code and in other situations, like users using screen readers:

```
.media-player .button:before {
    font-family: mediaplayer;
    display: block;
    font-size: 1.25em;
}

.media-player .face-play:before {
    content: '\e800';
}

.media-player .face-pause:before {
    content: '\e801';
}

.media-player .face-vol-full:before {
    content: '\e804';
}

.media-player .face-vol-mid:before {
    content: '\e803';
}

.media-player .face-vol-mute:before {
    content: '\e802';
}

.media-player .face-expand:before {
    content: '\e805';
}

.media-player .face-restore:before {
    content: '\e806';
}
```

The exact codes for `content` may vary depending on the order of the icons chosen in Fontello. These strange codes refer to Private Use Area Unicode. It's an empty space available in the Unicode table that can be used forthese kinds of purposes without the need to touch the space dedicated to the alphabet, numbers and punctuation. That way, screen readers won't try to read these "letters" we are presenting on the page.

Add classes to each button in the HTML file:

```
            <button type="button" class="controls button controls-play-pause face-play">
                Play / Pause
            </button>
            <input type="range" class="controls controls-seek" min="0" max="100" step="1" value="(
            <button type="button" class="controls button controls-mute face-vol-full">
                Mute
            </button>
            <input type="range" class="controls controls-volume" min="0" max="100" step="1" value=
            <button type="button" class="controls button controls-fullscreen face-expand">
                Toggle Full-screen
            </button>
```

The icons should be looking good, but we have to remove the text, preferably not hiding it completely from screen readers and other similar software. For that, we'll need a bit of extra markup and an extra class. Here is this class that I've copied from HTML5 Boilerplate:

```
.visually-hidden {
    border: 0;
    clip: rect(0 0 0 0);
    height: 1px;
    margin: -1px;
    overflow: hidden;
    padding: 0;
    position: absolute;
    width: 1px;
}
```

This CSS code hides an element from view, but keeps it readable by search engines, screen readers and others, as the code is present in the markup and does not have a `display: none` declaration.

With this in place, we can add a span with a class around the text we want to hide:

```
<div class="media-player-controls clearfix">
    <button type="button" class="controls button controls-play-pause face-play">
        <span class="visually-hidden">Play / Pause</span>
    </button>
    <input type="range" class="controls controls-seek" min="0" max="100" step="1" value="0">
    <button type="button" class="controls button controls-mute face-vol-full">
        <span class="visually-hidden">Mute</span>
    </button>
    <input type="range" class="controls controls-volume" min="0" max="100" step="1" value="100">
    <button type="button" class="controls button controls-fullscreen face-expand">
        <span class="visually-hidden">Toggle Full-screen</span>
    </button>
</div>
```

That will hide only the text and the icon will still be visible.

Finishing the CSS

There are two more CSS rules we can use to make accessibility and browser support better. The first one is this:

```
.media-player {
    min-height: 2em;
}
```

This is done to correct a problem that can happen with some browsers when using this media player on `audio` tags. The `audio` tag may have no height at all, and that can make the media player disappear.

Here is the second style:

```css
.media-player button:focus, .media-player input:focus {
    outline: 1px dotted red;
}
```

With this, we put a red outline around the buttons when they are focused with the keyboard using the TAB key. This make accessibility better; otherwise, the user may not know what he or she is focusing on.