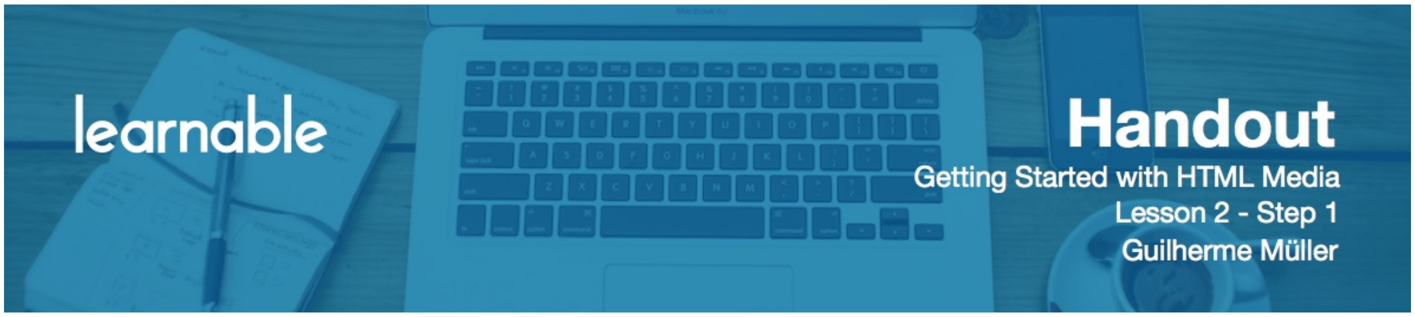

Handout

Getting Started with HTML Media
Lesson 2 - Step 1
Guilherme Müller

Introduction

In this lesson, I'm going to show you the basics of the `video` tag. Open the *videobasics.html* file provided in the lesson files before proceeding.

In the *media* there are three different video formats: each browser that supports video can use one or more of these formats.

Video Tag

Let's start by writing the simplest version of the `video` tag. You'll notice that it's similar to the `img` tag.

I'll create a `div.video` for each video that we're going to insert in the page, so that the base CSS may apply some basic styles to it. This CSS is just resetting margins, applying `box-sizing: border-box` so we have no problems with the box model, and some width and margin bottom to the videos.

`.video` will contain the video itself:

```
<div class="video">
    <video src="media/video.mp4"></video>
</div>
```

In the browser, if it supports the MP4 format, you'll see that the first frame of the video is already displayed. Currently all modern browsers support mp4 in some way.

Boolean Attributes

However, the video is not playing yet. For that to happen, we have some options like adding specific attributes to the tag.

One of those attributes is `autoplay`. This one, like many others we are going to see in this course, is called **boolean attribute**. Boolean attributes only have two states: `true` and `false`. `true` is when the attribute is added to the element; `false` is when it's not present. Such attributes don't have any values.

Add it like this:

```
<div class="video">
    <video src="media/video.mp4" autoplay></video>
</div>
```

If you are using XHTML syntax, you have to add a value, which can be anything, but generally it's coded like this:

```
<div class="video">
    <video src="media/video.mp4" autoplay="autoplay"></video>
</div>
```

This is done to respect XHTML rules.

Reload the page in your browser. You'll see that now the video starts to play automatically. However, we still don't have much control over the video, like pausing or changing the volume. Let's change that by adding another boolean attribute called `controls`.

```
<div class="video">
    <video src="media/video.mp4" autoplay controls></video>
</div>
```

This time, besides playing automatically, the video will also have browser-specific controls for play, pause, seeking, volume and full screen. It's possible to use JavaScript plugins or create custom controls that override these default ones, as we'll see later in the course.

Some other attributes that can be applied to the `video` tag:

- `loop`. When applied, the video will loop indefinitely.
- `mute` will mute the video by default.
- `poster` can be used to provide an image that will be shown before the video is played for the first time.
- `preload` is used to tell the browser how we want the video to be preloaded. This attribute is very limited and it won't work in most mobile devices because of bandwidth issues. It accepts three values: `auto` (preload the video if possible), `metadata` (load at least the metadata of the video, for example, its length), none (do not preload the video). If you have `autoplay` in the `video`, it will override the `preload` attribute as the browser will have to download the media anyway.

Modify your code like this:

```
<div class="video">
    <video controls poster="media/poster.png">
</div>
```

In the browser you will see that the image is being shown instead of the first frame of the video.

Source tags and Codecs

There's one more thing that we need to do: serve different video formats and provide a fallback to browsers that do not support HTML video.

Providing fallback is very simple. Between the tags, put some fallback text, like this:

```
<div class="video">
    <video controls poster="media/poster.png">
        <p>
            Your browser doesn't support HTML5 video playback.
            You can <a href="media/video.mp4">download the video instead</a>.
        </p>
    </video>
</div>
```

Anything that's between the `video` tags and is not a `source` tag will be interpreted by the browser as fallback content.

The `source` tag is like an expanded `src` attribute. This tag can be used multiple times inside the `video` tag to provide different video formats for each of the browsers. Browser support is much better now than it was some years ago, but even so, we need to serve at least two video formats for the user to ensure maximum compatibility. In this example, we are going to use three formats: mp4, Ogg video and WebM. Check out the lesson notes for a link to a compatibility table for `video` and `audio`.

Before adding the `source` tags, remove the `src` attribute from the `video`. Then, add the `source` tag:

```
<div class="video">
    <video controls poster="media/poster.png">
        <source src="media/video.webm" type='video/webm;codecs="vp8, vorbis"'>
        <source src="media/video.mp4" type='video/mp4;codecs="avc1.42E01E, mp4a.40.2"'>
        <source src="media/video.ogv" type='video/ogg;codecs="theora, vorbis"'>
        <p>
            Your browser doesn't support HTML5 video playback.
            You can <a href="media/video.mp4">download the video instead</a>.
        </p>
    </video>
</div>
```

`source` tags are void tags: they don't have a closing tag and we use only attributes in them.

Now we have a cross-browser `video` tag. To make identification easier, we specify the MIME types for each of the formats, and the codecs used for video and audio. This extra information can help to identify the type of file you are embedding, but keep in mind that if there's a typo, the video may not load. You don't need to memorize these MIME types.

Notice that we have double quotes inside the single quotes in this example. This is done so that the attribute is not broken by ending single or double quotes earlier.