|  |  |  |  |
| --- | --- | --- | --- |
| **Unit:** | 6G4Z2101 Introduction to Web Design & Development: HTML5/JavaScript Instructions & Assignment | Weeks | 6-9 |
| **Learning goals:** | * Understand HTML5 video element and use markup to embed video that will play across different browsers * Use controls and poster image * Write efficient, modern, and readable event-driven JavaScript to create custom controls for HTML5 video * Understand and use JavaScript event listeners and handlers to provide functionality and interaction | | |

|  |  |
| --- | --- |
| **Tools required:** | **Sections:** |
| - Web Browser(s)  - text/web editor (e.g. Notepad++, Dreamweaver [code mode], TextWrangler) | Part A: **Using Markup** |
| Part B: **Using JavaScript** |

*You must read ALL of this document!*

*Topics covered:* HTML5 Video, canvas; event listeners

*Learning Outcome:* write and understand efficient, readable client side JavaScript that is event- and object-driven, and portable*.*

*Graduate Outcomes:* Express ideas effectively and communicate information appropriately and accurately using a range of media including ICT; Find, evaluate, synthesise and use information from a variety of sources

***Learning and Assessment***

*These instructions guide you through a set of specific learning activities. After a topic has been covered and step-by-step instructions successfully followed, you are required to transfer your understanding to a new problem. Once you have done this, you will then complete a formally assessed task for which marks are awardable: you will not get help with this, but will get general feedback and the opportunity to improve your mark if you do the scheduled work by the two checkpoints (your lab in weeks commencing 21 Nov & 5 Dec). The different stages of learning are colour-coded in the instructions as follows:*

|  |  |
| --- | --- |
| *Colour key* | |
| *Learning activity* |  |
| *Transfer understanding* |  |
| *Assessment* |  |
| *Reference* |  |

*The learning activities teach you best-practice and the most up-to-date techniques: you need to work through them* ***thoroughly*** *in order to achieve the unit’s learning outcomes and to enrich your ePortfolio pages via client-side scripting.*

**The Coursework Assessment is also built into these materials,** *so you* ***must*** *work through the instructions in order to complete the Assessment. You can* ***not*** *proceed directly to the assessed task and skip the learning materials. The Coursework Assessment components are referred to in these notes as ‘Assessed Tasks’, and shaded red.*

*You will need to work on the materials in your own time, as well as during your lab sessions. You should bring questions and problems to your tutor in the lab session.*

*Put your video and canvas content in separate pages, and link to them from your ePortfolio pages.*

***Support***

*Do the activities, get feedback and help when you need it, and also get help for the transfer-of-understanding challenges that follow up learning activities to give you supervised practice during classes in transferring your understanding to new situations. The unit requires that you understand, and are able to transfer and repurpose, the content of this tutorial. These are therefore key learning opportunities and it is important not to miss them – as you will* ***not*** *be able to get help on assessed work. You can get help with the unshaded and green-shaded sections; but not if you’re attempting to patch it into the assessed work. If you present your assessed work at the checkpoints indicated, you will receive feedback and the opportunity to improve your grade before the final coursework deadline. Follow the instructions carefully and learn, then apply your understanding independently!*

**Feedback**

*Please check as you go with your lab tutor. Communicate any misunderstandings or ambiguities that you find in the instructions, along with any other suggestions, so that we can improve these materials.*

***Comments***

*When following the step-by-step instructions – and when writing code for transfer-of-understanding and assessed tasks - you should relate the code as you type it to the explanations, lecture material, and reference sources given, writing comments as you go (‘//’ for single lines in JavaScript, /\* \*/ for multi-line comments; html comments use <!-- -->). You should do this to articulate your understanding, but also to develop the important graduate skill of documenting your code and accurately communicating how it works so that it will be maintainable. Work towards great comments by reading explanations carefully, attending and reflecting on lectures, and asking questions: the goal is not practice in copy-and-pasting or typing, but engagement with and comprehension of the code!*

**Plagiarism**

You must write all assessed work yourself based on these materials, along with your own clear and concise comments explaining what you’ve done. Your comments are a key criterion in assessing whether you understand what you’ve done. The notes indicate reference sources on the internet: use these sources but do not copy examples from them, and do not use material derived from the internet to solve the assessed tasks. Do not give your work to other students.

**Screencasts**

*These are text-based instructions, as the development medium we are working with is essentially text-based. I have also produced some very short screencasts to explain and illustrate three key assessment criteria (*[*Meaningful Identifers*](https://moodle.mmu.ac.uk/mod/kalvidres/view.php?id=1054162)*,* [*camelCase*](https://moodle.mmu.ac.uk/mod/kalvidres/view.php?id=1056116)*, and* [*Comments*](https://moodle.mmu.ac.uk/mod/kalvidres/view.php?id=1056117)*). These are available on Moodle.*

*If there is any particular process that you think you would understand better via screencast, I can produce additional demonstrations if there is sufficient demand around a particular topic.* [*Email*](mailto:P.McKenna@mmu.ac.uk)*, feed through your lab tutor, or mention in a lecture. Note that most if not all online and youtube screencasts do not use the best-practice and modern techniques taught on this unit!*

# Part A. Using Markup

### 1. HTML5 Video

Interactive multimedia capability is HTML5’s most significant innovation. Video is the most obvious case where the new open standard can compete with the former proprietary standard (Flash video) by providing native playback of video directly within the browser. We will explore the new HTML5 <video> element over two tutorial sections: the first using only markup, and the second using JavaScript to create our own custom playback controls.

One of the main issues with distribution of video content across the web without a self-contained player such as Flash plugged into the browser, is inconsistent browser support. This is one of several reasons why YouTube (up until 2015) as well as many advertisers, news providers, and TV catchup players, continued where possible to play Flash first. However, these issues are being resolved and HTML5 video is becoming more widespread. Video control solutions are often implemented via simple use of built-in browser controls, or libraries such as Video.js, JW Player, and Kaltura; but a good web developer should be able to create their own video controls using event listeners. The directed activity in these instructions will therefore enable you to understand how to provide HTML5 solutions for cross-platform compatibility of native video. It will also enable you to understand the DOM, how elements can be manipulated using JavaScript, and how to use DOM Level 2 event Listeners to make website front-ends interactive. These skills are fundamental: when you grasp them, you will be able to transfer what you know to a wide range of web development tasks.

### 2. Activity Brief

You wish to include a snappy one minute introductory video in your ePortfolio website. You want the video to be viewable on as many devices as possible, including the iPad, and to create your own controls so that you can style them later to match the individual look and feel of your website. You will also be able to reuse these controls when you place other video content (e.g. an interview video) on your portfolio website.

As a trial, and pending the creation of your own video content, you’ve been provided with a dummy video: a not-so-snappy 1 minute video tutorial on Photoshop.

Most browsers now support mp4 (H.264 codec), but you have also been given a copy of the video file in the WebM (VP8 codec) format. The latter is not necessarily supported by Safari, but VP8 is royalty-free, has better compression, and is promoted by Google: working with it keeps our future options open.

Your initial task is to create a website simply to display the video content without a plugin and regardless of the browser or device used. (A ‘codec’ en**cod**es and **dec**odes digital video for playback, usually also compressing and decompressing it).

Once you have completed this task, you will write JavaScript to create your own custom video controls.

### 3. Video files and folder structure

In the first tutorial you will use markup-only HTML5 to display video content within a web browser.

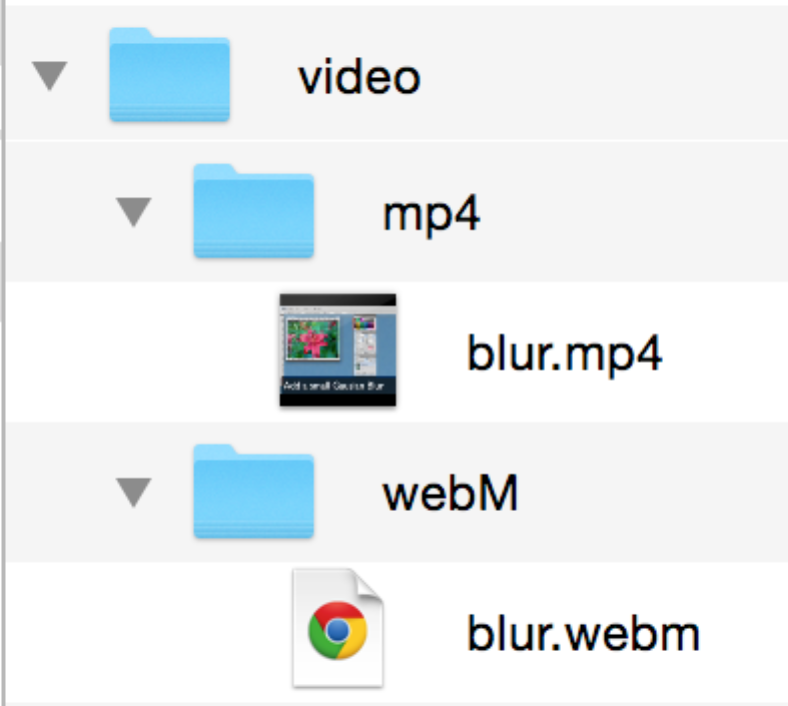
For the content to be displayed, you need to provide the sources for the video files.

Download the file named **videoFiles.zip** from **Moodle Week 6** (w.c. 31 October 2016)and extract to a specific disk location of your choice. This video content is provided just for us to work with, and you can replace it later with your own video content.

The two files contained within the videoFiles archive are named:

blur.webm

blur.mp4

**Action**: In your workspace, create a new folder called **Week6**, then within that a new folder called ‘video’. Inside the ‘video’ folder create two new folders, called MP4 and webM. You should have a folder structure with the directories indicated below (a single folder ‘video’, with 2 subfolders):

**Week6** > video/MP4

> video/webM

1. Copy the downloaded video files into the indicated folders:

blur.webm > webM (folder)

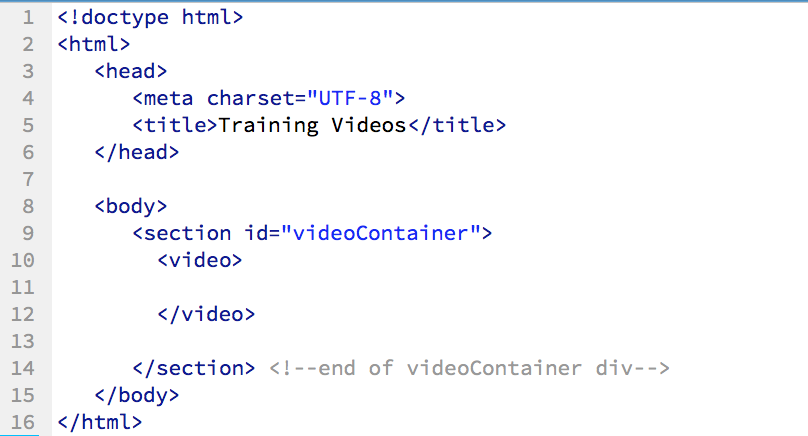
blur.mp4 > mp4 (folder)

Copying the files to the named folders will keep our information organised and structured so we can reference the video content correctly in our HTML5. You can of course structure your folders some other way – as long as the structure is meaningful and the location of the files is reflected in your markup.

Note that not every media player will play both of these files (e.g. Apple’s QuickTime Player will not play webm, in line with Apple’s support for mp4).

### 4. Creating the main index page

**Action**: Open a html editor (e.g. Notepad++, Dreamweaver in code mode, TextWrangler, Brackets, Atom, Sublime), **create a new document, and enter the basic markup to create a web page with a structure as follows**:

****

You are already familiar with HTML5’s simpler character set and doctype definitions, and the semantic <section> element. It is used to group related content. A div element could alternatively be used – though they have no semantic value and are now more often used in HTML5 as generic containers for styling purposes. We will place inside it first the HTML5 video element, and later, our own custom control bar buttons and other controls that we will then activate with JavaScript.

Also note that the tags should be systematically organised in order to maximise readability, with element contents indented and opening and closing tags aligned.

**Action**: **Save the file as index.html in your *Week6* folder.**

Here, we have created the basic HTML5 document structure for our web page. When you load it in a browser, you will see only a blank page with the title ‘Training Videos’. We will now add the video.

### 5. Placing content in the <video> element

To display video, the most basic requirement is to specify where the video file is, relative to where our html document is. The <video> element is now the ‘standard’ way of embedding a video on a web page. The most basic form of the <video> element is similar to the <img> element, and it simply specifies the path to the video using a src attribute.   
  
**Action**: **Modify line 10 from the screengrab above as follows:**

<video src="video/h264/blur.mp4">   
</video>

The path assumes that your video folder is located in the same place as your html document, and that it contains a further folder named h264 that in turn contains the actual video file named blur.mp4. Make sure you understand this before continuing, and know how to change the file path if you have a different folder setup.

We could place text between the video tags, to display in IE8 and other browsers that don’t support the video element, but all modern browsers will support HTML5 video. The tag also supports multiple <source> elements, which we will return to – as not all browsers necessarily support mp4. If your browser *does* support mp4 video (and most do), this will now display the video content of ‘blur.mp4’.

**Action**: **Save the file, then open it in Chrome and check whether or not it plays.**

The first frame of the video will display, but you have no means of playing or otherwise controlling its playback. We need to provide the user with playback controls such as pause, stop and play.

Video playback controls do not display by default. We can use the video element’s controls attribute to display the browser’s default video playback controls. The controls vary from browser to browser, which is one reason why we will create our own controls later using JavaScript.

In the meantime we will active the browser control bar: the controls attribute is Boolean, so it’s simple to use – when it’s present, video controls should display; when it’s not present, no video controls will display.

**Action**: **Amend the <video> element further by inserting the word controls as follows:**

<video controls src= “video/h264/blur.mp4”>

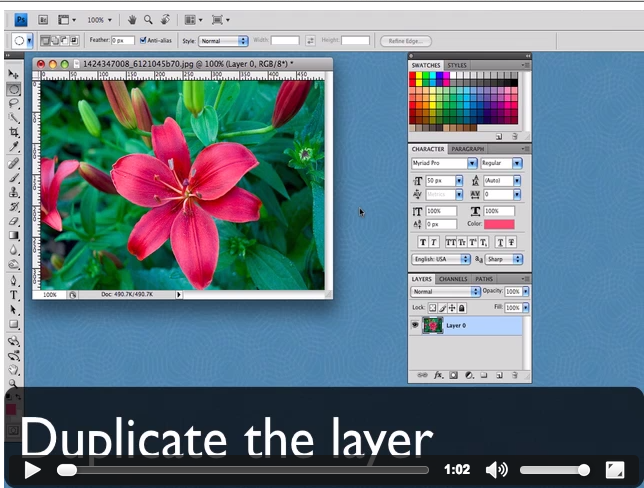
</video>

Be sure to type the above markup yourself: *don’t* copy and paste it.

Basic HTML5 Video browser controls usually include:

* Play
* Pause
* Seeking
* Volume

### 6. Testing in Chrome

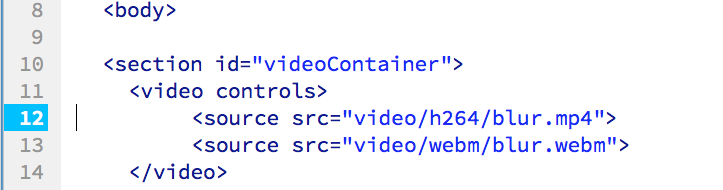
**Action**: **Now save and test in Chrome.** You should see the video displayed with Chrome’s video control bar, which enables you to control its playback. 

Note that the appearance of the playback controls will vary from browser to browser.

### 7. Adding multiple formats

The <source> element can be used to specify the multiple formats we need in order to ensure the video displays in the maximum possible number of browsers. We will replace our src attribute with a <source> element specifying each video file format.

**Action**: **Amend the video element as follows:**

****

We have replaced the single src attribute of the <video> element with multiple source elements, each of which has its own src attribute. You could also add a line to handle the ogg theora version. At present ogg theora support is similar to webm, so it can be considered redundant: check by comparing <http://caniuse.com/#search=theora> to <http://caniuse.com/#search=webm> .

We have specified the source directories for the video files to be played back in the browser. The HTML5 will automatically assign and playback the first video file that is compatible with the user’s browser.

**Action**: **Save and test.** The video should display within Firefox, Safari, Opera and Chrome.

### 8. Browser compatibilities

It is advisable to put the MP4 first, as above: some iPads do not like it if you don’t. And it is currently the most widely supported format.

### 9. Other Video Element Attributes

Although adding the controls attribute to your <video> element offers standard controls to the user, it is possible to use further video attributes that affect the video playback behaviour. We will look at a couple of examples next to demonstrate their use.

1. **Autoplay**

Specifies that the video will start playing as soon as it is ready.

**Action**: **Amend the <video controls> in your HTML5 script as indicated below:**

|  |
| --- |
| <video controls **autoplay**> |

Here, we set the **autoplay** Boolean attribute to true (we could alternatively, but less concisely, set autoplay = autoplay, or autoplay =”true” – XHTML would require one of these), as we have already done with the controls attribute, which is also Boolean: if present, video controls will be displayed.

**Action**: **Save & Test.**

Note that many consider it ‘polite’ to allow the user to actively choose to play a video - ie playback is click-initiated. It is good etiquette not to force video playback on the user - though one pattern of behaviour on ads (as required by DoubleClick and other ad servers) is that video will autoplay without sound, and it is the sound that is click-initiated. This can be partly accomplished by setting the <video> element’s muted attribute, but for now it is best to **remove the autoplay attribute** and just to be aware of it.

**Action**: **delete the autoplay attribute.**

1. **Width/Height**

We will now set the height and width of the video player.  
As with many other types of content that can be displayed in the browser, we can rescale the video by specifying a width & height within the control tag.

Note that in general video should be produced to the dimensions you intend to display it at: resizing will not only degrade the quality, but it also means you use more bandwidth than you need – and in the case of video, this can be very significant. Given the proliferation of differently-dimensioned mobile devices, scaling is not uncommon. If you include the video’s native width and height as attributes, the browser will know how much space to reserve when loading the page.

**Action: Find out the actual width of the video clips provided**. (e.g. you can use QuickTime Player’s Movie Inspector to inspect the properties of a movie.)

**Action: Amend the <video controls> in your HTML5 script (ie between** video **and** controls) **to indicate the width and height of the video (replace xxx with the actual width and height):**

|  |
| --- |
| <video width = xxx height = xxx controls autoplay> |

**Action: Save & Test.**

1. **Adding a Poster Image**

Before the user plays a video, browsers will either display its first frame or a black box. The first frame of a compressed video is not always of the best quality when presented as a still – and a black box is not particularly attractive! The HTML5 <video> element has a poster attribute which allows us to display an initial bitmap instead of the first frame until such time as the user clicks the play button.

**Action:** Create a new folder in the same directory as your html file (*Week6*), and name it images. Now take the image file poster.jpg from Moodle Week 6 (VideoFiles.zip) and place it in your images folder.

**Action: Amend the <video> tag in your HTML5 script as indicated below:**

|  |
| --- |
| <video width = 640 height = 480 controls **poster="images/poster.jpg**"> |

Note: HTML5 video has a preload attribute; as with autoplay, it is potentially a burden on bandwidth or the user, or both. Mobile devices will not preload, but on other devices then these two attributes are a particular problem if you have multiple videos on the same page.

**Action: Save and Test.**

You should now see the still image when the page first loads.

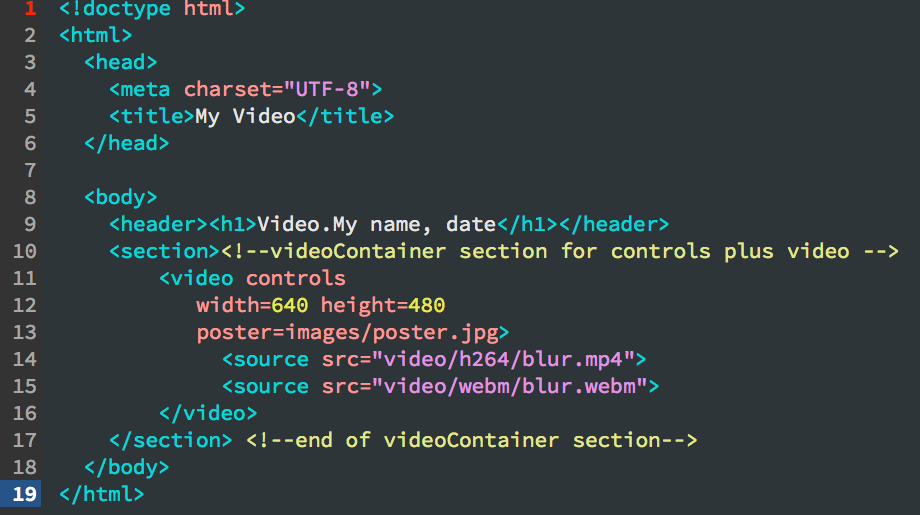
|  |
| --- |
| A complete list of <video> attributes can be found at the W3C site: <http://www.w3.org/wiki/HTML/Elements/video>. See also Opera Web Standards: <http://dev.opera.com/articles/view/introduction-html5-video/#anatomy>  and Refsnes’ W3Schools website: <http://www.w3schools.com/tags/tag_video.asp> |

**Review**: you have learned how to use the HTML5 <video> element to include video playback in a web page.

If you’re not sure about anything, check with your lab tutor before progressing onto the **Task 2 Activity**.

|  |
| --- |
| The <audio> element is almost identical to the <video> element. In your own time, you can transfer what you have learned to the <audio> element and create the means of playing an mp3 file of your choice on a web page. |

*Specimen solution (markup-only video playback)*

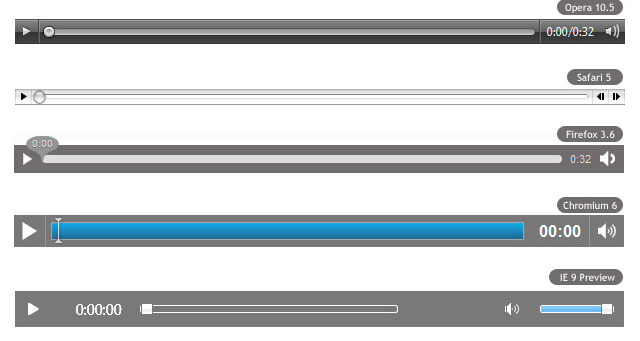


# Part B: HTML5 Video Custom Controls with JavaScript

These instructions assume that you have completed - and understand - *HTML5 Video Part A* above, covering the basic HTML5 <video> element.

While you may use the same video files as in Task 1, you will find this tutorial more engaging if you create your own short video. If you need to convert formats, use tools such as Adobe Media Encoder, Firefogg and CloudConvert.

### The task.

We have noted that the default controls for HTML5 video playback look different in different browsers:

If you want your video playback controls to be consistent across browsers, and/or to be consistent with the look and feel of your website, you will want to create your own controls. The controls themselves can be implemented quite easily in HTML, but to make them work we will need to write some JavaScript and link it to the controls.

### Preparing the HTML to interface with JavaScript

**Action:** Start by opening up in your editor your basic HTML5 from Task 1. Our first point of interest is the <video> </video> element. If JavaScript is to control the video, it needs to be able to refer to it.

Because our JavaScript is going to manipulate this video element, it needs to be able to identify it from the HTML. We will do this shortly by using the JavaScript method getElementById. At present we cannot get it by its ID – because it doesn’t have one in the html. We can resolve this by adding id= “video” (we could call it anything that is reasonably descriptive) to the opening tag of the <video> element.

**Action: Edit the beginning of your video tag (i.e. immediately after** video**) to insert an id as follows (and be careful with your casing of id):**

<video id = "video" controls

(Note that we could simply use the video element without an id: however, we are going to use the most straightforward JavaScript method, getElementById – which, as its name suggests, needs an id).

At this point, you should also **remove the controls attribute**, as we are going to create and code our own. As you know, this can be done by simply deleting the word ‘controls’ from the tag.

**Action: delete the controls attribute**

### Creating custom controls in HTML

### For now we have no control over the playback of the video. Our first step will be to add our custom controls to our own markup. These can go in a separate div, which we will first set up and name videoControls (again, we could call it anything reasonably descriptive), within our existing videoContainer element and after the video element (ie between lines 15 and 16 in the screengrab above):

<div id="videoControls">

</div>

The most basic control we want is a play button. We will use it to toggle between playing and pausing the video. To create the actual control on the page, HTML has a button element: we can create an element of type button, with a meaningful id (such as playPause) and initial caption via its inner text content:

<button type="button" id="playPause" >Play</button>

**Action: Inside the** videoControls **div, type in the following line:**

**<button type="button" id="playPause" >Play</button>**

**Examine it after you have typed it**: make sure you understand what each part of it does, and that you could create a similar button for another purpose if needed. Also check that it displays a button with the expected caption. It will not do anything, of course, until we write our JavaScript.

### Linking the control elements to JavaScript:

### Now that we have the basic play/pause control in place on the html page, we can proceed to writing the scripts to give it functionality.

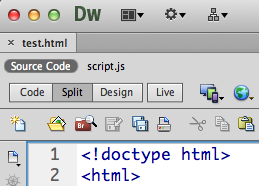
### We can put our JavaScript anywhere – including inside the html itself – but it is good practice to a) place it in a separate file; and b) organise your JavaScript files in a dedicated folder. JavaScript should not be mixed up with html: separating it out into files improves readability and maintainability, and also enables your JavaScript to be cached on the client’s side rather than reloaded with the page.

### You can also call files whatever you want, but it is good practice to think of names that indicate the purpose of each file.

### Actions: Use the Finder or file explorer to open up your Week6 folder. Create a new folder called scripts. In your editor create a new JavaScript file (there is a JavaScript file option in Dreamweaver and other tools, this is initially just a blank text document). Then save it as videoScript.js inside your scripts folder.

### We will first need to link our HTML page to this file, so that it knows where to find the scripts when it loads. To do this we use a <script> element in our html file. We will return to the best place to put this element later on in the unit, but for now type in the following link, between the opening of the body tag and the videoContainer section of the HTML file, as follows (note that the default type is text/javascript, so this does not need to be specified; no need to specify the charset either):

<script src="scripts/videoScript.js"></script>

(If you are using Dreamweaver and have videoScript.js open, close it. With just the html file saved and open with the script link included as above, you should be able to toggle between the html ‘source’ and the linked .js file, as in the screenshot to the right)

Now that the html knows where to find the JavaScript, we need to start writing some JavaScript that connects with the controls in the html, and enables them to work.

*Event-driven scripting: listeners*The way the video controls will work will be in response to user **events**, such as a mouse click on the play button. The code to perform these actions therefore needs to execute on demand: not in a sequence, but when that event occurs to that particular control.

The **DOM Level 2 Event Model** allows registration of event listeners on specific event targets, binding the event to the target element. You will see on the internet other, perhaps simpler, ways of binding events to elements – by assigning handlers to HTML attributes or DOM events – but listeners are more recent and represent best-practice: you should use them. For assessment purposes, you **must** use them rather than html attributes or traditional DOM event handlers. The reasons are explained in more depth in the relevant lecture, which is essential to getting to grips with listeners.

A JavaScript listener can ‘listen’ to a particular element for a particular event, and at the same time specify what code to run when that event occurs. We will therefore attach a listener to each control, specify what event it is listening for, and what code to execute when the event is heard.

*Initialisation*As well as listening for user events like click, listeners can also listen for system events – such as, for example, the HTML page loading completely. It is advisable to be sure that the DOM has fully loaded before activating any more listeners or code. The DOMContentLoaded event occurs after the document has been loaded and parsed, but not necessarily the styles and media, have finished loading. We can ‘listen’ to the document for this event to occur and then call such a function to handle the event. The addEventListener method will do the listening for us, then specify the code that will handle the event.

**Action: Type the following code in as the first line of your JavaScript file:**

document.addEventListener (“DOMContentLoaded”, handleDocumentLoad)

This event listener will ‘listen’ to the main document for the ‘DOMContentLoaded’ event. When it is heard – that is, when the HTML document is fully loaded and parsed – it will handle this event by executing a function. I have chosen to give the indicative name handleDocumentLoad, but you can call it anything meaningful: ‘initialise’ (or ‘init’) is a common name for this sort of handler.

The naming convention we have used, is **camelCase**: the identifier begins with a lower case letter, but the first letter of any subsequent word is capitalised: in this example, handleDocumentLoad. This naming convention enhances the readability of code, and you are expected to follow it throughout the unit. Your identifiers should be as descriptive as possible, without being too verbose. Function identifiers normally involve a verb, and variable identifiers a noun.

handleDocumentLoad will be our main function, determining what happens when the page loads, and we will write it so that it contains the video playback functionality.

**Action: Start off by keying in the function header and braces, after the listener:**

function handleDocumentLoad ()

{

}

All the code we are going to write will be contained within this function.

### Linking individual HTML controls to JavaScript

### Before we can code the control that we have placed on our html5 page, we need to connect our JavaScript to it. We have already given the button and the video elements IDs in the html for this purpose – including the video element. As indicated then, we can use JavaScript methods such as getElementById or querySelector to hook up our JavaScript to our markup. The querySelector method is actually more modern - and in our example would dispense with the need to give the video element an id at all – but it does tend to be slower and less intuitive than getElementById. We could also omit the id and alternatively use getElementsByTagName ("video")[0] – given that we know we only have one video element on the page, it will be the first element in the array the method returns). For now, we will use getElementById.

First, we need our JavaScript to get hold of the video itself. It will be convenient to store a reference to the video element in a JavaScript variable so it can be used again. We will choose to call this variable by the identifier ‘video’, and assign our html5 video element to it. We can get this for JavaScript by using JavaScript’s getElementById function along with the video element’s ID from the html. This will enable JavaScript to identify and then control the video just by using the variable name.

**Action: Type the following variable declaration on the first line inside our handleDocumentLoad function**:

var video = document.getElementById ( "video" );

The ‘Id’ is provided as a string corresponding to the unique id of the element we now want. This was given in our markup as “video”. *The id is case-sensitive and must be exactly the same as in the markup***.** Note that the function identifier is also case-sensitive, and must be spelt getElementById (‘getElementByID’, for example, will not work, and is a common head-against-brick-wall bug for beginners).

Note that you could first check if the video exists or not by using if (document.getElementById ('video')). Error-handling is good practice, but we will keep the code simple for now while we learn the basics.

Now that we have stored the video in a JavaScript variable, we can easily refer to the video just by using that variable.

getElementById works with any html element, so we can also use it to associate our controls with the control elements we declared in our HTML (refer to the IDs specified in the previous page).

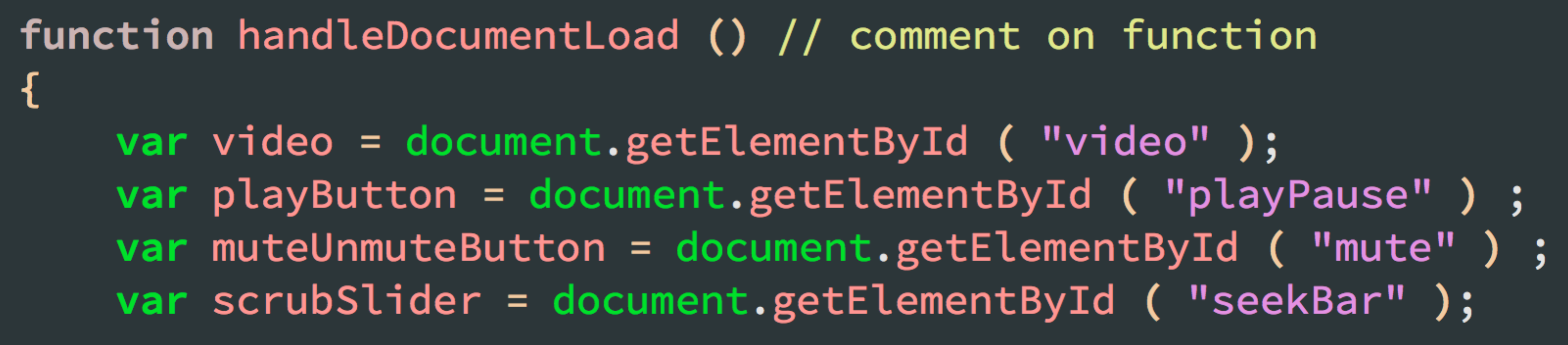
**Action: Enter the following line, after the previous, to get the play button control by its id, and put it in a variable called playButton:**

var playButton = document.getElementById ( "playPause" );

It will be more readable (and more reusable) if we create variables like this and then add listeners to them - rather than adding listeners directly to the result of the getElementById function.

While we are declaring a variable for the play control element, we may as well complete the variable declarations for the other controls we will be creating for now. It will get you used to this pattern.

**Action: Key in the other variable declarations and assignments as follows:**



Here we have created a number of variables and associated them with the video and controls from our HTML markup. Note that there are no types – JavaScript is not strongly typed.

### Handling the Play/Pause Button

Having stored the elements in JavaScript variables, we need to *listen* to each of them for the event we expect (e.g. ‘click’), and then specify the code that will handle the event.

The first video control we are going to implement is the play/pause button. To get this working, we will need to follow the same pattern as we used for the window load: listen to the play button; when it receives a click event, check whether the video is currently playing; and then play or pause the video as appropriate. The object we need to add the event listener to, is the button now stored in the JavaScript variable playButton, and the event it needs to listen for is (unsurprisingly) called click. The click event is fired when a mouse button (or other pointing device) is pressed and released on an element; this is what we expect the user to do to our play button, and we need to listen to the button for when it happens so that we know when to play the video.

We also need to name a function that will handle the event (by playing or pausing the video) if and when it happens to that button.

**Action: In the JavaScript file, add the following just below the variable declarations:**

playButton.addEventListener ( "click", playPauseVideo ) ;

The function name can be anything, but should be meaningful; and good function identifiers should normally use a doing-word, or verb. I have chosen to call it ‘playPauseVideo’.

Now we need to actually implement the handler function, *playPauseVideo*. Video is a HTML5 Media Element, and we can look up its methods at <https://developer.mozilla.org/en/docs/Web/API/HTMLMediaElement#Methods>.

**Action:** Open this link and check out the different methods available. You will see that the play and pause video methods in the HTML5 Media Elements API are (again unsurprisingly) called play() and pause(). They work with their object via dot notation: objectName.play() etc.

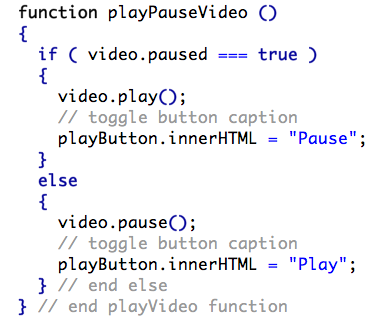
Before we know whether to call play() or pause(), however, we need to know the current state of the video – that is, whether it is already playing or already paused. This is a **property** of the video, which we can check.

**Action:** Go back to MDN and check out the Media Elements **properties**: <https://developer.mozilla.org/en/docs/Web/API/HTMLMediaElement#Properties>. You will see that the relevant property is called paused, and checking its value will tell us the current playback state of the video. Unsurprisingly, paused is a Boolean value – the video is either paused or it’s not paused (i.e. playing).

Object properties, like methods, are accessed via dot notation: we need to check the value of video.paused. To do so, we will use the equals operator in JavaScript, === , then call its play or pause method as appropriate.

We will also need to change the button caption accordingly; the DOM gives us three options: innerHTML, textContent, and innerText. We are using the first, which is the best-supported as well as being style-sensitive. The inn er html of an element is the content inside its tags.

**Action: Type in the code below, just after the playbutton’s event listener (and still inside the handleDocumentLoad function):**



(Note again that ‘video’ here is the variable identifier, which we have – for want of a more original name – chosen as the name for our video variable.) Here we have set up an event listener for the click event on our play/pause button, and used an if statement to toggle the video playback. We update the text displayed on the button by changing the element’s content via its innerHTML property. (There are better ways of doing this, but for now using innerHTML is the most straightforward).

|  |
| --- |
| *Some reusable techniques and principles learned from the tutorial you’ve just done:* |
| * + 1. Using the HTML button element; use reference sources to identify other input element types and events |
| * + 1. getElementById: understanding how JS hooks into HTML |
| * + 1. JavaScript should be in an external .js file and linked via a script element |
| * + 1. Use DOM Level 2 listeners to handle events |
| * + 1. Be able to find appropriate methods and properties from the HTML5 Media Elements API documentation - e.g. play(), pause(), paused() – and use them |
| * + 1. Using a condition depending on the state of the property |
| * + 1. innerHTML |
| * + 1. good identifers: camelCase; verbs for functions, adjectives/participles for variables; balance between verbose and descriptive |
| * + 1. good comments are essential for readability and maintainability: they should not just repeat code, and while learning you should use them to articulate your understanding of the code mechanics |

**Action: Save and test the play/pause button.**

Before progressing to the Transfer-of-Understanding task below, go over the key features of what you have just done again. They are very common in writing interactive event-driven JavaScript for interactive and dynamic pages, and it is important to understand them. Write down a summary list, and try to organise these into a mental template for handling events. You will need to use this understanding throughout your web development career.

**Note on Transfer-of-Understanding Task**

Your first ‘Transfer-of-Understanding’ challenge follows. Challenges are what your education is all about: they challenge you to transfer what you have understood to a slightly different situation. To solve new problems based on what you have learned. This is essentially what you will be asked to do for the unit assessment, so this kind of challenge is practice for that – but with the opportunity to receive help and feedback on your effort (you are on your own with the assessment!).

It is therefore **vital** that you work out a solution based on what you have learned, and that you do **NOT** attempt to copy or adapt someone else’s solution from the web!! If you feel the need to do this, **don’t**: go back over the last 8 pages then discuss with your tutor. Shortcuts – including using Internet tutorials, or skipping any part (explanations as well as actions) of the notes – are dead-ends.

|  |  |
| --- | --- |
| **Transfer-of-Understanding Task:**  You should be able to complete the following task based on your understanding of the previous task. If not, go back over the previous task. You can get help as necessary with the process of transferring understanding from that task, as well as formative feedback on your completion of this task. Handle the Mute Button To get the mute button working you need to follow a process very similar to the one you used for the play/pause button:   * 1. create the button in the html   2. in your JavaScript get it by id   3. in your JavaScript, assign a variable to it   4. add a listener to this variable   5. for the listener’s first argument, specify the event it’s listening for   6. for the listener’s second argument, specify the name of the function you will create to handle it when it happens: use a verb-based name that reflects what the function will do   7. declare and write the function; this should      1. mute if not muted, unmute if muted (this needs a property, **not** a function)      2. change the caption to ‘unmute’ if muted, and to ‘muted’ if unmuted   *You will also need:*  The video **property** that tells us whether the video is muted or not, and that can also be changed to do the actual muting and unmuting. Note: you will need a read-write property to do this, **not** a function.  *Where to find this:*  The same place of reference where we found the paused property: <https://developer.mozilla.org/en/docs/Web/API/HTMLMediaElement#Properties>.  *Hints:*   * Remember the difference between = and === (or ==) * You can explicitly set the property to the true or false values within an if/else. * If you want a shorter solution, a Boolean value can be toggled just by setting it to the opposite of its current state (the NOT operation in JavaScript is “!”). | |
| [7. Mute Button] Done |  |

### Creating and handling the Seek slider

With the basic buttons sorted, we can turn to a different kind of control in the seek bar. This will allow the user to ‘scrub’ through the video playback, as well as indicate how far the playback has progressed. The choice of HTML element is less obvious than with the button: we need a numeric input type, with the numbers being varied along a scale by a slider in relation to the playback position within the video.

The button element was a form element, and the one that can serve us as a slider is also a form input element. One of HTML5’s new <input> types is range (https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Forms/The\_native\_form\_widgets#Sliders). This is the one we want, as - apart from IE - browsers render range input elements as sliders. We will initialise our scale at zero (the beginning of the video).

**Action: In your html file, type this in after your button tags:**

<input type="range" id="seekBar" min="1" max="100" step="1" value=”0”>

Note the minimum, maximum and step values; and that the value attribute is initialised at zero. Note in general the value attribute at this stage, as the corresponding JavaScript value property can be very handy for getting and setting the value of other html elements. Remember it when you come to do assessed work. **Save your file, and check that the slider control displays.**

*Adding functionality*

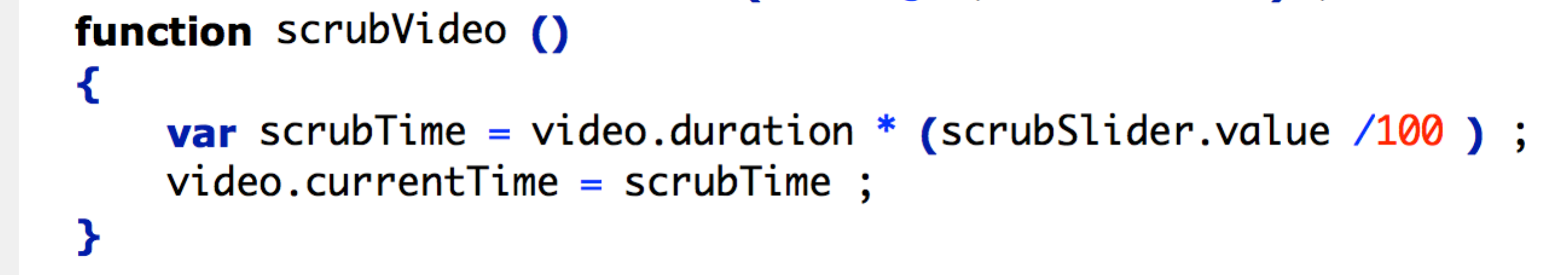
Getting a slider control to work is not necessarily difficult; however, this particular one is trickier than the buttons (and trickier than the slider control which forms part of your assessed work), as there is more than one event and kind of functionality involved. We will add the most basic functionality first, so that dragging the slider handle changes the playback position in the video.

To do this we need once again to set up an event listener, this time on the scrubSlider variable we created earlier to store the seekBar element in. This time the event that it will listen for is not click: bearing in mind that the slider is actually an input element, look at MDN’s list of events (<https://developer.mozilla.org/en-US/docs/Web/Events>). A few up from click is the change event: we can listen for this event and handle it by calculating the time in the video that playback should skip to. If and when we create a time display field, we would also want to update that in line with the slider.

**Action: Enter the listener as below:**

scrubSlider.addEventListener ( "change", scrubVideo ) ;

Within the handler function that I have chosen to call scrubVideo, we want to calculate the time-point in the video that has been chosen by dragging the slider. We can do this by relating it to the total duration of the video. We can find out from our MDN reference point (<https://developer.mozilla.org/en/docs/Web/HTML/Element/Input>) that the range of the range element attribute is all whole numbers between 0 and 100. We can calculate the time in the slider by converting the 0-100 value to a percentage and then multiplying it into the total duration (accessed via the duration property).

**Action:** Key in the function code below, immediately after your function that mutes the sound: 

Having calculated the time by taking the total duration of the video (accessible via the duration property) and multiplying that by the value of scrubSlider (the seekbar element in the HTML) as a percentage (i.e. divided by 100), to update the position in the video we set the currentTime property of the video to the point thus calculated. (We get the value of the slider element simply by accessing its value property in JavaScript - again, make a note of this for future reference!)

This go-to should work, but if you test it you will notice that the slider control’s handle does not change position when the video plays normally. To fix this we need to detect when the video’s time position changes, and move the slider accordingly. That change is another event – one that happens to the video element: in addition to using MDN as a source, the Web Hypertext Application Technology Working Group (WHATWG) is also useful: it provides a definitive guide to web standards, including all events for HTML5 media. Check out the timeupdate event [[https://html.spec.whatwg.org/multipage/embedded-content.html - event-media-timeupdate](https://html.spec.whatwg.org/multipage/embedded-content.html#event-media-timeupdate)]. Think about what the handler needs to do when this event is fired, and what would be a good name for that handler.

The additional listener is straightforward now.

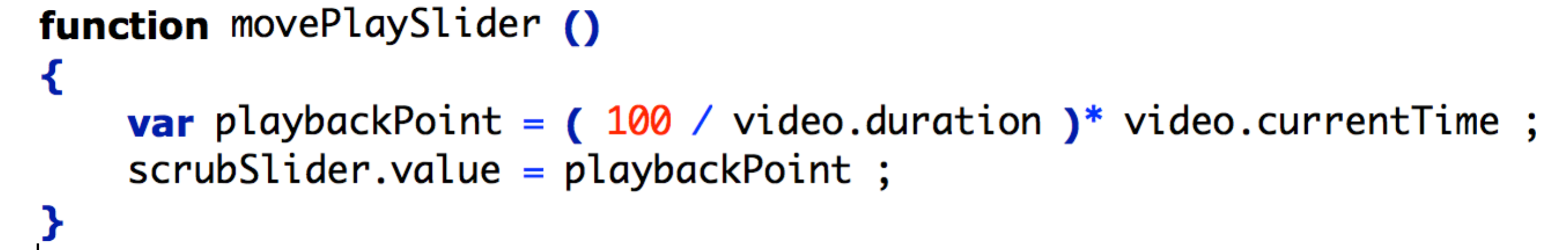
**Action: Add the following to your existing event listeners:**

video.addEventListener ( "timeupdate", movePlaySlider ) ;

We want to listen to the video for timeupdate to happen (it is fired continuously as the video plays), and move the slider handle accordingly.

To work out the position of the slider handle (remember the range is between 0 and 100) in relation to the current playback point in the video we can reverse the calculation we performed earlier, dividing the percentage progress of the video by its duration and multiplying this by the currentTime property of the video. Then we can simply update the value of scrubSlider.

The code below will implement our handler and update the seek bar as the video plays. **Action: Type this in immediately after your scrubVideo function (or after the timeupdate listener)**:



Now if you reload the page and start the video, you should see that the slider handle moves along as the video plays.

*More issues*If you drag the slider handle slowly, you may notice it doesn’t behave itself?

The handle bobs back and forward repeatedly? It doesn’t always stay? Playback doesn’t respond? This seek bar is hard work! To fix its behaviour we need to affect behaviour during the drag process: **pause** the video when the slider handle starts to be (and continues to be) dragged, and then play it again only once the handle is released.

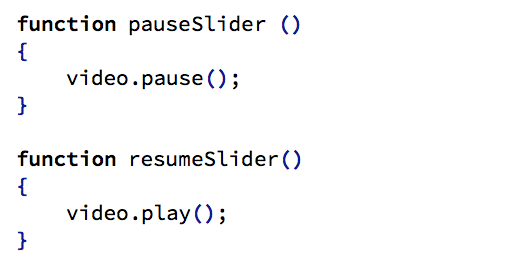
Think about the mouse actions when you start to drag the handle and when you release it. The events are respectively mousedown and mouseup. So now we listen to the seek bar for mousedown then **pause** the video when it happens; and listen also for mouseup, and **play** the video when the seek bar hears that event. If we **write the two listeners as follows** -

scrubSlider.addEventListener("mousedown", pauseSlider);

scrubSlider.addEventListener( "mouseup", resumeSlider ) ;

* the corresponding handler functions will look like this.

**Action:**  **key the following handler functions in after your moveSlider function**:



Now **save and test by dragging the seek bar slowly**.

**Review:**

You should now be able to get html elements by their ids, add listeners to them, and determine what happens when events occur to them.

### Assessed Task: creating and handling a volume slider

|  |  |
| --- | --- |
| To be done **without assistance**, and using the fundamental techniques learned in sections 1-9 of the unit notes.  This task is not as tricky as the scrub bar, but you will be transferring the principles you’ve learned from implementing it: from the scrub bar we have learned the essential pattern we need – create the slider, listen to it for change; and handle it by transferring the value from the slider to the value of the property we want to affect. Work it out – don’t just cannibalise previous code; in this case the property is obviously concerned with the video volume, not with the playback position or anything to do with the video duration!  Do it. Check it. Then get it marked.  *What you will need:*  Create a slider in the html, assign it to a meaningful variable in the JavaScript.  Find out what property holds the volume of the video.  Find out the range of values this property can have.  Add a listener to the slider – along the same lines as the scrub bar  Change the volume of the video to the value of the slider.  Do **NOT** use stackoverflow etc! If you feel the urge to reach for google, think through what you did to create the playback slider, and look up the property for the video volume and range of values (see <https://developer.mozilla.org/en/docs/Web/API/HTMLMediaElement>).  **Save, test, check.** **And again.** Make sure you understand the implementation**, then write a one-line comment in your own words**, indicating how this listener and handler work.  **Getting it marked:**   1. Fill out your Assignment Checklist/Self-Review Form. 2. Get your work checked against the checklist by your lab tutor, and act on feedback 3. Ask your lab tutor to complete the Summative Assessment Form and award your marks | |
| [10. Volume Slider] Done |  |

1. **Guided task: creating a video duration field**

The duration of the video is missing from some browsers’ video control bars. We will display the duration of our video. (Once you have learned how to do this, you will be able to display the current playback time on your own, for the assessed task). Displaying the duration will involve putting the duration property of the video into a text field. We encountered this property while implementing the play slider, so it might sound simple enough: assign the video duration to the text field value. However as we will see, a couple of issues arise.

a. Markup

First we need to create the display field in our markup. There are several options for this, including the new-to-HTML5 time element: this is more useful for calendar and clock times, but see <https://wiki.whatwg.org/wiki/Time_element#duration>. Previously we have used the elements available for forms, and we can use one of these again: see <https://developer.mozilla.org/en-US/docs/Web/HTML/Element#Forms> . The <input> element is versatile, depending on its type: a full list of possible controls is available at <https://developer.mozilla.org/en-US/docs/Web/HTML/Element/input#Attributes> . While intended primarily to accept user input, we can also use its text type to accept any text data.

The default size of text is a single line, so we will set the initial size to four to make it smaller. So that we can work with it easily in JavaScript, we will also give it an id. As it is not easy to get the duration of the video until it starts playing, we will also initialise the contents of the field.

**Action: In your html file, add the following to your** videoControls **div**:

<input type="text" id="durationField" size="4"/>

**Reload the page in a browser.** You should see a field displayed.

However, if you click inside the field, you will notice that you can type inside it. We may as well deal with this now, and **add the Boolean readonly attribute to the element in our markup.**

**Action: modify the line you’ve just entered so it reads as follows:**

<input type="text" readonly id="durationField" size="4" >

b. JavaScript

By now you are familiar with the scripting that needs to be done: get the element by its id, then set its value to the duration of the video.

This seems simple enough, but a couple of further issues will need to be sorted.

**Action:** To discover what those issues are, **key in the following two lines just after the variable declarations in your main handler:**

var durationDisplay = document.getElementById ( "durationField" ) ;

durationDisplay.value = video.duration ;

**Then reload the page.**

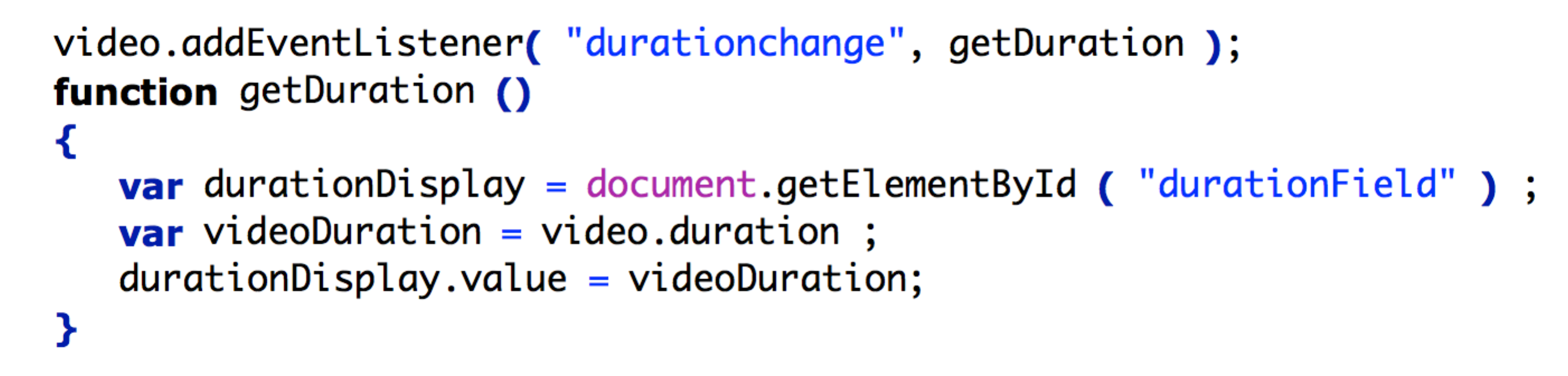
You may notice (depending on what browser you’re using) that the duration field display shows NaN – ‘not a number’. This would be because the video has not yet loaded and its duration is not yet known.

One way round this problem might be to relocate this code so it only kicks in once the video has started to play. However, the field would still be blank until such time as the user starts playing the video.

A better solution would be if we could bind the code to an event that occurs before playback starts but after information about the video has been loaded. MDN list events that occur before media playback, in the order they occur, here: <https://developer.mozilla.org/en-US/Apps/Build/Audio_and_video_delivery/Cross-browser_audio_basics#Media_loading_events>

Which one can we use? The loadedmetadata event would do – as would timeupdate - but the most specific as well as the earliest event is durationchange. We can listen to the video for this event to happen, and bind it to the code that will assign the duration to the field value.

**Action: Key in the following code after your existing listeners/handlers:**

** Now reload the page**.

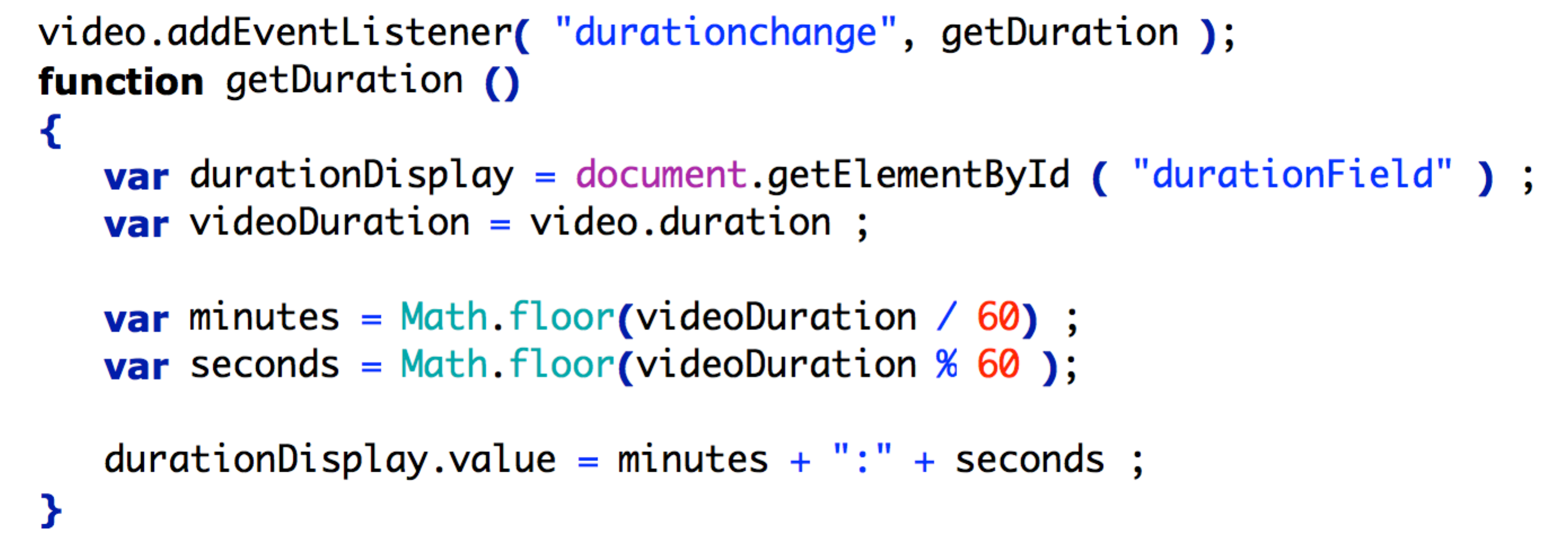
You should now see that the duration in seconds displays in the field.

There are a thousand milliseconds in a second, so the mantissa (the part of the number after the decimal point) is not a problem. However, for lengthier videos we would definitely want to format the duration as mm:ss (minutes and seconds) rather than just seconds.

We can get the number of minutes by dividing number of seconds by 60, with the remainder representing the seconds.

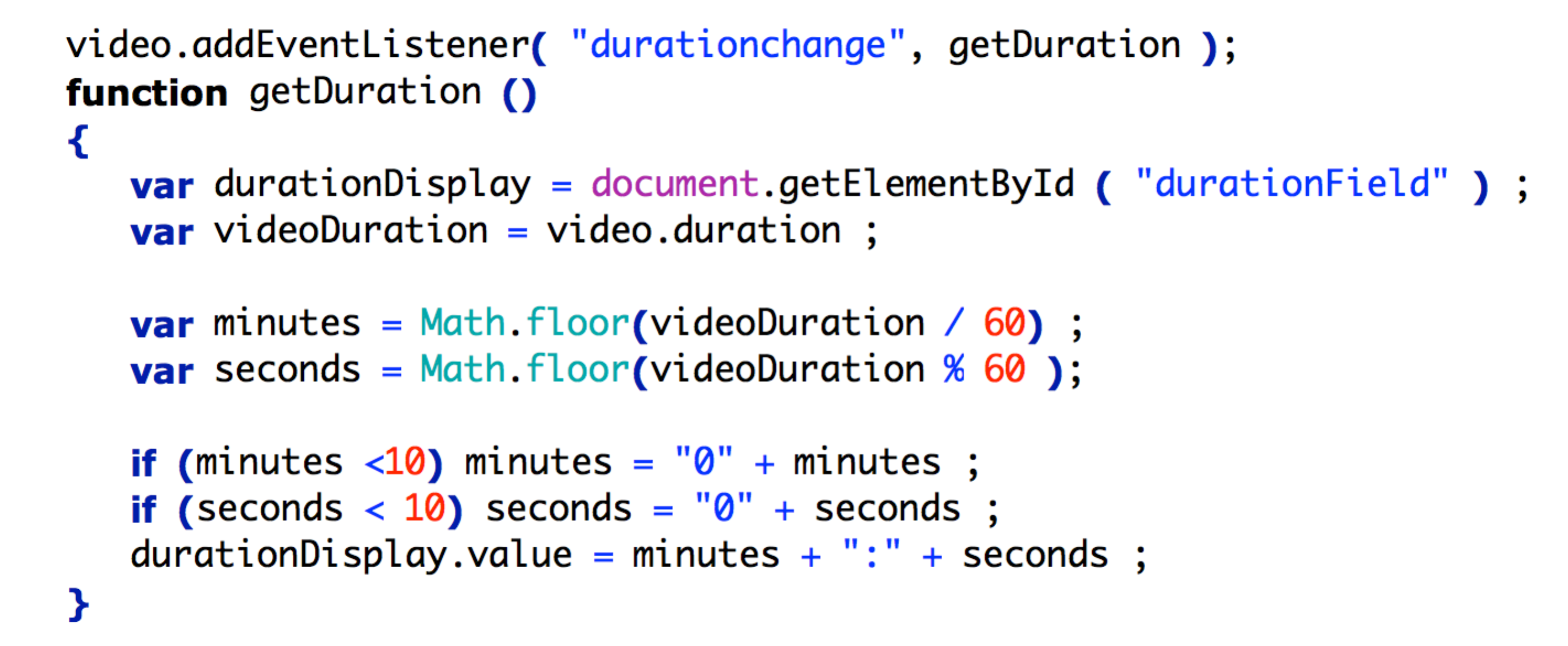
However, JavaScript works with real numbers, and doing this in a simple way (var minutes = video.duration / 60; var seconds = video.duration % 60 ; durationDisplay.value = minutes + ":" + seconds ) will produce a real rather than a whole/cardinal result for the minutes. We need to round it down to a whole number of minutes. JavaScript has a built-in Math object, and we can use one of its methods - Math.floor().

**Action: Change your code to insert the Math.floor() function:**



Load the page now, and you will see another problem: a number of seconds below 10 is displayed without any leading zero. A duration of 1 minute and 2 seconds would be easier read as 01:02 rather than 1:2.

So if the minutes or seconds are less than 10, we want to add a leading zero. We can do this by concatenating ‘0’ as a string character with the single digit number representing the minutes or seconds (note that this is obviously not a mathematical operation, adding 0!). **Action: Amend the code again so it reads as follows:**

Note that in the last line of the above that we could alternatively have used durationDisplay.setAttribute (“value”, (minutes + “:” + seconds) ); . In this example it seems more verbose, but setAttribute() method is potentially quite powerful, as it enables us to update the value of any attribute.

Make sure that you understand the above explanations, as you will need to understand this code in order to implement and comment the next assessed task.

Note also that proper error handling would check first that video.readyState > 0 before any attempt to access the video object.

### Assessed Task: implement a current playback time field

|  |  |
| --- | --- |
| Display the current playback time of the video (i.e. the current playback position in seconds/minutes:seconds) as part of your controls. To be done **without assistance**, and using the fundamental techniques learned in sections 1-9 of these notes.  Go over what you’ve done so far and work out   1. what event you need to listen for 2. what element the event will occur to 3. what property you will need to change   Create a HTML element and assign it to a variable in your JavaScript (you have now done this repeatedly)  Check out a HTMLMediaElement reference (e.g. MDN) again for an attribute/**property** that will give you the current playback time of the video.  Check out a Media event reference (e.g. https://developer.mozilla.org/en-US/docs/Web/Events#Media\_events) again for an **event** that is fired when this property is updated  Write a listener and handler to format the time and put it in your html element.  You may also wish to set the initial default playback time.  Note that the current playback time has no relation to the duration.  **Getting it marked:**   1. Fill out your Assignment Checklist/Self-Review Form. 2. Get your work checked against the checklist by your lab tutor, and act on feedback 3. Ask your lab tutor to complete the Summative Assessment Form and award your marks | |
| [12. Time Field] Done |  |

### Assessed Task: implementing a playback speed changer

|  |  |
| --- | --- |
| To be done **without assistance**, and using the fundamental techniques learned in sections 1-9 of the unit notes.  Learners work at different speeds. Literally. Video tutorials often include a control that allows the user to speed up or slow down the playback.  The lynda.com control is pictured to the right as an example (you do not necessarily need the x suffixes).  Your task is to implement such a control: it should allow the user to choose one of seven playback speeds stepped at even intervals from 0.5x of normal speed to 2x of normal speed, indicate the current playback speed, and default to x1.  Hints:   * Use a Forms reference source (e.g. <https://developer.mozilla.org/en-US/docs/Web/HTML/Element#Forms>) to find an element that will provide a dropdown menu containing different option items like the lynda one. * You do not need any conditional statements.   **Make sure you explain your solution with a comment.**  **Getting it marked:**   1. Fill out your Assignment Checklist/Self-Review Form. 2. Get your work checked against the checklist by your lab tutor, and act on feedback 3. Ask your lab tutor to complete the Summative Assessment Form and award your marks | |
| [13. Speed control] Done |  |

## <--------------STOPPED HERE------------->

1. **Assessed Task: implementing fast-forward and rewind buttons**

|  |  |
| --- | --- |
| To be done **without assistance**, and using the fundamental techniques learned in sections 1-9 of the unit notes.  Implement fast-forward and rewind (skip to start) buttons as follows:  Fast-forward should be straightforward in principle if you have already done the playback speed control.  However, your fast-forward button will enable triple speed when it is held down, reverting to double speed when it is released. Double-clicking the button should restore to normal speed.  You will need to check out your [event reference](https://developer.mozilla.org/en-US/docs/Web/Events) for events like click but which are fired when the mouse is pressed on an element, released over an element, and clicked twice on an element. Each of these events will need to be listened for separately.  You should also indicate status (this can be done most simply by changing the caption of the button as appropriate).  This is an interesting challenge – it allows you to explore the potential of listeners to listen to the same element for multiple events. It is also possible to add multiple listeners for the same event, and at different levels of the DOM hierarchy.  Rewind will simply return the playhead to the start: it does not have to play in reverse, just revisit the property you manipulated in scrubVideo.  **Getting it marked:**   1. Fill out your Assignment Checklist/Self-Review Form. 2. Get your work checked against the checklist by your lab tutor, and act on feedback 3. Ask your lab tutor to complete the Summative Assessment Form and award your marks | |
| [14. FF-REW buttons] Done |  |

### Handling JavaScript disablers

One final problem: at some point you will have disabled the native video controls – i.e. removed the controls attribute from the HTML5 video tag. This is of course required, as two sets of controls would be confusing. It is very unlikely that users will have disabled JavaScript – and it is becoming increasingly difficult to do so – but because it is possible, it is as well to provide the native video controls as a fallback, available only when JavaScript is disabled. To check out this scenario, disable JavaScript. In Firefox, enter about:config into the address bar. Promise to be careful, then search for JavaScript.enabled. Double-click on JavaScript.enabled, or right click on it and click "Toggle". (In Chrome, go to chrome://settings/content and turn off JavaScript at the bottom). If you now open your page, you will find that the video controls do not work and there is no way of playing the video!

A solution to this problem is to first restore the controls attribute, and then write some code to remove it if JavaScript **is** enabled. That way, the native controls will be present if JavaScript is disabled, but will not be present if JavaScript is enabled.

removeAttribute can remove the video’s controls attribute. **We can write this code in the JavaScript file, at the start of our main function:**

video.removeAttribute ( 'controls' ) ;

Now when your page loads, there will be no native controls, just your custom controls. But whenever JavaScript is disabled, the native controls will appear. This is consistent with the principles of graceful degradation and progressive enhancement.

Note that as we proceeded in this tutorial, we used documentation from reliable sources such as MDN, W3C and WHATWG to explore elements, attributes, events and methods. You should now be in a position to explore documentation independently and experiment to develop your understanding.

**---------------------------------------------------------------------------------------------------**

*Video Extension Tasks:*

1. Investigate querySelector and use it instead of getElementById
2. Programmatically add video to your web page (look up createElement, appendChild).
3. Create a video playlist by changing the src attribute dynamically
4. So that the JavaScript-disabled do not have to see controls that don’t work, move the creation of the controls to JavaScript.
5. Select and use a library (e.g. JW Player, Sublime) to implement a video player.

## Video encoding and editing resources:

Adobe Video Encoder, Premiere, Final Cut Pro in the Mac labs.

Open source, free or commercial video encoding software, such as Handbrake (free, open source) or Super (free). YouTube can also encode your video in several resolutions and codecs.

|  |  |
| --- | --- |
| **Additional Learning Resources** | |
| [Introducing the JavaScript Language with Joe Chellman](http://www.lynda.com/JavaScript-tutorials/Introducing-JavaScript-Language/123563-2.html?org=mmu.ac.uk) (Lynda.com) | Status: **background** |
| [JavaScript for Web Designers with Joe Chellman](http://www.lynda.com/JavaScript-tutorials/JavaScript-Web-Designers/144203-2.html?org=mmu.ac.uk),  Lessons 1, 2 and 3. (Lynda.com) | Status: **reinforcement** |

**Key Assignment Requirements [video scripting]:**

There are four key pre-requisites:

1. Your code contains **no** assignment to html event attributes or properties, e.g. onclick=”playVideo”, or playBtn.onclick = “playVideo”.
2. Your solution is complete, and [for the video] includes the guided video controls solutions from the tutorial
3. Your attempts at the assessed components are based on the good-practice from these tutorial instructions: they must contain **no** code copied or adapted from the internet or other sources.
4. You have written concise comments that demonstrate your understanding of the code you’ve written, and what you have learned from the tutorial and the lectures.

*Assignment Checklist/Self-Review Form:*

Implementation of each control is assessed on the basis of code quality criteria as well as functionality. You must achieve these to pass and clearly demonstrate your understanding of what you have learned specifically from the tutorial and the lectures.

|  |  |  |  |
| --- | --- | --- | --- |
| Code quality criteria and checklist | | | If No, what is the room for improvement and what needs to be done? {complete following review\*, and advice from tutor} |
|  | Yes | No |  |
| Code is in separate .js file(s) |  |  |  |
| JavaScript code uses DOM Level 2 event listeners. Contains **no** assignment to html event attributes or properties, e.g. onclick=”playVideo”, or playBtn.onclick = “playVideo” |  |  |  |
| **All** identifiers use camelCase where appropriate |  |  |  |
| **All** identifiers clearly indicate purpose to anyone unfamiliar with the code |  |  |  |
| **No** code is copied or adapted from the internet or other sources. |  |  |  |
| Code structure is made clear by systematic use of indentation |  |  |  |
| **Comments** communicate clearly, effectively, and accurately what is happening in the code USD L4.3 |  |  |  |
| Comments don’t just repeat the code |  |  |  |
| Comments demonstrate **understanding** (theory and implementation) |  |  |  |
| Comments are **concise**: notes, not sentences |  |  |  |
| Solutions are complete: in the case of the video, include the guided solutions from the tutorial; in the case of the flag, colours and dimensions are exact. |  |  |  |
|  | | | |

*\* You should self-review prior to checking with your lab tutor on the specified checkpoint date.*

You should achieve all of the above; if you have, and have done all of the assessed components 1-4 as indicated in the weighted list below, you will be awarded **full marks** for each weighted component. This is the objective for the unit – that you independently apply what you have learned to new situations, and achieve the learning outcomes and obtain full marks.

If you have not achieved all of the above, and/or have not done all the components, you will be awarded a partial mark based on the extent to which you have met the criteria in relation to the weighting below. The formative feedback Checkpoint gives you an opportunity to improve your work based on the feedback given.

Significant copying of material (from the internet, books, or fellow students), and other indications that you have not learned what has been taught, is likely to attract a **zero mark**.

**Summative Assessment form and weighting of components**

|  |  |  |  |
| --- | --- | --- | --- |
|  | *If <100%, unresolved room for improvement* | *mark* | *weight* |
| *Video* |  | | |
| 1. *volume slider* |  |  | *5* |
| 1. *current time* |  |  | *5* |
| 1. *playback speed changer* |  |  | *10* |
| 1. *fast-forward + reverse control* |  |  | *10* |
| *Checkpoint :* | *Lab in week commencing 21 November* |  |  |
| *Canvas* |  | | |
| 1. *map of Brazil* |  |  | *10* |
| 1. *Mondrian* |  |  | *10* |
| *Checkpoint:* | *Lab in w.c. 5 December* |  |  |
| *total* | |  | */50* |

**Graduate Outcomes**

Many of the specific criteria above map onto the following Graduate Outcomes and accompanying Standard Descriptors. Graduate Outcome 3 and its grade descriptors reflect in-code comments and identifiers that communicate clearly, effectively, and accurately to other coders what is happening in the code; Graduate Outcome 6 your ability to apply understanding based on learning and reference material.

|  |  |  |
| --- | --- | --- |
| Graduate  Outcome  Grade  Grade range | 3 | 6 |
| Express ideas effectively and communicate information appropriately and accurately using a range of media including ICT | Find, evaluate, synthesise and use information from a variety of sources |
| 86%-100% | Ideas are presented creatively to an audience of peers using a defined range of strategies and media. | Information from primary and secondary sources is carefully collected, analysed, interpreted and applied to new problems under supervision, bringing insight to the analysis. |
| 70%-85% | Ideas are presented fluently to an audience of peers using a defined range of strategies and media. | Information from primary and secondary sources is meticulously collected, analysed, interpreted and applied to specific problems under supervision, bringing insight to the analysis. |
| 60%-69% | Ideas are presented convincingly to an audience of peers using a defined range of strategies and media. | Information from primary and secondary sources is thoroughly collected, analysed, interpreted and applied to specific problems under supervision. |
| 50%-59% | Ideas are presented confidently to an audience of peers using a defined range of strategies and media. | Information from primary and secondary sources is carefully collected, analysed, interpreted and applied to specific problems under supervision. |
| 40%-49% | Ideas are presented adequately to an audience of peers using a defined range of strategies and media. | Information from primary and secondary sources is collected, analysed, interpreted and applied to specific problems under supervision. |
| 35%-39% | Unclear or confused ideas are presented to an audience of peers using a defined range of strategies and media. | Insufficient information from primary and secondary sources is collected, analysed, interpreted and applied to specific problems under supervision, or the analysis of such information is inadequate or incomplete. |
| 20%-34% | Unclear and confused ideas are presented to an audience of peers using a defined range of strategies and media. | Errors are made when information from primary and secondary sources is collected, analysed, interpreted and applied to specific problems under supervision, and the analysis of such information is inadequate and incomplete. |
| 0%-19% | Unstructured or wrong ideas are presented to an audience of peers using a defined range of strategies and media. | Extremely limited or no information from primary and secondary sources is collected, analysed, interpreted and applied to specific problems under supervision. |