Multimedia Applications using HTML 5, CSS & JS

Aim: To learn the basics of creating media-rich and dynamic web pages with HTML 5, Cascading Style Sheets (CSS) and JavaScript (JS).

Register your attendance

Please make sure we register your attendance.

HTML 5, CSS and JS

HTML 5, Cascading Style Sheets (CSS) and JavaScript (JS) are used together to create web-based applications. As well as traditional web pages, many mobile apps are built using these technologies. In these practical sessions you will gain some experience in how they can be used to create dynamic, media-rich web applications.

HTML 5 is the latest version of the HyperText Markup Language (HTML) that is used to specify the structure and content of web pages.

CSS is a separate computer language for specifying the style and layout of web pages.

JS is a fully-fledged programming language that can be executed by web browsers. It is used in many ways, but particularly to add functions for reacting to events, such as button clicks and data entry in forms, to perform any needed calculations, and to alter dynamically the HTML and CSS of a page to change its content, presentation and layout.

In these practicals you will start with a skeleton web application that is built with HTML5+CSS+JS and you will go through a series of exercises to add content to this application. Complete instructions are not given and you will need to create your own HTML5+CSS+JS code to implement what is asked for. So you may need to consult learning resources as you proceed. Some suitable resources are listed below.

Resources

To learn more and to complete the more advanced parts of this practical you should consult the ***W3Schools*** tutorials and references on HTML, CSS and JS at **http://www.w3schools.com/**

Introduction to Our Multimedia Web Application

The web page you will construct has been designed to look more like a self-contained application than a traditional web page. It contains different screens that are accessed by buttons. Each screen will give an example of different uses of HTML5+CSS+JS. You will go through a series of exercises that will enable you to enhance each screen in turn. Take your time, and be prepared to work on it in your own time, if need be. The skills learnt here will be employed in the module assignment.

Preparatory work

Starter files for this practical session are in the **WebMM** ZIP file on the module's **Practicals** page on **Canvas**. Download this file to somewhere convenient in your file space and unzip it.

A variety of development tools are available for use in the lab, and you can choose which you would prefer. **TextPad** is a fairly intuitive text editor, and provides syntax colouring (that is, different parts of the text are coloured differently so that you can easily differentiate, for example between HTML tags and text). Alternatively you can use a fully-fledged IDE such as **Netbeans** or **Eclipse**: these tools make it easier to keep track of files in single projects, and provide useful editing features, such as code completion. Note that **Dreamweaver** in the labs is **not appropriate** as it does not support HTML5.

To view your application and to **debug** it, you need to display it in a web browser, such as **Firefox** or **Internet Explorer 11 (IE)**. The **developer tools** in these browsers will help you to debug your code, particularly your Javascript, by enabling you to set breakpoints and watch variable values.

To start things off:

1. Open your copy of the **WebMM** folder. You will see a couple of HTML files and a subfolder containing some media to use.
2. The file **MMfinal.html** is a *finished* version, so this will give you an idea of what you are aiming for. Open this file in a browser by opening the browser and then clicking-and-dragging the file onto the open browser window.
    a. Select "**Yes**" if you are asked to "**Allow blocked content**".
3. Check out briefly what there is by clicking each of the buttons in the left-hand menu in turn.
    a. On the Video screen, you may need to scroll the mouse over the video to see the play controls.
    b. On the Animation screen, click on the gray "bullet" to the left of the cowboy.
4. You might like to leave this open in your web browser so you can refer back to it as you progress with building your own application.
    a. Note of course that this file contains code for all of the exercises given below, but you are strongly advised NOT to look at this unless you are really stuck. What you will build does not have to be an exact replica of this version, and you are encouraged to develop your own code (there will be more than one way to do many of the things you are asked for).

This is a single web page that is designed to look like a self-contained multimedia application. JavaScript is used to change the content in the main panel when a new button is clicked (rather than displaying a whole new web page). It makes use of the popular **JQuery** library of JS functions.

You are going to start with a *skeleton* version of this, stored in the file **MMapp.html**, that you will gradually modify according to the instructions given below to produce your own customised version.

5. Open **MMapp.html** in your browser. You will discover it only has a partially-complete Home screen and button.

Leave this page open in your browser for the duration of this practical session. You can just **refresh (press F5)** the browser when you make changes to the page to see the updates. You will edit this starting web page using your preferred development environment (IDE):

6. Open **MMapp.html** in your IDE.

7. Take a careful look through this file: it contains computer code written in HTML, CSS and JS. This code provides the basic structure for our multimedia app, but most of the content is missing.
    - The HTML element is used to divide the page into a main content area and left-hand menu panel.
    - The position and style of each is provided by CSS in the embedded style sheet.
    - Button operation is controlled using JS. Take particular note of the *clickBtn(btn)* function in the header script. It makes use of the **JQuery** library to do some clever things: in short, it hides all HTML elements with IDs containing "scene" and then shows the element with ID "scenex", where x is the value of the argument btn. (Now is a very good time to explore a bit about JQuery in the W3Schools site, if you are not familiar with it.)

So let's start to make some changes to this basic page.

Home Screen

The opening screen in this multimedia application corresponds to the Home button and shows some text in different styles. Do the following:

8. The home screen is defined by the with ID "scene0": examine the HTML code for this as this structure will be used to create the other screens of your app.
9. The main heading has some CSS styling applied to it (see the <style> sheet in the document <head>), but the remaining text does not. Add some CSS style rules to make the paragraph text more interesting. You might like to give the two paragraphs different styles (as in the example finished app), and put your name in place of the text in the second paragraph.
10. **Refresh** the web page in your browser to see the change that you have made.
    - **Note** that every time you make a change to the code in your IDE, you have to save the file then refresh its display in the browser to see the effect of the changes. (Alternatively, Netbeans enables you to load the file into the browser in the IDE).
11. In the finished app, you might have noticed that if you hover the mouse cursor over the "WELCOME!" text it spins around (if you have not seen this, then try it now). This is an example of a CSS *transition*. Let's add this to your app:
    a. Add the style definition, "**transition: 2s;**" to the style rules for *h1.welcome* in the embedded style sheet. This specifies that any change made to the style of h1 while it is being displayed should take effect over a time interval of 2 seconds. Such a change could be made dynamically via JS code, but we will use the HTML *hover* state here.
    b. Add the following style rule to the embedded style sheet that specifies a "**transform**" to apply when the mouse hovers over the h1 text, "WELCOME!":

        h1.welcome:hover { transform: rotateY(360deg); }

        Try this out in the browser.

    c. Try changing the transition time in the h1 style and see what happens.

d. Change the rotation that is applied in the *hover* condition for h1 from "rotateY(360deg);" to "rotateX(360deg);". How does the text rotate now? Try also "rotateZ(360deg);".
e. You can add other style definitions to *h1.welcome:hover* and see them applied during the transition as well. For example, set the text colour to yellow by adding "color: yellow;" to the *h1.welcome:hover* style rules. What happens now when the mouse is over the heading?

Feel free to play with different CSS styles for this page before moving on to the next exercise.

Slide show

Now we are going to add a new scene, with its associated button, to our app. This scene will contain a slide show of images.

12. Directly below the closing tag that finishes "scene0" (but before the that finishes the "stage") create a new scene with ID "scene1", following the template of the home screen (you could cut-and-paste the entire "scene0" … and edit as needed e.g. delete the heading and paragraphs, as you will provide new content below).
13. As content for this scene, add a heading, "Slide show", using for example <h2>, and give it some nice styling using CSS.
14. Now add a paragraph whose content is not text, but an image (<img> with ID="slide", source of "Media/image1.jpg", width of 288px and height of 209px):

    <img id="slide" src="Media/image1.jpg" width=288 height=209 alt="Image 1">

15. To make this scene accessible, create a new button immediately below the Home button in the menu , with the appropriate call to *clickBtn()*.
16. Save what you have done and try it in the browser. Hopefully you can click on your new button and go to this scene showing the first image.
17. Now let's turn it into a slide show. Helpfully, the Media folder contains three images: image1.jpg, image2.jpg, image3.jpg. These simple names will make it easy to write some Javascript code to update the source for the "slide" <img> element and cycle through these images. For a first attempt, try:
    a. Add a button labelled "Next" to this scene by putting a <button> element immediately below the paragraph containing the image. On clicking, this button should call the JS function, *nextSlide()*, which you will now define.
    b. In a new <script> element, which you can place below the end of the "scene1" , define a function *nextSlide()* that changes the image source of "slide" in sequence, returning to image1 after image3. You will have to think about this and write suitable JS code, with variables and control logic, to achieve this functionality. Note that JS can get a pointer to the <img> element by calling *document.getElementById("slide")*, and then the source attribute can be changed by calling *setAttribute()* as a method on the returned <img> object. Please ask a demonstrator if you need help, but also be prepared to look up JS and HTML documentation and tutorials (such as on W3Schools).

c. When you have code to test, reload your file in the browser and give it a go. If things do not work as expected and you cannot work out why by looking at your code, you could try using the debugger in the Developer Tools of your browser (either Firefox or IE). Again, please ask if you need help.

18. Once you have that working, here are some variations you could attempt (you can come back to this after you have completed the rest of your app, if you like):
    a. Add a "Previous" button with suitable JS code so that this button cycles through the images in reverse order.
    b. Now alter your code so that the "Next" button will halt and be disabled at image3, and the "Previous" button will halt and be disabled at image1. You could also add a text display that indicates what slide is being shown eg "2 of 3".
    c. Add a new button, "Auto", and a suitable JS function(s) to show the slides automatically, with a suitable time delay between each. The user should click the button once to start the show, and again to stop it. A time delay can be achieved by using the inbuilt "*setTimeout()*" function (you can find documentation on this at e.g. W3Schools).

Video

HTML5 allows the easy inclusion of videos in web pages via the <video> element. The next screen you will build will display a short video clip and provide suitable user controls to play it.

19. Following the template for screens 0 and 1, create a new with ID "**scene 2**". Include a sensible <h2> heading and a single paragraph with a style that will centre its text (but with no content yet).
20. A video clip is stored in the file "Media/LMFAO.mp4". It is 423 pixels wide by 240 pixels high. Add a suitable <video> tag to the empty paragraph to include this video, using the "controls" attribute to give it user controls (you might have to look up some HTML documentation to do this).
21. As before, to make this scene accessible, create a new button in the menu , with the appropriate call to *clickBtn().* Save your work and try it in the browser. Note you might have to scroll over the video with your mouse to see the user controls (this depends on which browser you are using).
    - If you have headphones you can listen to the soundtrack as well as watch the video.
22. Try replacing the attribute "controls" with "autoplay" and see what happens.
23. It is also possible to add your own buttons for user control. Let's add a "Play" button. Give your <video> tag an ID by including the attribute id="vid". This allows us to refer to this tag by name. Create a new button for this scene by adding the following line after the paragraph containing the video:

```
<button onclick="document.getElementById('vid').play();">Play</button>
```

24. Remove the "autoplay" attribute from the video tag. Save and try your app in the browser. Now you should see a button at the bottom of the Video page. Click it and the video should start to play.

25. Now, in a similar fashion, add buttons to pause and rewind (this is a little tricky) the video and give them a try.

Canvas Graphics

You can create drawings directly in a web page, using the HTML5 Canvas object and JavaScript (JS) programming. This exercise provides an introduction to this approach to web graphics.

26. Create a new "scene 3" with a heading as content, and an associated menu button, as you have done for the other scenes.
27. Below the heading in this scene, create an HTML canvas with a suitable ID, a width of 400 pixels and a height of 280 pixels. You might like to give it a border too.
28. Now after the end of the "scene 3" div, create a <script> element to which you will add JS code to draw to your canvas. It will probably start something like the following (depending on your canvas ID and what variable names you choose to use):

```
<script>
var c = document.getElementById("myCanvas");
var ctx = c.getContext("2d");
ctx.fillStyle = "blue";
</script>
```

29. The exercise is to add code to draw a simple Scottish Saltire with some text: take a look at the finished application to see what we are aiming at. Add code to your script to achieve this (consulting appropriate <canvas> documentation (e.g. on W3Schools) as the need arises; and feel free to ask for help as well!)

The End

That's all for the moment. Keep playing and modifying as much as you like. But…

MAKE SURE YOU KEEP THIS PRACTICAL WORK, AS YOU WILL RETURN TO IT LATER!!

In a later practical session you will add more buttons to this application that will demonstrate different techniques for adding animation to your web pages. Two of these buttons are already active in MMfinal.html, so you can try them out now.