Developing Web Applications using HTML5

Exercise Guide

01 – Introducing HTML5

Objectives

This practical session will introduce you to the building blocks of HTML 5. You will also explore some of the new HTML5 structural tags, and look at supporting IE8 and below.

Reference materials

This practical session is based on the material in the first module: Introducing HTML5.

Overview

The practical session contains three exercises:

- Exercise 1 Creating a HTML5 Skeleton
- Exercise 2 Exploring the new structural elements
- Exercise 3 Supporting IE8 and below

Exercise 1 Creating a HTML5 Skeleton

1. Open your preferred IDE. Your instructor will give you a brief demo of them if you are unfamiliar. If you are unsure of anything, please ask your instructor to come and show you where things are.
2. Navigate to folder EXERCISE\01 – Introducing HTML5\Starter.
3. Create a new file and name it **skeleton.html.**
4. There may be markup already in the file. Hit **ctrl+a** – this will select all the information currently there. Once you have done this delete any markup currently present.
5. First we need to set a doctype for the file:

```
<!doctype html>
```

6. Why do we need to add a **doctype** tag to a browser?

7. Now add the opening and closing markup for a **<html>** element. Follow this up with **<head>** then **<body>** tags within the **<html>** element.
8. Within the <head> section, add a **<title>** tag. Leave its contents blank for now.
9. To the opening **<html>** tag, add a **lang="en"** attribute.
10. Within the head element add a **<meta charset = "utf-8" />** element.
11. Why should you add a **charset** attribute?

12. Save the file.

Exercise 2 Exploring the new structural elements

In this part of the exercise you are going to add header, nav, footer, article and section to the page using the new structural elements of HTML5.

1. Add title text of **Exploring HTML5.**
2. First of all, we will create the header section for the page. Within the <body> tag add a <header> tag.
3. Within the header tag add a <h1> tag with the text of *The HTML5 Structure*.
4. Beneath the <h1> tag let's add a <h2> with the value of *Exploring the new structural tags of HTML5*.
5. That will do for the header, now we will add the navigation section of the page. Below the header tag add a <nav> tag. This section of the page will host the core section of a page or a site's navigation. This should refer to your own site and not to links that would arise from advertising and the like.
6. Within the <nav> tag create an unordered list with two list elements.
7. Inside the first list item add a link to the current page.
8. In the second list item add a link to http://www.qa.com, opening in a new window.
9. Below the navigation section add an <article> tag. This will be a simple setup at the moment and represents the content of our page. Within the article tag create a <h1> with a value of *From the HTML5 spec.*
10. Finally we need to add a <footer> element. This element will hold some copyright information and a link to QA. Add the following inside the footer:

```
&copy;
<a href="http://www.qa.com" target="_blank">QA</a> 2016
```

11. Save the file and open it in IE, Firefox, Safari, Chrome and Opera. Unfortunately the appearance is somewhat underwhelming right now. We are going to have to style the page a little to position the elements correctly.

12. Either in an external CSS file or an internal **<style>** tag add the following CSS:

```
header, nav, footer, article {

              border: 1px solid #000000;
  }
  nav {
            float: left;
            width: 25%;
 }
 article {
            float: right;
            width: 70%;
 }
 footer {
            clear: both;
 }
```

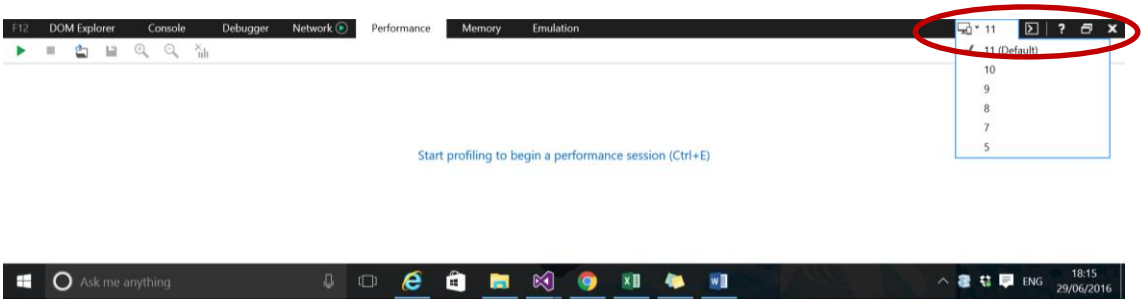
13. Save the page and view it in the browser. It should now be positioned correctly.

Exercise 3 Supporting Internet Explorer 8 and below

IE6 through 8 are still widely available browsers and we have to support them. This is one of the most significant challenges with HTML5. Internet Explorer 9 was the first version of this browser capable of understanding some HTML5. Prior to version 9, Internet Explorer did not apply any styling on unknown elements. If IE 8 doesn't explicitly recognise the element name, it will insert the element into the DOM as an empty node with no children. All the elements that you would expect to be direct children of the unknown element will actually be inserted as siblings instead.

In this part of the exercise we will explore how serious an issue this is if we choose not to handle it. We will learn how to fix it effectively in the next module.

1. Open same web page from last exercise in IE. Note how well it works. Now press F12 to bring up the developer console. You can choose to change the rendering mode for the browser. This is one of the most useful functionalities of the IE developer tools. With it we can ask IE to render the page as if it was in a previous version.



2. Step the rendering mode down to IE5 from IE11. Note that it starts breaking when you select IE8.

3. This means every user on that version of IE and down would get a broken version of our page. This is because older browsers ignore tags they do not recognise. This problem can be solved with JavaScript and some CSS as we will examine in the next chapter.

02 – HTML5 Mark-up

Objectives

This practical session you will be looking at the new HTML5 Outlining algorithm. You will also learn to use the new HTML5 semantic elements to lay out a web page. Lastly, you will download and examine the use of HTML5 boilerplate.

Overview

This practical session contains three exercises:

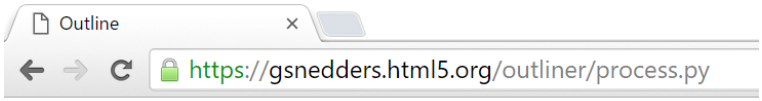
- Exercise 1 Examining the HTML5 Outlining algorithm
- Exercise 2 Using the new semantic elements to lay out a page
- Exercise 3 Examine the HTML5 boilerplate

Exercise 1 Examining HTML5 outlining algorithm

1. Navigate to **02 - HTML5 Markup\Starter** folder.
2. Open ***html5Outline.html*** into any browser and take a look at the code. Note that we have one <h1> and a series of <h2> and <h3> elements. The remaining mark-up belongs to one of these headings. We will start by investigating how this works.
3. Type the following URL into your browser:

```
http://gsnedders.html5.org/outliner/
```

4. Then copy and paste the source code into the HTML section of the page. And hit the ***'Outline This!'*** button. You should then see a page outline like below:



1. Main heading
    1. Level 2 heading
        1. Level 3 heading
    2. Another level 2 heading

5. In a plain document with no other sectioning, the outline will match the heading levels. This is similar to the way a table of contents in Wikipedia.

6. Let's look at how section affect the outline. Edit the code to look like the following:

```
<h1>Sections</h1>
  <section>
      <h1>Main heading</h1>
      <p>Some text</p>
      <h2>Level 2 heading</h2>
      <p>Some more text</p>
      <h3>Level 3 heading</h3>
      <p>A bit more text</p>
      <h2>Another level 2 heading</h2>
      <p>The last bit of text</p>
  </section>
  <section>
      <h1>Main heading</h1>
      <p>Some text</p>
      <h2>Level 2 heading</h2>
      <p>Some more text</p>
      <h3>Level 3 heading</h3>
      <p>A bit more text</p>
      <h2>Another level 2 heading</h2>
      <p>The last bit of text</p>
  </section>
```

7. Run the new code in Outliner again. As you can see, there are now multiple <h1> elements in the document, but they don't sit in the same level of the document outline. In fact, you can do without any heading element other than <h1>.

8. Replace <section> with <article>. Do you notice any difference?

9. Next, consider the <header> element. Wrap <header> around <h1> and <p> like the following:

```
<header>
     <h1>Main heading</h1>
     <p>Some text</p>
</header>
```

10. Run the code again in outline. What do you notice?

Exercise 2 – Using the new semantic elements

1. Navigate to 02 – HTML5 Markup\Starter folder.
2. Open html5-blog.html.
3. Locate <!-- Main Site Nav here -->. Insert appropriate elements tags and attributes to create links to home, blog and contact.
4. Locate <!-- Sidebar links -->. Insert appropriate elements tags and attributes to create links to archives for March, April and June.
5. Locate <!-- Footer links -->. Insert appropriate elements tags and attribute to link to the QA web site. (http://www.qa.com)
6. Locate <!-- Copyright info -->. Using appropriate HTML character entities and text, create copyright information for QA HTML5 blog.
7. Rather than using classes and IDs for sections like headings, navigation and footers, replace all the s with appropriate, new HTML5 structural elements.
8. Replace class=”meta” with appropriate HTML5 semantic elements to present the date/time format with an appropriate attribute.

Exercise 3 Examine the HTML5 Boilerplate

1. Navigate to 02 - HTML5 Markup\html5-boilerplate folder.
2. Take a look at the file/folder structure.
3. Open and examine index.html.
4. What is the Document Type Declaration used?

5. Why is it necessary to declare the charset?

6. Why is this meta tag being used?

```
<meta http-equiv="x-ua-compatible" content="ie=edge">
```

7. What is the purpose of the following meta tags?

```
<meta name="viewport" content="width=device-width, initial-
scale=1">
```

Why do we use *normalize.css*?

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

8. What is the purpose of using *Modernizr*?

```
<script src="js/vendor/modernizr-2.8.3.min.js"></script>
```

9. What is the purpose of the following "*Conditioning commenting*" used?

```
<!--[if lt IE 8]>

<p class="browserupgrade">You are using an
<strong>outdated</strong> browser. Please <a
href="http://browsehappy.com/">upgrade your browser</a> to
improve your experience.</p>

<![endif]-->
```

(Answers for the above questions can be found at the following page.)

Exercise 3 (Answers) Examine the HTML5 Boilerplate

1. <!doctype html>
2. If the charset isn't declared within the first 512 bytes of your HTML document, your site is vulnerable to malicious code and hijacking!
3. Another IE fix to ensure the latest rendering engine version of IE (Internet Exploder) is being used.
4. It's basically a message to the mobile browser that says, "Render me differently, I'm designed for mobile screens too!"
5. Normalize.css makes browsers render all elements more consistently and in line with modern standards. It precisely targets only the styles that need normalizing.
6. Modernizer helps older browsers handle HTML5 elements and it must load at the top of the page before any HTML5-specific code can be read.
7. Conditional commenting only evaluated by Internet Explorer browsers. Kindly inform your site visitor that their choice of browser is antiquated.

03 – Learning CSS3

Objectives

This practical session you will use CSS3 to style a web page investigating some of the powerful new selectors and features. You will then enable media queries to allow the page to react to different device proportions.

Overview

This practical session contains two exercises:

- Exercise 1 Style web page using CSS3 selectors and features
- Exercise 2 Add Media Queries for responsive design

Exercise 1 Style web page using CSS3 selectors and features

1. Navigate to **03 – Learning CSS3\starter** folder.

2. Open **css3.html** and **main.css**. View the HTML page in Chrome. The markup has been partially styled by the CSS for you.

3. Examine the CSS and HTML. You should be familiar with the HTML.

4. In **main.css**. Locate **/*Header Style*/.** Add the following code and the appropriate selectors.

```
header h1{
     margin-bottom: -0.5em;
}
/*Select all h1, h2 and h3 */
?{
    font-family: 'ChunkFiveRegular', Arial, sans-serif;
}
/*Selects any paragraph elements that immediately follow a
level-one and level-two heading elements. */
    ?{
        color: rgba(136, 176, 199, 0.61);
        padding-top: 0.2em;
    }
```

5. In ***main.css.*** Locate ***/*Pseudo-element*/.*** Add the following code and the pseudo-element that applies rules to the portion of a document that has been highlighted.

```
? {
    background: #b3d4fc;
    text-shadow: none;
}
```

6. In ***main.css***. Locate ***/*Add Print Media Queries here*/*** towards the end of the page***.*** Add the appropriate media queries.

7. View ***css3.html*** in Chrome. Press ***ctrl+p*** to bring the print dialog box. You will see the print preview box which is showing the page in a very different way than your browser is choosing to display it to you.

Exercise 2 Add Media Queries for responsive design

In this exercise, we will use the mobile first approach.

1. Open ***css3***.html in Chrome. View the page using Chrome, “Toggle device toolbar”.

2. In ***main.css***, Locate ***/*Add Media Queries for intermediate devices menu here*/.*** Add the following code, and the appropriate media queries for screen which has min-width up to 480px.

```
?{
   body {
     background-color: #ccc;
    }

    nav a {
        float: left;
        width: 27%;
        margin: 0 1.7%;
        padding: 5px 2%;
        margin-bottom: 0;
    }
    nav li:first-child a {
        margin-left: 0;
    }
    nav li:last-child a {
        margin-right: 0;
    }
    /*IE fixes*/
    nav ul li {
        display: inline;
    }
    .oldie nav a {
        margin: 0 0.7%;
    }
}
```

3. Save the page. View the page using Chrome, “Toggle device toolbar”. Select “ipad”mode. You should now see a different navigational menu.

04 – Learning CSS3 Features

Objectives

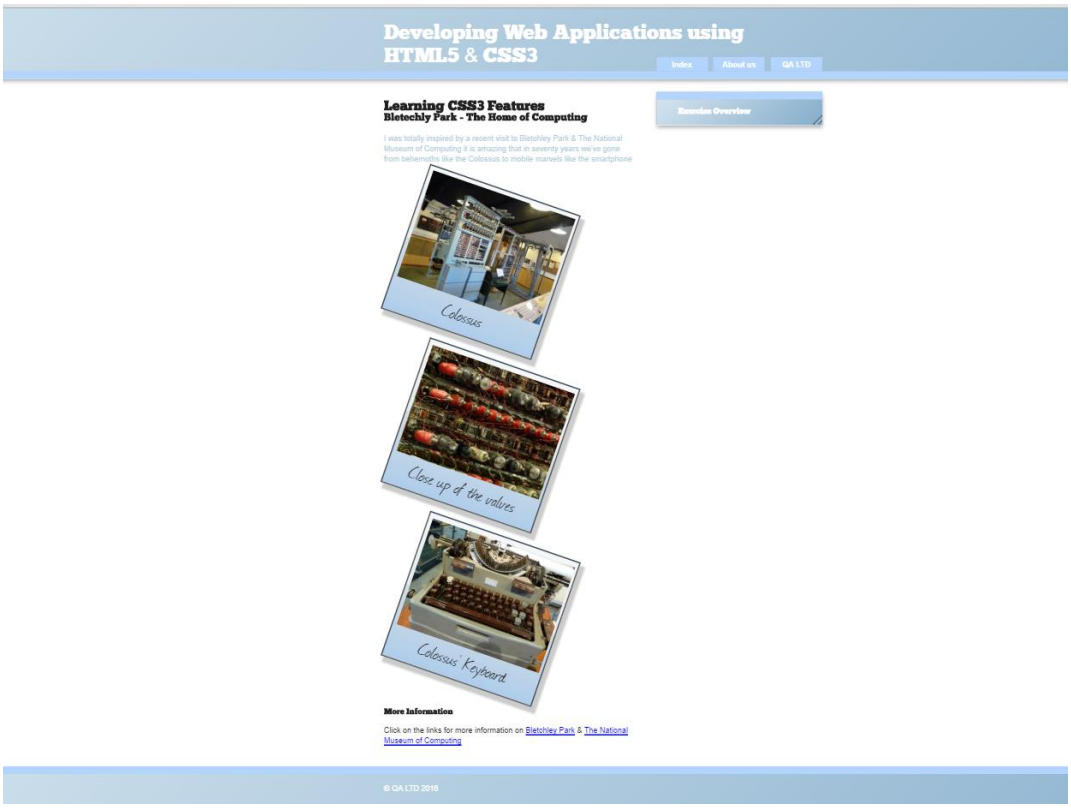
This practical session you will use CSS3 features to style a web page.

Overview

This practical session contains one exercise:

- Exercise 1 you will be using several CSS3 features to style a web page.

The final web page will look similar to the following figure.



Exercise 1 Style web page using CSS3 features

1. Navigate to **04_Learning_CSS3_Features/starter** folder.
2. Open **index.html** and **main.css**. View the HTML page in Chrome. The markup has been partially styled by the CSS for you.
3. Examine the CSS and HTML. You should be familiar with the HTML.
4. In **index.html,** locate **<!-- figure element to go here -->.** Add figure element and a figure caption within the figure element. Within the figure element, but above the figure caption, add an <img> tag. From the img folder within the starter folder, set the source to be **img1.jpg**. Repeat this for **img2.jpg** and **img3.jpg.**
5. Save the page, and open it in Chrome. You will notice the text immediately beneath each graphic. We will format these object using CSS.
6. Open **main.css** from the CSS folder. Beneath the transitions and transformations comment add a selector for figure. Within add the following CSS:

```
    margin-top: 50px;
    border: 3px solid rgba(0,0,0,0.6);
    padding: 10px 10px 30px 10px;
    width: 373px;
    margin-bottom: 50px;
```

7. In the above CSS, add a box-sizing property and set the CSS border model to border-box.
8. Then add a box-shadow property with a suitable rgba().
9. Save all files, and view the HTML page in Chrome. You should note that there is now a border around the graphic giving us the edge for a Polaroid effect.
10. Next, improve the background of the Polaroid by adding a linear gradient. Add the following code within the linear gradient.

```
to bottom, #f6f8f9 0%,#e5ebee 50%,#d7dee3 51%,#afceec 100%
```

11. Save all files, and open the HTML page in Chrome. You will see a light blue background across the Polaroid.

12. Now we will improve the font-face of the Polaroid text. Open fonts folder located within the CSS folder. You will see a ***journal-webfont*** and the file types we need to support for all the fallback considerations. This is the font we are going to use for our Polaroid caption.

13. In main.css, locate ***/*Author's custom styles - Font faces*/.*** Use a suitable CSS rule, and add the following code:

```
   font-family: 'JournalRegular';
    src: url('fonts/journal-webfont.eot');
    src: url('fonts/journal-webfont.eot?#iefix')
format('embedded-opentype'),
         url('fonts/journal-webfont.woff') format('woff'),
         url('fonts/journal-webfont.ttf') format('truetype'),
         url('fonts/journal-webfont.svg#JournalRegular')
format('svg');
    font-weight: normal;
    font-style: normal;
```

14. Now we will add this rule to the figure caption text. Scroll back to the top of the stylesheet, and locate the figure selector. Add a suitable selector for figure caption. Then add the following font properties.

```
    text-align: center;
    margin-top: 25px;
    font-family: 'JournalRegular', Arial, sans-serif;
    font-size: 48px;
```

15. Save all files, and open the HTML page in Chrome. You should see the font has applied.

16. Next, add a transform property to the figure element to rotate it “19deg” from its normal position. Use suitable prefixes.

17. Then, add a *hover* pseudo selector for the figure element. Within it add a transform property, and add the following code. Use suitable prefix.

```
rotate(0deg) scale(1.5, 1.5);
```

18. In the same selector, add a *transform-origin* property, and add the following code. Use suitable prefix.

```
-5% 0%;
```

19. Save all files and check the HTML page in Chrome. It is rather abrupt at the moment.

20. Next, we will a transition property to the figure selector and add the following code.

```
transform 0.5s ease-out;
```

21. Save all file and view the HTML file in Chrome. You will find the animation much smoother.

22. To improve the stacking orders of the elements as we hover, set the hover to have a z-index of 500, and the position to be relative. Save and retest.

05 – JavaScript for developer

Objectives

This practical session you will be exploring the building blocks of reusable code, arrays, functions and object.

Overview

This practical session contains two exercise:

- Exercise 1 you will be exploring indexed arrays
- Exercise 2 you will be creating semantic arrays using object literals

Exercise 1 Exploring indexed arrays

1. Navigate to 05_JavaScript_for_Developer/starter folder. Open arrays.html.
2. Within the script block, create an array with the short hand approach, and assign with a variable sentence, and the following value:

```
"Luke", "I", "am", "your", "father"
```

3. We now have an array of String. Print this to the console using the following:

```
  console.log(sentence.toString());
```

4. Save and test the page in the browser.
5. Now, we will get rid of the commas using sequential code.
6. Create a function named `displayArray`, and add parameter, `array`.
7. Within the function, create a variable name `output`, and assigned with an empty String.
8. Add a `return` for the function to show the `output`.
9. We will now add a "for" loop to iterates over the array. Enter the following code:

```
   for (var i = 0; i < array.length; i++) {
                output += array[i] + " ";
                }
```

10. Display the result in the console by calling the function and passed in "`sentence`" as its argument. Test in the browser.
11. However, we would also want to end the statement with an exclamation mark. Add an if statement to check if this is the last element and append an exclamation mark to the end of the output string. (Hint: use array.length).
12. Test in the browser again. However, there is a space before the exclamation mark.
13. Now we will use an else statement with the if to get rid of the space if it is the last entry in the array.
14. Test in the browser again. You should see the console as follow:

```
Luke I am your father!
```

Exercise 2 Using object literal

1. Navigate to 05_JavaScript_for_Developer/starter folder. Open object.html.

2. Within the script block, create a new object name objStudent with the following object literal:

```
{name: "Harry Potter", address: "4 Privet Drive", email:
"hp@hogwarts.ac.uk"}
```

3. Print out the name of the student in the console with the following syntax:

```
objectName["propertyName"]
```

4. A great way to extend objects with new value at any point is known as “expando property”. Add the following code:

```
objStudent.schoolHouse = "Gryffindor"; //expand property
```

5. Print `objStudent` in the console to test a new property is added.

6. Using a `for in` loop, print out the following in the console:

```
name : Harry Potter
address : 4 Privet Drive
email : hp@hogwarts.ac.uk
schoolHouse : Gryffindor
```

06 – Advanced JavaScript

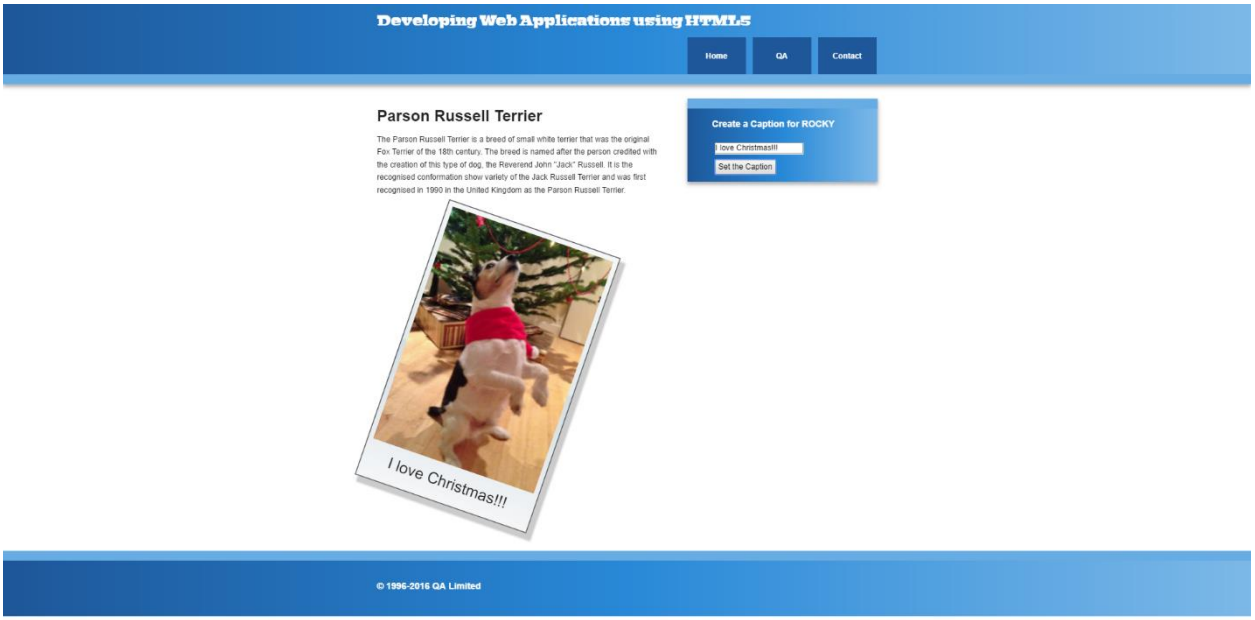
Objectives

This practical session you will be exploring the building blocks of advanced JavaScript.

Overview

This practical session contains two exercises:

- In Exercise 1 you will be using a Polaroid picture figure, and will allow a user to set the caption of the picture with event driven JavaScript and a simple JavaScript Object
- The final web page will look similar to the following figure:



- Exercise 2 (optional) you will set a different gradients depending upon the day of the week for the header and footer of the same web page. You will need to use a switch statement with a Date object, and will be using HTML5 JavaScript API that allows us to easily add and remove CSS classes

Exercise 1 exploring event driven JavaScript

1. Navigate to 06_Advanced_JavaScript/starter folder. Open polariod-caption.html.
2. Locate <figcaption>, and set its id to caption1.
3. Locate <aside>, towards the bottom of the HTML. Nested within is an input box and a button. Set the input with an id captionText and the button with an id setCaption.
4. Navigate to 06_Advanced_JavaScript/starter/js folder. Open main.js.
5. Create a JavaScript object called photoEditor.
6. Now extend the photoEditor object with a property of setCaption which points to a function.

```
photoEditor.setCaption = function () {

}
```

    1. Set two parameters for the function:
        a. `captionID` – provide the ID of the element we wish to change the text value of
        b. `captionTextID` – the text we wish to insert
    2. Within the function, add the following code and missing methods:

```
var captionID = document.???(captionID);

captionID.innerText = document.???(captionTextID).value;
```

    3. Next, create a variable named `setCaption` and use `document.querySelector` to point to the button with ID of `setCaption`.
    4. Enter the code below to associate the click event of the button to the `setCaption` function. What is the missing event handler?

```
setCaption.???('click', function () {

    photoEditor.setCaption('caption1', 'captionText');

}, false);
```

    5. Save the HTML page, and open it in Chrome. Enter your caption into the text box and press the button. The caption of the figure should update.

Exercise 2 setting a gradient based upon the day of the week

1. Navigate to 06_Advanced_JavaScript/starter folder. Open polariod-caption-gradient.html.

2. Examine the HTML code with class, header-container and footer-container, where currently we apply the background gradient with a class called “gradients”.

3. Navigate to 06_Advanced_JavaScript/starter/css folder. Open gradients.css. Examine the different CSS written for each day of the week.

4. Navigate to 06_Advanced_JavaScript/starter/js folder. Open gradient.js.

5. Select all the elements using “gradients” class, and assign it to the variable shadedItems. What is the missing method?

```
var shadedItems = document.???('.gradients');
```

6. Use `console.dir` to examine the array, and see what we have selected.

7. Next, to find the day of the week we will use JavaScript object called `Date`. The Date object has a function called `getDay()` that returns a value between 0 and 6, where 0 is Sunday and 6 is Saturday. Enter the following code to create the `Date` object and then invoke the `getDay()` function.

```
var theDate = new Date();

var theDay = theDate.getDay();
```

8. We need to ensure that every element we selected has its class changed whatever the day of the week. We will use a for loop iterates using an integer counter.

```
for (var i = 0; i < shadedItems.length; i++) {

}
```

9. Within the `for` loop, we will be using HTML5 API for adding and removing CSS classes. Each element has a `classList` object that includes methods for adding and removing classes. If the day of the week is not Sunday, we want to remove the gradient class so that we can replace it with a day specific gradient:

```
  if (theDay > 0) {

        shadedItems[i].classList.remove('gradients');

    }
```

10. Add an if statement that checks the day of the week and applies a class. Use the code below as a guide to add additional if statements that apply the correct class for Tuesday through Sunday.

```
    if (theDay === 1) {
          shadedItems[i].classList.add('monGradient');
     }
```

11. Save all the files, and test the web page in Chrome. A different class should now be applied.

07 – Introducing jQuery

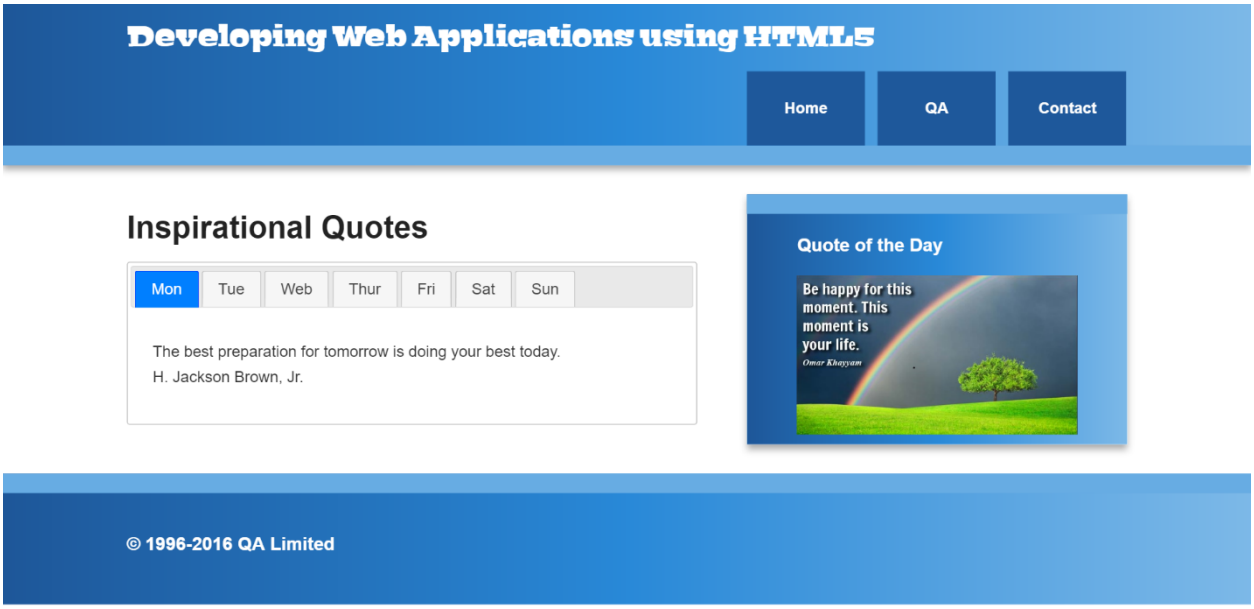
Objectives

In this practical session you will be using jQuery and its library: jQuery UI. You will investigate how to simplify JavaScript programming and use it to simplify cross-browser development.

Overview

This practical session contains two exercises:

- In Exercise 1 we will animate the aside, allowing it to scroll up and down based upon a mouse click
- In Exercise 2 we will be using the jQueryUI library to turn the body of the page into a tab system
- The final web page will look similar to the following figure:



Exercise 1 using jQuery

1. Navigate to 07_Introducing_jQuery/starter folder. Open jquery.html.

2. The entire HTML you are going to need is currently in place, including the link to the jQuery library. Scroll down to the script tags and you will notice the following mark-up:

```
  <script src="https://code.jquery.com/jquery-
1.12.0.min.js"></script>

        <script>window.jQuery || document.write('<script
src="js/vendor/jquery-1.12.0.min.js"><\/script>')</script>
```

3. Navigate to 07_Introducing_jQuery/starter/js folder. Open main.js.

4. We are going to use jQuery library to animate the aside allowing it to slide up and down based upon a mouse click. Firstly, we need to ensure jQuery only executes after the DOM has loaded. Add the following code:

```
$(function () {

});
```

5. Next, we will make the `div` within `aside` slide away as the page loads. What is the missing selector?

```
$('??').slideUp();
```

6. Save the page, and view it in Chrome.

7. To allow the div to slide back down, we will use the `on()` function to hook up an event. Add the following code.

```
   $('aside').on('click', function () {

        $(this).children('div').slideDown();

    });
```

8. Save the page again, and view in it in Chrome. However, it does not slide back up.

9. Change the `slideDown` function call to be:

```
slideToggle(2000);
```

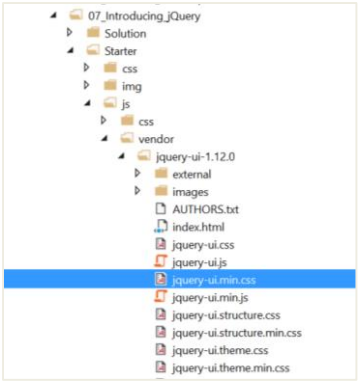
10. Save the page, and view it in Chrome. jQuery will now work out if the div is slide up or down and reverse the animation.

Exercise 1 Using jQueryUI

11. Navigate to 07_Introducing_jQuery/starter folder. Open jquery.html.

12. Examine the with an id of “mytabs”. Each tab has an ID and is referenced by an appropriate href in the <ul>.

13. jQueryUI has two parts, a CSS file and a JavaScript file. Both are needed. Within the head of the html. Locate <!--Add jQueryUI CSS here-->. Add a reference to the jQueryUI CSS file.



14. Locate `<!--Add jQueryUI here-->` which is be beneath the jQuery file reference. Add a reference to the jQueryUI file.

15. Navigate to 07_Introducing_jQuery/starter/js folder. Open main.js.

16. With all the mark-up in place, add this single line of code:

```
$('#mytabs').tabs();
```

17. Save the page, and view it in Chrome. You should now have a tabbing system hooked up and working.

08 – HTML5 Forms

Objectives

In this practical session you will be using HTML5 form control to build a payment form.

Overview

This practical session contains two exercise:

- In Exercise 1 you will be building a web form with HTML5 form controls, and also style it with CSS3 pseudo-classes
- In Exercise 2 you will use JavaScript to customise validation message for the name input field
- The final web form will look similar to the following figure

Exercise 1 Building a HTML5 payment form

1. Navigate to **08_HTML5_Forms/starter** folder.
2. Open **index.html** and **formStyle.css** in the CSS folder.
3. In **index.html,** locate <!--Add Form Style here-->. Link **formStyle.css.**
4. Create a form element beneath <h2>Please fill in your details below</h2> and give it an id of payment.
5. Fieldsets allow you to group the form controls and improve the user interface. Add fieldset element, and immediately after the opening tag, add a legend element with a text node “Your details”.
6. Next create an ordered list and add a <li> within. Inside add the following code:

```
 <label for="name">Name</label>
<input id="name" name="name" type="text >
```

7. Now add some HTML5 functionalities to the form. Add the attribute *autofocus* to the name field. Save the file, and test it in Chrome.
8. Make the field mandatory with a *required* attribute.
9. Before the end of the form, add another fieldset element. Within it, add a submit button, so that we can test the form submission.
10. Inside the first fieldset, repeat the previous steps. Add an input element with a name, id and type of *email*. Add a placeholder attribute with a value of “*you@domain.com*” and a *required* attribute.
11. Repeat once more, adding an input of type *tel* with a name and id of *phone*. Make it mandatory, and add a placeholder with a value of “Eg. +447800000000”.
12. This will give us the basis of the form. To save you typing, open *formFilelds.txt* from the starter folder. Open and paste the code below the first fieldset.
13. Save the files, and test it in Chrome. You should see a fully functional HTML5 form.
14. Next, we will style invalid elements using CSS3 pseudo-classes. Open *formStyle.css.* At the end of the CSS, add an invalid pseudo-class for all input and textarea with the following style.

```
background-color: #ffd4d4;
border: 1px solid maroon;
```

15. Save the file, and test it in Chrome.

Exercise 2 customising the validation message (optional)

In this exercise, you will customise the payment form name field with its own validation message.

1. Navigate to 08_HTML5_Forms/Starter/js folder.

2. Open main.js. Add the following code, and fill in the missing id for name.

```
var fldName = document.getElementById('?');
```

3. We will use ***setCustomValidity*** property and ***oninvalid*** event to create a custom message for the name input field. Add the following code, and fill in the missing event with your own custom message.

```
fldName.??? =
 function () {
     fldName.setCustomValidity("");
     if (!fldName.validity.valid) {
          fldName.setCustomValidity(
           "???"
          );
     }
 };
```

4. Save the files, and test the form. You should get your own custom message when you submit the form.

09 – Geolocation

Objectives

This practical session you will be using Google Maps Geolocation API to create a web page that locate your location.

Overview

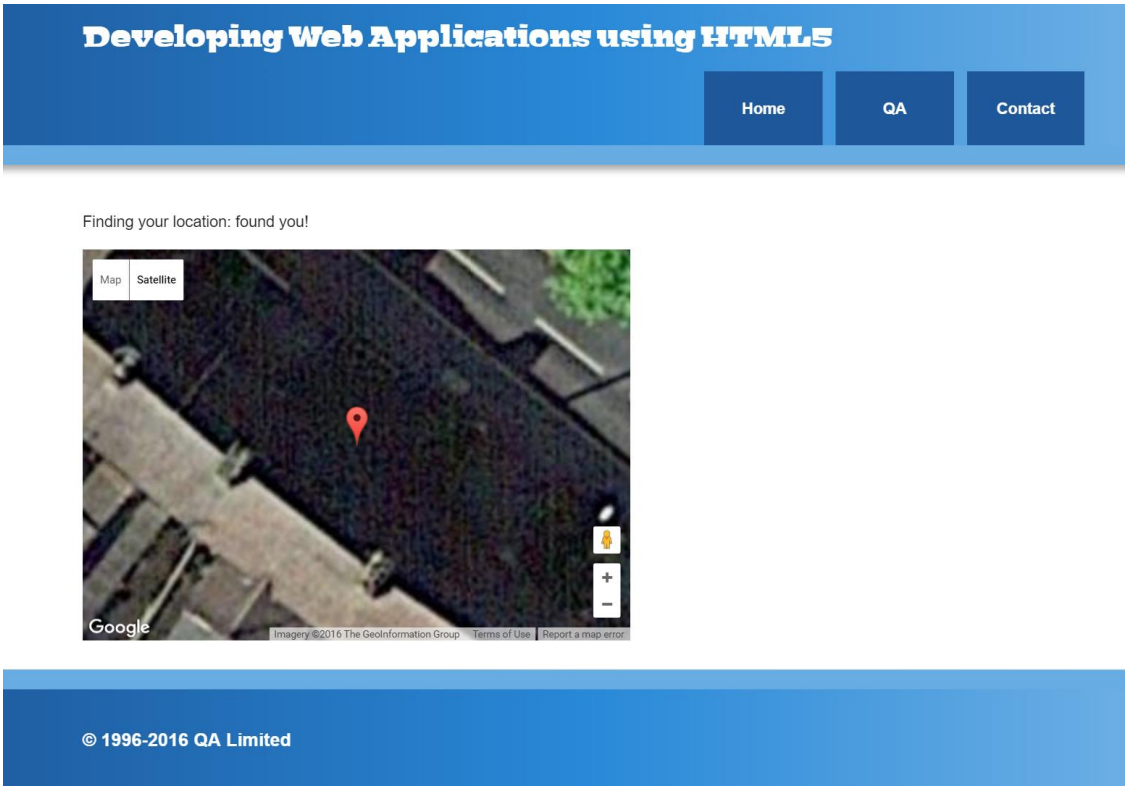
This practical session contains two exercise:

- Exercise 1 you will be building an application that will show your location on a web page

Note: you would need a Google Map API key for this exercise. Please obtain one from the following web address:

https://developers.google.com/maps/documentation/geolocation/get-api-key

The final web page looks similar to the following figure.



Exercise 1 using Google Maps geolocation API

1. Navigate to 09_Geolocation/starter folder. Open locate.html.
2. In locate.html, locate `<!--insert article here-->`. Insert an article element. Within it, create text node that reads “Finding your location: checking...”
3. Wrap the text “checking” with a span element of an id “status”.
4. Locate `<!-- insert your API key-->`. Insert the Google Map API key within the script tag in “YOUR_API_KEY_HERE”.
5. Next, navigate to 09_Geolocation/starter/js folder. Open main.js.
6. In main.js, create a self-executing anonymous function.

```
(function(){
  //Normal code goes here.
})()
```

7. Within the function, add the following code. Fill in the missing method, and the properties of its third optional parameter.

```
if (navigator.geolocation) {
   navigator.geolocation.??(success, error, {??});
    } else {
        error('Your browser does not support geolocation');
    }
```

8. Next, create a function named “success” and pass a “position” object as parameter.
9. Within the “success” function, we will select the span element with the id “status” and assign it to a variable “s”. (Hint: use querySelector.)
10. Now parse the string “*found you*” into the HTML element using the quick and dirty way. What is the missing function?

```
s.?? = “found you!”;
```

11. Immediately after this, we will create a map canvas for the google map. We will create a "`div`" element, and with an id '`mapcanvas`'. The height will be `400px` and with `560px`. Fill in the missing code.

```
var mapcanvas = document.createElement('??');

        mapcanvas.id = '??';

        mapcanvas.style.height = '??';

        mapcanvas.style.width = '??';
```

12. We will next append the mapcanvas to the article element. What is the missing method?

```
      document.querySelector('article').??(mapcanvas);
```

13. Add the following code, and fill in the missing properties for the coordinate object.

```
var latlng = new google.maps.LatLng(position.coords.?,
position.coords.?);
```

14. We will now create an object and assign it to the variable "`myOptions`". Within it, we will use the following options for our map.

```
            zoom: 25, //The initial Map zoom level.

            center: latlng,    //The initial Map center.

            mapTypeControl: true,    //The initial
enabled/disabled state of the Map type control.

            navigationControlOptions: { style: 'DEFAULT',
position: 'TOP_RIGHT' },

            mapTypeId: google.maps.MapTypeId.SATELLITE    //
displays Google Earth satellite images
```

15. Next we will create a new map inside the "`mapcanvas`" container. The second argument will be our map options.

```
var map = new google.maps.Map(document.getElementById("??"),
??);
```

16. We will also create a new google map `Marker` class and assign it the variable `marker`.

```
var marker = new google.maps.??({

        });
```

17. The marker will have the following options.

```
            position: latlng,
            map: map,
            title: "You are here!"
```

18. Lastly, we will create a function named “error” with a parameter “msg”. Within it, add the following code.

```
        var s = document.querySelector('#status');
        s.innerHTML = typeof msg == 'string' ? msg : "failed";
```

19. Save all your files, and check `locate.html` in Chrome. You should see the google map on the web page now.

10 – Drag and Drop

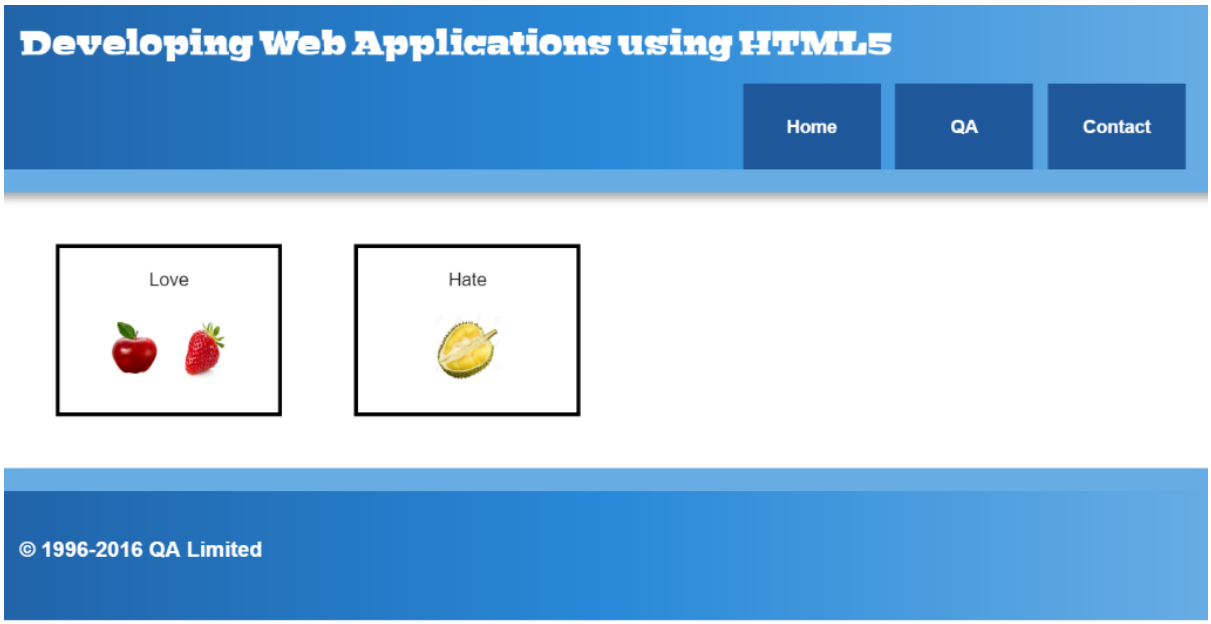
Objectives

This practical session you will be using the Drag and Drop API to make images draggable and to drop into another area.

Overview

This practical session contains two exercise:

- Exercise 1 you will be building a web application that allow you to drag some fruits images and drop them into a love and hate container
- The final web application looks similar to the following figure



Exercise 1 building a drag and drop application

1. Navigate to `10_Drag_and_Drop/starter` folder. Open `container.html`. You should see a love and hate container, and three fruits images beside them. We want to be able to drag and drop the fruit images into the love and hate container.

2. First, set all the fruit images with `draggable` attribute to value `true`.

3. The images will also listen to the `dragstart` event. Add the following code to all the fruit images.

```
ondragstart="drag(this, event)"
```

4. Next, navigate to 10_Drag_and_Drop/starter/js folder. Open main.js.

5. Define the JavaScript function that will run when we start to drag the image element.

```
  function drag(target, e) {
    }
```

6. Within this function, first declare what type of effect we allow in the operation, and set that to move. What is the required method?

```
e.dataTransfer.? = '';
```

7. Then set the data for the operation. In this case the type is `text` and the value is the ID of the element we are dragging. What is the missing method?

```
e.dataTransfer.?('', target.id);
```

8. We will use the `setDragImage` method to set what we will be dragging and then where the cursor will be while dragging.

```
e.dataTransfer.setDragImage(e.target, 20, 20);
```

9. Return to `container.html`. In order for an element to accept a drop it needs to listen to three different events: `dragEnter`, `dragOver` and also the `drop` event. Add the following code to the love and hate `section`.

```
ondrop="drop(this, event)" ondragenter="dragEnter(event)"
                        ondragover="return dragOver(event)"
```

10. We have added event listeners we need, next we will create these functions. We will start by the `dragenter` and `dragover` events. Return to `main.js` and add the following function. In this function we define what we want to happen when the element we are dragging reaches the element it is supposed to be dropped in, in this case we also want to prevent the default behaviour of the browser. What is the missing method?

```
 function dragEnter(ev) {
      event.??;
      return true;
  }
```

11. Next in the `dragOver` function we simply prevent the default to allow for the drop.

```
function dragOver(ev) {
      event.??;
  }
```

12. The next part is where we define the function for when we actually drop the element on the desired target.

```
function drop(target, e) {
  }
```

13. Within the function, firstly set a variable called data in which we get all the data that is available for the `text` format.

```
var id = e.dataTransfer.getData('??');
```

14. And then we append that data which will be the element that we are dragging to the section where we wish to drop the element.

```
target.appendChild(document.getElementById(id));
```

15. Finally, some final touches to prevent the default behaviour of the browser.

```
e.??;
```

16. Save all the files, and test in Chrome. You should be able to drag and drop the fruit images into the love or hate container.

11 – AJAX

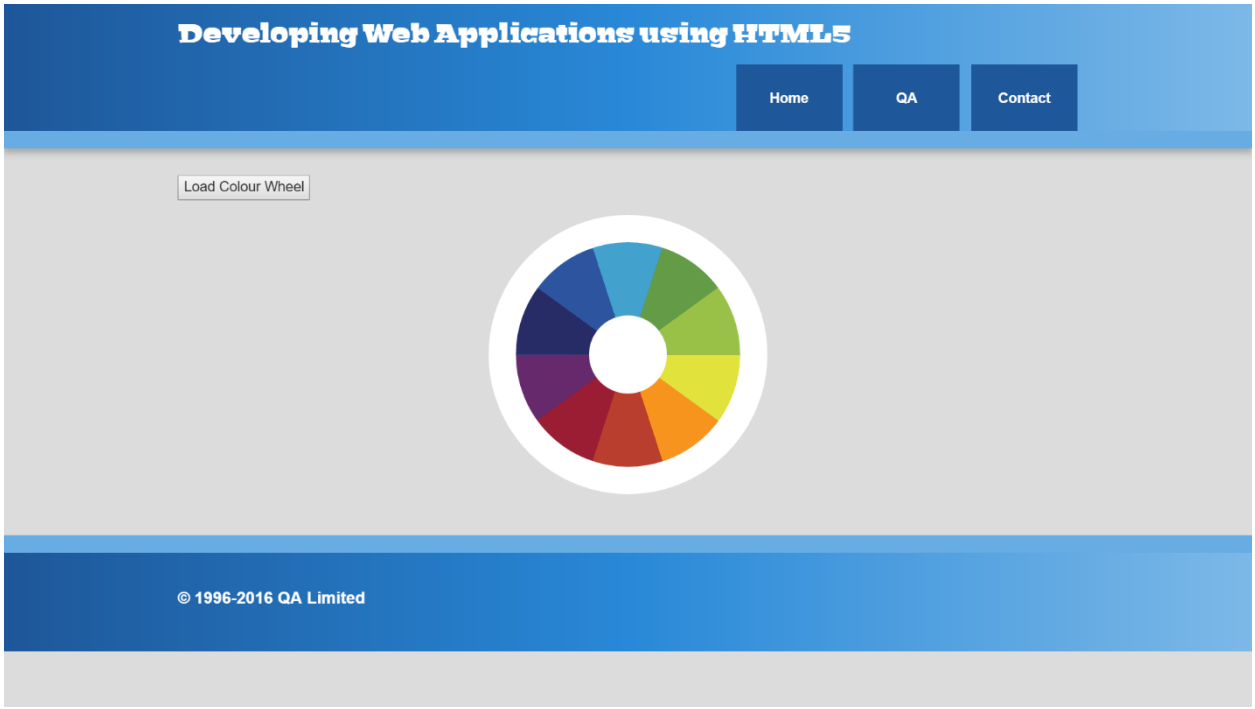
Objectives

In this practical session you will be using the Fetch API to load a colour wheel page.

Overview

This practical session contains two exercise:

- In Exercise 1 you will be loading a colour wheel page using the Fetch API
- The final web application looks similar to the following figure



Exercise 1 Loading a colour wheel page using the Fetch API

1. Navigate to `11_AJAX/starter` folder. Open `index.html`.

2. Locate `<!--load colour wheel here -->`.
3. Create a `button` with `id=loadWheel` and text, `Load Colour Wheel`.
4. Below the `button`, create a `div` with `id=output`.
5. Navigate to `11_AJAX/starter /js` folder. Open `main.js`.
6. Create a self-executing function.

```
(function () {
})();
```

7. Using the fetch() API, load `colourWheel.html`, and assign it to a new variable `result`.

```
var result = ??('colourWheel.html');
```

8. Add the following code to return the `Promise`, and return the response as text. Use `console.log` to check the `response` and the `header`.

```
   result.then(function (response) {
        console.log('response', response);
        console.log('header', response.headers.get('Content-
Type'));
        return response.text();
    })
```

9. Chain the `Promise` together.

```
.then(function (text) {
    })
```

10. Within the above `Promise`, add the following code:

```
        p.addEventListener("click", function () {
            var output = document.querySelector('#output');
            output.innerHTML = text;
        })
```

11. The catch statement will execute if there are any errors:

```
.catch(function (ex) {
```

```
        console.log('failed', ex);
    });
```

12. Pass the selection of the button as a parameter into the second parenthesis of the self-executing function.

```
 document.querySelector("#loadWheel")
```

13. Please note that you will also need to add the parameter to the anonymous function.

```
(function (p) {
})();
```

14. Save all the files, and load `index.html` in Chrome. Click the button, and it should load the `colourWheel.html` page.

12 – Web Worker

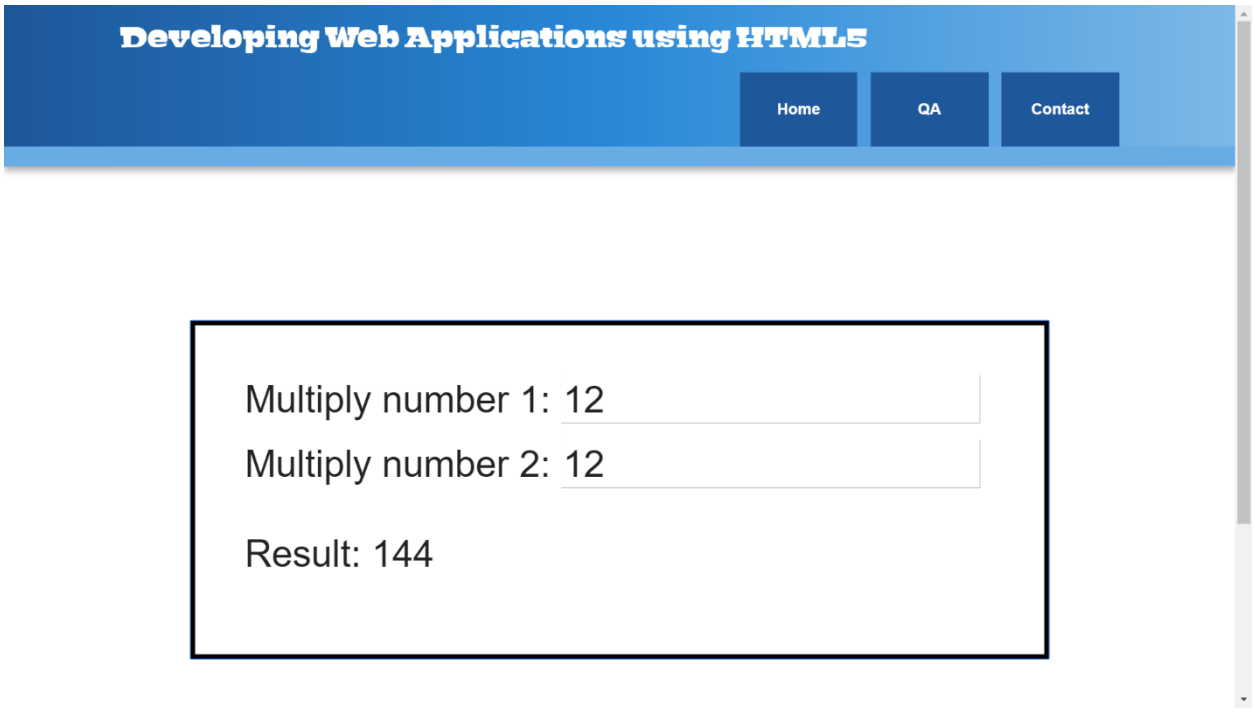
Objectives

This practical session you will be using web workers to do some multiplication in the backgrounds.

Overview

This practical session contains two exercise:

- Exercise 1 you will be creating a simple web application which allows you to enter two numbers to be multiplied together. The numbers are sent to a dedicated worker, multiplied together, and the result is returned to the page and displayed.
- The final web application looks similar to the following figure



Exercise 1 Web Worker multiplication

1. Navigate to `12_Web_Worker/starter` folder. Open `index.html`. Locate
2. `<!--Input for number-->`. The input boxes for number and output box for result have been created for you.
3. Navigate to `12_Web_Worker/starter/js` folder. Open `main.js`.
4. Select the two input boxes and output box using `document.querySelector`, and assign them with variables `first`, `second` and `result` respectively.
5. Next, check whether the browser supports the Web Worker API.

```
if (window.Worker) {
}
```

6. Create a new `Worker` object and assign it to `myWorker` variable. Specify the url of the JavaScript file. (`js/worker.js`)
7. Add the following code for the first input box. When the value of the input box is changed, `postMessage` is used to send the value inside to the worker as an array. Console.log is used to check message is posted to worker.

```
first.onkeyup = function () {
        myWorker.postMessage([first.value, second.value]);
        console.log('Message posted to worker');
 };
```

8. Do the same for the second input box.
9. Navigate to `12_Web_Worker/starter/js` folder. Open `worker.js`.

10. In the worker, we can respond when the message is received with the following code. The `onmessage` handler allows us to run some code whenever a message is received, with the message itself being available in the message event's data attribute. Here we simply multiply together the two numbers then use `postMessage()` again, to post the result back to the main thread.

```
 onmessage = function(e) {
  console.log('Message received from main script');
  var workerResult = 'Result: ' + (e.data[0] * e.data[1]);
  console.log('Posting message back to main script');
  postMessage(workerResult);
}
```

11. Open `main.js` again. Add the following code. Here we grab the message event data and set it as the `textContent` of the result paragraph, so the user can see the result of the calculation.

```
myWorker.onmessage = function (e) {
        result.textContent = e.data;
        console.log('Message received from worker');
    };
```

12. Save all files, and test the files in Chrome. View the messages in the console log.

13 – Web Socket

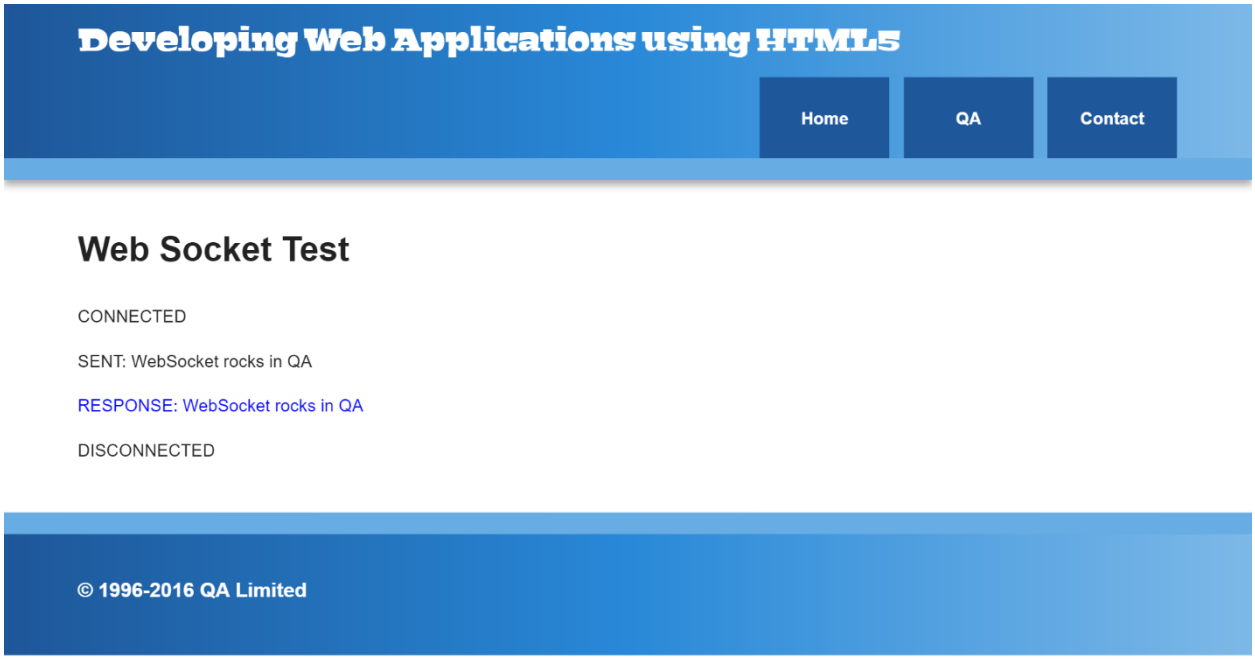
Objective

This practical session you will be using web socket to establish a connection and send message.

Overview

This practical session contains one exercise:

- Exercise 1 you will be creating a simple web application that will automatically connect using web socket, send a message, display the response, and close the connection
- The final web application looks similar to the following figure



Exercise 1 Using Web Socket

1. Navigate to `13_Web_Socket/starter` folder. Open `socket.html`.
2. Locate `<article>` element within `section`. Set it with an id "`output`".
3. Navigate to `13_Web_Socket/starter/js` folder. Open `main.js`.
4. To save time from setting up a web socket server, we will be using the server from websocket.org.

```
var wsUri = "ws://echo.websocket.org/";
```

5. Next, create a variable for `output`.
6. Enter the following code for `init()` function. What is the missing method?

```
function init() {
    output = document.??("output");
    testWebSocket();
}
```

7. Now, create the `testWebSocket()` function.

```
function testWebSocket() {
}
```

8. Within the function, create a new `WebSocket` object, specifying the url to connect to.

```
websocket = new WebSocket(wsUri);
```

9. WebSocket object has the following properties for handling web socket events:
    a. onopen -- indicate the connection has been opened, so you can start sending data to the server
    b. onclose == indicate the connection has closed
    c. onmessage = indicates a message has arrived from the server
    d. onerror = indicates an error has occurred

```
  websocket.onopen = function (evt) { onOpen(evt); };
```

10. Using the above code as a guide, create similar codes for `onclose`, `onmessage` and `onerror`.

11. Now, we will create the `onOpen()` function which will include two callback functions, `writeToScreen()` and `doSend()`.

```
function onOpen(evt) {
    writeToScreen("CONNECTED");
    doSend("WebSocket rocks in QA")
}
```

12. The `onClose()` function will include one callback function, `writeToScreen()`.

```
function onClose(evt) {
    writeToScreen("DISCONNECTED");
}
```

13. The `onMessage()` function includes the `writeToScreen()` callback function, and also the `close()` method from web socket object that will close the `WebSocket` connection. `evt.data` allow us to access the whole message from the main page.

```
function onMessage(evt) {
    writeToScreen('<span style="color: blue;">RESPONSE: ' +
evt.data + '</span>');
    websocket.close();
}
```

14. The `onError()` function include the `writeToScreen()` callback function.

```
function onError(evt) {
    writeToScreen('<span style="color: red;">ERROR: ' + evt.data
+ '</span>');
}
```

15. Next, we will create the `doSend()` callback function. This function includes `writeToScreen()` callback function, and also invoke `send()` method on the `WebSocket` object, where the data is buffered and sent asynchronously.

```
function doSend(message) {
    writeToScreen("SENT: " + message);
    websocket.send(message);
}
```

16. Now we will create the `writeToScreen()` callback function which will write the message on the screen. We create a new `p` element and assign it with a variable `pre`. The `word-wrap` property allows long words to be broken and wrap onto the next line, and the `break-word` value allows unbreakable words to be broken. We then return the message as HTML using `innerHTML` and append it to the `output`.

```
function writeToScreen(message) {
    var pre = document.createElement("p");
    pre.style.wordWrap = "break-word";
    pre.innerHTML = message;
    output.appendChild(pre);
}
```

17. Lastly, we will load the `init()` function.

```
window.addEventListener('load', init, false);
```

18. Save the file, and test it in Chrome. The web application should automatically connect using web socket, send a message, display the response, and close the connection.

14 – Client side storage

Objectives

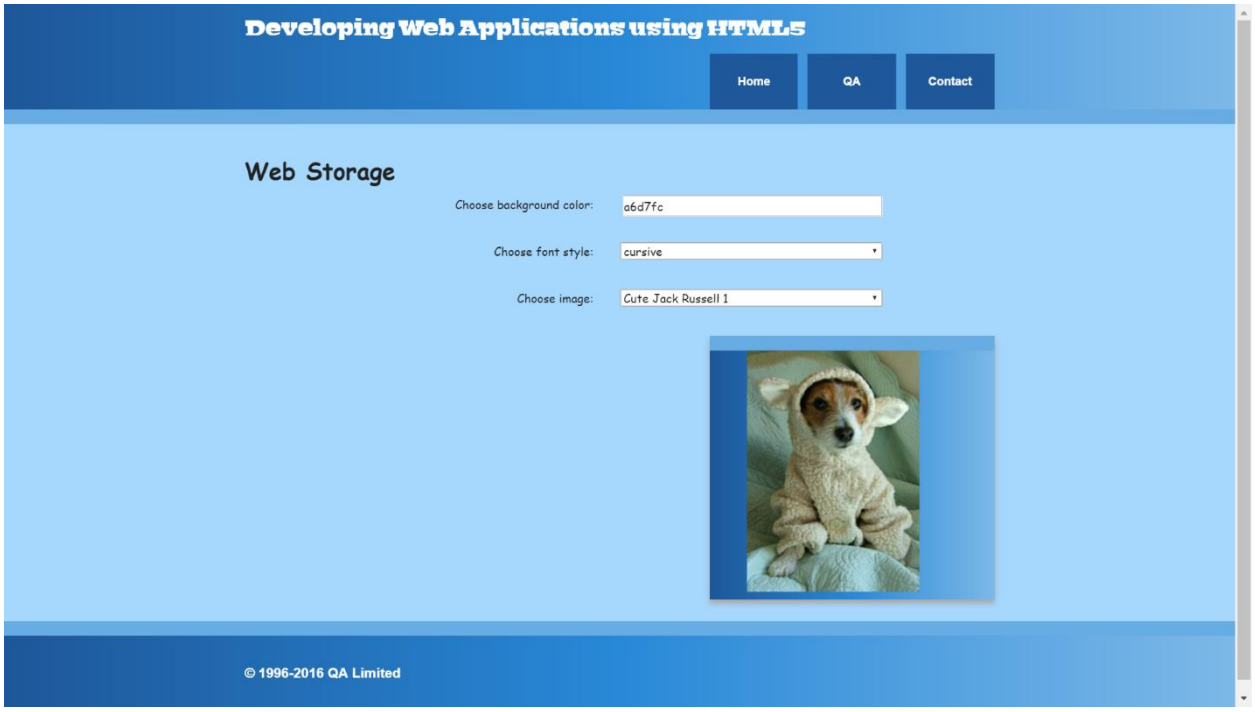
In this practical session you will learn to use web storage to store your choices for styling the landing page.

Overview

This practical session contains one exercise:

- In Exercise 1 you will create a landing page which provides controls which you can use to customise the colour, font and decorative image. When you choose different options, the page is instantly updated. In addition, your choices are stored in localStorage, so that when you leave the page and load it again later on, your choices are remembered

- The final web application looks similar to the following figure



EG_11_AJAX.docx

Exercise 1 web storage

1. Navigate to `14_Client_Side_Storage/starter` folder. Open `index.html`. Locate `<!-- Web Storage-->`. A form with various customised controls have been created for you.

2. Navigate to `14_Client_Side_Storage/starter/js` folder. Open `main.js`.

3. Select the class `main-container` with `document.querySelector`, and assign with the variables, `mainContainer`.

4. Repeat for the `main` class and the `img` element, and assign with `main` and `imgElem` variables respectively.

5. Select the `input` with `id=bgcolor`, assign with variable `bgcolorForm`. Repeat for `select` with `id=font` and `id=image`, and assign them with variables `fontForm` and `imageFont` respectively.

6. Add the following code where we will test whether the storage object has already been populated (i.e. the page was previously accessed). The `Storage.getItem()` method is used to get a data item from storage. In this case we are testing to see whether the `bgcolor` item exists; if not, we run `populateStorage()` to add the existing customisation values to the storage. If there are already values there, we run `setStyles()` to update the page styling with the stored values.

```
if (!localStorage.getItem('bgcolor')) {
    populateStorage();
} else {
    setStyles();
}
```

7. Now, we will create the `setStyles()` function:

```
function setStyles() {
}
```

8. We will first grab the values from local storage.

```
    var currentColor = localStorage.getItem('bgcolor');
```

9. Do the same for `font` and `image`.

10. Next, we set the values displayed in the form elements to those values, so that they keep in sync when you reload the page.

```
bgcolorForm.value = currentColor;
```

11. Do the same for `font` and `image`.

12. Finally, we update the styles/decorative image on the page, so your customisation options come up again on reload.

```
    mainContainer.style.backgroundColor = '#' + currentColor;
    main.style.fontFamily = currentFont;
    imgElem.setAttribute('src', currentImage);
```

13. Next, we will create the `populateStorage() function`.

```
function populateStorage() {
}
```

14. The populateStorage() function sets three items in local storage – the background colour, font and image path.

```
localStorage.setItem('bgcolor', bgcolorForm.value);
```

15. Do the same for `font` and `image`.

16. It then runs the `setStyles()` function to update the page styles.

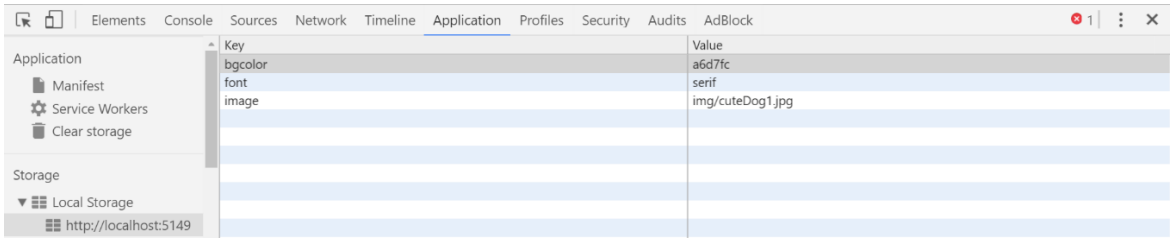
```
   setStyles();
```

17. Lastly, we will include an `onchange` handler on each form element, so that the data and styling is updated whenever a form value is changed:

```
bgcolorForm.onchange = populateStorage;
```

18. Do the same for `fontForm` and `imageForm`.

19. Save all files, and test it in Chrome. Check the local storage in Chrome debugger under “Application”. Close the browser and open the same page again, your choices are remembered.



15 – SVG

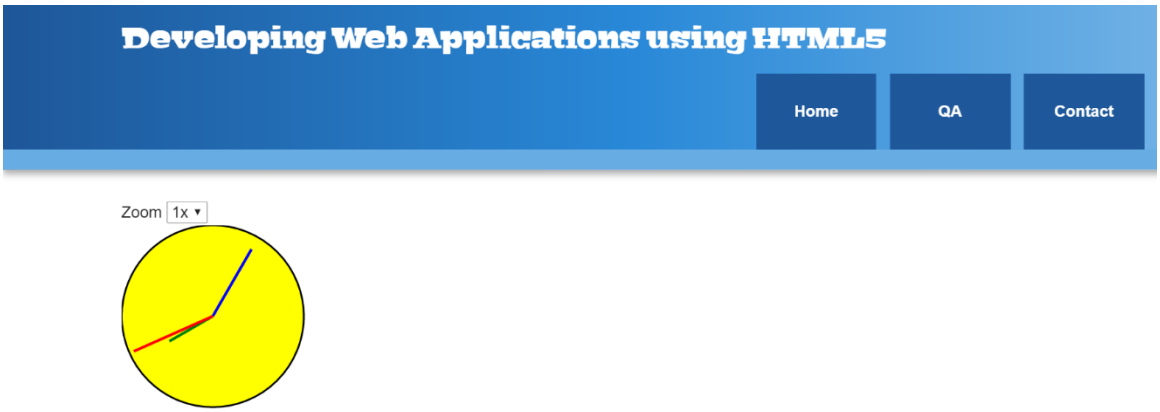
Objectives

In this practical session you will be using the Scalable Vector Graphic to create a clock and add interactivity using JavaScript.

Overview

This practical session contains two exercise:

- In Exercise 1 you will be building a clock web application
- The clock web application looks similar to the following figure



Exercise 1 building a clock web application

1. Navigate to `16_SVG/starter` folder. Open `clock.html`. A drop-down selection menu has been built for you to select the zoom level of the clock.

2. Locate `<!--set svg element here-->`. Add svg element with the viewBox attribute and value "0 0 640 480".

3. Next, add two `<g>` elements so we can apply properties to a group of element.

4. In the first `<g>` element, add a `circle` element with the following properties: fill = yellow, x-coordinate = 100, y-coordinate = 100, radius = 100, and stroke-width = 2px.

5. Locate `line.txt` in starter folder. Copy and paste the code just below the circle element. Can you figure out what these codes are doing?

6. In the second `<g>` element, add three `line` elements.

```
<line x1="100" y1="100" x2="100" y2="45" stroke-width="6px"
stroke="green" id="hourhand" />

<line x1="100" y1="100" x2="100" y2="15" stroke-width="4px"
stroke="blue" id="minutehand" />

<line x1="100" y1="100" x2="100" y2="5" stroke-width="2px"
stroke="red" id="secondhand" />
```

7. Save the file, and view it in Chrome. You should just see a yellow clock with a red second hand.

8. Next, navigate to `16_SVG/Starter/js` folder. Open `main.js`.

9. Create a self-executing anonymous function and assign to a `CLOCK` variable.

10. We will be using JavaScript Object Oriented Style to create the rest of the function. Create an anonymous function and assign it to `drawClock` variable.

11. Within it, add the following code, which help to set the zoom level of the clock.

```
var INITIAL_R = 100;

var zoom = document.getElementById("rangeinput").value;

var r = INITIAL_R * zoom;
```

12. Next, we will draw a circle for the clock with the following code. What are the missing attributes?

```
        var circle = document.getElementById("circle");
        circle.setAttribute('?', r);
        circle.setAttribute('?', r);
        circle.setAttribute('?', r);
```

13. Following the above, let's draw the 12 hours' hand with code below.

```
  for (var i = 0; i < 12; i++) {
            var hour = document.getElementById("hour" + i);
            var degrees = i * 30;
            hour.setAttribute('x1', getX(degrees, r, 0.9)); //
90% of radio's length.
            hour.setAttribute('y1', getY(degrees, r, 0.9)); //
90% of radio's length.
            hour.setAttribute('x2', getX(degrees, r));
            hour.setAttribute('y2', getY(degrees, r));
        }
```

14. We will now create the `drawHands` function. Create an anonymous function and assign it to `drawHands` variable.

15. Within it, add the following constants for hand's sizes.

```
        var SECONDS_HAND_SIZE = 0.95,
        MINUTES_HAND_SIZE = 0.85,
        HOURS_HAND_SIZE = 0.55;
```

16. Next, we will draw the circle. What are the missing methods?

```
       var circle = document.?? ("circle");
```

17. Following the above, add the code below for the clock circle's properties.

```
        var r = circle.getAttribute('r'),
        cx = parseInt(circle.getAttribute('cx')),
        cy = parseInt(circle.getAttribute('cy'));
```

18. Create a JavaScript `Date` instance now and assign it to `currentTime` variable.

19. Next, create a `drawHand` function with the following parameters: hand, value, size, degrees. Within it, add the following code:

```
            var deg = degrees * value;
            x2 = getX(deg, r, size, cx),
            y2 = getY(deg, r, size, cy);

            hand.setAttribute('x1', cx);
            hand.setAttribute('y1', cy);
            hand.setAttribute('x2', x2);
            hand.setAttribute('y2', y2);
```

20. Now, call the drawHand functions three times respectively for the `secondhand`, `minutehand` and the `hourhand`.

```
drawHand(document.getElementById("??"),
            currentTime.getSeconds(),
            SECONDS_HAND_SIZE,
            6);
      drawHand(document.getElementById("??"),
            currentTime.getMinutes(),
            MINUTES_HAND_SIZE,
            6);
      drawHand(document.getElementById("??"),
            currentTime.getHours(),
            HOURS_HAND_SIZE,
            30);
```

21. Open `function.txt` from the `starter` folder. Copy and paste the function `getX, getY`, and `getRad` after the `drawHands` variables.

22.Right after this, add the following return statement and the functions within.

```
  return {
        init: function () {
            drawClock();
            setInterval(drawHands, 1000);
        },
        zoom: function () {
            drawClock();
        }
    };
```

23. Before we finish, add this at the end of the code.

```
CLOCK.init();
```

24. Save the files, and test in Chrome. You should see a working web clock.

16 – Canvas

Objectives

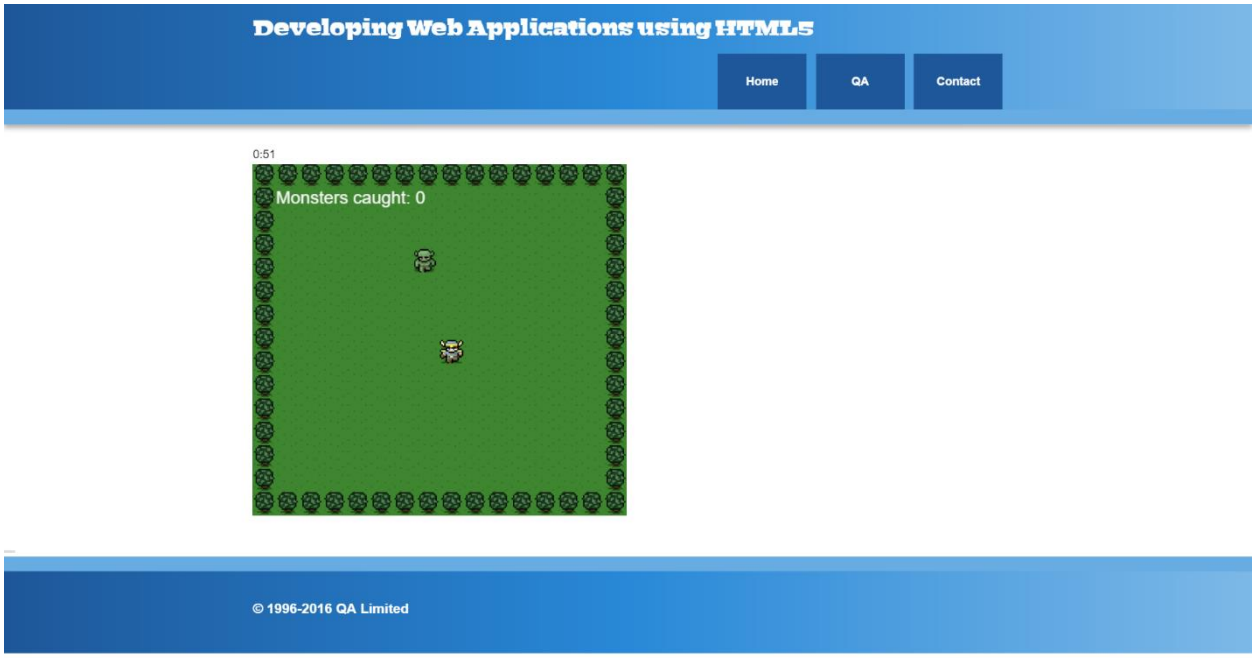
This practical session you will be using the HTML5 Canvas and JavaScript to create a “Catch the Monster” game.

Overview

In this practical session contains two exercise:

- **In Exercise 1 you will be building a “Catch the Monster” Game**
- **In Exercise 2 you will add a count-down timer to the “Catch the Monster” Game (optional)**

The “Catch the Monster” game looks similar to the following figure.



Exercise 1 building the “Catch the Monster” game

1. Navigate to `16_Canvas/starter/js` folder. Open `main.js`.
2. The first thing we do is to create a canvas element using JavaScript.

```
var canvas = document.createElement("canvas");
var ctx = canvas.getContext("2d");
canvas.width = 512;
canvas.height = 480;
```

3. Navigate to `16_Canvas/starter/` folder. Open `index.html`.
4. Locate the “main” class. Append the canvas element to the main class.

```
document.querySelector('???').appendChild(???);
```

5. A game needs graphics, so let’s load up some images. `bgReady` is used to let us know when it's safe to draw the image, as trying to draw it before it's loaded will throw a DOM error.

```
// Background image
var bgReady = false;
var bgImage = new Image();
bgImage.onload = function () {
     bgReady = true;
};
bgImage.src = "img/background.png";
```

6. Using the above code, do this for another two images that we need: `hero.png` and `monster.png`.
7. Now we will define some variables we'll need to use later. `hero` gets setup with speed which is how fast it'll move in pixels per second. `monster` won't move so it just has coordinates. Lastly, `monstersCaught` stores the number of monsters the player has caught.

```
// Game objects
var hero = {
     speed: 256, // movement in pixels per second
     x: 0,
     y: 0
};
var monster = {
     x: 0,
     y: 0
};
var monstersCaught = 0;
```

8. Now for input handling, we just want to store the user input for later instead of acting on it immediately. To accomplish this, we simply have a variable `keysDown` which stores any event's `keyCode`. If a key code is in the object, the user is currently pressing that key.

```
// Handle keyboard controls
var keysDown = {};

addEventListener("keydown", function (e) {
     keysDown[e.keyCode] = true;
}, false);

addEventListener("keyup", function (e) {
     delete keysDown[e.keyCode];
}, false);
```

9. Next, we will create a `reset` function. The `reset` function is called to begin a new game, or level, or whatever you'd like to call it. It places the hero (the player) in the centre of the screen and the monster somewhere randomly.

```
// Reset the game when the player catches a monster
var reset = function () {
     hero.x = canvas.width / 2;
     hero.y = canvas.height / 2;

     // Throw the monster somewhere on the screen randomly
     monster.x = 32 + (Math.random() * (canvas.width - 64));
     monster.y = 32 + (Math.random() * (canvas.height - 64));
};
```

10. Following that we will create an `update` function. It is called at every single interval execution. The first thing it does is it checks the up, down, left, and right arrow keys to see if the user has pressed them. If so, the hero is moved in the corresponding direction. `modifier` is a time-based number based on 1. If exactly one second has passed, the value will be 1 and the hero's speed will be multiplied by 1, meaning he will have moved 256 pixels in that second. This function gets called so rapidly that the modifier value will typically be very low, but using this pattern will ensure that the hero will move the same speed no matter how fast (or slowly!) the script is running.

```
// Update game objects
var update = function (modifier) {
     if (38 in keysDown) { // Player holding up
          hero.y -= hero.speed * modifier;
     }
     if (40 in keysDown) { // Player holding down
          hero.y += hero.speed * modifier;
     }
     if (37 in keysDown) { // Player holding left
          hero.x -= hero.speed * modifier;
     }
     if (39 in keysDown) { // Player holding right
          hero.x += hero.speed * modifier;
     }
```

11. Now that we've moved the hero according to the player's input, we can check to see if it caused anything to happen. If there was a collision with the hero and monster, that's it! That's pretty much the game. We tally the score (+1 to monstersCaught) and reset the game.

```
// Are they touching?
     if (
          hero.x <= (monster.x + 32)
          && monster.x <= (hero.x + 32)
          && hero.y <= (monster.y + 32)
          && monster.y <= (hero.y + 32)
     ) {
          ++monstersCaught;
          reset();
     }
};
```

12. Games are more fun when you get to see the action going down, so let's draw everything to the screen. First we take the background image and draw it to the canvas. Repeat for the hero and monster.

```
// Draw everything
var render = function () {
     if (bgReady) {
          ctx.drawImage(bgImage, 0, 0);
     }

     if (heroReady) {
          ctx.drawImage(heroImage, hero.x, hero.y);
     }

     if (monsterReady) {
          ctx.drawImage(monsterImage, monster.x, monster.y);
     }
```

13. Next we change some contextual properties, related to how to draw the font, and we make a call to fillText to display the player's score.

```
// Score
    ctx.fillStyle = "rgb(250, 250, 250)";
    ctx.font = "24px Helvetica";
    ctx.textAlign = "left";
    ctx.textBaseline = "top";
    ctx.fillText("Monsters caught: " + monstersCaught, 32, 32);
};
```

14. The main game loop is what controls the flow of the game. First we want to get the current timestamp so we can calculate the delta (how many milliseconds have passed since the last interval). We get the modifier to send to update by dividing by 1000 (the number of milliseconds in one second). Then we call render and record the timestamp. The `requestAnimationFrame()` method tells the browser that you wish to perform an animation and requests that the browser call a specified function to update an animation before the next repaint. It helps to make your animations perform smoothly, synced with your GPU (Graphic Processing Unit) and hog much less CPU.

```
// The main game loop
var main = function () {
    var now = Date.now();
    var delta = now - then;

    update(delta / 1000);
    render();

    then = now;

    // Request to do this again ASAP
    requestAnimationFrame(main);
};
```

15. Almost there, this is the last code snippet! First we will set our timestamp (with the variable `then`) to seed it. Then we call `reset` to start a new game/level. (Remember that this centres the hero and places the monster randomly for the player to find.)

```
// Let's play this game!
var then = Date.now();
reset();
main();
```

16. Save the files, and open `game.html` in Chrome browser.

Exercise 2 (optional) add a count-down timer to the "Catch the Monster" game

1. Add a `div` within the `main` class with an id "`counter`".
2. Navigate to 16_Canvas/starter/js folder. Create countdown.js.
3. Add the following code:

```
function countdown() {
    var seconds = 60;
    function tick() {
        var counter = document.getElementById("counter");
        seconds--;
        counter.innerHTML = "0:" + (seconds < 10 ? "0" : "") +
String(seconds);
        if (seconds > 0) {
            setTimeout(tick, 1000);
        } else {
            alert("Game over");
        }
    }
    tick();
}
```

4. Call `countdown()` at the end of the code.

5. Link `countdown.js` to `game.html`.

6. Save all files, and open `game.html` in Chrome. You should see a count-down timer added.

17 – Video and Audio

Objectives

This practical session you will be using building custom controls for HTML5 video.

Overview

This practical session contains two exercise:

- Exercise 1 you will be building play, pause and full screen custom control for HTML5 Video
- Exercise 2 (optional) you will be building seek bar and volume control for the HTML5 video
- The final HTML5 video with custom controls looks similar to the following page

Exercise 1 building custom control for HTML5 Video (play, pause and full-screen)

1. Navigate to `17_Video_and_Audio/starter/` folder. Open `video.html`.
2. Locate `<!-- Video -->`. Create a `video` element with the following attributes:
    a. id = “video”
    b. width = “640”
    c. height = “365”
    d. poster = “img/frog.jpg”
3. Next, using `source` element, include all three different formats of the video in `video` folder.
4. Provide a link for the browser to download the mp4 format of the video if the browser does not support HTML5 video.
5. Locate `<!-- Video Controls -->`. Create a div with `id = video-controls`.
6. Within the div, build the following video controls:
    a. A button, id=play-pause, class=play, with text “Play”.
    b. Range input type, id=seek-bar, default value is 0.
    c. A button, id=mute, with text “Mute”.
    d. Range input type, id=volume-bar, min=0, max=1, step=0.1, default value is 1.
    e. A button, id=full-screen, with text “Full=Screen”.
7. Navigate to `17_Video_and_Audio/starter/js` folder. Open `main.js`.
8. Create the following function:

```
window.onload = function () {
}
```

9. Within the function, create the following variables, and initialise with the video and controls from the HTML markup. The first one has been done for you:

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

10. Do the same for the following:

    a. playButton, with id=”play-pause”
    b. muteButton, with id=”mute”
    c. fullScreenButton, with id=”full-screen”
    d. seekBar, with id=”seek-bar”
    e. volumeBar, with id=”volume-bar”

11. The first video control is to hook up the Play/Pause button. We will set up an event listener that checks whether the video is currently playing and then toggles appropriately. We will check the `paused` property of the video to examine the current playback state. Use the `play()` and `pause()` functions to control playback.

```
// Event listener for the play/pause button
playButton.addEventListener("click", function () {
    if (video.paused == true) {
        // Play the video
        video.play();

        // Update the button text to 'Pause'
        playButton.innerHTML = "Pause";
    } else {
        // Pause the video
        video.pause();

        // Update the button text to 'Play'
        playButton.innerHTML = "Play";
    }
});
```

12. To get the mute button working, we will need to follow a similar process to the one we used for play/pause button. This time, however, you need to examine the `muted` property of the video and toggle it appropriately.

```
// Event listener for the mute button
muteButton.addEventListener("click", function() {
  if (video.muted == false) {
    // Mute the video
     ??

    // Update the button text
    ??

  } else {
    // Unmute the video
    ??

    // Update the button text
    ??
   }
});
```

13. Next, we will implement a FullScreen API that allows you to give full screen focus to certain HTML elements. To get the full screen button working you need to set up another event listener that will call the `requestFullScreen()` function when the button is clicked. To ensure that this will work across all supported browsers you are also going to need to check to see if the `requestFullScreen()` is available and fallback to the vendor prefixed versions (`mozRequestFullScreen` and `webkitRequestFullscreen`) if it is not.

```
// Event listener for the full-screen button
fullScreenButton.addEventListener("click", function() {
  if (video.requestFullscreen) {
    video.requestFullscreen();
  } else if (video.mozRequestFullScreen) {
    video.mozRequestFullScreen(); // Firefox
  } else if (video.webkitRequestFullscreen) {
    video.webkitRequestFullscreen(); // Chrome and Safari
  }
});
```

Exercise 2 (optional) building seek bar and volume control for HTML5 video

1. We have sorted all the buttons. Now we will work on the seek bar. To start with let's hook it up so that dragging the slider handle changes the position in the video. To do this you need to set up an event listener on the `seekBar` that will execute when the change event is fired. You then need to calculate the time in the video that playback should be skipped to and update the time of the video accordingly.

```
// Event listener for the seek bar
seekBar.addEventListener("change", function() {
  // Calculate the new time
  var time = video.duration * (seekBar.value / 100);

  // Update the video time
  video.currentTime = time;
});
```

2. This basic implementation works but you might have noticed that the position of the slider handle doesn't move as the video plays. To fix this you need to set up an event listener on the video that executes when the `timeupdate` event is fired. You then reverse the calculation that was used previously in order to get the value that the `seekBar` should be set to. The `timeupdate` event is fired continuously as the video plays.

```
// Update the seek bar as the video plays
video.addEventListener("timeupdate", function() {
  // Calculate the slider value
  var value = (100 / video.duration) * video.currentTime;

  // Update the slider value
  seekBar.value = value;
});
```

3. There's just one more bug that we need to fix with the seek bar. If you slowly drag the slider handle you will notice that the video keeps trying to play, resulting in a stuttering playback. To fix this you need to pause the video when the slider handle starts to be dragged and then play it again once the handle is dropped. You can do this by using two event listeners that execute when the `mousedown` and `mouseup` events are fired. Pause the video on `mousedown` and play it again on `mouseup`.

```
// Pause the video when the slider handle is being dragged
seekBar.addEventListener("mousedown", function() {
  video.pause();
});

// Play the video when the slider handle is dropped
seekBar.addEventListener("mouseup", function() {
  video.play();
});
```

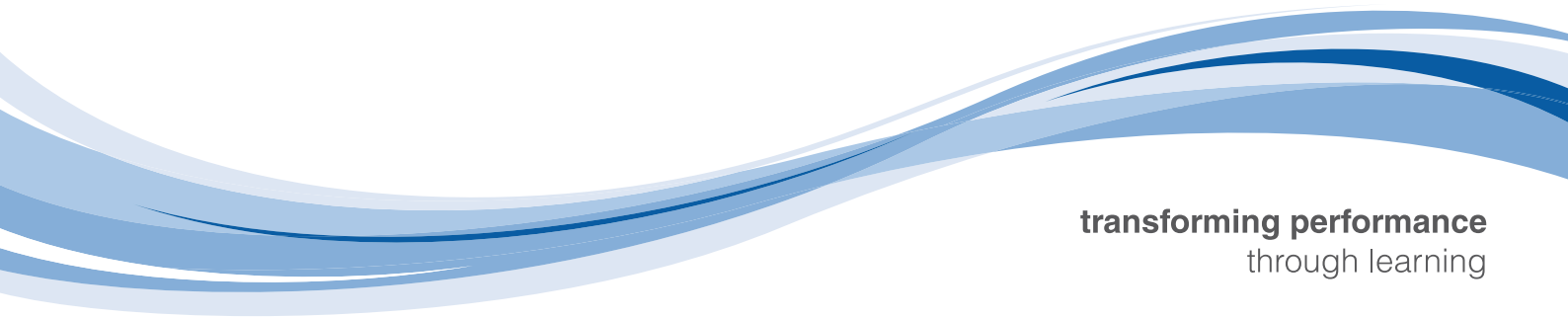
4. The final control to implement is the volume slider. For this you need to set up an event listener for the change event. When this event fires you need to take the value from the slider and use it to update the value of the volume property on the video.

```
// Event listener for the volume bar
volumeBar.addEventListener("change", function() {
  // Update the video volume
  video.volume = volumeBar.value;
});
```

5. That's it. Save all files and test it in Chrome browser.

Contact us:

0845 757 3888
info@qa.com
www.qa.com



transforming performance
through learning