Exercise Unit01 – Laying out a Page

1. Add one here

Exercise Unit 2 – Tools and Resources

1. Logo Hijack – The Logo Bandit

In this exercise, with your newfound DOM knowledge, you are going to hijack google.com and change it into a Yahoo search site instead.

1. Open up [www.google.com](http://www.google.com) in Chrome, and open up the elements.
2. Find the Google logo using the Chrome Development tools.
3. Modify the source of the logo IMG so that it’s a *Yahoo* logo instead
4. Find the Google Search button using the same tools.
5. Modify the test of the button so that it says “Yahoo!” instead.
6. Drawing out a DOM Tree

In this exercise, you are going to draw out a DOM tree from wab page provided for you

1. Open Exercises/Unit02-ToolsAndResources/dom-exercise.html using Chrome in your browser.
2. Using the Chome Development Tools, Elements panel, identify the elements that make up the DOM Tree.
3. Write these element using paper and a pencil or in your text editor.

Exercise Unit 3 –Sections

1. Replacing the meaningless divs

In this exercise, you are going to replace the divs

1. Open Exercises/Unit02-ToolsAndResources/01-html5-layout-styled.html.
2. All of the divs have classes that are named the same as structural divs, replace the divs with structural tags like (nav, header, hgroup, etc.)
3. Write these element using paper and a pencil or in your text editor.

b. Sections

In this exercise, you will modify an HTML page we worked on earlier in the course to replace meaningless div elements with meaningful section and article elements.

1. Open html5-sections/Exercises/html5-layout.html.
2. Replace meaningless div elements with meaningful section and article

elements. Note that there is room for interpretation here, so there is no one correct solution.

1. To keep the page looking as it did before, you will also need to modify html5-sections/Exercises/style-html5.css.

c. Determining the Outline

In this exercise, you will try to determine the outline of an HTML page.

1. Review the code below.

2. Create a list either on paper or in a text editor or word processor that shows the HTML outline as specified by the HTML5 specification

Exercise Unit 4 – Audio and Video

In this exercise, you will create an HTML5 file from scratch that plays video files.

a. Video Multiple Sources

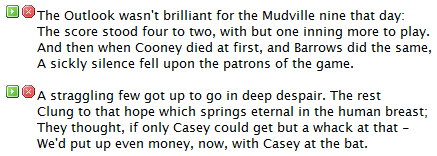
1. Create a new HTML5 file called video-multiple-sources.html in the Exercises/Unit04-AudioAndVideo directory.
2. Write the code to include the justin.mp4 (mime type is video/mp4) and

justin.ogv (mime type is video/ogg) files as source options. Both files are located in the html5-audio-and-video/Media directory.

1. Play around with video attributes such as: controls, autoplay, and loop.

b. Audio Javascript

1. Open the audio-javascript.html file that’s in Exercises/Unit04-AudioAndVideo directory and the audio-javascript.js file in the scripts folder. Notice that each <p> tag now has an id of the form pos- and a number. This number represents the time (in seconds) at which this stanza begins.
2. You will add JavaScript code to insert working play and pause images (found in the images folder) at the beginning of each stanza, like this:



When the play image is clicked, the audio should jump to that stanza and play, when the pause image is clicked, the audio should pause

1. Challenge:
   1. Add code to the reportTime() function so that the stanza currently being played is given the “highlight” class (already defined in style.css
   2. Make sure to remove the class when the stanza is no longer being played.

Exercise Unit 5 – Forms

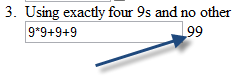
In this exercise, you will create an HTML5 quiz that validates form entries and reports the percentage of both the valid (but not necessarily correct) answers and the percentage of correct answers.

1. Open Exercises/Unit05-Forms/quiz.html in your editor.
2. Make the following changes to the form:
3. Add placeholders to all questions.
4. Make all questions required.
5. Question 1 should only accept valid colors.
6. Question 2 should only accept integers greater than or equal to 20.
7. Question 3 should only accept the pattern shown in the footnote[[1]](#footnote-1) (don't look if you want to figure out the pattern yourself).
8. Question 4 should only accept valid dates.
9. Question 5 should only accept valid URLs and should provide a list of common search engines to choose from, but should not limit the answer to those shown in the list.
10. At the bottom of the form:
11. Add a bar showing the percentage of valid (but not necessarily correct) answers answered. Give it an id of "quiz-progress".
12. Add a bar showing the percentage of correct answers answered. Give it an id of "quiz-success".
13. Finish the updateMeasures() function so that it correctly updates the two bars added above on every form change.

**Hint:** one way to do this is to loop through the input fields stored in the questions variables.

5. Challenge:

1. Add code so that the result of the formula the user enters in question 3 is displayed next to the input field like this:



1. Look into Bootstrap form components (<http://www.w3schools.com/bootstrap/bootstrap_forms.asp>) and give your form some bootstrap styling

Exercise Unit 6 – JavaScript

1. Hello World

To get started with writing JavaScript, open Exercises/Unit06-JavaScript/hello-world.html.

1. Add a script tag that calls to hello-world.js in hello-world.html
2. Add code into the function in hello-world.js add an call to alert a sentence when you open the document
3. Call the function so that when you open the page a window alerts you with “Hello World”
4. Add a line in the function so that “Hello World” is written in your console
5. Creating an object

Open Exercises/Unit06-JavaScript/film.html.

1. Add a script tag that calls to film.js in your html.
2. In film.js define an object called film
3. This object contains several parameters, the first being director and give it the value of Alfred Hitchcock as a string
4. Assign another parameter named data with an empty array. This array has three objects with two parameters, title and year. Give the first object the title of *North By Northwest*, as a string and the year of 1959 as a number.
5. Repeat the object with the title of *To Catch a Thief* and 1955 as the year. Add a third movie, if you’d like.
6. Add one more parameter to your object film and call it consoleFunction. This parameter is a function. In this function, state an object called theFilmObject. It should equal to this.
7. You’re almost done! This calls to the film object itself, which means that you can call any parameter that you have defined and it will give it to you. Let’s try this by including data to this and for each of the data, define a function that takes the object of movie.
8. In the console have the movie object also adopt the title param and complete a sentence like this:

title + “ is a film directed by ” + director

Hint: director is not part of the movie object …

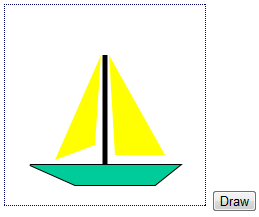
1. Don’t see anything in the console? In the last statement: call upon the film object and the consoleFunction() so that you can see what it is that you have written.

Challenge:

1. Create a empty div tag with the id=”filmList” and have the list of films show on your web page using getElementById() and innerHTML
2. Use JSON.Stringify to translate the array into a string.

Exercise Unit 7 – Canvas

1. Sailboat

Use HTML5 canvas to draw a simple sailboat like the one shown below: 

1. Open Exercises/Unit07-Canvas/sailboat.html in your editor.
2. Add the JavaScript code necessary to draw the sailboat pictured above.

Challenge:

Have the left sale blink between different colors. You'll need to use some JavaScript skills to make this happen.

If JavaScript isn't your thing, try adding a little person on your sailboat.

1. Snowman

Use circles and squares to create a snowman like the one pictured below:

1. Open Exercises/Unit07-Canvas/snowman.html in your editor.
2. Add the JavaScript code necessary to draw the snowman pictured above. You will need to add:
   * A layer of snow on the ground.
   * Three balls for the body and head.
   * Eyes, mouth and nose.
   * A hat.
   * Arms.
   * Buttons.
   * A sun.

 3. South America

1. Use two images found in the Exercises/Unit07-Canvas/images folder:

* south-america.gif - a map of South America.
* flags.png - a picture containing small graphics of country flags.

1. Open Exercises/Unit07-Canvas/south-america.html in your editor. You will be recreating the following drawing.
2. Add the JavaScript code necessary to:
   * Create the image objects and set their source values.
   * Draw the backdrop (the map).
   * Place the flags using the sprite method shown earlier. Each flag is 18 pixels wide and 13 pixels high. The source and destination positions are shown in the table below.
   * Add the country names.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Country** | **Source X** | **Source Y** | **Destination X** | **Destination Y** |
| **Chile** | 283 | 88 | 100 | 250 |
| **Argentina** | 255 | 4 | 130 | 300 |
| **Brazil** | 171 | 60 | 200 | 170 |
| **Paraguay** | 59 | 452 | 170 | 250 |
| **Uruguay** | 59 | 564 | 185 | 310 |
| **Bolivia** | 59 | 200 | 135 | 210 |
| **Peru** | 31 | 424 | 75 | 170 |

Exercise Unit 8 – APIs

In this exercise we are going to add a google map onto a web page.

1. Open Exercises/Unit08-APIs/weather.html
2. In the html reference the weather.js file in a script tag
3. In weather.js create a new XMLHttpRequest() object in a variable called xmlhttp
4. xmlhttp should open and get from this website: <http://api.openweathermap.org/data/2.5/weather?q=Toronto,on&appid=2de143494c0b295cca9337e1e96b00e0>
5. For the xmlhttp to work it needs to send this request.
6. Once you have this working with no errors, let’s test out the results in the console. Create a function called reqListener. In this function console.log the request’s responseText.
7. Add an event listener to xmlhttp that loads and listens to it’s request (reqListener).
8. Next we want to view the results on our webpage. Create a button with an onclick function called getWeather(). Underneath the button put a div with the id of weather.
9. If you click the button now… you will get *UNCAUGHT ReferenceError: getWeather is not defined*. We have to define the function. In weather.js create the getWeather(). function. The document should get the element by id and use a function called innerHTML which should be equal by xmlhttp’s responseText.
10. Wow! You’ve got a Toronto weather app… but the information that’s being put on the page is awfully ugly. Let’s parse through it and create a nice display for the information.

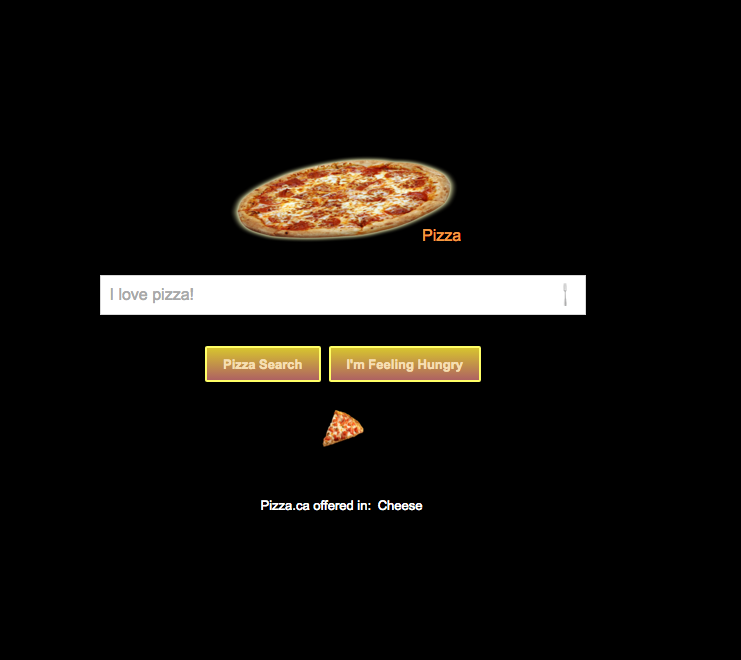
Exercise Unit 9 – CSS3

1. The logo bandit is back! And he’s making Google into a Pizza haven.
2. Open up [www.google.com](http://www.google.com) in Chrome, and open up the elements panel.
3. Using the HTML and CSS panels, you will be changing everything into pizza themed.
4. Change the entire background to black.
5. Find the Google logo using the elements cursor and change it to a picture of a pizza.
6. Next to the logo is a google’s country at that moment. Find that word and change it to “Pizza”. Change the color of the word to a good cheesy orange.
7. Give the input space a placeholder saying “I love Pizza…”
8. Next we’ll be styling the buttons. First change the wording to say “Pizza Search” and “I’m Feeling Hungry”. Make sure to change both the text and the aria-label. Aria Labels are an attribute designed to help assistive technology or screen readers.
9. Give the buttons a border of solid yellow. Make sure to use the Hex of yellow and not the word. Change the background color and font colorof the button too to whatever gradients you choose.

Hint: the color for the buttons may not be under background-color.

1. Change the font at the bottom of the page to “Pizza.ca offered in Cheese”. Make sure you don’t accidently click any buttons, your changes won’t be remembered. Change the writing to white
2. The final part of this exercise is changing the “Search by Voice” icon at the end of the input into a fork. Find the image using the element’s cursor and change the image url into a fork.

In the end, your google page should look something like this:

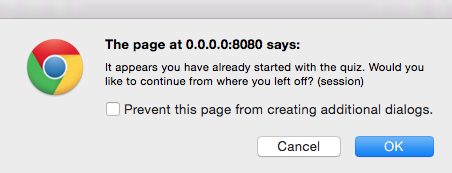


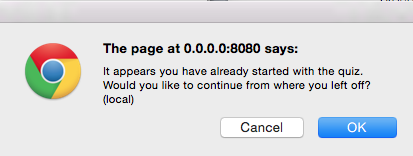
1. The logo bandit is back! And he’s making Google into a Pizza haven.

Exercise Unit 6 – Storage

In this exercise, you will create a quiz application that allows the user to save and resume later. It also protects the user from losing data if he/she accidentally refreshes.

1. Open Solutions > 6.b. Saving Quiz Challenge in your browser and play with the application:
   1. Answer one or more questions and then refresh the browser. You will get a dialog giving you the chance to use refill the form with values stored in sessionStorage:



* 1. Press OK and your values get added back.
  2. Refresh again and press Cancel on the dialog. Your values do not get added back.
  3. Refresh again. You don't get the JavaScript dialog because the sessionStorage key/value pairs were removed when you pressed Cancel in the previous step.
  4. Answer one or more questions again and click the Save My Answers for Later button.
  5. Close and reopen the browser. You will get a dialog giving you the chance to use refill the form with values stored in localStorage:
  6. If you click Cancel, the localStorage key/value pairs will be removed and you'll have to start the quiz over.
  7. If you click OK, the form will be refilled with your previous answers and they will be saved into sessionStorage.
  8. Also notice that the footer contains the time and date the quiz answers were last saved

1. Now open Exercises/Unit06-Storage/saving-quiz.html in your editor.
2. In the addLoadEvents() function:
   1. Loop through the inputs and add event listeners that capture change events to save the associated key/value pair in sessionStorage. Don't forget to use the prefix.
   2. Add an event listener to the Save button to capture a click event and call saveAnswers.
   3. Call the refill() function at the end.
3. Write the code in the saveAnswers() function to save all the answers in localStorage.
4. The refill() function:
   1. calls hasAnswers(), which returns "session", "local", or false, depending on if and where it finds saved answers. If there are no saved answers, it returns without doing anything.
   2. declares some variables:
      1. confirmed - we'll change it to true if the user wishes to refill the form.
      2. msg - the message to ask the user if he/she wants to refill the form.
      3. questions - the question inputs
   3. loops through the inputs. On the first iteration, it prompts the user with the message. If the user clicks Cancel, the key/value pairs are deleted (via the deleteAnswers() function) and the function returns/ends. Otherwise, we iterate through the questions. This is where you come in.
   4. Add code to populate the question inputs from the appropriate storage location (based on the value of fillFrom).

Challenge:

1. Notice that an external script called dateFormat.js is included. That extends the Date object prototype with a format() method, which you use as follows:

var now = new Date();

var dateMask = "yyyy-mm-dd H:MM:ss";

var formattedNow = now.format(dateMask);

1. Use this to write out the date last saved to the output element below the form.
2. You will need to store the date in sessionStorage and localStorage as appropriate. Don't forget the prefix.
3. Note that the dateMask variable is already set in the code.

Exercise Unit 9 – Animation

In this exercise, you will convert a basic HTML 4 page to an HTML5 page. The

CSS documents have already been created for you, such that, when you're finished, your HTML5 page should render exactly like the HTML 4 page.

1. Open html5-laying-out-a-page/Exercises/html4-layout.html.
2. Save the file as html5-layout.html.
3. Turn the page from an HTML 4 page into an HTML5 page. Make sure to change the stylesheet reference to point to style-html5.css.

Exercise Unit 10 – Unit Testing

ADD..

1. [9\+\-\\*/]{4,7} [↑](#footnote-ref-1)