Exercise 09 – The Document Object Model

Objectives

In this exercise, you will be manipulating rendered HTML using the core JavaScript skills you learned on day one of the course and the new concepts regarding DOM manipulation.

Overview

This exercise has two sections. In the first part, you will generate new content through DOM manipulation. In the second part, you will create an 'image of the day' that amends an **<img >** element to show a random choice from an array of object literals.

This exercise is scheduled to run for around **40 minutes**.

Part 1 – Creating new Content using DOM programming.

In this part of the exercise, you will be creating new **<p>** elements via the DOM API. We will create a series of **<p>** elements, append them to a ** ** but randomly set their colour from a pre-prepared array.

Exercise Set Up

1. In VSCode, open the file called **NewContent.html** from the **Starter** folder inside **Ex09** of the **Exercises** folder.
2. If you have rebooted your computer or stopped *live-server* from running since the last exercise, then use VSCode's integrated terminal to navigate to the **Exercises** folder and run the command **live-server**.
3. Open your browser (if it doesn't automatically) and navigate to **http://localhost:8080** and follow the path to open **Ex09→Starter→ NewContent.html**.

Exercise Instructions

1. Under the array declaration, create a function called **buildP** that takes *2 parameters*:
   a) One called **placeholder**;
   b) One called **num**.

The parameter placeholder will hold a reference to the DOM element we want to append new content to. The DOM programming mechanism allows us to create new elements by type using the **createElement** function. Any new elements have to be added to the DOM or it will not display on the page and it will not have any content, unless we give it some! When DOM programming, it is important to work in the correct order:

   i) Create the new element;
   ii) Create any text nodes that need to be attached to the new element;
   iii) Attach the text nodes to the new element;
   iv) Attach the new element to the appropriate existing DOM element;

2. Create a **new element** called **p** of type **p** using the **createElement** function:

```
document.createElement('p')
```

3. Create a *text node* to attach to the new paragraph called **text** with *any string you want in it* using the line of code below (the text given is an example):

```
document.createTextNode(`Have you tried turning it off and back on again?`)
```

4. Append the *text node* to the paragraph **p** using the **appendChild** function.
5. Append the paragraph **p** to the **placeholder** that was passed into the function.

The function is now complete and ready to go. All we need to do is call it and pass in the correct parameters. The new **<p>** element we are going to create needs to be appended to a container in the page. This could be another element on the page or even the document root itself. We will add it to the ** .**

6. Under the function definition, call the function setting the placeholder argument with the element found by using:

```
document.querySelector('#placeholder')
```

When we created the buildP function, we added a second parameter to the function that we have not used, or even set in the function so far. This is one of JavaScript's nice features. You can either add none, the first, or all parameters.

7. Save your code and refer to your browser.

You should see that the text and paragraph you created with JavaScript are now part of the page. Verify this by inspecting the elements in the Developer Tools.

The second parameter of the **buildP** function is going to be used to create and append multiple paragraph elements. We will always want *at least 1 paragraph* but maybe more - time to use the **do...while** loop.

8. At the top of the **buildP** function, declare a variable **i** initialised to **0**.
9. Surround the rest of the code in the **buildP** function with a **do...while** loop that exits when **i** reaches the **number** supplied to the function (remember to increment i).
10. Amend the call to the **buildP** function to add a second parameter of any number you choose.
11. Save and refer to your browser, checking the output is as expected.
12. Try a few different values and observe the page.

The final piece in this part of the exercise is to randomise the colours used to display our text in. To do this, we will use the array of colours already defined and a randomly generated value in the range of the indexes of the array.

Math.random gives us a value somewhere between 0 and 1. Indexes of arrays are integer values as are their length. The parseInt function will be used to return the nearest integer to a value we are going to calculate using the randomly generated number and the length of the colours array.

13. Before the line that appends the text node to the paragraph add the following line:

```
p.style.color = colours[parseInt(Math.random() * colours.length)]
```

14. Save and refer to the output of the browser.

The random number is generated in an algorithm based upon the system clock. Sometimes, rapid random calls can leave us with very similar values.

Part 2 – Image of the Day.

As we increase our JavaScript knowledge, we are able to do more interesting things. Some of the exercises that follow after coming chapters will build on the basic Image of the Day you are about to create to produce a slideshow type image gallery.

Exercise Set Up

1. In VSCode, open the file called **ImageLiterals.html** from the **Starter** folder inside **Ex09** of the **Exercises** folder.
2. If you have rebooted your computer or stopped *live-server* from running since the last exercise, then use VSCode's integrated terminal to navigate to the **Exercises** folder and run the command **live-server**.
3. Open your browser (if it doesn't automatically) and navigate to **http://localhost:8080** and follow the path to open **Ex09→Starter→ ImageLiterals.html**.

Exercise Instructions

1. Familiarise yourself with the HTML and JavaScript that already exists in the file.
2. Have a look at the CSS that is contained in the file styles.css.

If you are unsure of anything, please ask your instructor. Whilst we are not going to be changing any of the CSS, it is useful to know what it will be doing.

The JavaScript contains an array of objects that define a **src** and an **alt** for an image, along with a property called **caption** that contains some *text related to the image*.

3. The HTML contains an **<img>** with an **id** of **slideShow**, make this a variable called **i** in the script.
4. Make another variable called **caption** and point this to the DOM element with an **id** of **captionText**.
5. Assign a variable called **currentPosition** to a *random number* generated using the **length** of the **images** array. *It must be an integer (see Instruction 13 of Part 1).*

We can now change the image using 2 of the variables we have just created and reassigning some of their properties.

6. Change the value of the **src** property of the **img** to the **src** value *obtained from the object at the index of* **currentPosition** from the **images** array.

7. Change the value of the **alt** property of the **img** to the **alt** value *obtained from the object at the index of* **currentPosition** from the **images** array.
8. Change the value of the **innerText** of the **span,** so that it is set to the value of **caption** *if it exists* and **alt** *otherwise*, *obtained from the object at the index of* **currentPosition** from the **images** array. *(Hint - use a ternary statement!)*
9. Save your code and observe the output of the browser. Refresh the page several times to see the image, its alt text and its caption change.

Further activities

If you finish ahead of schedule, take a shot at the following activity. At this point, you should be able to write the code with no instructions, but your instructor can help if there are questions.

In Part 1 of the exercise, we made a function that created **<p>** elements. With an additional parameter that represents the number of html elements we want to build, we would have a much more powerful function. Copy **buildP**, rename it as **buildAnything**, amend it and call it.