Exercise 11 – Events

Objectives

In this exercise, we are going to take the Image of the Day viewer we created back in the DOM exercise and move it forward to become a fully-fledged image viewer.

Overview

We already built the core of this exercise in an earlier exercise. Now, we take it a step further by defining functions to navigate up and down the image stack and associate them to events.

This exercise is scheduled to run for around **30 minutes**.

Exercise Set Up

1. In VSCode, open the file called **ImagesWithEvents.html** and **styles.css** from the **Starter** folder inside **Ex11** 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 **Ex11→Starter→ ImagesWithEvents.html**.

Exercise Instructions

When the page first runs, a randomly-selected image is loaded, it survives at script scope even after it has been created. This means it can double up as a way of flipping through the images in our gallery. We want to be able to cycle through the images using some next and previous buttons and we need to write a couple of functions to do this.

1. At script level, create a function called **previousImage** that has a body that:
   a) Post-decrements the value of **currentPosition**;
   b) Sets the **src** of **i** to be the value of **src** of the object at the index of **currentPosition** from the **images** array;
   c) Sets the **alt** of **i** to be the value of **alt** of the object at the index of **currentPosition** from the images array;
   d) Sets the **innerText** of **caption** to be the value of **caption** of the object at the index of **currentPosition** in the **images** array, *if it exists* or the **alt** value *if it doesn't*.

All good so far, but what would happen when the **currentPosition** hits **-1**? We need to restrict that from happening. We will solve the ticking time bomb by adding an **if** statement that cycles the array back around to the last element.

2. Add an **if** statement to check for this condition and set the value of the **currentPosition** to *one less than* the **length** of the **images** array. It should be placed *immediately after the post-decrement* of **currentPosition** in the function body.

We can hook this up as a hard-coded event handler to test the logic quite simply. However, this approach is not best practice, hard-coding JavaScript execution against an element makes your code quite brittle. It will do at the moment to test the previous button.

3. Locate the **<button id="previous">** and add an **onclick** attribute calling the **previousImage** function.

Now we need to navigate to the next image.

4. Create a function called **nextImage** that:
   a) Increases the value of **currentPosition** by **1**;
   b) Checks to see **if** the value of **currentPosition** is the *same as* the **length** of the **images** array and sets **currentPosition** to *zero* if it is;
   c) Sets the **src**, **alt** and **caption** of **i** appropriately.
5. Add appropriate **onclick** event to the *next button*.
6. Save your code and check the buttons work in the browser.

It is time to improve the event handling approach by creating some programmatic event handlers. If you check the global variables at the top of the script block, you will see there is a **previous** and **next** variable.

7. We need next and to point to their corresponding buttons in the html – you should now be familiar enough to do this without prompting.
8. Add a **console.dir** to let you inspect **previous**.

Check the page in Chrome. Scroll down until you find the **onclick** property. You will find it is pointed to a function. At run time the Browser behind the scenes has effectively added a function to the object. Sneaky old JavaScript!

We can leverage this exact same behaviour ourselves.

9. Remove the hardcoded **onclick** attributes from each button.

Events are often a one-to-one relationship and only one object needs to raise the event. That is certainly the situation in our code, so we will be creating anonymous functions (using the arrow function implementation). Anonymous functions provide a function that only the pointer to it can call.

We are going to switch to a programmatic hook up of the events.

10. Use an arrow function to set the **next** and **previous onclick** attributes to call the appropriate functions.

In good web design, we separate functionality from the markup. HTML is for structure, CSS for presentation and JavaScript for behaviour (although CSS can make behavioural changes as well). We will step this concept even further forward in the object-orientated chapter of this course and further improve this design pattern.

One final change can be made to the code. Currently, the image array initialisation and the code that first sets the image element are executed as soon as the page hits the code. Let's improve this by using an event of the **window** object.

11. Wrap the code that fills the array and sets the image in an anonymous function responding to the **window.onload** event.

Further activities

If you are finished ahead of schedule and want to push yourself a bit, consider this design improvement:

- The code to switch the image **alt** and **src** is repeated in three locations. Extract the code into a separate function and remove the repetition.