# Array Enhancements

## Overview

In this lab you'll make use of some of the new features for dealing with arrays in modern JavaScript. If time permits, you'll also see how typed arrays are used in the HTML File API.

## Source folders

* C:/JsDeepDive/Labs/Student/04-Arrays
* C:/JsDeepDive/Labs/Solutions/04-Arrays

## Roadmap

There are 4 exercises in this lab, of which the last exercise is "if time permits". Here is a brief summary of the tasks you will perform in each exercise; more detailed instructions follow later:

1. Refactoring code to use new array features
2. Copying within an array
3. Finding an element in an array
4. (If Time Permits) Using typed arrays with the File API

## Exercise 1: Refactoring code to use new array features

Open a Command Prompt window, go to the C:/JsDeepDive folder, and run the following command to start the Babel transpiler:

npx gulp

Now go to the *student* folder for this lab. Note there are two subfolders – you want the ProductSuggestions folder for the first three exercises.

Take a look at the files in the ProductSuggestions folder. This is the same web application you saw earlier in the course – it allows the user to enter details for product suggestions. Reacquaint yourself with how the code works in es6scripts/script.js.

Refactor the code to make use of ES6 array features where possible. For example:

* Use for-of loops rather than for-in loops. Try having a go at using the version of for-of that gives the index and value of every element, and call console.log() on each iteration to display indexes/values on the console.
* Also try out forEach() as an alternative iteration mechanism somewhere in the code.

## Exercise 2: Copying within an array

In the *All Product Suggestions* panel, add a *Repeat* buttonto repeat the last product in the array. The idea is that if the array contained [A,B,C] initially, then it'll contain [A,B,C,C] afterwards. Here's what you need to do:

* Increase the size of the array by 1 element, via push().
* Clone the element that used to be at the end position to the new slot at the end of the array. Use copyWithin() to do this.

## Exercise 3: Finding an element in an array

In the *Matching Product Suggestions* panel, add a *Find First* buttonto find the first product matching the text entered by the user. Use find() to do the work. If a match is found, display it in the panel, otherwise display an error message (e.g. in an alert).

## Exercise 4 (If time permits): Using typed arrays with the File API

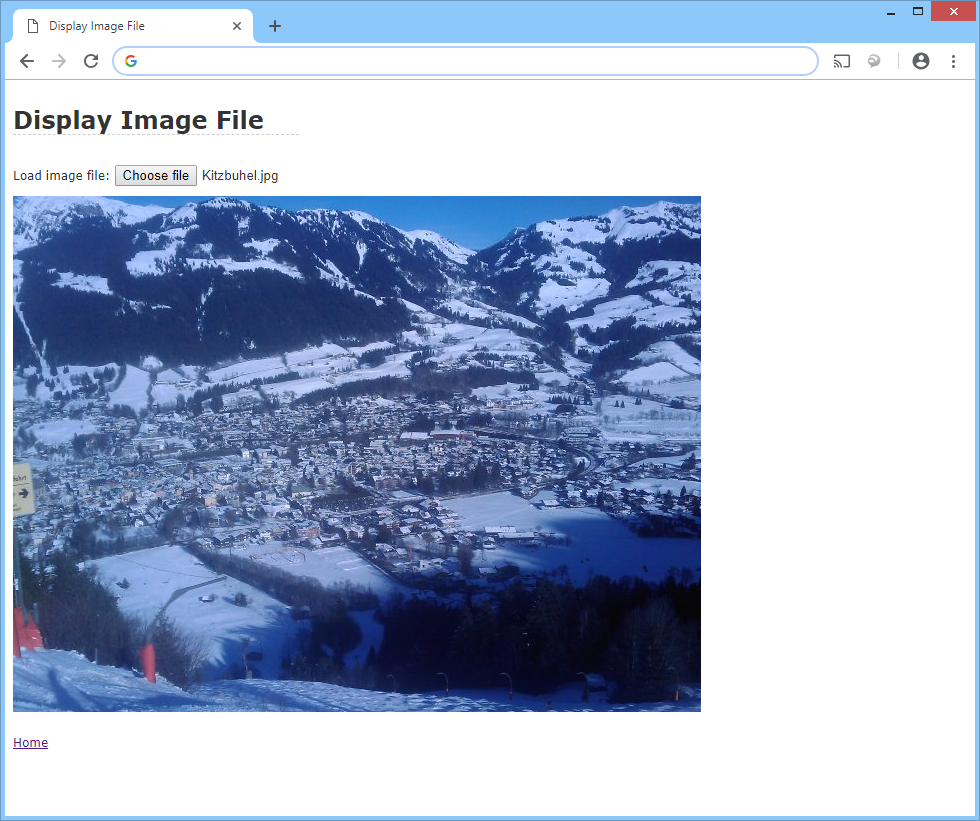
HTML5 has a standardized File API that allows a web app to interact with files on the local file system. You'll explore this API in this exercise.

In the *student* folder, go to the ReadingFiles folder and open index.html in a browser. There are 3 hyperlinks that take you to separate pages, to read a file in 3 different ways:

* As an image file (already implemented)
* As a text file (already implemented)
* As a binary file (you will implement this)

Click the *Display image file* link first, which takes you to displayImageFile.html. The web page has a *Choose file* button – this is actually an HTML5 <input type='file'> element. When you click it, a file chooser dialog box appears. Choose an image file in the filessubfolder, e.g. Kitzbuhel.jpg and click *Open.* This generates a change event, which is handled in onLoadImageFile() in es6scripts/processImageFile.js. This function does the following tasks:

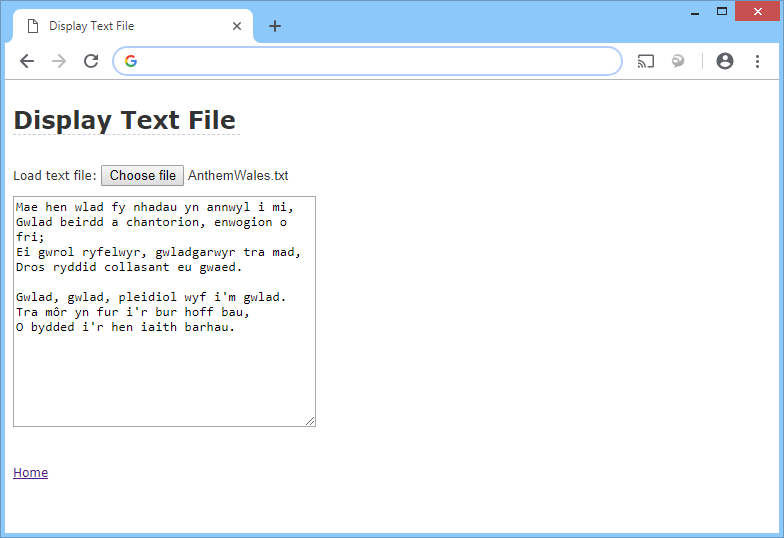
* It checks the user selected an image file, and gets the File object representing the file.
* It creates a FileReader object to read the file contents. Specifically, FileReader has a readAsDataURL() method that reads a file asynchronously and exposes the contents as a URL, which can be assigned to the 'src' attribute on an <img> element. The net effect is that the web page displays the image you selected.



When you're happy with this, click the *Home* link to return to the home page.

Back on the home page, click *Display text file* which takes you todisplayTextFile.html. This web page is similar to the previous one, except it reads a text file via the readAsText() method on FileReader.

To see how it works, see onLoadTextFile() in es6scripts/processTextFile.js. The net effect is that the web page displays the text file you selected, e.g. here's the content of the AnthemWales.txt file:



When you've had a play with this, click *Home* to return to the home page.

Back on the home page, *Display binary file* which takes you todisplayBinaryFile.html. The purpose of this web page is to read a binary file via the readAsArrayBuffer() method on FileReader.

You'll write the code to do this in es6scripts/processBinaryFile.js. Add the following code, where indicated by the TODO comment in onLoadBinaryFile():

* Create a FileReader object.
* Set up event handlers for the onload and onerror events (see below for details of how to handle the onload event)
* Call readAsArrayBuffer() on the FileReader object, to read the file contents into an ArrayBuffer.

When the file has been completely loaded, the onload event occurs. Handle this event as follows:

* Get the result property from the FileReader object. This is an ArrayBuffer object, containing the raw contents of the file.
* You can't manipulate an ArrayBuffer directly – it doesn't have any methods and you can't iterate over it. Instead, you must create a typed array or a DataView, to wrap it up. We suggest you create a byte array that points to the ArrayBuffer as follows:

let array = new Uint8Array(arrayBuffer)

* Iterate through the byte array. Convert each byte into a 2-digit hex string and display the final string in the <textArea> element named binaryFileOutput.

When you're ready, make sure Babel has transpiled your ES6 code successfully, then refresh the web page in the browser and try to read a file as binary data. It should look something like this:

