# Destructuring

## Overview

In this lab you'll use destructuring to access array elements and object properties. You'll display the resulting values on a simple web page.

## Source folders

* C:/JsDeepDive/Labs/Student/02-Destructuring
* C:/JsDeepDive/Labs/Solutions/02-Destructuring

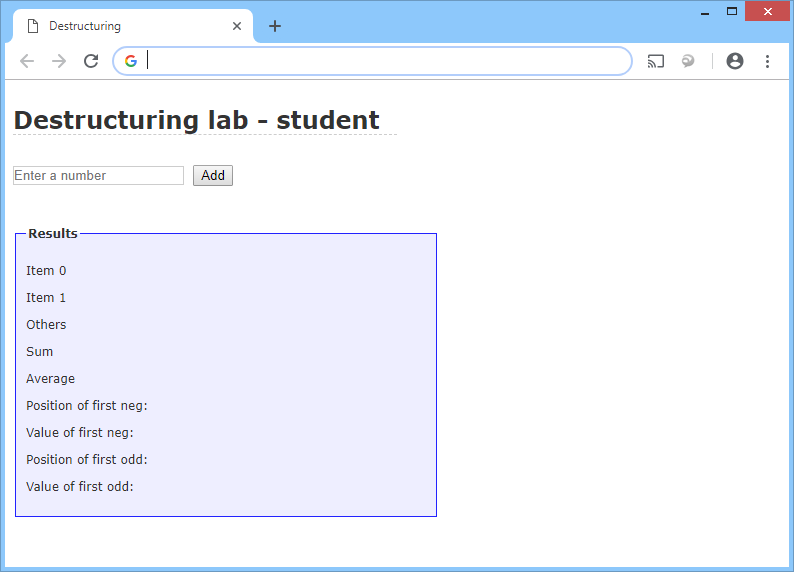
## Roadmap

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. Getting started
2. Destructuring an array
3. Destructuring an object
4. (If Time Permits) More destructuring an object

## Exercise 1: Getting started

In Windows Explorer, go to the *student* folder for this lab. The folder contains an HTML page and a simple CSS stylesheet. Open the HTML page in a browser – it allows the user to enter a series of numbers (one at a time), and displays various details about the numbers in a results panel.



The web page doesn't do anything at the moment – you'll need to add some script to make it work. You'll write your code in es6scripts/script.js, and then use Babel to transpile it into es5scripts/script.js (the HTML page picks up the ES5 version of the code).

To get the Babel transpiler running as a Gulp task, open a Command Prompt window in the C:/JsDeepDive folder and run the following command:

npx gulp

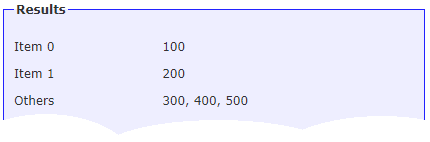
## Exercise 2: Destructuring an array

Take a look at the code in es6scripts/script.js. We've written some starter code to handle clicks to the *Add* button. The code gets the number entered by the user, converts it into an integer, and adds it to a global array named numbers.

Where indicated by the "Exercise 2" comment, add code as follows:

* Destructure numbers into three variables holding item 0, item 1, and all the remaining elements from the array.
* Display these three variables on the HTML page, in the <span> elements named item0, item1, and itemOthers. We've provided a function named setHtml() to make it easier to display some HTML content in an element; the 1st parameter is a CSS selector for an element, and the 2nd parameter is the HTML you want to display.

Save your file, and verify Babel transpiles it successfully. Then refresh the HTML page in the browser and add some numbers. Verify the results are displayed correctly. For example, if you entered numbers such as 100, 200, 300, 400, 500, you should see something like this:



## Exercise 3: Destructuring an object

Write a function named stats() that returns an object containing 2 properties:

* sum – The sum of all the values in the numbers array
* average – The average of all the values in the numbers array

Call stats() from the doAdd() function, where indicated by the "Exercise 3" comment. Destructure the returned object into two separate variables, holding the sum and average respectively. Display these values in the <span> elements named sum and average.

Save your file, and verify Babel transpiles it successfully. Then refresh the HTML page in the browser and add some numbers. Verify the sum and average values are updated every time you add another number.

## Exercise 4 (If time permits): More destructuring an object

Write a function named find() that finds the first element in the numbers array that matches a specified test. Here are some hints:

* find() should take a "predicate" function as a parameter, which specifies the test to perform upon each element in the array.
* find() should iterate through the numbers array and invoke the predicate function upon each element until the predicate function returns true. At that point, return an object containing the index of the matching element, along with its value.
* If no match is found, return an object with suitable "error" properties.

Call find() from the doAdd() function, where indicated by the "Exercise 4" comment. Pass an arrow function as a parameter, to test if a value is negative. The find() function will return an object indicating the position of the first negative element, plus its value. Destructure this object and display its constituent fields in the <span> elements named negPosition and negValue.

Call find() again, this time passing in an arrow function that tests if a value is odd. The find() function will return an object indicating the position of the first odd element, plus its value. Destructure this object and display its constituent fields in the <span> elements named oddPosition and oddValue.

Save your file, and verify Babel transpiles it successfully. Then refresh the HTML page in the browser and add some numbers. Verify the Web page displays the correct info about the first negative value and the first odd value.