Introduction to the JavaScript Array reduce() method

JavaScript's Array reduce() method can be a bit confusing. It doesn't "reduce" any of the array elements, in fact they are not changed. This method "reduces" the array to one value.

An array is essentially a list, and in JavaScript, that list can contain any valid value. Most array-specific tasks involve "doing something" with the array, or you may need to "do something" with every array element. But what if you need to convert the array into one value? In other words, if the array contains a list of values, what one value could represent all of those values? In this article, I'll explain how the JavaScript **Array reduce()** method provides a way to solve this problem.

An important concept to think about is how you can translate a list of values into one, that represents all of them. In some languages, all arrays must be "typed". In other words, every array element must be of the same type. In JavaScript, you're not limited in this way; a JavaScript array can contain any number of numbers, strings, arrays or any other valid JS value type.

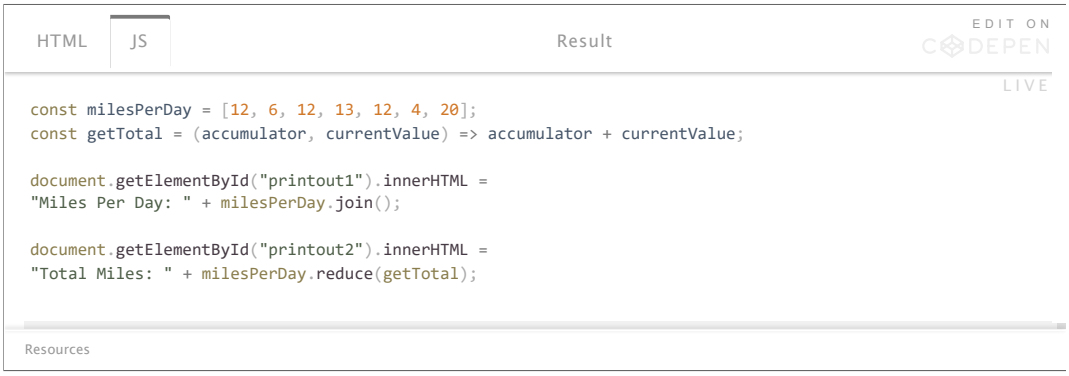
It's true that Typescript can limit this kind of freedom, but you're not required to use Typescript and ultimately, Typescript compiles down to JavaScript. So, the reality of dynamic arrays in JavaScript has not changed. Therefore, if you want to reduce all of the values in your array to one, it's important to consider the type of each element. If all of the elements in your array are of the same type, things are fairly simple. If the types vary, however, that can get complicated. Of course, that discussion is beyond the scope of this article, but I just wanted to point it out.

Using a while-loop – Example # 1

A "while" loop is a valid way to iterate a JavaScript array and that approach has worked for us here. We effectively "reduced" all of the elements of our array to the one value of 79, and we did this by adding each array element to the "getTotal" variable.

When I said that this approach "*...has worked for us*", you may notice that I didn't say that it "*...worked well*". This is because the approach we took solved a problem, but created a couple of issues, the main one being boilerplate code. Most of the code in our "**while**" loop will be repeated **if / when** we need to solve the same problem elsewhere in our code. In addition to that, we have created an "**i**" variable to keep track of our counting during our "while" loop. It may seem innocent, but the more variables we create, the more variables we need to manage. Fewer variables is better.

Using the JavaScript Array reduce() method – Example # 2



The **Array reduce()** method has dramatically simplified things in Example # 2. Notice how, in the getTotal the variable is now an anonymous function. In this function, we add the **accumulator** and **currentValue** arguments. Now you may be wondering: "*What are the **accumulator** and **currentValue** arguments?*" Well, take a look at the last line of code: **milesPerDay.reduce(getTotal)**. What's happening here is, we're executing the **.reduce()** method of our "**milesPerDay**", passing it the **getTotal** as an argument. This is the anonymous function that we detailed a few sentences ago.

So, the **reduce** method iterates the array and provides access to not only the currently iterated value (i.e. the current array element), but also the value that was returned by the previous execution of this function. This allows you to "*accumulate*" values across the array.

The arguments passed to the JavaScript Array reduce() method – Example # 3

Example # 3 does not provide any additional value with respect to solving the problem; the solution from Example # 2 is solid. The purpose of Example # 3 is to offer more detail about the arguments that are passed to the function provided to the **Array reduce()** method.

As demonstrated in the UI, the first argument is the accumulated value that's being created as you iterate the array (i.e. the value returned by the last execution of this same function). The second argument, "**currentValue**", is the value of the current array element. In the third, the index of the current array element is provided, and in the last, we see the array over which you are iterating.

I decided to show the value of the "**array**" argument to demonstrate that the **Array reduce()** method does not itself make any changes to the original. You can do that within the function passed to the **Array reduce()** method, but **reduce()** itself does not make any such changes. What you choose to do with these arguments is up to you, but just know that they're helpful in building the logic you need to reduce your array to one final value.