JavaScript for App Development

By Mark Lassoff, Founder Framework Television

Section Six

It's time for us to tackle arrays.

Arrays allow us to store multiple values referenced by a single variable. For example, we could store a bunch of names, or addresses, or grades.

Arrays are also objects in JavaScript, meaning that they have more power than a variable. They have properties associated with them such as `Array.length`, which tells you how many items are stored in the array. They also have methods that do some kind of work on the array like `Array.sort()` which sort the members of the array alphabetically or numerically.

Arrays are data structures-- A template for storing specific kinds of data. They are often the first data structure that most developers encounter in their coding career.

Section 6 Goals

In this section of the course your goals are:

□ Create a Simple Array
□ Identify Array Members by Index
□ Change Array Members via Index
□ Use a Loop to Access Each Member of the Array
□ Use Array Properties and Methods

Watch This: Section 6 Video

As always your course videos are available on YouTube, Roku and other locations. However, only those officially enrolled have access to this course guide, are able to submit assignments, work with the instructor, and get this guide.

Watch this section video at: https://www.youtube.com/watch?v=83PyNXkjfWc

Creating a Simple Array

In its simplest form, an array is a list of values. The values can be strings, floating point numbers or integers. You can use an array to store just about any set of values.

Let’s start by defining an array.

```
<div id=”output”></div>
<script>
   var computers = [“Apple”, “IBM”, “Commodore”, “TRS-80”, 
“Amiga”];
</script>
```

In this code segment, we’ve declared an array called computer and add a number of string values to the array.

Arrays can also be declared more formally using the `new` keyword. This is less common and you’ll run in to it with older code.

> Even though the new keyword is not used much in modern coding contexts, I want you to be aware of it. Chances are as a new developer, you won’t be writing code from scratch but working with an older code base that you’ve inherited.

When defined with the `new` keyword the array definition will look something like this:

```
<div id=”output”></div>
<script>
   var family = new Array(“Mark”, “Joan”, “Rick”, “Brett”, “Kerry”);
</script>
```

It’s also possible to define an array as empty initially using something like `var family = new Array` or `var family = []`.

Understanding Array Indexes

When you create an Array, the individual members are assigned numerical indexes. JavaScript (like most modern coding languages) is zero-indexed, meaning that the first index in an array will be zero. In the code sample above the members of the array are: “Mark”, “Joan”, “Rick”, “Brett” and “Kerry”.

When that array is declared with `var family = new Array(“Mark”, “Joan”, “Rick”, “Brett”, “Kerry”);` the array indexes are stored as follows:

Index	Value
0	“Mark”
1	“Joan”
2	“Rick”
3	“Brett”
4	“Kerry”

To access a member of an array we use the array name and the index like this:

```
<div id=”output”></div>
<script>
   var family = new Array(“Mark”, “Joan”, “Rick”, “Brett”, “Kerry”);
   document.getElementById(“output”).innerHTML = family[0];
</script>
```

This code is accessing the zeroth member of the array which is Mark. if `family[3]` were accessed, “Brett” would be the result.

Do This: Accessing Array Members and the Entire Array

First define an array that includes the names of five or six family members or friends. Using the numerical indexes output each member of the array. Then try the following:

1. Access an array that doesn’t exist, for example, `family[44]`.
2. Access the array by name without an index like this, `family`.

Make a note of what happens in each case.

Modify Array Members By Index

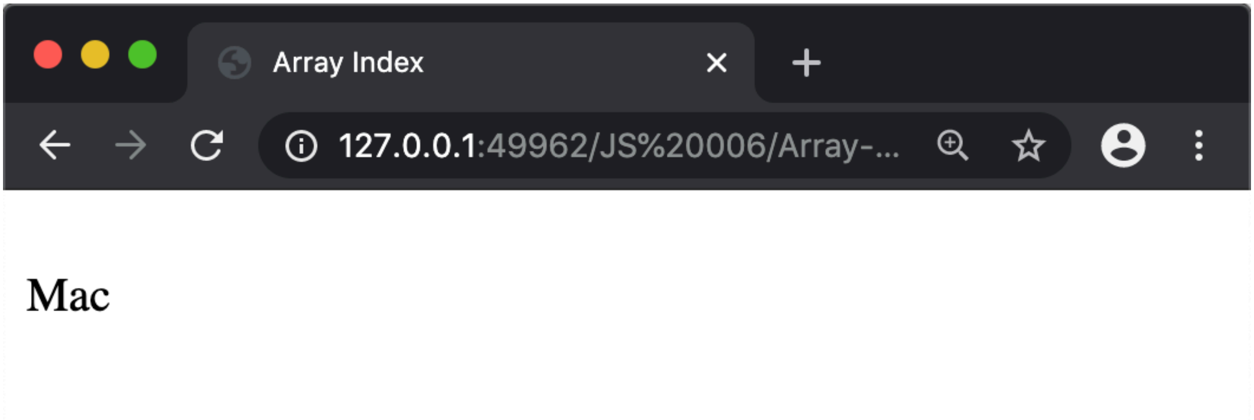
Let’s modify the Array members by using the array index. Let’s take a look at another short code sample:

```
<div id=”output”></div>
<script>
   var computers = [“Apple”, “IBM”, “Commodore”, “TRS-80”,
“Amiga”];
   computers[0] = “Mac”;
   document.getElementById(‘output’).innerHTML += “<br/>” +
computers[0];
</script>
```

In this example we declare the array `computers` and the initial value of `computers[0]` is “Apple”. However, after declaring the array the following modification is made:

```
computers[0] = “Mac”;
```

This writes over the zeroth index of the array and inserts “Mac” instead of “Apple”. So the output of this looks like this:



Mac

We can also add to the array if we know the final index. For example in this case we could add the `computers[5] = “Atari”;` which would add an additional value on to the computers array.

If we were to dump the entire contents of the computers array with the code `document.getElementById(“output”).innerHTML += “<br/>” + computers;` it would look something like this:

```
Mac,IBM,Commodore,TRS-80,Amiga,Atari
```

> In reality, adding elements by adding an index is a lousy way to do this. It means our program must always be aware of the length of the array and there is too much chance of accidentally deleting a member or adding to the wrong array index. The solution is to push() members on to the array and we’ll learn how to do that in just a bit.

Looping Through Arrays

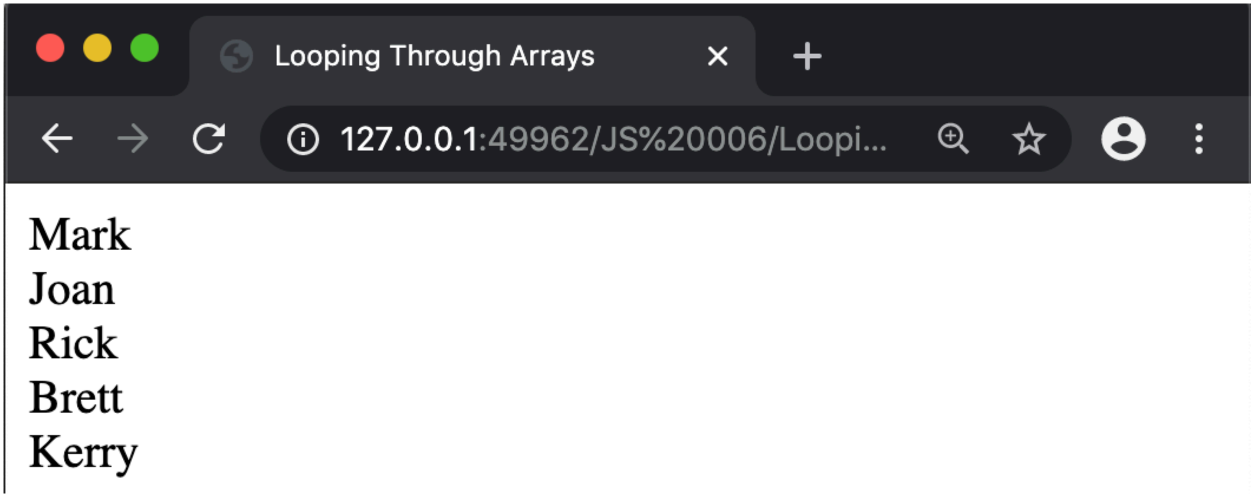
An array dump as demonstrated above isn’t the best way to access all the members of an array. More frequently you’ll want to access all the members of an array by looping through that array.

Let’s take a look at exactly how you accomplish that:

```
<div id=”output”></div>

<script>

//Old School Array Declaration
var family = new Array(“Mark”, “Joan”, “Rick”, “Brett”, “Kerry”);

var out = “”;
```

```
for(var i=0; i < family.length; i++)
  {
    out += family[i];
    out += “<br/>”;
  }
document.getElementById(‘output’).innerHTML = out;

</script>
```

The result of this code looks like this in the browser window.



```
Mark
Joan
Rick
Brett
Kerry
```

Two items worth noting here. First is the continuation condition of our loop. You’ll recall that the loop continues to iterate as long as the continuation condition is `true`. Our continuation condition is `i < family.length`. This insures that the condition is false when i is greater than the number of members of the array.

We also access the individual member of the array with `family[i]`. The value of `i` begins at zero and increases to one and so on as we iterate through the array. You’ll note the array counter is i++ indicating we’ll increment i each time through the loop, accessing each array member.

Do This: Loop Through an Array

Using the array you previously created, create a loop that loops through each member of the array and outputs it to an `alert()` box.

Properties of an Array

As you'll recall earlier I mentioned that arrays have properties, which describe the array and methods, which act on the array. In a generic context I like to think of properties and methods as the adjectives and verbs of objectives, respectively.

There's only one property of `Array` that we'll work with here and it's one we've already used: `Array.count`. This simply returns the number of elements in the array. For my `family` array above, `family.count` returns 5. **Keep in mind this is the number of array members—not the index of the last element. Since JavaScript is zero-indexed, this can throw you off!**

`Array.count` is a relative expensive operation from a resource point of view. That's because the JavaScript processor has to count each and every element of the array each time the `count` property is encountered. For longer arrays this can be a burden. You might see a structure like this that is more efficient:

```
var out = “”;
var length = family.length;
for(var i=0; i < length; i++)
  {
     out += family[i];
     out += “<br/>”;
  }
```

With this structure, the number of members of the array only have to be counted once, alleviating the burden on the processor.

Array Methods

There are a number of Array methods that you can use to manipulate your arrays. I won't go over them exhaustively here. However, you should be aware of what's out there and available for you to use.

Do This: Access JavaScript Documentation

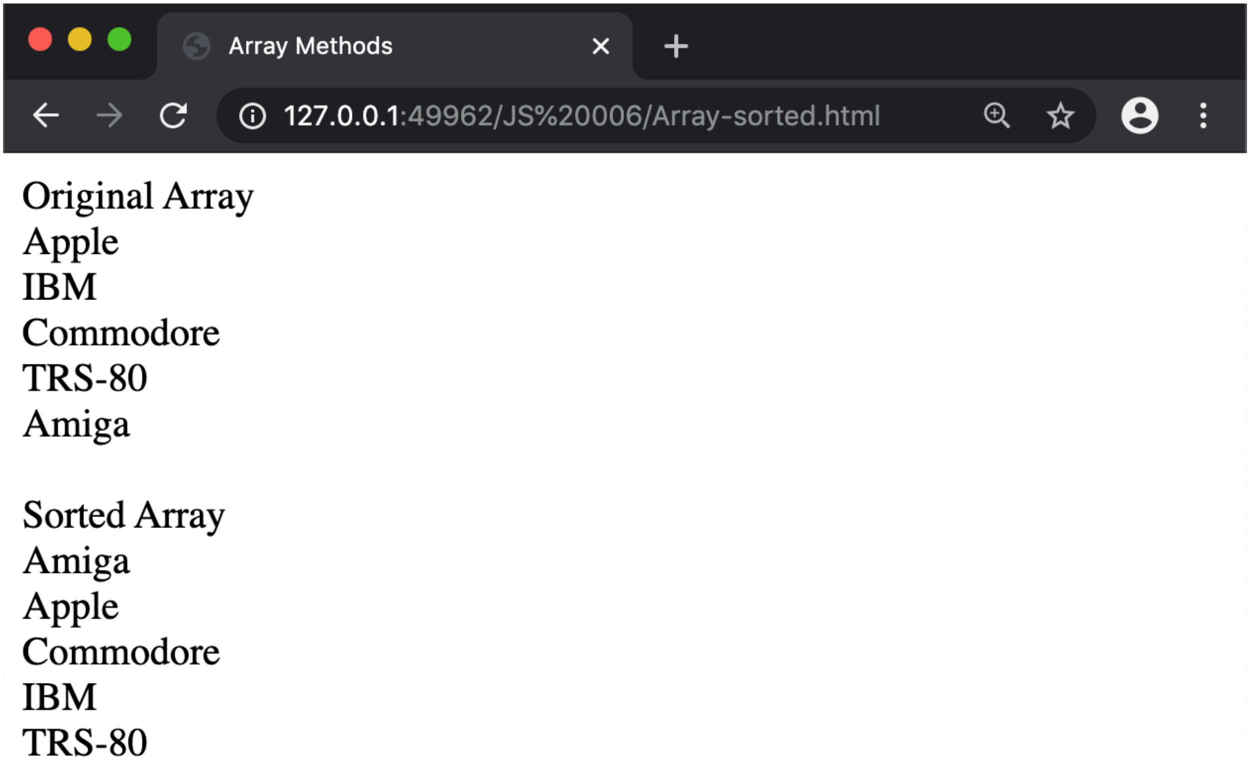
Even experienced pros have to access language documentation every once in a while. There are several versions of documentation out there, however, I like to use the Mozilla "official" documentation.

Read the Mozilla documentation on arrays at https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array.

To get a feel for how methods work examine the following code:

```
<div id="output"></div>
<script>
  var computers = ["Apple", "IBM", "Commodore", "TRS-80",
"Amiga"];
  var out = "Original Array<br/>";
   //Loop Through the array
  for(var x = 0; x < computers.length; x++)
  {
    out += computers[x];
    out += "<br/>";
  }
  document.getElementById(‘output’).innerHTML = out;

  out = “<br/>Sorted Array<br/>”;
  computers.sort();
   for(var x = 0; x < computers.length; x++)
  {
     out += computers[x];
     out += “<br/>”;
  }
     document.getElementById(‘output’).innerHTML += out;

</script>
```

Your output should look like this:



```
Original Array
Apple
IBM
Commodore
TRS-80
Amiga

Sorted Array
Amiga
Apple
Commodore
IBM
TRS-80
```

Obviously we used `Array.sort()` to sort the members of the array alphabetically. This actually reindexed the array so Amiga, the first member of the array alphabetically was reassigned index 0, Apple was assigned index 1 and so on.

As you saw from the documentation, `Array.sort()` is one of many individual methods that you can use with Arrays.

Do This: Use Array.pop() and Array.push()

The `array.pop()` method is used to delete the last member of the array. The `array.push()` method is used to add a member to the array. Using your family array, use both methods and then loop through the array again to output it after the pop and push operations.

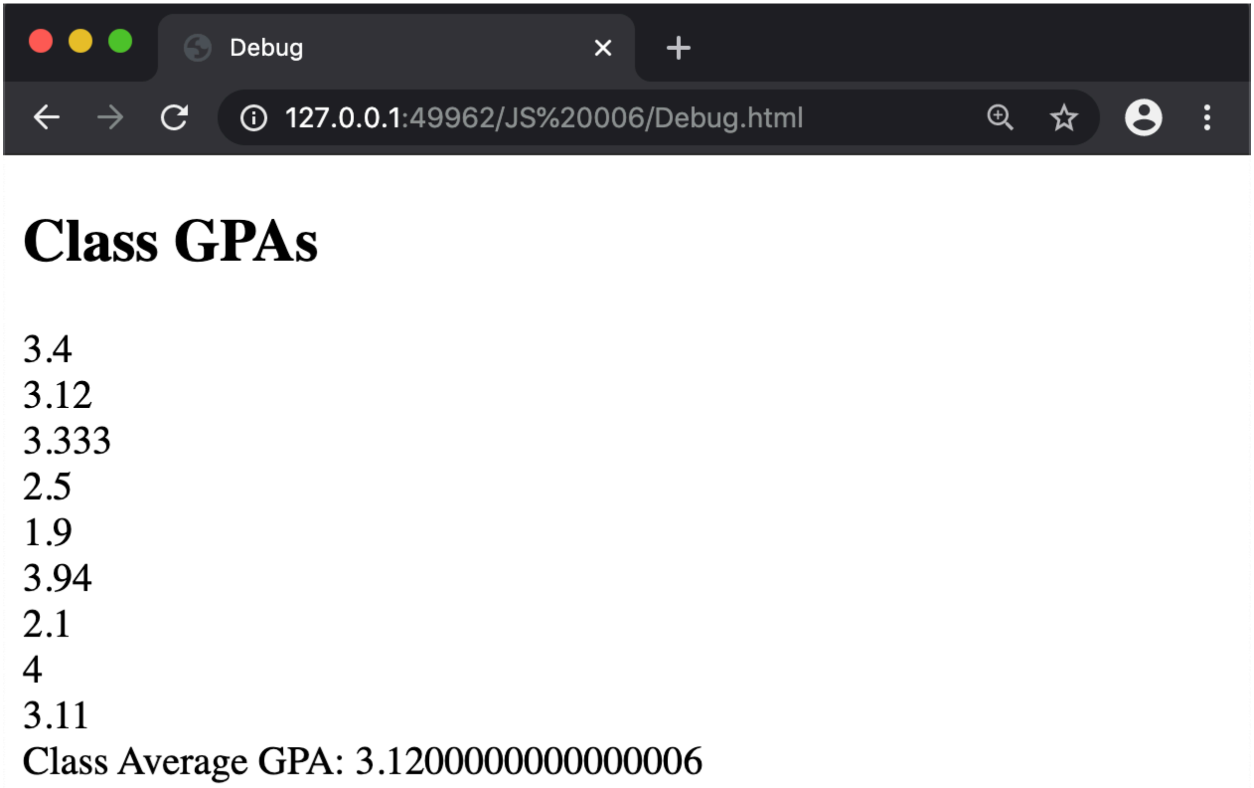
Do This: Debugging

This code isn’t working.

```
<div id=”output”></div>

<script>

var gpas = (3.4, 3.12, 3.333, 2.5, 1.9, 3.94, 2.1, 4, 3.11);

var out = “<h2>Class GPAs</h2>”;
var total = 0;

var length = gpas.length;
for(var i=0; i < length; i++)
   {
       var out += gpas[i];
       out += “<br/>”;
       total += gpas[1];
   }

var average += total/length;
out += “Class Average GPA: “ + average;
```

```
document.getElementById('output').innerHTML = out;

</script>
```

Debug the code and correct the errors so we get output that looks like the screenshot below.



Class GPAs

```
3.4
3.12
3.333
2.5
1.9
3.94
2.1
4
3.11
Class Average GPA: 3.1200000000000006
```

Submit This: Lab Exercise

Consider the following code:

```
<script>
   var names = [];
   var theName = "";
   while (theName != "XXX")
   {
      theName = prompt("Enter a name or XXX to Stop");
```

```
            if(theName != “XXX”){
                names.push(theName);
            }
        }

        for(var x = 0; x < names.length; x++)
        {
            alert(names[x]);
        }
    </script>
```

Using this code as a baseline, please write a program that asks the user to enter a list of GPA’s (and XXX to quit). Once the GPA’s are entered output the following information:

A) The GPA’s in order from largest to smallest.
B) The average of the GPA’s
C) The highest and lowest GPA for the class
D) A list of the outstanding GPA’s (Defined as GPA’s over 3.4).

Good luck!

Please save your file in the following format to insure proper credit:

`LastName_Exercise6.`

Remember every exercise must be submitted in order for you to earn certification. Once you’ve completed this exercise, you’re ready to move on to Section 7.