# JavaScript for App Development

By Mark Lassoff, Founder Framework Television

## Section Eight

### Section 8 Goals

In this section of the course your goals are:

- ☐ Understand Variable Scope
- ☐ Use the `let` Statement to Declare Variables
- ☐ Distinguish `let` from `var`
- ☐ Create a Constant with const
- ☐ Understand Scope as Related to Constants
- ☐ Declare a Constant Object
- ☐ Declare a Constant Array

### Watch This: Section 8 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=Co4KF1CrSfE

## Variable Scope

Variable Scope determines where variables are visible and accessible as your code executes.

Limiting scope is usually a good idea to prevent accidental collisions between variables with the same names. For example, let's say you used code from a game library. That game library has a variable called `playerScore` and your own code has a variable with the same name.

Can you imagine what would happen if the code in the library and your code were both manipulating the same version of `playerScore`? It wouldn't work well.

However, assuming variables are properly scoped, this won't happen and each variable will only be accessible in its own relevant domain.

Examine the code block below.

```
<script>
  //Global Variables, accessible everywhere
  var x = 0;
  var name = "Mark";

  function myFunction(){
    //Local Variables only accessible within the function
    var z = 25;
    var exxes = 12;

    //Local Version of x is different from Global x
    //This is only accessible within the function
    var x = 19;
  }

<script>
```

Fundamentally, there are two variable scopes when you declare variables with `var`. Variables declared outside a function are global and can be accessed from anywhere within your Javascript. Variables declared inside a function are local and can be accessed only from inside the function.

If you declare variables without the benefit of the `var` statement (which for reasons beyond explanation you CAN do in JavaScript, these rules don't apply. Promise me you won't try this.

If you declare the same variable within a function and globally you are creating two distinct variables. *This is obviously not a best practice*.

## `let`

ES6 has introduced the keyword `let`. It is now the preferred way to declare your variables.

You may be wondering, why we didn't start with `let`. The reason is that there is a couple decades worth of Javascript out there that predates ES6. You have to know the old and understand the new. Likely your first jobs in interview will be maintaining code others wrote—not writing your own code from scratch.

Declaring variables with `let` is no different than declaring variables with `var`.

```
<script>
  let name = “Mark Lassoff”;
  let favoriteBand = “Journey”;
  let age = 44.5;
</script>
```

There are some differences in how the JavaScript processor handles your variable behind the scenes.

First, when you declare your variable with `let` it is not part of the Window object. (We haven't covered the Window object yet, but consider it the object that maintains state and status of the browser and your app). Technically you could access variables declared with `var` through the window object like this: `window.yourVariable`.

The second major difference between `var` and `let`, is scope. We'll talk about that next.

## `let` Versus `var`

Variables declared with `let` are block scoped. What this means, is that the variables are accessible within the block in which they are declared. Examine the following code:

```
<script>
let y = 100;
   {
      let x = 15;

      //Can reference x here
      alert(x);

      //Can reference y here
      alert(y);
   }
//x is undefined here.  It was declared in the block
alert(x);
</script>
```

In the example above, `y` is globally scoped so it's accessible everywhere. `x`, however, is block scoped. References made to `x` outside the block in which its declared will be `null`.

## Creating Constants with `const`

Every programming language has the ability to create Constants. Now with ES6, Javascript has joined the rest of the programming world!

Constants are essentially variables whose value does not change throughout the life of your program execution. They used to store any type of value that you'll need to reference but won't change. They are declared with the `const` keyword.

```
<script>
   const APPLES = 12;
   alert(APPLES);   //12

   APPLES = APPLES + 2; //Not allowed.. Value is constant
   APPLES = 2; //Not allowed.
</script>
```

By convention, the names of constants are typed in all caps.

## Constant Scope

Constants are scoped the same as variables declared with `let`. Constants declared with `const` may be block scoped or globally scoped.

> While there is no hard and fast rule, if I declare a constant, I'll do it in the global scope. This is a matter of design as I'll want the constant to substitute for a value in several locations within the code.

## Do This: Discuss Constants

In the class Slack discussion, give a few examples of values you might declare as constants within a program and why. Be creative. For example: *I would declare the size of a game board in a constant, so this way if I wanted to change the size of the game board, I'd only have to do it one place*.

## `const` Objects

While of limited utility, objects can be declared with const in ES6 JavaScript. You might think that these objects would then be immutable; however, that's not the case. The reality is a bit confusing.

```
const wine = {type:"Chardonnay", brand:"McGivneys",
color:"white",
 price: 9.99};

// In a const Object you can modify a property
wine.price = 10.99;

// In a const Object you can add a property
wine.starRating = 5;
```

Kind of bizarre, right? (Keep in mind I didn't create Javascript, I just teach it!)

The one thing you cannot do in this case is redefine `wine` in this script. That would result in an error.

As I mentioned, I find `const` objects to be of limited utility.

## `const` Arrays

Like `const` objects, Arrays can be declared as constants. Arrays declared as constants cannot be redeclared, but the array itself is mutable.

```
const coffee = [“Starbucks”, “Peete’s”, “Dunkin’ Donuts”,
“Caribou Coffee”, “Gloria Jeans”, “McDonalds”];

//We can modify an array member
coffee[2] = “Dunkin”;

//We can add to the array
coffee.push(“Coffee Beenery”);
```

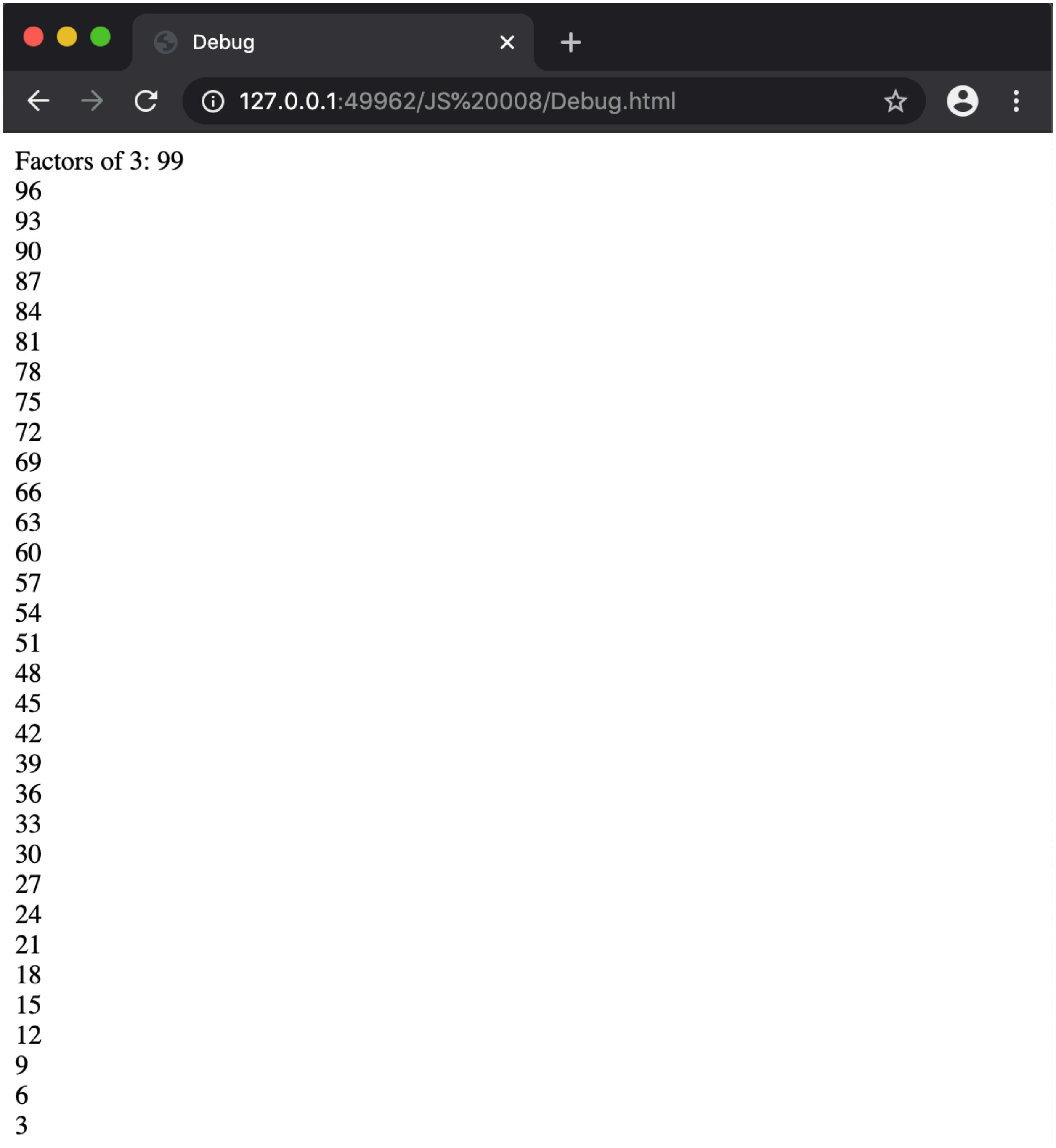
In practice, you’ll rarely run into Arrays or Objects declared as constants, but, now you know how to handle them if you do!

## Do This: Debugging

This code isn’t working. There’s a problem with scope. And that’s not the only problem!

```
<div id=”out”></div>
<script>

   init();

   function init()
   {
   let factor = 3;
    const maxValue = 100;
     const minValue = 0;


      for(let x = maxValue; x > minValue; x--;)
      {
         if(x % factor == 0)
         {
            let output = “”;
            output += x;
            output += “<br/>”;
         }
      }
   }

   document.getElementById(“out”).innerHTML = “Factors of “ +
factor
   + “:<br/> “;
   document.getElementById(“out”).innerHTML += output;

</script>
```

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



```
Factors of 3: 99
96
93
90
87
84
81
78
75
72
69
66
63
60
57
54
51
48
45
42
39
36
33
30
27
24
21
18
15
12
9
6
3
```

## Submit This: Lab Exercise

We're going to redo the calculator exercise from earlier in the course, but this time, please don't use `var` to declare your variables. Use `let` and, if appropriate, `const`. Here are the instructions:

Create a calculator. You will prompt the user three times. For the first prompt, ask the user for the first number to be calculated. For the second prompt, ask the user for the second number to be calculated. On the third prompt ask for an operation ( + , - , * , / , %).

Once the user entries to the prompts are stored, perform the operation requested and output the entire equation. For example if the user enters 4, 5 and + output "9 + 4 = 13"

**One additional caveat: The results of the calculation must be shown using a function called showResult(). Lab submissions not using this function will be returned for correction.**

Good luck!

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

`LastName_Exercise8.`

*Remember every exercise must be submitted in order for you to earn certification.*