## **JAVASCRIPT FUNDAMENTALS PART 4**

### **Arrays**

Strings and numbers may be our building blocks but as your scripts get more complex you’re going to need a way to deal with large quantities of them. Luckily, JavaScript has a couple of data types that are used for just that. An Array is simply an ordered collection of items (Strings, numbers, or other things).

1. [This tutorial](https://www.w3schools.com/js/js_arrays.asp) is a great introduction.
2. [This article](https://www.w3schools.com/js/js_array_methods.asp) covers some of the most useful built-in array methods. These fundamentals are something you’ll use every day, so don’t rush too much and miss out!

### **Loops**

Computers don’t get tired, and they’re really *really* fast! For that reason they are well suited to solving problems that involve doing calculations multiple times. In some cases a computer will be able to repeat a task *thousands* or even *millions* of times in just a few short seconds where it might take a human many hours. (obviously speed here depends on the complexity of the calculation and the speed of the computer itself). One way to make a computer do a repetitive task is using a loop

1. Read this [MDN article](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Looping_code). It’s a longer one, but make sure you tackle the ‘Active Learning’ sections at the bottom of the page.
2. Once again, same info, slightly different context from [JavaScript.info](http://javascript.info/while-for) (Skim the info if you think you know it all, but don’t forget the tasks at the end of the page. You learn best by *doing*)

### **Practice**

### **Test Driven Development**

Test Driven Development (TDD) is a phrase you often hear in the dev world. It refers to the practice of writing automated tests that describe how your code should work before you actually write the code. For example, if you want to write a function that adds a couple of numbers, you would first write a test that uses the function and supplies the expected output. Before you write your code the test will fail, and you should be able to know that your code works correctly when the tests pass.

In many ways TDD is much more productive than writing code without tests. If we didn’t have the test for the adding function above, we would have to run the code ourselves over and over, plugging in different numbers until we were sure that it was working… not a big deal for a simple add(2, 2), but imagine having to do that for more complicated functions, like checking whether or not someone has won a game of tic tac toe: (game\_win(["o", null,"x",null,"x",null,"x", "o", "o"])) If you didn’t do TDD then you might actually have to play multiple games against yourself just to test if the function was working correctly!

We will teach you the art of actually writing these tests later in the course. The following exercises have the tests already written out for you. All you have to do is read the specs and write the code that makes them pass! The very first exercise (01-helloWorld) is intentionally very simple and walks you through the process of running the tests and making them pass.

### **Good Luck!**

Check out our exercises repository [here](https://github.com/TheOdinProject/javascript-exercises) and follow the directions in the [README](https://github.com/TheOdinProject/javascript-exercises#how-to-use-these-exercises) for setting up Jest. Solutions for the exercises can be found on the ‘solutions’ branch of that repo.

Complete the following exercises:

* helloWorld
* repeatString
* reverseString
* removeFromArray
* sumAll
* leapYears
* tempConversion