CODE ACADEMY JAVASCRIPT

* There are three essential *data types* in JavaScript: strings, numbers, and booleans.
* We can write out anything to the console with console.log.
* We can do math with operators like +, -, \*, and /.
* We can find the remainder after dividing two numbers with a modulus: %.
* We can generate a random number with Math.random, then make it a whole number with Math.floor.
* We write a single line comment with // and a multi-line comment with /\* and \*/
* if/else statements make binary decisions and execute separate code based on a condition.
* We can add extra conditions with to if/elsestatements with else if conditions.
* switch statements make complicated if/elsestatements easier to read, however they achieve the same result as if/else statements.
* *Comparison operators*, like <, >, <=, and >= can compare two variables. After they compare, they always return either true or false.
* *Logical Operators*, like &&, ||, !==, and !, can compare two variables to see if a certain condition exists:
  + && checks if both sides are true.
  + || checks if either side is true.

This unit introduced you to functions.

* *Functions* are written to perform a task.
* Functions take data or variables, perform a set of tasks on them, and then return the result.
* We can define parameters when calling the function.
* When calling a function, we can pass in *arguments*, which will set the function's parameters.
* We can use return to return the result of a function which allows us to call functions anywhere, even inside other functions.
* *Arrays* are lists and are a way to store data in JavaScript. Each item inside of an array is at a numbered position. Arrays are created with brackets [].
* We can access one item in an array using it's numbered position, with syntax like: myArray[0].
* Arrays have a length property, which allows you to see how many items are in an array.
* Arrays also have their own methods, including pushand pop, which add and subtract items from an array, respectively.

FOR LOOP

Since this syntax is a little complicated, let's break it into 4 parts:

1. Within the for loop's parentheses, the *start condition* is var i = 0, which means the loop will start counting at 0.
2. The *stop condition* is i < animals.length, which means the loop will run as long as i is less than the length of the animals array. When i is greater than the length of the animals array, the loop will stop looping.
3. The *iterator* is i++. This means that each loop, iwill have 1 added to it.
4. And finally, the code block is inside the { ... }. The block will run each loop, until the loop stops.

The secret to loops is that i, the variable we created inside the for loop's parentheses, is always equal to a number. To be more clear, the first loop, i will equal 0, the second loop, i will equal 1, and the third loop, iwill equal 2.

This makes it possible to write animals[0], animals[1], animals[2] programmatically instead of by hand. We can write a for loop, and replace the hard coded number with the variable i, like this: animals[i].

If we can make a for loop run forwards through an array, can we make it run backwards through it? Of course!

We can make out loop run backwards by modifying the start, stop, and iterator conditions.

To do this, we'll need to edit the code between the forloop's parentheses:

1. The start condition should set i to the length of the array.
2. The stop condition should end when i is 0.
3. The iterator should subtract 1 each time, which is done with i--.

Great job! In this unit we learned how to write less repetitive code with loops.

* for loops allow us to repeat a block of code a known amount of times.
* We can use a for loop inside another for loop to compare two lists.
* while loops are for looping over a code block an unknown amount of times.

SCOPE

Scope is a big idea in programming, so let's start at a high level.

Scope refers to where in a program a variable can be accessed. The idea is that some variables are unable to be accessed everywhere within a program.

Think of it like a an apartment building. Everyone who lives in the apartment building is under the *global scope* of the building and its manager. So, if there are rats in the shared laundry room, everyone has access to the laundry machines, and the rats.

If you write a variable outside of a function in JavaScript, it's in the *global scope* and can be used by any other part of the program, just like the laundry room can be used by everyone in an apartment.

This unit introduced you to scope.

* *Scope* is the idea in programming that some variables are acessible/inaccessible from other parts of the program.
* *Global Scope* refers to variables that are accessible to every part of the program.
* *Functional Scope* refers to variables created inside functions, which are not accessible outside of its block.