**SCOPE**

**Review: Scope**

In this lesson, you learned about scope and how it impacts the accessibility of different variables.

Let's review the following terms:

* **Scope** is the idea in programming that some variables are accessible/inaccessible from other parts of the program.
* **Blocks** are statements that exist within curly braces {}.
* **Global scope** refers to the context within which variables are accessible to every part of the program.
* **Global variables** are variables that exist within global scope.
* **Block scope** refers to the context within which variables that are accessible only within the block they are defined.
* **Local variables** are variables that exist within block scope.
* **Global namespace** is the space in our code that contains globally scoped information.
* **Scope pollution** is when too many variables exist in a namespace or variable names are reused.

As you continue your coding journey, remember to use best practices when declaring your variables! Scoping your variables tightly will ensure that your code has clean, organized, and modular logic.

**ARRAYS**

# Arrays

Organizing and storing data is a foundational concept of programming.

One way we organize data in real life is by making lists. Let's make one here:

New Year's Resolutions: 1. Keep a journal 2. Take a falconry class 3. Learn to juggle

Let's now write this list in JavaScript, as an array:

let newYearsResolutions = ['Keep a journal', 'Take a falconry class', 'Learn to juggle'];

Arrays are JavaScript's way of making lists. Arrays can store any data types (including strings, numbers, and booleans). Like lists, arrays are ordered, meaning each item has a numbered position.

Here's an array of the concepts we'll cover:

let concepts = ['creating arrays', 'array structures', 'array manipulation']

By the end of this lesson you'll have another tool under your belt that helps you manage chunks of data!

**ARRAYS**

**Create an Array**

One way we can create an array is to use an *array literal*. An array literal creates an array by wrapping items in square brackets []. Remember from the previous exercise, arrays can store any data type — we can have an array that holds all the same data types or an array that holds different data types.

Let's take a closer look at the syntax in the array example:

* The array is represented by the square brackets [] and the content inside.
* Each content item inside an array is called an *element*.
* There are three different elements inside the array.
* Each element inside the array is a different data type.

We can also save an array to a variable. You may have noticed we did this in the previous exercise:

let newYearsResolutions = ['Keep a journal', 'Take a falconry class', 'Learn to juggle'];

Let's practice by making an array of our own.

Remember to surround your array with brackets: []. That's what makes it an array! Look at the follow syntax:

const sampleArray = [1, 'Hello World!', true];

However, in the hobbies array, there should be three strings!

**Accessing Elements**

Each element in an array has a numbered position known as its *index*. We can access individual items using their index, which is similar to referencing an item in a list based on the item's position.

Arrays in JavaScript are *zero-indexed*, meaning the positions start counting from 0 rather than 1. Therefore, the first item in an array will be at position 0. Let's see how we could access an element in an array:

In the code snippet above:

* cities is an array that has three elements.
* We're using bracket notation, [] with the index after the name of the array to access the element.
* cities[0] will access the element at index 0 in the array cities. You can think of cities[0] as accessing the space in memory that holds the string 'New York'.

You can also access individual characters in a string using bracket notation and the index. For instance, you can write:

const hello = 'Hello World'; console.log(hello[6]); // Output: W

The console will display W since it is the character that is at index 6.

**ARRAYS**

**Accessing Elements**

Each element in an array has a numbered position known as its *index*. We can access individual items using their index, which is similar to referencing an item in a list based on the item's position.

Arrays in JavaScript are *zero-indexed*, meaning the positions start counting from 0 rather than 1. Therefore, the first item in an array will be at position 0. Let's see how we could access an element in an array:

In the code snippet above:

* cities is an array that has three elements.
* We're using bracket notation, [] with the index after the name of the array to access the element.
* cities[0] will access the element at index 0 in the array cities. You can think of cities[0] as accessing the space in memory that holds the string 'New York'.

You can also access individual characters in a string using bracket notation and the index. For instance, you can write:

const hello = 'Hello World'; console.log(hello[6]); // Output: W

The console will display W since it is the character that is at index 6.

**1.**

Individual elements in arrays can also be stored to variables.

Create a variable named listItem and set it equal to the first item in the famousSayings array using square bracket notation ([]).

Then use console.log() to print the listItem variable to the console.

Hint

The first item within an array is always at position [0].

const alphabet = ['a', 'b', 'c']; const firstLetter = alphabet[0]; // 'a'

**2.**

Now, console.log() the third element in the famousSayings array using bracket notation to access the element.

Do not save the element to a new variable before you log it.

Hint

We can directly print to the console by using the index to access the element in the array.

const greeting = 'Hello World!'; console.log(greeting[1]); // Output: e

**3.**

Awesome, you can access each element in an array using the index. But what happens if you try to access an index that is beyond the last element?

Try to log the item at index [3] of famousSayings to the console. What is logged to the console?

Hint

When you try to access an index in an array that does not contain an element you will get a value of undefined.

Since we're trying to access an the element in index 3 in the famousSayings, the value will be undefined.

Your code should use the following syntax:

console.log(famousSayings[3]);

**ARRAYS**

# Update Elements

In the previous exercise, you learned how to access elements inside an array or a string by using an index. Once you have access to an element in an array, you can update its value.

let seasons = ['Winter', 'Spring', 'Summer', 'Fall']; seasons[3] = 'Autumn'; console.log(seasons); //Output: ['Winter', 'Spring', 'Summer', 'Autumn']

In the example above, the seasons array contained the names of the four seasons.

However, we decided that we preferred to say 'Autumn' instead of 'Fall'.

The line, seasons[3] = 'Autumn'; tells our program to change the item at index 3 of the seasons array to be 'Autumn' instead of what is already there.

**1.**

Change the second element of the array groceryList to 'avocados'.

Hint

To reassign an element of an array, access the element by index and then use the = operator to assign a new value. Remember JavaScript is zero-indexed, the first element has an index of 0.

Answer: let groceryList = ['bread', 'tomatoes', 'milk'];

groceryList[1] = 'avocados';

Example

let sampleArr = ['one', 'two', 'three']; sampleArr[0] = 1; console.log(sampleArr); // Output: [1, 'two', 'three']

**ARRAYS**

# Arrays with let and const

You may recall that you can declare variables with both the let and const keywords. Variables declared with let can be reassigned.

Variables declared with the const keyword cannot be reassigned. However, elements in an array declared with const remain mutable. Meaning that we can change the contents of a const array, but cannot reassign a new array or a different value.

The instructions below will illustrate this concept more clearly. Pay close attention to the similarities and differences between the condiments array and the utensils array as you complete the steps.

**1.**

Below the two existing arrays, re-assign the element in index 0 of condiments to 'Mayo'.

Log the updated array, condiments, to the console.

Hint

To re-assign an element using its index, use the bracket notation:

let fictionalChars = ['Harry Potter', 'Hans Solo', 'Eva Luna']; fictionalChars[1] = ('Lauren Olamina'); console.log(fictionalChars); // ['Harry Potter', 'Lauren Olamina', 'Eva Luna']

Remember to log condiments after you use re-assign the element inside condiments.

**2.**

Below your code from Step 1, reassign condiments to be a new array that contains a single string ['Mayo']

Log the result to the console.

Notice that you can re-assign elements in an array and re-assign an entire new array to a variable declared using the let keyword.

Hint

You can reassign an element to another array using the following syntax:

let sampleArr = [1, 2, 3]; sampleArr = ['a string'];

Remember to log condiments after you reassign it.

**3.**

Below your code from Step 2, re-assign the last item from the utensils array to 'Spoon'.

Log the updated array to the console.

let condiments = ['Ketchup', 'Mustard', 'Soy Sauce', 'Sriracha'];

const utensils = ['Fork', 'Knife', 'Chopsticks', 'Spork'];

condiments[0] = 'Mayo';

console.log(condiments);

condiments = ['Mayo'];

utensils[3] = 'Spoon';

console.log(utensils);

// utensils = ['Spork'];

**The .length property**

One of an array's built-in properties is lengthand it returns the number of items in the array. We access the .length property just like we do with strings. Check the example below:

const newYearsResolutions = ['Keep a journal', 'Take a falconry class']; console.log(newYearsResolutions.length); // Output: 2

In the example above, we log newYearsResolutions.length to the console using the following steps:

* We use *dot notation*, chaining a period with the property name to the array, to access the length property of the newYearsResolutions array.
* Then we log the length of newYearsResolution to the console.
* Since newYearsResolution has two elements, so 2 would be logged to the console.

When we want to know how many elements are in an array, we can access the .lengthproperty.

**Instructions**

**1.**

Find the length of the objectives array and log it to the console

Answer:

const objectives = ['Learn a new languages', 'Read 52 books', 'Run a marathon'];

console.log(objectives.length); //Find length of objective array//

**The .push() Method**

Let's learn about some built-in JavaScript methods that make working with arrays easier. These methods are specifically called on arrays to make common tasks, like adding and removing elements, more straightforward.

One method, .push() allows us to add items to the end of an array. Here is an example of how this is used:

const itemTracker = ['item 0', 'item 1', 'item 2']; itemTracker.push('item 3', 'item 4'); console.log(itemTracker); // Output: ['item 0', 'item 1', 'item 2', 'item 3', 'item 4'];

So, how does .push() work?

* We access the push method by using dot notation, connecting push to itemTrackerwith a period.
* Then we call it like a function. That's because .push() is a function and one that JavaScript allows us to use right on an array.
* .push() can take a single argument or multiple arguments separated by commas. In this case, we're adding two elements: 'item 3' and 'item 4' to itemTracker.
* Notice that .push() changes, or *mutates*, itemTracker. You might also see .push()referred to as a *destructive* array method since it changes the initial array.

If you're looking for a method that will mutate an array by adding elements to it, then .push()is the method for you!

**1.**

Add two elements to the chores array using .push().

Stuck? Get a hint

**2.**

Use console.log to print your choresarray to make sure your items were added.

Answer:

const chores = ['wash dishes', 'do laundry', 'take out trash'];

chores.push('dust', 'make bed');

console.log(chores);

**ARRAYS**

**The .pop() Method**

Another array method, .pop(), removes the last item of an array.

const newItemTracker = ['item 0', 'item 1', 'item 2']; const removed = newItemTracker.pop(); console.log(newItemTracker); // Output: [ 'item 0', 'item 1' ] console.log(removed); // Output: item 2

* In the example above, calling .pop() on the newItemTracker array removed item 2 from the end.
* .pop() does not take any arguments, it simply removes the last element of newItemTracker.
* .pop() returns the value of the last element. In the example, we store the returned value in a variable removed to be used for later.
* .pop() is a method that mutates the initial array.

When you need to mutate an array by removing the last element, use .pop().

**1.**

Use the .pop() method to remove the last element from chores.

Hint

To remove an element from an array using .pop() check out the following example:

let carrot = ['pointy part', 'mid section', 'the end'] carrot.pop(); console.log(carrot); // Output: ['pointy part', 'mid section']

**2.**

In a line after you called chores.pop(), log chores to the console to make sure it worked.

Stuck? Get a hint

const chores = ['wash dishes', 'do laundry', 'take out trash', 'cook dinner', 'mop floor'];

chores.pop();

console.log(chores);