Intro to JavaScript Arrays Lab

Exercise

Introduction

This lab provides an opportunity to practice defining, accessing, and manipulating arrays.

A quick note before you dive in

Don't hesitate to collaborate with your classmates when working through labs.

If you get stuck during the lab, we recommend revisiting the lesson materials first. They're designed to provide you with the information and examples that will help you complete the exercises.

The internet is filled with resources specific to JavaScript arrays. The [MDN web docs on arrays] and [Stack Overflow] are just a few clicks away. Use these as a last resort before reaching out for help.

Happy coding!

Lab exercises

Copy and paste each of the following exercises into your JavaScript file. To check your work, open the file in a web browser and inspect the console output, or run the file in your terminal using `node app.js`.

Exercise 1: Define an empty array

```
/*
Exercise 1: Define an empty array

1) Create an empty array and assign it to a variable called `foods`.

Exercise 1 has been completed for you:
*/

const foods = [];

console.log('Exercise 1 result:', foods);
```

Exercise 2: Add strings to the array

```
/*
Exercise 2: Add strings to the array

1) Add 'pizza' and 'cheeseburger' to the `foods` array.

Note: 'pizza' should be the first item in the array, followed by 'cheeseburger'.

Complete Exercise 2 in the space below:
*/



console.log('Exercise 2 result:', foods);
```

Adding to the front of an array

We know that `push()` and `pop()` are helpful methods for adding and removing items at the end of an array. `shift()` and `unshift()` accomplish the same tasks at the beginning of an array!

The `shift()` and `unshift()` methods provide an easy way to manipulate elements at the beginning of an array. The `unshift()` method ***adds*** one or more elements to the start of the array and returns the new length, while the `shift()` method ***removes*** the first element from an array and returns that element.

Check out the docs on `shift()` and `unshift()` for more info.

Exercise 3: Insert at the beginning

```
/*
Exercise 3: Insert at the beginning

1) Insert the string 'taco' at the beginning of the `foods` array.

Complete Exercise 3 in the space below:
*/



console.log('Exercise 3 result:', foods);
```

Exercise 4: Access an array element

```
/*
Exercise 4: Access an array element

1) Retrieve the 'pizza' string from the array based on its position (index) in
   the array.

2) Assign it to a variable called `favFood`.

Complete Exercise 4 in the space below:
*/



console.log('Exercise 4 result:', favFood);
```

Adding and removing elements anywhere in an array

The `splice()` method can ***add or remove*** any number of elements inside an array. You can even take both actions simultaneously with a single line of code!

If we read the docs, we'll find the syntax to be:

```
splice(start)
splice(start, deleteCount)
splice(start, deleteCount, item0)
splice(start, deleteCount, item0, item1)
splice(start, deleteCount, item0, item1, /* …, */ itemN)
```

This syntax may seem overwhelming at first glance, but it means that all but the first parameter is optional - so when we call `splice()`, its behavior will change depending on how many arguments we pass to it. Give it a try!

Exercise 5: Insert an element between two others

```
/*
Exercise 5: Insert an element between two others

1) Insert the string 'tofu' between 'pizza' and 'cheeseburger' in the array.

Complete Exercise 5 in the space below:
*/



console.log('Exercise 5 result:', foods);
```

Exercise 6: Replace elements

```
/*
Exercise 6: Replace elements

1) Replace 'pizza' in the `foods` array with 'sushi' and 'cupcake'.

Complete Exercise 6 in the space below:
*/



console.log('Exercise 6 result:', foods);
```

Copying parts of an array

The `slice()` method extracts a portion of an array and creates a new array without modifying the original. Check out the docs!

Exercise 7: Using the `slice()` method

```
/*
Exercise 7: Using the `slice()` method

1) Use the `slice()` method to create a new array that contains 'sushi' and
   'cupcake'.

2) Assign it to a variable named `yummy`.

Complete Exercise 7 in the space below:
*/



console.log('Exercise 7 result:', yummy);
```

Searching for items within an array

The `indexOf()` method searches for a specified element within an array and returns the index of the first occurrence. If the element is not found, it returns -1. Here are the docs.

Exercise 8: Finding an index

```
/*
Exercise 8: Finding an index

1) Using the `indexOf()` method, find the index of the string 'tofu' in the
   `foods` array.

2) Assign it to a variable named `soyIdx`.

Complete Exercise 8 in the space below:
*/



console.log('Exercise 8 result:', soyIdx);
```

Turning an entire array into a string

The `join()` method combines all the elements of an array into a single string, separated by a specified *delimiter*. Find the `join()` docs here.

> 📚 A *delimiter* is a character or group of characters used to separate elements within a string.

Exercise 9: Joining elements

```
/*
Exercise 9: Joining elements

1) Use the `join()` method to concatenate the strings in the `foods` array,
   separated by ' -> '.

2) Assign the result to a variable called `allFoods`.

Note: The final result should log as:
'taco -> sushi -> cupcake -> tofu -> cheeseburger'

Complete Exercise 9 in the space below:
*/



console.log('Exercise 9 result:', allFoods);
```

Checking for specific elements in an array

The `includes()` method checks if an array contains a specific element, returning a boolean value (true or false). Read more about the `includes()` method here.

Exercise 10: Check for an element

```
/*
Exercise 10: Check for an element

1) Using the .includes() method, check if the `foods` array contains the string
   'soup'.

2) Assign the result to a variable called `hasSoup``.

Complete Exercise 10 in the space below:
*/



console.log('Exercise 10 result:', hasSoup);
```

Looping and manipulating

These last few exercises are more difficult. Don't forget to take breaks as you're working on these!

Several methods exist for iterating through arrays in JavaScript.

`for` loop

The traditional `for` loop can iterate through an array by index.

```
for (let i = 0; i < arr.length; i++) {
  console.log(arr[i]);
}
```

`for...of` loop

The `for...of` loop allows you to loop through the values of an array directly without using an index.

```
for (const element of arr) {
  console.log(element);
}
```

`forEach()` method

The `forEach()` method performs a given function on each element of an array.

```
arr.forEach((element) => {
  console.log(element);
});
```

Choose one to practice with for the following exercises!

Exercise 11: Odd numbers from an array

```
/*
Exercise 11: Odd numbers from an array

1) Choose a method to iterate through the `nums` array.

2) Push each odd number to a new array named `odds`.

Hint: Initialize the `odds` variable to an empty array before the iteration.

Complete Exercise 11 in the space below:
*/

const nums = [100, 5, 23, 15, 21, 72, 9, 45, 66, 7, 81, 90];



console.log('Exercise 11 result:', odds);
```

Exercise 12: FizzBuzz with arrays

```
/*
Exercise 12: FizzBuzz with arrays

1) Choose a method to iterate through the `nums` array.

2. As you loop, sort the numbers into new arrays based on the following rules:

   - Push any number evenly divisible by 3 to an array called `fizz`.
   - Push any number evenly divisible by 5 to an array called `buzz`.
   - Push any number that is evenly divisible by 3 and 5 to an array called
     `fizzbuzz`.

   Note: A single number may meet more than one of the above rules. If it does,
         it should be placed in multiple arrays. For example, the number `15`
         will appear in the `fizz`, `buzz`, and `fizzbuzz` arrays.

Complete Exercise 12 in the space below:
*/



console.log('Exercise 12 Results:');
console.log('  fizz:', fizz);
console.log('  buzz:', buzz);
console.log('  fizzbuzz:', fizzbuzz);
```

Working with two-dimensional arrays

You've worked with one-dimensional arrays a lot already. Here's one:

```
const oneDArray = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

One-dimensional arrays may be written split across multiple lines if the contents don't fit on one line or if it enhances readability. For example:

```
const oneDArray = [
  1, // 0 index
  2, // 1 index
  3, // 2 index
  // and so on
]
```

This changes nothing about the array. It only changes how we write it.

A two-dimensional array is essentially an array of arrays. The inner arrays are referred to as nested arrays. This creates a structure similar to a grid or a table.

```
const twoDArray = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];
```

- In a one-dimensional array, you access elements using a **single index**. For example, to access the number `3` in `oneDArray` above, you would use `oneDArray[2]`.
- In a two-dimensional array, you use **two indices**: The first specifies the specific array (row), and the second specifies the element in that array (column). To access the number `3` in `twoDArray` above, you would use `twoDArray[0][2]`. See more below.

```
const twoDArray = [
  [1, 2, 3], // index 0
  [4, 5, 6], // index 1
  [7, 8, 9]  // index 2
];
```

And finally, below, we see the indexes of the elements inside of the nested arrays:

```
const twoDArray = [
  [1, 2, 3], // index 0
// 0  1  2
  [4, 5, 6], // index 1
// 0  1  2
  [7, 8, 9]  // index 2
// 0  1  2
];
```

Exercise 13: Retrieve the last array

```
/*
Exercise 13: Retrieve the Last Array

1) Assign the last nested array in the `numArrays` below to a variable named
   `numList`. As you do this, also fulfill these goals:

   - Assume you don't know how many nested arrays `numArrays` contains.
   - Do not alter the original `numArrays` array.

Complete Exercise 13 in the space below:
*/

const numArrays = [
    [100, 5, 23],
    [15, 21, 72, 9],
    [45, 66],
    [7, 81, 90]
];

console.log('Exercise 13 result:', numList);
```

Exercise 14: Accessing within nested arrays

```
/*
Exercise 14: Accessing within nested arrays

1) Retrieve the number `66` from the `numArrays` array. As part of this process
   do not alter the original `numArrays` array.

2) Assign it to a variable called `num`.

Complete Exercise 14 in the space below:
*/



console.log('Exercise 14 result:', num);
```

Looping through nested arrays

To loop through a two-dimensional array, you'll typically use nested loops: one loop for the **rows** and another for the **columns** within each row.

Exercise 15: Nested array sum

```
/*
Exercise 15: Nested array sum

1) Use nested loops or `forEach()` methods to sum up all numbers within
   `numArrays` nested arrays.

2) Assign the sum to a variable called `total`.

Hint: Be sure to declare and initialize the total variable before the iterations.

Complete Exercise 15 in the space below:
*/



console.log('Exercise 15 result:\n', total);
```

🎉 **The End. You did it!** 🎉