Arrays

JavaScript

Arrays

In many programming languages, arrays are their own data-type. JavaScript technically treats arrays as objects, so when you use the typeof operator you may not see the result you expect.

Arrays are, basically, a list of values. Whenever you need to hold multiple values in one variable, consider using an array!

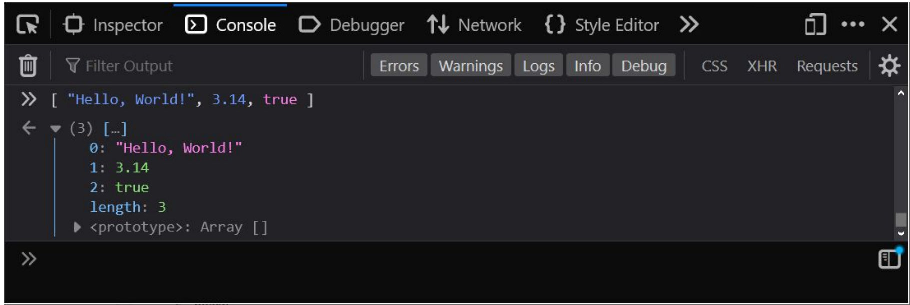
You can store any other data-type within an array, including other arrays.

A JavaScript array will look something like so…

["Hello, World!", 3.14, true]

This array has 3 values inside:

- String
- Number
- Boolean



```
» [ "Hello, World!", 3.14, true ]
← ▼ (3) […]
      0: "Hello, World!"
      1: 3.14
      2: true
      length: 3
    ▶ <prototype>: Array []
»
```

Array Output in the Web Console

Take a close look at the output in the Web Console.

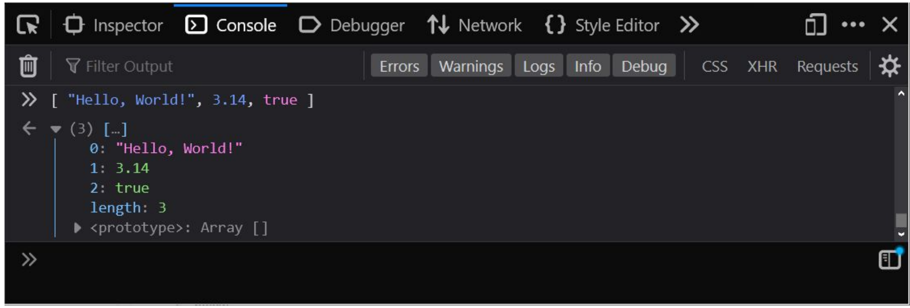
Notice that when you see an array output, you can click the small dropdown array to expand and view the array details.

Each value in the array will be listed, one after the other. In our example, you'll see that each value is prepended with a 0, 1, and 2.

These numbers are the value indexes. Indexes start at 0, not one. Array indexes in most programming languages follow this convention!

Beneath the index numbers, you'll find the array's length property.

Length represents how many values are in the array. In this case we can see that the example contains three values.



```
>> [ "Hello, World!", 3.14, true ]
<- ▼ (3) […]
     0: "Hello, World!"
     1: 3.14
     2: true
     length: 3
   ▶ <prototype>: Array []
>>
```

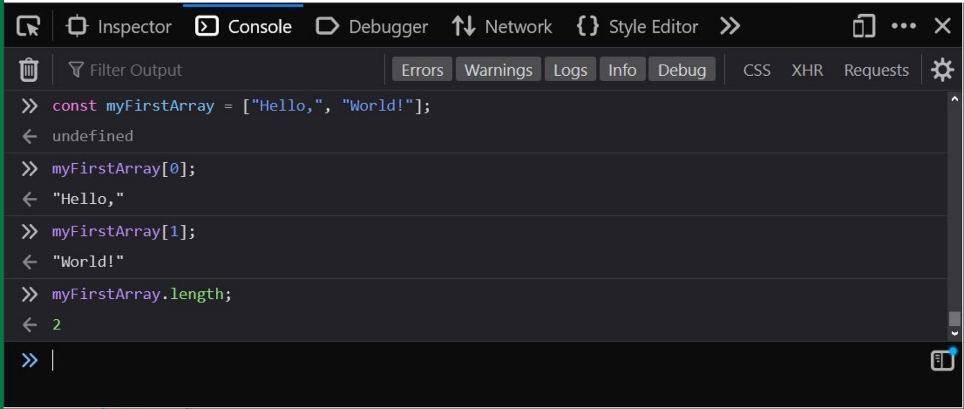
Declaring and Accessing an Array

We can use variables to store arrays, much like any other value we've wanted to store.

Typically you'll use the const keyword when declaring arrays. You can add and remove items, as long as you don't assign a completely new array to the variable.

You can access individual values by index—simply call the variable by name followed by square brackets containing the index number!

If you want the length, call the array by name followed by a period and the word length.



```
>> const myFirstArray = ["Hello,", "World!"];
<- undefined
>> myFirstArray[0];
<- "Hello,"
>> myFirstArray[1];
<- "World!"
>> myFirstArray.length;
<- 2
>>
```

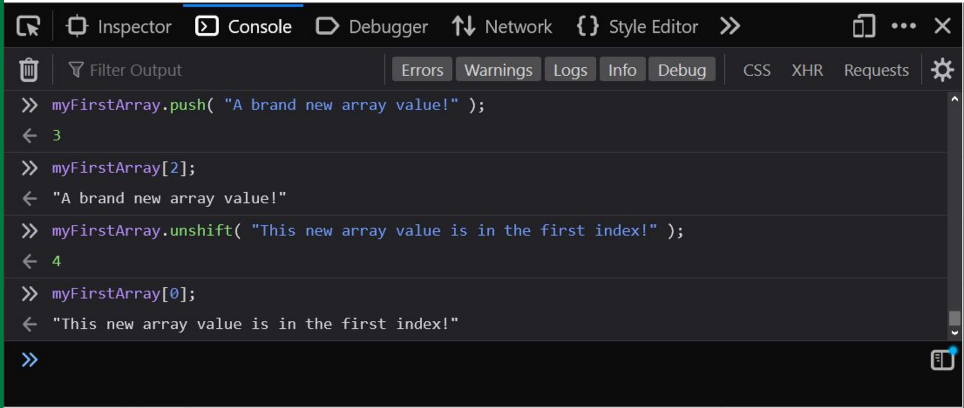
Adding an Item to an Array

We push items to add them to the end of an array, or unshift them to place a new item at the beginning instead.

Array.push will add a new value as an item in your array.

This method will place the new item at the end of the array (it will have a new index number.)

Array.unshift will add a new value as the first item in the array; the other items in the array will make room and adopt new index values so that your new item can use index #0.



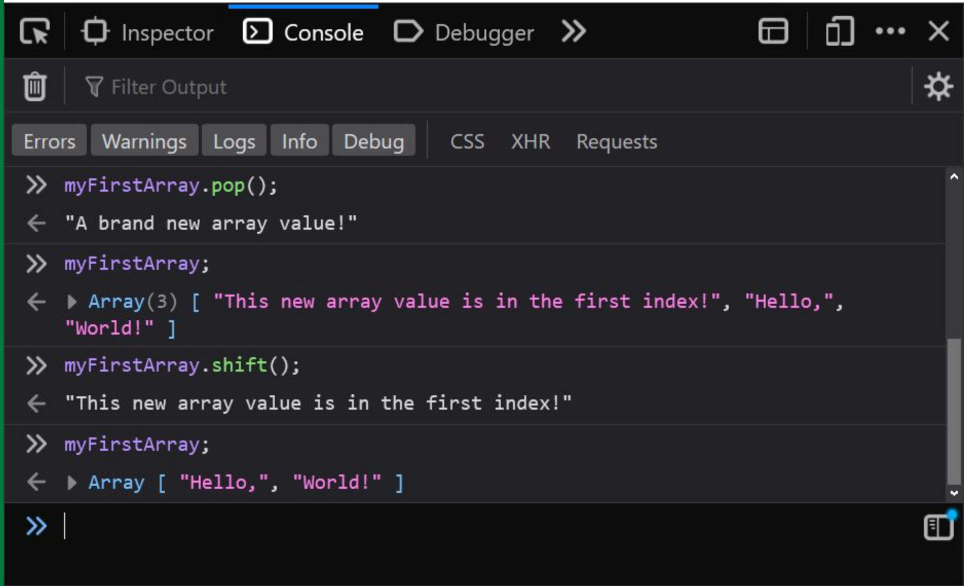
```
>> myFirstArray.push( "A brand new array value!" );
<- 3
>> myFirstArray[2];
<- "A brand new array value!"
>> myFirstArray.unshift( "This new array value is in the first index!" );
<- 4
>> myFirstArray[0];
<- "This new array value is in the first index!"
>>
```

Removing an Item from an Array

We pop items to remove them from the end of an array, or shift them to remove a value from the beginning instead.

Array.pop will remove a value as an item in your array.

Array.shift will remove the value located at the beginning of the array; the other items in the array will move as needed and adopt new index values.



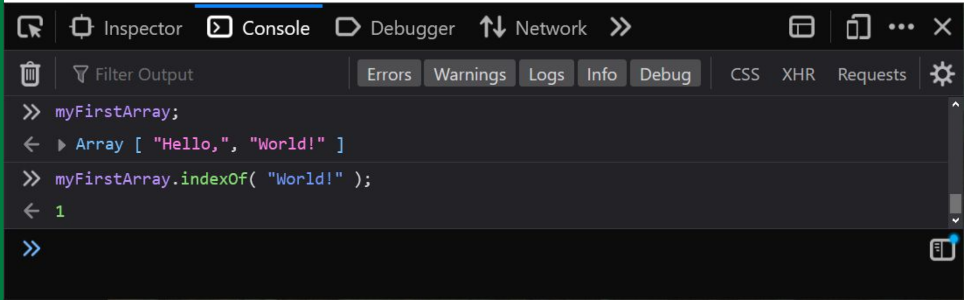
```
>> myFirstArray.pop();
← "A brand new array value!"
>> myFirstArray;
← ▶ Array(3) [ "This new array value is in the first index!", "Hello,",
  "World!" ]
>> myFirstArray.shift();
← "This new array value is in the first index!"
>> myFirstArray;
← ▶ Array [ "Hello,", "World!" ]
>>
```

Find the Index of an Item in an Array

The indexOf method can be used to get the index number for a value in an array.

To use Array.indexOf, you must pass in a value you expect to be inside of the array.

If that value is found, the method will return to you the index number for that value.



```
>> myFirstArray;
<- ▸ Array [ "Hello,", "World!" ]
>> myFirstArray.indexOf( "World!" );
<- 1
>>
```

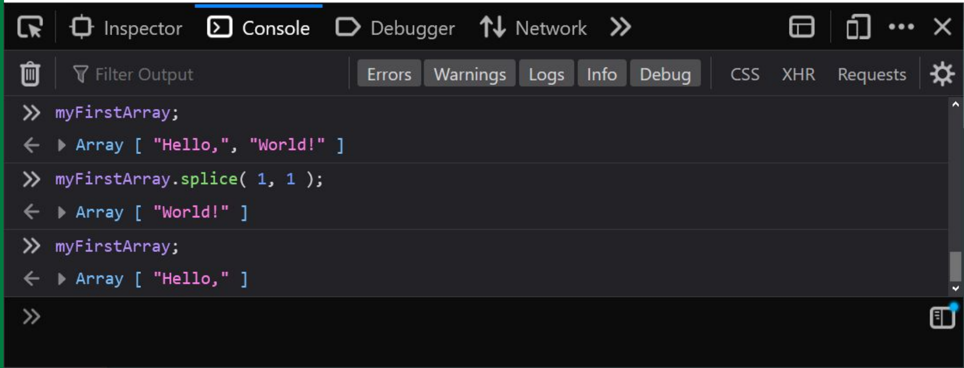
Remove a Specific Item from the Array

Array's splice method is flexible, and can be used for multiple purposes.

Array.splice can take 3 arguments…

- Position (index number)
- Number of items to remove starting at the provided position
- New values to enter into the array

If we wanted to use this to remove the second item in an array (index position 1), you can pass it position 1 and another 1 as the number of items you'd like removed.

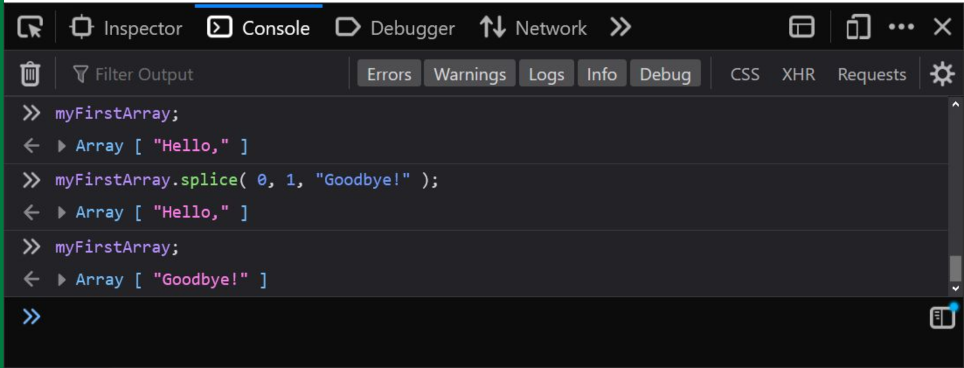


```
>> myFirstArray;
<- ▶ Array [ "Hello,", "World!" ]
>> myFirstArray.splice( 1, 1 );
<- ▶ Array [ "World!" ]
>> myFirstArray;
<- ▶ Array [ "Hello," ]
>>
```

Replace an Item in an Array

Array's splice method can also be used to replace values in your array.

If we wanted to use this to replace the first item, we can target it by position 0, set the method to delete 1 item, and enter a value to add!



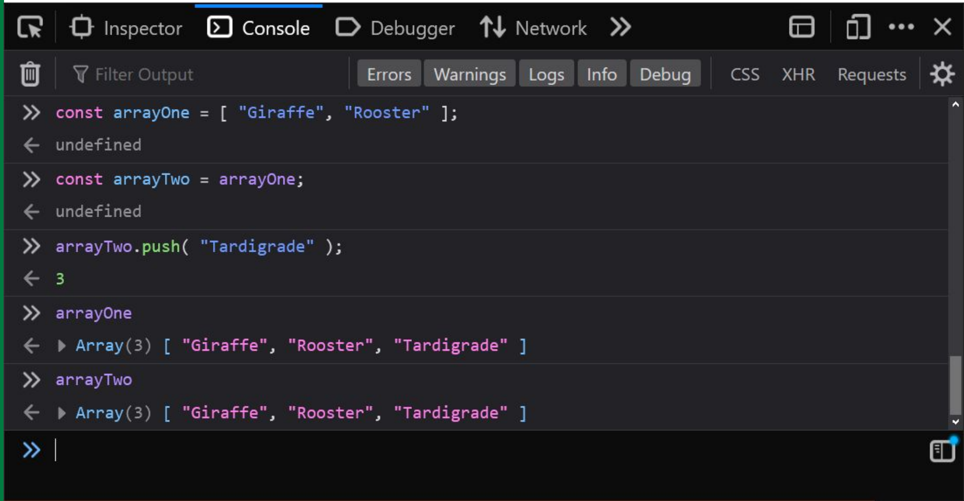
```
>> myFirstArray;
<- ▸ Array [ "Hello," ]
>> myFirstArray.splice( 0, 1, "Goodbye!" );
<- ▸ Array [ "Hello," ]
>> myFirstArray;
<- ▸ Array [ "Goodbye!" ]
>>
```

When Might we Want to Copy an Array?

If you assign a variable containing an existing array—careful—both variable names will actually represent the same array!

Run an experiment…

1. Create a basic array and assign it to a variable.
2. Assign that variable as the value to another variable.
3. Make a change to one, and check on the values of both variables.



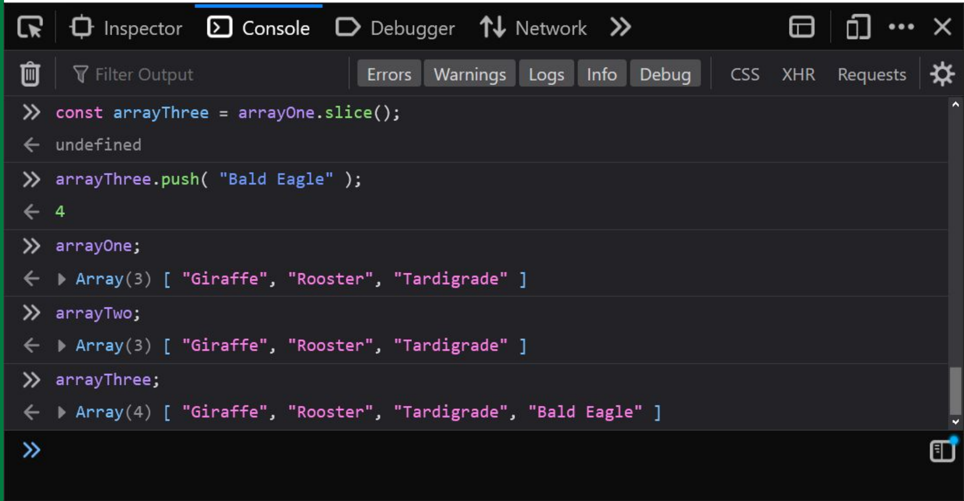
```
>> const arrayOne = [ "Giraffe", "Rooster" ];
<- undefined
>> const arrayTwo = arrayOne;
<- undefined
>> arrayTwo.push( "Tardigrade" );
<- 3
>> arrayOne
<- ▶ Array(3) [ "Giraffe", "Rooster", "Tardigrade" ]
>> arrayTwo
<- ▶ Array(3) [ "Giraffe", "Rooster", "Tardigrade" ]
>>
```

Copy an Array

We can make a copy of an array by using something like the slice method. It returns to you a copy of the array!

Run an experiment…

1. Make a new array, but assign it a copy of one of the arrays from the previous example.
2. Check the values of your three array variables.

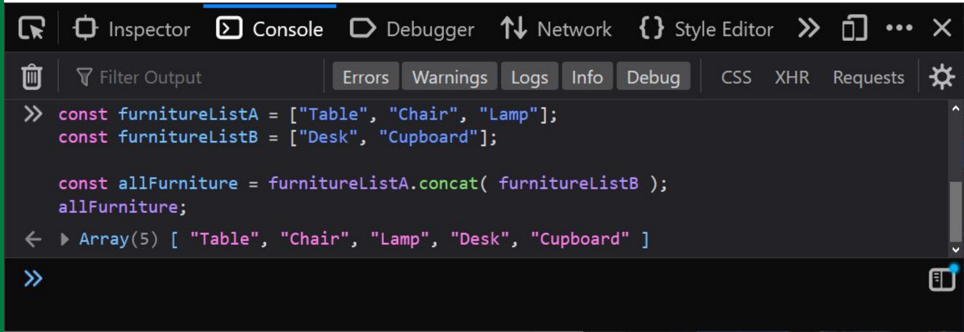


```
>> const arrayThree = arrayOne.slice();
<- undefined
>> arrayThree.push( "Bald Eagle" );
<- 4
>> arrayOne;
<- ▶ Array(3) [ "Giraffe", "Rooster", "Tardigrade" ]
>> arrayTwo;
<- ▶ Array(3) [ "Giraffe", "Rooster", "Tardigrade" ]
>> arrayThree;
<- ▶ Array(4) [ "Giraffe", "Rooster", "Tardigrade", "Bald Eagle" ]
>>
```

Combining Arrays

We can combine, or concatenate, arrays if we'd like the contents of multiple lists to appear in a single array.

To merge arrays we use the Array.concat method, passing in as arguments one or more (comma-separated) arrays to join together.



```
>> const furnitureListA = ["Table", "Chair", "Lamp"];
   const furnitureListB = ["Desk", "Cupboard"];

   const allFurniture = furnitureListA.concat( furnitureListB );
   allFurniture;
<- ▶ Array(5) [ "Table", "Chair", "Lamp", "Desk", "Cupboard" ]
>>
```

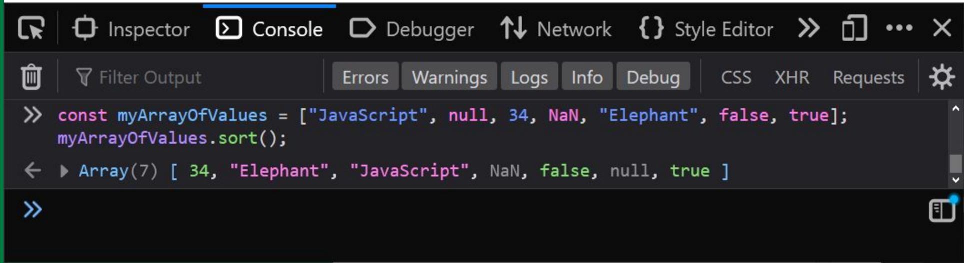
Sorting Arrays

If you need an array re-ordered, the sort method is just what the doctor ordered!

Using the Array.sort method you can re-order an array's contents. Note that this will not create a new copy of an array, but instead re-order the existing array.

By default it will convert each item in the list to a string and compare the string value with each other—re-ordering all contents in ascending order based on the characters available in each stringified item. Note that values that won't stringify nicely, may not order as you'd hope.

You can pass in a sorting function as an argument. We'll be covering functions later in this module.



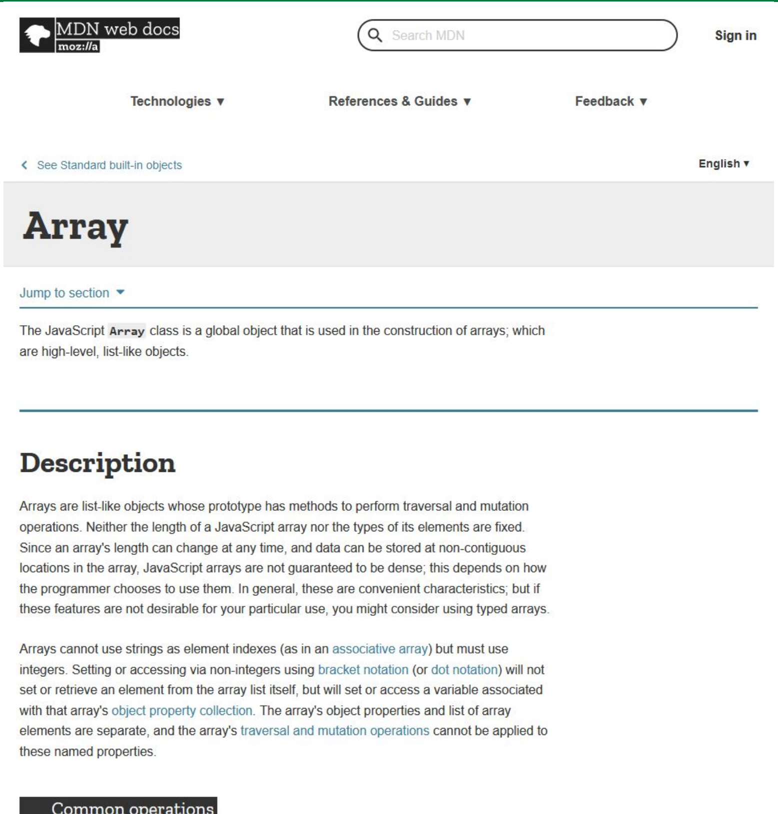
```
>> const myArrayOfValues = ["JavaScript", null, 34, NaN, "Elephant", false, true];
   myArrayOfValues.sort();
<- ▸ Array(7) [ 34, "Elephant", "JavaScript", NaN, false, null, true ]
>>
```

More on JavaScript Arrays

There's plenty more to learn about arrays in JavaScript...

Check out docs to see what else they're capable of!

W3Schools also offers some nice examples and coverage of the topic.



Array

The JavaScript `Array` class is a global object that is used in the construction of arrays; which are high-level, list-like objects.

Description

Arrays are list-like objects whose prototype has methods to perform traversal and mutation operations. Neither the length of a JavaScript array nor the types of its elements are fixed. Since an array's length can change at any time, and data can be stored at non-contiguous locations in the array, JavaScript arrays are not guaranteed to be dense; this depends on how the programmer chooses to use them. In general, these are convenient characteristics; but if these features are not desirable for your particular use, you might consider using typed arrays.

Arrays cannot use strings as element indexes (as in an associative array) but must use integers. Setting or accessing via non-integers using bracket notation (or dot notation) will not set or retrieve an element from the array list itself, but will set or access a variable associated with that array's object property collection. The array's object properties and list of array elements are separate, and the array's traversal and mutation operations cannot be applied to these named properties.

Common operations