Arrays Basics

Introduction

Variables are fine if you know how many primes you need at the outset

```
const firstPrime = 2;
const secondPrime = 3;
const thirdPrime = 5;
const fourthPrime = 7;
const fifthPrime = 11;
```

But what if you didn't know how many values you needed to store

Introduction

```
const primes = [2, 3, 5, 7, 11];
const names = ['Alice', 'Bob', 'Charlie'];
const booleans = [true, false, false, true];
const mixedTypes = [1, 'sweet', true, null, NaN, 'bye!'];
const woahhh = ['Whats up with this? -->', ['Woah', 'crazy!']];
const emptyArray = [];
```

To write an array in JavaScript, you use square brackets [] and comma separate each value in the array

Accessing and updating array values

To access an element in an array, we specify the name of the array followed by square brackets brackets and the position (also called the index) of the element we're trying to access.

```
const arr1 = [5, 3, 10];
console.log(arr1[0]); // should equal 5
console.log(arr1[1]); // should equal 3
console.log(arr1[2]); // should equal 10
console.log(arr1[3]); // should be undefined -- remember, arrays
                      //are zero-indexed!
console.log(arr1[1 + 1]); // the same as arr[2], which is 10
console.log(arr1[arr1.length - 1]); // shorthand for the last
                                    //element of an array, in
                                    //this case 10
```

Arrays are zero-indexed, which means that the first element is accessed at index 0

Accessing and updating array values

To update a value in an array, we can simply assign an element at a given index to a new value

```
const arr2 = [5, 3, 10];
arr2[0] = -1000;
arr2[2] = 'dope';
console.log(arr2); // should be [-1000, 3, 'dope']
```

Adding to arrays

There are a number of ways you can add elements to an array.

One way is by setting a value at a new index in the array.

```
const arr3 = [1, 2, 3];
arr3[3] = 4;
console.log(arr3); // [1,2,3,4]
```

Adding to arrays

```
const arr3 = [1, 2, 3];
arr3[3] = 4;
console.log(arr3); // [1,2,3,4]
```

Be careful with this approach

You can add an element at any index, and any elements that don't have values in them will be filled with undefined values.

```
const arr4 = [1, 2, 3];
arr4[5] = 'whoa';
console.log(arr4); // [1, 2, 3, undefined, undefined, 'woah']
```

Adding to arrays

If you want to add to the end of an array, a better approach is to use the **push** function

This function returns the new length (the number of elements) of the array.

```
const arr5 = [3, 2, 5];
arr5.push(7); //
console.log(arr5); // [3, 2, 5, 7]
```

Adding to arrays

if you want to add to the beginning of an array, you can use the **unshift** function.

As with push, unshift returns the length of the modified array

```
const arr6 = [1, 2, 3];
arr6.unshift(0);
console.log(arr6); // [0,1,2,3]
```

Removing from arrays

One (not common) way to remove elements is to manually set the length of the array to a number smaller than its current length

```
const arr7 = [1, 2, 3];
arr7.length = 2; // set the new length
console.log(arr7); // [1,2]
```

Removing from arrays

A more common way to remove elements from the back of an array is to use **pop**().

This function works in sort of the opposite way as push, by removing items one by one from the back of the array.

```
const arr8 = [1, 2, 3];
arr8.pop(); // returns 3
console.log(arr8); // [1,2]
```

Unlike push, however, pop doesn't return the length of the new array; instead, it returns the value that was just removed.

Removing from arrays

If you want to remove an element from the front of an array, you should **shift()** (like unshift, but the opposite)

As with pop(), shift() returns the removed value.

```
const arr9 = [1, 2, 3];
arr9.shift(); // returns 1
console.log(arr9); // [2,3]
```

Removing from arrays

To remove an element of an array at an index i:
array.splice(i, 1);

```
const arr10 = [5, 4, 3, 2];
arr10.splice(1,2); // Remove two element at position 1
console.log(arr10); // [5, 2]
```

Removing from arrays

delete replaces the value at the index where you delete with empty (undefined)

```
const arr10 = [5, 4, 3, 2];
delete arr10[1];
console.log(arr10); // [5, empty, 3, 2]
```

This usually isn't what you want, which is why you won't often see people use delete on array