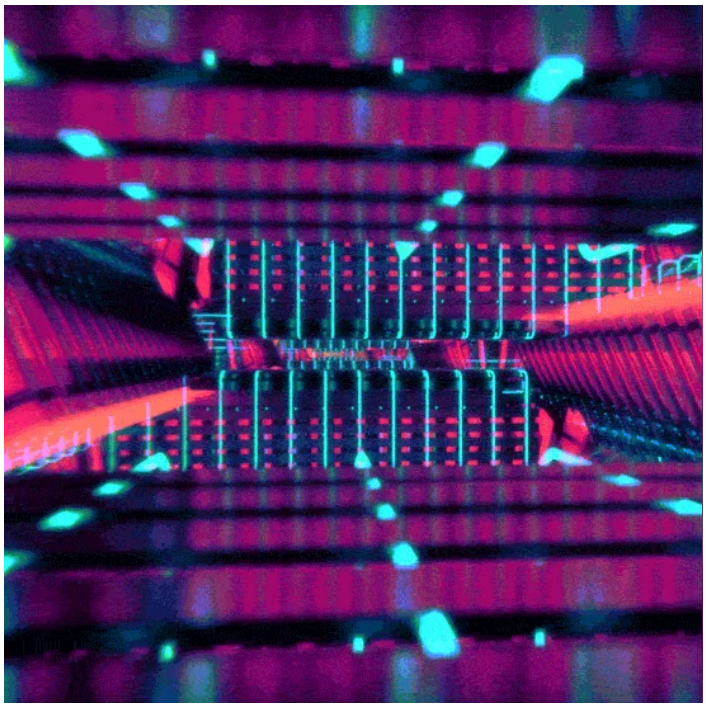
2 — Data Structures

Christian Grewell Follow
Feb 6 · 7 min read

Arrays, objects and the methods attached to them are so important that they deserve their own lab. Unfortunately, we don't have a lot of time, but don't worry, you'll become quite comfortable with some of these in the next few months as you build your applications.



source: https://media.giphy.com/media/dE0DoplZp2cSs/giphy.gif

What is an array?

JavaScript arrays are used to store multiple values in a single variable. An array can hold many values under a single variable name, and you can access the values by referring to an index number. Arrays are extremely useful, and you'll be working with them all the time. Some important properties of arrays:

- They store the *order* that items are added to them
- They can store a huge amount to items
- They can also store nested properties
- They can be very efficient, especially when paired with methods that operate on them.

Declaring Arrays

Open your browser's console and type the following:

```
> let cats = ['fluffy','bobby','whiskers']
> console.log(cats)
```

```
> let cats = ['fluffy','bobby','whiskers']
  console.log(cats)
  ▼(3) ["fluffy", "bobby", "whiskers"]
     0: "fluffy"
     1: "bobby"
     2: "whiskers"
     length: 3
```

an array of cats

What are we doing here?

Array Syntax

To declare an array in JavaScript, use the following syntax

```
let arrayName = [] //declares an empty array named arrayName
const arrayName = [] //declares an empty array named
arrayName
```

Of course declaring an empty array isn't normally the end in and of itself. Instead, you'll want to sometimes declare an array with things (values or even objects) inside of it. In the cat example, we created an array of `strings`. But it could have been an array of numbers too:

```
let luckyNumbers = [8, 88, 888, 8888, 8888] //an array of
numbers
console.log(luckyNumbers[1])
```

```
> let luckyNumbers = [8, 88, 888, 8888, 8888]
<· undefined
> console.log(luckyNumbers[1])
  88
<· undefined
>
```

What is this [1] thing and why is it 88 instead of 8?!

Binding an array

An important point to mention is that bindings to arrays can be declared with `let` or `const`. Even when they are declared with `const`, arrays are still mutable; they can be changed. However, a variable declared with `const` cannot be reassigned:

```
const cats = ['Bobby','Fredrick','Happy Tim']
dogs = [];
```

```
cats = dogs;
```

```
//Error:

TypeError: Assignment to constant variable.
```

Adding values to an array

To add values to the array we can supply initial elements in the brackets. In this example, we're adding three *strings*

```
let cats = ['Mark', 'Patches', 'Alvin'];
```

Array elements are numbered, starting with zero. We can get an element by its number in square brackets:

```
let cats = ['Mark', 'Patches', 'Alvin'];
```

```
console.log(cats[0]); // Mark
console.log(cats[1]); // Patches
console.log(cats[2]); // Alvin
```

You can replace an element by setting it equal to something else. You can also add a new item to the array

```
//replaces Patches with Peaches
let cats[2] = 'Peaches'
```

```
//adds a new cat
let cats[3] = 'Marmalade'
```

Array Properties

Arrays also have their own properties and methods (functions) you can call to get information back. Some important and useful ones

```
array.length
```

```
array.push()
```

```
array.pop()

array.shift()
```

```
array.unshift()
```

```
array.toString()
```

A full list is here, I recommend reading the definitions: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

. . .

```
Advanced Challenge: You are a volunteer at your local animal
shelter. Your job is to help as many animals get adopted as
possible, but sometimes you are tasked with admitting a new
shelter animal :(
```

```
Create three functions, addPet(name) to add a new shelter
pet to the list of pets, adoptPet(name) to remove the
shelter pet from the array of pets and currentPets() to
print out a list of all pets for adoption.

SOLUTION:
```

```
const pets = ['Bob Ross', 'Patches', 'Doofus'];
```

```
function addPet(name) {
     pets.push(name);
     console.log(pets);
}

function adoptPet(name) {
     let petPos = pets.indexOf(name);
     pets.splice(petPos,1);
     console.log(pets);
}
```

```
function currentPets() {
     pets.forEach(function(item, index, array) {
     console.log(index, item);
     });
}

addPet('Freddy Mercury');
adoptPet('Bob Ross');
currentPets();
```

```
//[ 'Bob Ross', 'Patches', 'Doofus', 'Freddy Mercury' ]
//[ 'Patches', 'Doofus', 'Freddy Mercury' ]
//0 'Patches'
//1 'Doofus'
//2 'Freddy Mercury'
```

Arrays can store objects!

So far, we've just been manipulating a one-dimensional arrays (a list). In fact, arrays can also store multi-dimensional lists (objects). We'll cover objects in the next lab,

For example, let's say we want to have a program for a vet that returns the cat's name, and whether it loves catnip. There are a few ways to do this, one using objects, the other a classic array:

```
const cats = ['Tabby', true, 'Bobby Brown', false,
'Peaches', true];
console.log(cats);
```

```
const cats = [
{name: 'Tabby',lovescatnip: true},
{name: 'Bob Ross',lovescatnip: false},
{name: 'Peaches',lovescatnip: true},
];
console.log(cats);
```

We've just created an object in the second example :)

Accessing arrays with bracket [ ] and dot . notation and properties

Now that we have an array of objects above, how might we access their *properties* so that the vet can figure out if a particular cat likes catnip?

We've seen a few suspicious-looking expressions like `myName.length` (to get the length of a string) and `Math.max` (the maximum function) in past. These are expressions that access a *property* of some value. In the first case, we access the `length` property of the value in a string called `myName` . In the second, we access the property named `max` in the `Math` object (which is a collection of math-related constants and functions). Almost all JavaScript values have properties.

Below is the bracket notation syntax for accessing an array index (remember they start from 0) and dot notation to access an array property (some are built-in, others we can declare on the object)

```
Array[index].propertyname
```

Here's an example of how we might access a property of our `cats` array at a index 0:

```
const cats = [
{name: 'Tabby',lovescatnip: true},
{name: 'Bob Ross',lovescatnip: false},
{name: 'Peaches',lovescatnip: true},
];

console.log(cats[0].name); //Tabby
console.log(cats[0].lovescatnip); //true
```

This is not a great solution for our vet…

. . .

```
CHALLENGE 2: go through the array methods here, can we find
anything that might allow the vet to pass in a cat name and
get back a true or false if they like catnip?)

Answer: https://developer.mozilla.org/en-
US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter

console.log(cats.filter(cat => cat.name === 'Bob Ross'));
```

```
//{name: 'Bob Ross',lovescatnip: false}


const cats = [
{name: 'Tabby',lovescatnip: true},
{name: 'Bob Ross',lovescatnip: false},
{name: 'Peaches',lovescatnip: true},
];
```

```
console.log(`today you have ${cats.length} cats to see! Here
they are:`);
console.log(cats);
console.log(`Here is Bob Ross cat's info:`);
console.log(cats.filter(cat => cat.name === 'Bob Ross'));
```

Array Summary

Below are some important things to remember about arrays.

- Arrays are lists and are a way to store values in JavaScript.
- Arrays are created with brackets `[]`.
- Each value in an array is in a numbered position, starting at `0`.
- You can access a single value in an array using its numbered position, with syntax like: `myArray[0]`.
- We can also change a value in an array using its numbered position, with syntax like `myArray[0] = "new string"`;
- Arrays have a `length` *property*, which allows you to see how many items are in an array.
- Arrays have their own methods, including `.push()` and `.pop()`, which add and remove items from an array, respectively.
- Arrays have many other methods that perform different functions, such as `.slice()` and `.shift()`. You can read the documentation for any array method and property on the Mozilla Developer Network website.
- Variables that contain arrays can be declared with `let` or `const`. Even when declared with `const`, arrays are still mutable; they can be changed. However, a variable declared with `const` cannot be reassigned.

Objects

You might have noticed that even with an ordered list of values (strings, numbers, etc…) and some methods and properties to add, change and delete them, we haven't yet *programmed* something truly useful, especially something that might help us later when we are doing our projects. Let's take the following scenario:

Last week, you spent time in a coffee shop observing behaviors with the goal of helping the business understand more about it's operations, products and customers. You wrote these observations down.

You *could have* made those observations into an array:

```
const journal =
['Male',2,'Americano',5,'Cash','Female',1,'Frap','2,'Mobile'
]
```

Can you see a problem with this? No, it's not the mixed-and-matched values, that's perfectly fine in JavaScript arrays (though not always a good practice).

- It's impossible for us to know what each of those values corresponds to, you can guess the keys for some but not all(male = gender, 2 = ?, Americano = order, 5 = ?, Cash = payment type…etc)
- Iterating through this would be a bad dream, but adding or removing individual values would be a nightmare (you'd shift the entire array)

For situations like this, we have ***Objects***, which are arbitrary collections of properties nicely grouped together. We worked with objects earlier in the vet example.

```
IMPORTANT: Objects are at the very heart of JavaScript - if
you can understand what an object is, you're going to go
places.
```

Declaring an Object

One way to create an object is by using braces as an expression.

```
let observation = {
  gender: 'Male',
  paymentType: 'Cash',
  drinks: ['Americano', 'Frap', 'Iced Coffee'],
  isTalkative: false
};
```

```
console.log(observation.gender);
// Male
console.log(observation.drinks);
// [Americano, Frap, Iced Coffee]

observation.gender = 'Female';
console.log(observation.gender);
// Female
```

In the example above, we have a `customer` object, with the following properties:

- gender
- paymentType
- drinks
- isTalkative

Let's say we are sitting down at the coffee shop, and would like a way to rapidly catalog our observations and store those for later analysis. What approach could we take?

. . .

```
CHALLENGE 3: You would like to create a program to help you
rapidly add observations on each customer that visits the
coffee shop.
```

```
Hint: this is the perfect job for a function, but recall our
object delcaration notation too.

const observations = [];
```

```
function
addObservation(gender,paymentType,drinks,isTalkative) {

observations.push({gender,paymentType,drinks,isTalkative});
}
```

```
addObservation("Male","Cash","Cappuccino",false);
console.log(observations);

//
```

```
[ { gender: 'Male',
    paymentType: 'Cash',
    drinks: 'Cappuccino',
    isTalkative: false } ]
```

There is something cool in this function.

Instead of declaring properties like `drinks: drinks`, we just have to give a property name as an argument to the function. This is shorthand that means the same thing—if a property name in brace notation isn't followed by a value, its value is taken from the binding with the same name.

Now we have a function that we can use to add observations in a structured way.