Arrays

Learning Objectives

- Explain what arrays are
- Create arrays
- Access an element in an array
- Add items to an array
- Call methods on arrays

Intro

We've been dealing with single concepts so far - a person, a number, a state of one thing. But just as with real life, we don't always have one of something - we could have a group of things. At home we don't just have one piece of food, we have many. We don't have one pair of shoes, we have many. And we definitely have more than one recipe, and each recipe has more than one ingredient. So how do we represent this in our code?

This is where arrays come in. An array is part of a subset of objects known as "collections". It is essentially a storage container like a big basket, or bag, or box. In real life we might be storing fruit, shoes or sweets but in our code we can store any object we like of any of the datatypes we have seen - `number` s, `strings` s, even `booleans` s.

Each item in an array is indexed - assigned an incrementing integer from 0 for the first item. Using this number we can retrieve the item located at that index number.

> Note: it is useful to retrieve values using the index, but the power of arrays will come when we learn about loops!

Arrays are usually denoted via square brackets: `[]`

And their contents are always comma separated: `["apple", "orange", "lemon"]`

Just like any other datatype, we can store an array in a variable:

```
const ingredients = ["chicken", "butter", "salt", "pepper"]
```

What are arrays for

Arrays are great for storing similarly shaped data together:

```
const favouriteNumbers = [1, 9, 12, 33]
```

> Try to avoid storing different datatypes - an array's name should indicate what is being stored in it, and if I assume that `favouriteNumber` will have numbers in it, but some cheeky dev hid a boolean or a string in their, that can cause a lot of unexpected bugs in my code!

Arrays are also great when you want to sort items based on something. Arrays are inherently ordered, since its a collection where values are indexed with incrementing integers (`0, 1, 2, 3...`)

A very common use case is that if we have want to do something to multiple items (like storing multiple ingredients in a database, or display multiple steps for a recipe, or display our images on a social media site) we can use arrays. Since they can store large amount of items in them, they are great for passing around and working with large datasets.

Accessing arrays

All "keys" to arrays are integers, we call this the "index". The first element in an array is at index zero, and the amount of elements can generally go up as high as your computer/programming language will allow. Some languages require that you specify how big is each new list you use (to know how much space to allocate in memory), but JavaScript is way more flexible, and will grow these collections automatically to fit as many items as you add to it.

So what does this look like?

First, let's create a day 2 directory, with a new file in it called `arrays.js`

In that file, let's create our first list!

```
const ingredients = ["eggs", "salt", "pepper", "butter"]
console.log(ingredients)
```

Arrays are characterised by the square brackets around the elements.

Array indexes in most programming languages start at 0. Why? More info here for the curious.

```
console.log(ingredients[0])
console.log(ingredients[2])
```

In JavaScript, if you try to access an element by its index that doesn't exist, the value returned will be undefined

```
console.log(ingredients[9])
```

Updating an item

We can update an item in an array by accessing an element at a particular index and giving it a new value, just like re-assigning a variable.

```
ingredients[0] = "tofu"
console.log(ingredients[0])
```

Creating an empty array

There will be times where you want to initialize an empty array. While there are many ways to achieve this, the most common one is simply to write `const myArray = []`

> Note that just because you created a list with `const`, does not mean you cannot modify the values *inside* the array. You are not reassigning the value of the variable, you are simply modifying its contents. This is only true for objects or arrays though!

Getting the number of items in an array

Every JS array comes with a property called `length`. Note that it is not a method - it does not need brackets. `python console.log(ingredients.length)`

Adding and removing items

Most of the time we do not want to care about the order of an array - especially not right before we want to modify it, since if we want to, we can just reorder the elements after adding a new one. Because of this, it's very common when adding a new element to simply use the `.push(item)` method. This places the new element at the last index position on the array. You also do not need to keep track of the number of items, because when you are coding, there is no way knowing of the actual number of items! The user might have already uploaded more photos, recipes, or there might already be more comments under a LinkedIn post since last checking - we have to assume that the number of items is unknown and flexible.

```
ingredients.push("cheese")
console.log(ingredients)
```

You can also remove the last item with the `pop()` method. This not only removes the last item from the array, but also returns it so we have a chance to store it in a variable

```
const removedIngredient = ingredients.pop()
console.log(ingredients)
console.log(removedIngredient)
```

There are also more methods to add/remove items from the beginning of the array, or to add/remove somewhere in the middle - we will not look at these, but you can read more here: [MDN Docs - Arrays]

Conclusion

We have seen how we can store collections of values using arrays and how to access said values.

An important note: Using arrays like this would be a huge hassle, since recipes are more complex than just a list of strings, or numbers. If I want to display many comments, there is more to it than just the text - who is the author, when was it created, what post doest it belong to, etc.

The complexity of the data needed to be used to display items is way higher than what we encountered - so how can we store entities with more details?

Also, how can we automate the whole process? If I do not know how many items will be present, how do I know how many items to access and interact with?

The next two lessons will give you all the answers you are looking for! For data complexity, `object`s will come to the rescue, and for automation, `loops` will give us a hand.