Arrays

Summary

This summary are the dot points that you'd need to have a go at the array challenges. There is more to say about arrays of course, but it's good to also get into using them and playing around with code. Then we can layer the deeper understanding over this baseline.

There is some mention of loops that can be ignored for now, and you can return to these sections after we learn loops.

- We have been thinking of variables as named storage spaces for data.
- Arrays are like having a whole warehouse of storage, or think of them as a series of storage buckets.
- They come pre-indexed, starting at 0, and then 1, then 2, and so on.
- We can put just a few things in, or many many things.
- We can create an array like this:

```
const myPets = ["Garfield", "Odie", "Snoopy", "Woodstock", "Bugs",
"Winnie"]`
```

- We can access the contents of the arrays we create by referencing the name of the array, and then the index, eg, `myPets[2]` would access the fourth element of the array that we called `myPets`, and that would be the string `"Snoopy"`.
- Once we have made our array, we have various tools at our disposal.
- We showed that we can search through our array using a `while` loop (and later we will show even simpler ways to loop through our array).
- We didn't discuss these much, but we can pop the last element off the end of the array: `let lastElement = myPets.pop()`.
- We can also add an element to the array: `myPets.push("Gromit")`
- Run the following code to see these in action:

```
const myPets = ["Garfield", "Odie", "Snoopy", "Woodstock", "Bugs",
"Winnie"]
pet = pets.pop()
console.log(pet)
pets.push("Snoopy")
console.log(pets)
```

- I would recommend having a look at the Mozilla docs on JS docs (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) for some more interesting functions that apply to arrays.

Some more detail

The following comes from a larger document that I wrote about datastructures in Ruby, and have adapted for JS here. It mentions loops on occasion, but we will be covering those next. To this point students had learned string and integers and similar concepts. We had been thinking of these as datatypes. Now we are looking to move on to some containers for this data.

The key points are above, and the next part is just there if you are interested.

Data Structures

We have been dealing with concepts like strings and numbers, and even undefined, and these are called datatypes in JS. Now we are going to look at ways to combine these types, and for now I will be calling these new containers 'data structures' - but the distinction is not always so clear.

These are really just constructs in coding - things made up by the programmers that came before you - to make shifting and mutating and combining those basic elements we have been discussing much easier. They are a means to an end. They are tools. In some ways can be thought of as storage facilities.

Arrays are the structures that were made to store things that are *similar* (we will discuss what this means in more detail below). They come built into JS. They are particularly helpful for storing data because they come indexed - someone did all this work for you, and you get to take advantage of it. By convention the first index is 0, and the index goes up by 1 for every element added to the array.

```
["cat", "dog", "fish"]
```

- This is an array with three elements.
- They are all strings.
- At the `0` index is the string `"cat"`. At index `1` the string `"dog"`, and at `2` is `"fish"`.

Arrays are useful because they come ready made with indexes, and because we tend to 'know' (or assume) that they will contain the same type of data. Although we could fill them with different types of things, we try not to do this because it ruins their benefits. Nothing actually stops us from putting any mix of anything we like into an array. We could put in strings and floats and booleans into one array, but instantly we have lost the power of the structure - a structure that is there merely to help us. We aren't obliged to use arrays - they are there for our benefit when we feel they could help us achieve a goal. We should attempt to program them to hold the same type of data so that when we loop through them they will behave as expected.

This is the key point I'm trying to get across here: these are tools, and as such they only express their benefits when used thoughtfully.

For example, let's say we have an array filled with numbers that we want to add together and find the total, but for some reason a string sneaks in. Everything will go well until we try to add a string to our running total, at which point our code breaks (usually - or we get unwanted behaviour). This is a problem for us, but it is of our own making. No point blaming the array or the `.each` loop.

To labour the point slightly, let's say that we have some data about a tennis player. Think about the problems with putting all that data into one array. It might look like this:

```
["Roger", "Federer", 20, 36, 8, 4]
```

The biggest problem here is that we have no way of knowing what the data is that we are extracting. `4` refers to his number of kids. Or at least that's what I had in mind. `20` refers to total grand slam titles (currently) - but how would anyone know this? And what is the point of indexing this data set in this way? How would this be used in a loop? We now have many problems (and we will return to these later).

That arrays contain the same datatype is really just the minimum requirement. Let's say you have a list of students, but are storing both the first names and last names in the array. Maybe it looks like this:

```
names = ["Roger", "Federer", "Rafael", "Nadal", ..]
```

You can see how this might be less useful than having:

```
firstNames = ["Roger", "Rafael", ..]
secondNames = ["Federer", "Nadal", ..]
```

Here you can see that it will be more readily useful to us in this format. In the first example, no matter which index we hit we are getting back a first name. And in the second we will always encounter a second name. And this is most of what coding is. Pushing and poking at data and trying to get a result. You do the work to put the data into an array so that you can perform useful tasks with ease. Sometimes getting the data in the array is the hard part, but that's part of the challenge of coding. Sometimes you have a perfect array of data, and have to think very hard about how to extract what you need.

We have a structure that we can use for heterogenous data, and in JS this is an 'object'. We will discuss this data structure soon.