Objects

Introduction

Objects are a way of storing heterogenous data. They give us the power to group together pieces of data that may be of different types, but that all fall under the one category.

Let's get one going first, and then discuss the details:

```
// We looked at this array:
rhyseArray = ["Rhyse", "Martin", 27, 183, 100.0, "Leeds Rhinos"]

// Let's put it in a more useful format:
rhyseObject = {
    firstName: "Rhyse",
    lastName: "Martin",
    age: 27,
    heightCm: 183,
    weightKg: 100,
    team: "Leeds Rhinos"
}

// Accessing a value:
console.log(rhysObject.lastName)

// Changing a value (change the age to 28):
rhyseObject.age = 28
// `rhyseObject.age += 1` also works

// We could push his height into the rugbyHeights array (assuming it's
been made previously):
rugbyHeights = [172, 199, 168]
rugbyHeights.push(rhyseObject.heightCm)
```

The comments explain things mostly. What I would recommend is not to get too wound up with everthing here, and trying to understand all the theory or whatever. Just make a few objects, access a few values, change a few values, and tinker around with them. You'll learn a lot more by making and breaking code, and it will start to all become clear. Then you'll have more questions, and can ask about the bits that you are having trouble with. Tinkering with code will be the fastest way to learn anything that we do.

Summary

- We talked about arrays, and how they are very useful for storing data that is similar. Arrays are simply a tool that we can use, and we need to understand why we do so. And as our code gets more complicated, we need to make use of these structures.
- We learned a new one, *objects*. These are used to store data that falls under the same topic, or that should be stored in the same place, but has datatypes that are dissimilar.
- Objects consists of key and value pairs. You access the value via the key of the object.
- Here are some examples:

```
console.log("objects.js")

// NOT a very useful structure
let freddyArray = [
    "Freddy Mercury",
    "Queen",
    15,
    false
]
// Better..
let freddy = {
    name: "Freddy Mercury",
    band: "Queen",
    numOfAlbums: 15,
    living: false,
}

let johnLennon = {
    name: "John Lennon",
    band: "The Beatles",
    numOfAlbums: 7,
    living: false
}

console.log(freddy)

console.log(freddy.band)
console.log(freddy.numOfAlbums)

console.log(johnLennon)

console.log(johnLennon.living)
console.log(johnLennon.name)
```

Building more complex structures

Now we have some really powerful constructs under our control: We have *control flow* (`if`/`else`; *loops*) to guide which parts of the program run; we have *arrays* to group similar elements; we now have *objects* to group heterogenous data; and we already learned *functions* and can make our code clearer by assembling code that performs a certain function, and making it reusable.

And these can be combined and recombined to make more and more complex programs. We can put arrays in arrays, and objects in objects, and arrays in objects, and so on.

```
// The object from before
let freddy = {
    name: "Freddy Mercury",
    band: "Queen",
    numOfAlbums: 15,
    living: false,
```

```
}

let johnLennon = {
    name: "John Lennon",
    band: "The Beatles",
    numOfAlbums: 7,
    living: false
}

let eddieV = {
    name: "Eddie Vedder",
    band: "Pearl Jam",
    numOfAlbums: 10,
    living: true
}

// Put our rock star objects into an array
let rockStars = [freddy, johnLennon, eddieV]

// Define a function to find all the living rockstars
const findLiving = function(array) {
    //set up our counter and array to put the selection in
    finalArray = []
    counter = 0
    // loop through the array of rockstars
    while (counter < array.length) {
        // In our rockstar objects, `living` is a boolean
        if (array[counter].living) {
            // Push into the final array
            finalArray.push(array[counter])
        }
        // Increment
        counter += 1
    }
    // Return the finalArray
    return finalArray
}
// Run our `findLiving` function with our `rockStars` array as the
argument, and store the return value (the result of the function) in a
variable.
const livingArray = findLiving(rockStars)
console.log(livingArray)
console.log(livingArray[0].name)
```

Hopefully you can get a sense of the possibilities we have now from the code above.

Excerpt from data structures essay

This is a bit more, and some of it is more of the same. Just thought I'd include it here for anyone who wants to read a bit more about these structures.

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.

Remember earlier when we had the data referring to Federer, but it looked silly in an array? `["Roger", "Federer", 20, 36, 8, 4]` was the array. We can use another datatype called a hash to make better sense of this. A hash is a data structure that helps us to store the information relating to one entity (in this case a tennis player), but where we have several datatypes that make up that entity.

In this case the data we are trying to represent could be first and last names, number of glad slams, number of kids, and age. Here, some of these are best represented as integers rather than strings. Now, you could potentially argue that we could just make them strings, and then we'd have the same datatype. But would this be much of an improvement?

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

If someone doesn't know that "36" is his age, then what does that number mean to them? How is this useful?

The *object* datatype (or data-structure) helps us out here because we have keys that can denote and describe the data we are looking at, and these keys have a value. So for the data above we might have something like:

```
federer = {
    firstName: "Roger",
    secondName: "Federer",
    grandSlamTitles: 20,
    age: 36,
    children: 4
}
```

You can see that here we have contained in one structure all the things that we need to cover, but the information about what these values refer to is not lost. Now we can access elements of this variable federer by referencing the key.

`federer.age` gives us back the integer 36, for example.

You can think of all that we have seen so far as falling into two levels. The elementary datatypes are the building blocks of everything else that happens. They are *string*, *number*(*float*, *integer*), *boolean*, *undefined*, and *null*. These are the basic structures: the elements that we can use to build our world.

And then we also have two useful structures that are inbuilt into JS to help us combine these lower level elements: *arrays* and *objects*.

In the same way as we are all made from the same elements, just in different combinations, and all code ever is is those basic datatypes being pushed around all over the place in larger structures.