Objects

Learning Objectives

- Understand what a JavaScript Object is
- Understand the advantages of a objects
- Be able to create an object
- Be able to retrieve items from an object
- Be able to add items to an object
- Be able to modify an item in an object

What Are Objects

We've seen how you can store (ideally) identical values in a list-like collection called arrays.

Arrays are fantastic for storing (and later, interacting with) data, because there could be a flexible amount of values in them.

But if I want to store something more complex than just names, our current options are lacking, and will be a bottleneck if we want to increase complexity.

If a user or employee has a lot of data about them, a bunch of variables, unstructured, could cause confusion. Consider the following:

```
const firstName = "John"
const lastName = "Doe"
const jobTitle = "Project Manager"
const salary = 60000
const currentProjects = ["FacebookForCats", "AmazonButBetter", "NotFlix", "Instant
Gran"]
```

This is all well and good, but what happens if I need another employee? Or a 100? Do I just keep adding numbers after variables and hope I can keep track of each number belonging to a single user?

This is where another data structure comes in - in JavaScript, we refer to them as objects. If you have experience with other languages, you may have heard about them as hashmaps, maps, or dictionaries, but the idea is the same.

Objects are essentially key-value pairs, similarly to indexed values in an array, but objects must have all their keys stored as strings.

Consider the previous employee:

```
const employee = {
    firstName: "John",
    lastName: "Doe",
    jobTitle: "Project Manager",
    salary: 60000,
    currentProjects: ["FacebookForCats", "AmazonButBetter", "NotFlix", "InstantGra
n"]
}
```

Each key must be unique, and their corresponding value can be of any datatype - booleans, strings, numbers, null, even an array or another object!

> Note: Keys are often referred to as properties in JavaScript - since most objects describe an actual object, like an employee, a person, a recipe or a post, these things have properties.

Using objects

Creating Dictionaries

New object in JS usually are created using the `{}` (curly brackets/braces), and they behave like any other data type - we can store them in variables. Values are comma separated, similarly to arrays.

Important to remember that you should name your properties like you name your variables - they should describe the intent and datatype. For example, salary should not be the string "a lot", and if something says `dateOfBirth`, it better not be an array!

Accessing Elements

We can access elements in a similar manner to arrays, using the square brackets - the main difference is, we have to use the key, instead of an index.

```
console.log(employee["firstName"])
```

If we try to access an element for which there is no key we get an `undefined`.

```
console.log(employee["dateOfBirth"])
```

A more common way to access values from object, however, is using the `.` notation:

```
console.log(employee.salary)
```

Be mindful, it only works if you use strings without special characters (no dots, dashes, commas, colons and the like. Just plain, simple text with characters). Going forward we will mostly use this syntax.

Since, like arrays, we usually do not have the contents of the object visible to us at all times, we could check if a key exists in an object. We can do that with the following:

```
console.log(employee.hasOwnProperty("firstName")) // true
console.log(employee.hasOwnProperty("height")) // false
```

Modifying Elements

Adding values to an object is similar to adding assigning values to a variable:

```
employee.department = "engineering"
console.log(employee)
```

We can also replace values

```
employee.firstName = "Jane"
console.log(employee)
```

We can remove items using the `delete` operator (mind the weird syntax, no brackets!)

```
delete employee.salary
console.log(employee)
```

Helpful Methods

An object has lots of helpful methods, including ways to list all the keys, or values, as arrays:

```
console.log(employee.keys())
console.log(employee.arrays())
```

Nested objects

The most important (and possbliy hardest) part of todays lesson is understanding the possible complexity of nested objects.

Consider the following:

```
const employee = {
    firstName: "John",
    lastName: "Doe",
    jobTitle: "Project Manager",
    salary: 60000,
    currentProjects: ["FacebookForCats", "AmazonButBetter", "NotFlix", "InstantGra
n"],
    address: {
        postCode: "EH6 4UH",
        city: "Edinburgh",
        street: "Newton Crescent",
        number: 15
    }
}
```

One of our properties now is actually another object. We could nest this as much as we would like, but for today, this is enough complexity for sure.

How do we access the city our beloved Joe lives in?

Consider that each property is its own value. For example if I'd like to upcase all letters in Joe's name, I could do the following:

```
console.log(employee.firstName.toUpperCase())
```

While `employee` is an object, `employee.firstName` is a string - keep that in mind. Always take your time when working with arrays or objects. This is one of the hardest parts to get the hang of in the beginning, but with patience and understanding, this becomes second nature in a couple of weeks!

So if I'd like to get back the address of Joe, I could do this:

```
console.log(employee.address)
```

And what is the type of data I got back? That's right, another object. Therefore I can chain this together like so:

```
console.log(employee.address.city)
```

We could always break it down to mutiple variables as we go along:

```
const addressObject = employee.address;
console.log(addressObject.city)
```

But in the long run, we are better off understanding the chaining, and keeping track of the data as it evolves!