How ImmutableJS works

In this lesson, we'll take a quick look at the internals of ImmutableJS so that we have a better understanding of how can we apply Immutable.js to our Redux app in a performant manner.

Immutable data structures can't be changed. So when we convert a regular JavaScript object with `fromJS` what ImmutableJS does is loop over every single property and value in the object (including nested objects and arrays) and transfers it to a new, immutable one. (the same thing applies in the other direction for `toJS`)

The problem with standard JavaScript objects is that they have `reference equality`. That means even when two objects have the same content, they're not the same:

```
var object1 = {
        twitter: '@mxstbr'
};

var object2 = {
        twitter: '@mxstbr'
};

console.log(object1 === object2); // -> false
```

In the above example, even though `object1` and `object2` have the exact same contents, they aren't the exact same object and thus aren't equal. To properly check if two variables contain the same thing in JavaScript we'd have to loop over every property and value in those variables (including nested things) and check it against the other object.

That's very, very slow.

Since immutable objects can't ever be changed again, ImmutableJS can *`compute a hash based on the contents of the object`* and store that in a private field. Since this hash is based on the contents, when Immutable then compares two objects it only has to compare two hashes, i.e. two strings!

That’s a lot faster than looping over every property and value and comparing those!

```
import { fromJS } from 'immutable';

var object1 = fromJS({
        twitter: '@mxstbr'
});

var object2 = fromJS({
        twitter: '@mxstbr'
});

console.log(object1.equals(object2)); // -> true 🎉
```

That’s nice and all, but how is this helpful in our app?