New Object Functionality

Objects have been upgraded with ES2015. We'll go over short-hand property assignment and short-hand method naming. We'll go over getters and setters and see how they allow us to implement cleaner interfaces for working with data.

Properties

If we're adding a variable as a property to an object, we'd do it like this.

```
var name = 'Sophia';
var age = 25;

var person = {
    name: name,
    age: age
};
```

We can now omit the colon and the variable name repeat, as long as the variable name is the same as the property name we want on the object. In other words, the block above is the same as this one.

```
const name = 'Sophia';
const age = 25;

const person = {
    name,
    age
};
```

It's just a shorthand to let you write a little less code.

Methods

Adding a method to an object is a little easier as well. We can drop the colon and the word `function`. The following two objects are equivalent.

```
const name = 'Christina';
const age = 23;
```

```
const obj1 = {
    name,
    age,
    growOlder: function() {
        this.age++;
    }
}

const obj2 = {
    name,
    age,
    growOlder() {
        this.age++;
    }
}
```

Setters & Getters

We now have access to special methods called setters & getters.

Getters

A getter is a method on an object created by placing the keyword `get` before the function. To invoke the method and get its return value, the method is called without the `()`, as if it were a standard static property.

```
const firstName = 'Brooke';
const lastName = 'Smith';

const person = {
    firstName,
    lastName,
    get fullName() {
        return this.firstName + ' ' + this.lastName;
    }
}

console.log(person.fullName); // -> Brooke Smith
```

Setters

Setters are special methods that take in a single argument. They are created by writing a method on an object using the keyword `set`. They are invoked through a normal `=` assignment.

```
const firstName = 'Brooke';
```

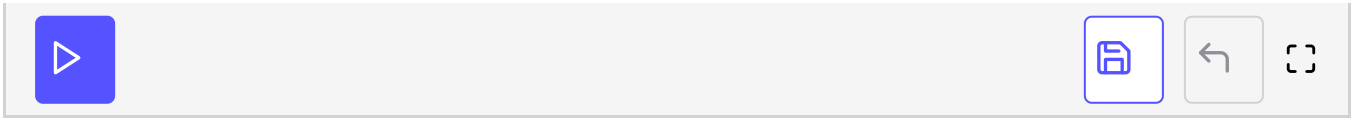
```
const lastName = 'Smith';

const person = {
    firstName,
    lastName,

    get fullName() {
        return this.firstName + ' ' + this.lastName;
    },

    set fullName(name) {
        const spaceIndex = name.indexOf(' ');
        this.firstName = name.slice(0, spaceIndex);
        this.lastName = name.slice(spaceIndex + 1);
    }
}

console.log(person.fullName); // -> Brooke Smith
person.fullName = 'Alex Johnson';
console.log(person.firstName); // -> Alex
console.log(person.lastName); // -> Johnson
console.log(person.fullName); // -> Alex Johnson
```



If an object has a setter method on it and we attempt to assign to the object using the name of the setter, as in `person.fullName = 'Alex Johnson'`, it triggers the setter. The setter must take in one parameter and is called with whatever is after the equals sign as its parameter.

Together, getters and setters create a powerful new way of dealing with objects. Someone using the object can simply attempt to use a property or assign to the property and internally, the object can be doing a whole lot more.

That's it.

Exercises

```
// Change this getter method such that
// when we attempt to access the property
// 'val', it multiplies by 2 every time.

// Ex:
/*
console.log(obj.val); -> 2
console.log(obj.val); -> 4
console.log(obj.val); -> 8
console.log(obj.val); -> 16
*/
```

```
const obj = {
    _val: 1,
    get val() {
        // Your code here
    }
};
```