Object and Strings

Level 3 – Section 1

Removing Repetition From Creating Objects

The *buildUser* function **returns an object** with the *first*, *last*, and *fullName* properties.

```
function buildUser(first, last){
  let fullName = first + " " + last;

  return { first: first, last: last, fullName: fullName };
}
```

Returning objects with keys and variables with the same name looks repetitive

Calling the *buildUser* function:

```
let user = buildUser("Sam", "Williams");

console.log( user.first );
console.log( user.last );
console.log( user.fullName );
```

```
> Sam
> Williams
> Sam Williams
```

The Object Initializer Shorthand

ES2015

We can remove **duplicate** variable names from object properties when those properties have the **same name** as the variables being assigned to them.

```
function buildUser(first, last){
  let fullName = first + " " + last;

  return { first, last, fullName };
}
```

Way cleaner!

Yields the same result:

```
let user = buildUser("Sam", "Williams");

console.log( user.first );
console.log( user.last );
console.log( user.fullName );
```

```
> Sam
> Williams
> Sam Williams
```

Assigning With Object Initializer Shorthand

The object initializer shorthand works **anywhere** a new object is returned, not just from functions.

```
let name = "Sam";
let age = 45;
let friends = ["Brook","Tyler"];

let user = { name, age, friends };

console.log( user.name );
console.log( user.age );
console.log( user.friends );
```

```
> Sam
> 45
> ["Brook", "Tyler"]
```

Same thing

```
let user = { name: name, age: age, friends: friends };
```

Object Destructuring

ES2015

We can use shorthand to assign **properties** from objects to **local variables** with the **same name**.

```
let user = buildUser("Sam", "Williams");

let first = user.first;
let last  = user.last;
let fullName = user.fullName;
```

Unnecessary repetition

This function returns `{ first, last, fullName }`

Same names as properties from return object

```
let { first, last, fullName } = buildUser("Sam", "Williams");

console.log( first );
console.log( last );
console.log( fullName );
```

```
> Sam
> Williams
> Sam Williams
```

Destructuring Selected Elements

Not **all** properties have to be destructured all the time. We can **explicitly select** the ones we want.

```
let { fullName } = buildUser("Sam", "Williams");

console.log( fullName );
```

```
> Sam Williams
```

Only grabbing fullName from the return object

```
let { last, fullName } = buildUser("Sam", "Williams");

console.log( last );
console.log( fullName );
```

```
> Williams
> Sam Williams
```

Grabbing last and fullName from the return object

Recap Object Initializer vs. Destructuring

Object Initializer Shorthand Syntax

```
let name = "Sam";
let age = 45;

let user = { name, age };

console.log( user.name );
console.log( user.age );
```

From variables to object properties

Object Destructuring

From object properties to variables

```
let { first, last, fullName } = buildUser("Sam", "Williams");

console.log( first );
console.log( last );
console.log( fullName );
```

Returns `{ first, last, fullName }`

Adding a Function to an Object

In previous versions of JavaScript, adding a function to an object required specifying the **property name** and then the **full function definition** (including the *function* keyword).

```
function buildUser(first, last, postCount){

  let fullName = first + " " + last;
  const ACTIVE_POST_COUNT = 10;

  return {
    first,
    last,
    fullName,
    isActive: function(){
      return postCount >= ACTIVE_POST_COUNT;
    }
  }
}
```

An anonymous function is assigned to an object property

Using the Method Initializer Shorthand

A new shorthand notation is available for adding a method to an object where the keyword *function* is no longer necessary.

```
function buildUser(first, last, postCount){

  let fullName = first + " " + last;
  const ACTIVE_POST_COUNT = 10;

  return {
    first,
    last,
    fullName,
    isActive(){
      return postCount >= ACTIVE_POST_COUNT;
    }
  }
}
```

Less characters and easier to read!

Template Strings

Template strings are **string literals** allowing embedded expressions. This allows for a much better way to do **string interpolation.**

```
function buildUser(first, last, postCount){

  let fullName = first + " " + last;
  const ACTIVE_POST_COUNT = 10;
  //...

}
```

```
function buildUser(first, last, postCount){

  let fullName = `${first} ${last}`;
  const ACTIVE_POST_COUNT = 10;
  //...

}
```

Enclosed by back-ticks, NOT single quotes, and code is wrapped inside dollar sign and curly braces

Writing Multi-line Strings

Template strings offer a new — and much better — way to write **multi-line strings**.

```
let userName = "Sam";
let admin = { fullName: "Alex Williams" };

let veryLongText = `Hi ${userName},

this is a very
very

veeeery
long text.

Regards,
  ${admin.fullName}
`;

console.log( veryLongText );
```

Newline characters are part of the template string

Newline characters are preserved

```
> Hi Sam,

  this is a very
  very

  veeeery
  long text.

  Regards,
    Alex Williams
```