Destructuring Examples

destructuring, associativity and ES6 notation

Consider the following examples:

```
let user = {
    name        : 'Ashley',
    email       : 'ashley@ilovees2015.net',
    lessonsSeen : [ 2, 5, 6, 7, 9 ],
    nextLesson  : 10
};

let { email, nextLesson } = user;
console.log(user);
// email becomes 'ashley@ilovees2015.net'
// nextLesson becomes 10
```

In a destructuring expression `L = R`, we take the right value `R`, and break it down so that the new variables in `L` can be assigned a value. In the above code, we used the object property shorthand notation.

```
let { email, nextLesson } = user;
```

Without this shorthand notation, our code will look like this:

```
let {
    email: email,
    nextLesson: nextLesson
} = user;

console.log(user)
```

In this case, the above code is equivalent with the following ES5 assignments:

```
let email = user.email;
let nextLesson = user.nextLesson;
```

Note that the above two lines are executed in parallel. First, the `R` value is fully evaluated before assigning it to left values. For instance, let's increment the variables `a` and `b` using destructuring:

```
let [a, b] = [5, 3];
[a, b] = [a + 1, b + 1];
console.log( a, b );
```

If we wanted to transform this destructuring assignment to ES5, we would write the following code:

```
let [a, b] = [5, 3];
[a, b] = [a + 1, b + 1];
var temp_a = a + 1;
var temp_b = b + 1;
a = temp_a;
b = temp_b;
console.log(a, b)
```

The value of a destructuring assignment of the form `L = R` is `R`:

```
console.log({email, nextLesson} = user);
```

As a consequence, don't expect destructuring to be used as an alternative for filtering values.

Destructuring is *right-associative*, i.e., it is evaluated from right to left. `L = M = R` becomes `L = R`, which in turn becomes `R` after evaluation. The side effect is that in `M` and `L`, variable assignments may take place on any depth.

```
let user2 = {email, nextLesson} = user;
console.log( user2 === user, user2.name );
```

In the above example, `{email, nextLesson} = user` is evaluated. The side effect of the evaluation is that `email` and `nextLesson` are assigned to `"ashley@ilovees2015.net"` and `10` respectively. The value of the expression is `user`. Then `user2 = user` is evaluated, creating another *handle* (or call it reference or pointer depending on your taste) for the object accessible via `user`.

Based on the above thought process, the below assignment should not surprise you:

```
let {name} = {email, nextLesson} = user;
console.log( name );
```

▷   🖫 ↶ ⌷

> Make sure you use the `let` keyword to initialize new variables. You can destructure an object or an array only if all the variables inside have been declared.

In the next lesson, we will discuss deeper destructuring.