Deeper Destructuring, destructuring functions, and pitfalls

using default values and assigning undefined in destructuring

Destructuring objects and arrays in any depth is possible. We can also use default values. Objects or arrays that don't exist on the right become assigned to `undefined` on the left.

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

let {
    lessonsSeen : [
        first,
        second,
        third,
        fourth,
        fifth,
        sixth = null,
        seventh
    ],
    nextLesson : eighth
} = user;

console.log( "first:\t\t"+first+"\nsecond:\t\t"+second+"\nthird:\t\t"+third+
              "\nfourth:\t\t"+fourth+"\nfifth:\t\t"+fifth+"\nsixth:\t\t"+sixth+
              "\nseventh:\t"+seventh+"\neighth:\t\t"+eighth);
```

Notice that the `null` value of the `sixth` field behaves in the same way as a default argument value of function arguments.

destructuring function arguments

The arguments in a function signature act as left values of destructuring assignments. The parameters of a function call act as the respective right assignments. The parameters of a function can act as the respective right values of destructuring assignments.

You will use destructuring function arguments in exercises 5 and 6.

```
function f( L1, L2 )
{
 console.log(L1, L2);
}
let R1 = "R1";
let R2 = "R2";
f( R1, R2 ); // executes L1 = R1, L2 = R2
```

Destructuring pitfalls

Now let’s talk about the possible bugs that can occur as a result of destructuring.

Software developers tend to make mistakes. Don’t overuse destructuring, always keep your code readable! Continuing the above example, suppose you make a typo, and write ‘neme’ instead of ‘name’.

```
let { neme } = user;
console.log( neme );
```

```
Error: undefined
```

The typo silently assigns the value undefined to neme, potentially causing trouble. Always pay attention to fine-tuning your debugging skills.

In an L = R destructuring expression, R cannot be null or undefined, otherwise a TypeError is thrown:

```
let testUser = null;
let { name, email } = testUser;
```

```
Uncaught TypeError: Cannot match against 'undefined' or 'null' (...)
Uncaught TypeError: Cannot match against 'undefined' or 'null'.(…)
```

Now, let’s do some exercises before learning new concepts.