JavaScript TUTORIAL

ES6 Destructuring Assignment

Summary: in this tutorial, you will learn how to use the ES6 destructuring assignment that allows you to destructure an array into individual variables.

ES6 introduces a new feature called destructuring assignment, which lets you destructure properties of an object or elements of an array into individual variables.

Let's start with the array destructuring.

Introduction to JavaScript Array destructuring

The following example shows how to destructures the elements of an array returned from a function into multiple variables:

```
function getScores() {
  return [70, 80, 90];
}
let scores = getScores();

let x = scores[0],
  y = scores[1],
  z = scores[2];

console.log({ x, y, z });
```

Output:

```
```

First, define a function that returns an array of numbers:

```
function getScores() {
   return [70, 80, 90];
}
```

Second, call the `getScores()` function and assigns the returned value to a variable:

```
let scores = getScores();
```

Third, get the individual score:

```
let x = scores[0],
    y = scores[1],
    z = scores[2];
```

Before ES6, there was no direct way to assign the elements of the returned array to multiple variables such as `x` , `y` and `z` .

Fortunately, starting from ES6, you can use the destructing assignment as follows:

```
function getScores() {
  return [70, 80, 90];
}

let [x, y, z] = getScores();

console.log({ x, y, z });
```

The variables `x` , `y` and `z` will take the values of the first, second, and third elements of the returned array.

> Note that the square brackets `[]` look like the array syntax but they are not.

If the `getScores()` function returns an array of two elements, the third variable will be `undefined` , like this:

```
function getScores() {
  return [70, 80];
}

let [x, y, z] = getScores();

console.log({ x, y, z });
```

Output:

```
{ x: 70, y: 80, z: undefined }
```

If the `getScores()` function returns an array with more than three elements, the remaining elements are discarded. For example:

```
function getScores() {
  return [70, 80, 90, 100];
}

let [x, y, z] = getScores();

console.log({ x, y, z });
```

Array Destructuring Assignment and Rest syntax

It’s possible to take all remaining elements of an array and put them in a new array by using the rest syntax `(...)`:

```
function getScores() {
  return [70, 80, 90, 100];
}

let [x, y, ...args] = getScores();

console.log({ x, y, args });
```

The variables `x` and `y` receive values of the first two elements of the returned array. The `args` variable receives all the remaining arguments, which are the last two elements of the returned array.

Note that it's possible to destructure an array in the assignment that separates from the variable's declaration. For example:

```
let a, b;
[a, b] = [10, 20];

console.log({ a, b });
```

Setting default values

See the following example:

```
function getItems() {
  return [10, 20];
}

let items = getItems();
let thirdItem = items[2] != undefined ? items[2] : 0;

console.log({ thirdItem }); // 0
```

Output:

```
{ thirdItem: 0 }
```

How it works:

- First, declare the `getItems()` function that returns an array of two numbers.
- Then, assign the items variable to the returned array of the getItems() function.
- Finally, check if the third element exists in the array. If not, assign the value 0 to the `thirdItem` variable.

It'll be simpler with the destructuring assignment with a default value:

```
let [, , thirdItem = 0] = getItems();

console.log(thirdItem); // 0
```

If the value taken from the array is `undefined`, you can assign the variable a default value, like this:

```
let a, b;
[a = 1, b = 2] = [10];
console.log(a); // 10
console.log(b); // 2
```

If the `getItems()` function doesn't return an array and you expect an array, the destructing assignment will result in an error. For example:

```
function getItems() {
    return null;
}

let [x = 1, y = 2] = getItems();
```

Error:

```
Uncaught TypeError: getItems is not a function or its return value is not iterable
```

A typical way to solve this is to fallback the returned value of the `getItems()` function to an empty array like this:

```
function getItems() {
    return null;
}

let [a = 10, b = 20] = getItems() || [];
```

```
console.log(a); // 10
console.log(b); // 20
```

Nested array destructuring

The following function returns an array that contains an element that is another array, or nested array:

```
function getProfile() {
    return [
        'John',
        'Doe',
        ['Red', 'Green', 'Blue']
    ];
}
```

Since the third element of the returned array is another array, you need to use the nested array destructuring syntax to destructure it, like this:

```
let [
    firstName,
    lastName,
    [
        color1,
        color2,
        color3
    ]
] = getProfile();

console.log(color1, color2, color3); // Red Green Blue
```

Array Destructuring Assignment Applications

Let’s see some practical examples of using the array destructuring assignment syntax.

1) Swapping variables

The array destructuring makes it easy to swap values of variables without using a temporary variable:

```
let a = 10,
    b = 20;

[a, b] = [b, a];

console.log(a); // 20
console.log(b); // 10
```

2) Functions that return multiple values

In JavaScript, a function can return a value. However, you can return an array that contains multiple values, for example:

```
function stat(a, b) {
    return [
        a + b,
        (a + b) / 2,
        a - b
    ]
}
```

Then you use the array destructuring assignment syntax to destructure the elements of the return array into variables:

```
let [sum, average, difference] = stat(20, 10);
console.log(sum, average, difference); // 30, 15, 10
```

In this tutorial, you have learned how to use the ES6 destructuring assignment to destructure elements in an array into individual variables.