Extracting data from arrays

Say you got an array of names.

const names = ['Luke', 'Eva', 'Phil'];

Next, let's use destructuring to extract data from the array.

Extracting elements from an array

Let’s start with the absolute basics — extracting the first element from the array.

const [first] = names;

console.log(first); // 'Luke'

Ok, let’s break down what this syntax actually does. By wrapping a variable declaration with square brackets, we’re saying that we want to take the first element from names and assign it to the variable — in our case assigning the first element to the variable first.

Now, what do we do if we want to extract several elements — say the first and the second element? Easy — we simply add more variables between the square brackets. This will result in new elements being fetched from the front of the array and assigned to the variables defined.

const [first, second] = names;

console.log(first, second); // 'Luke' 'Eva'

Default values when elements are missing

But what happens if we try to extract four elements from our array that contains only three elements?

const [first, second, third, fourth] = names;

console.log(fourth); // undefined

In this scenario fourth will be undefined.

This is in many scenarios undesired, so what we can do is to define a default value that will be used *if* the array doesn’t contain enough elements.

const [first, second, third, fourth='Martin'] = names;

console.log(fourth); // 'Martin'

Skipping elements in the array

Until now we’ve kept it simple and picked elements from the start of the list.

Now, there may be scenarios where we would like to skip elements. In this case, we would like to avoid assigning the unwanted elements to variables.

Luckily, destructuring got a nice solution for this problem.

var [first, , second] = names;

console.log(first, second); // 'Luke' 'Phil'

By simply adding another comma without assigning a variable, we’re able to jump to the next element. What if we want to skip more elements? Easy — simply add more commas!

Assign the rest of an array to a variable

We’ve seen that extracting different elements is easy, but often we would like to keep the part of the array that hasn’t been extracted.

Let’s see how we could do this using destructuring.

var [first, ...rest] = names;

console.log(rest); // ['Eva','Phil']

By prefixing the last variable with ... we’re able to say that we want to assign the rest of the array to a variable called rest.

Destructuring objects

Ok, now that we’ve got the array covered, let’s look at have we can extract data from objects.

Consider the following object describing a person.

const person = {

name: 'Luke',

age: '24',

facts: {

hobby: 'Photo',

work: 'Software Developer'

}

}

Extracting data from the object

Let’s again start with the basics by extracting the name and the age from person.

const {name, age} = person;

console.log(name, age); // 'Luke' '24'

As you can see, we almost got the same syntax as with the array — only that we now use curly braces instead of square brackets to say what we want to extract.

Extracting nested values

Now, what if we want to extract something deeper in the object structure — say the person’s hobby?

const {facts: {hobby}} = person;

console.log(hobby); // 'Photo'

By using the colon, we’re able to explain the path through the object, resulting in us being able to explain what property we’re interested in.

Default values when data is missing

As we saw when destructuring arrays, we’re able to give default values that will be used if the data to be extracted doesn’t exist.

To see how this is done, let’s attempt to extract the hometown, but with Unknown as the default value.

const {hometown = 'Unknown'} = person;

console.log(hometown); // 'Unknown'

Destructuring the function parameter

Before wrapping up, let’s look at one last handy usage — destructuring function parameters.

Say your function gets an object as a parameter. Then you can destructure the parameter straight in the parameter list!

Let’s give it a try by creating a toString function that prints the persons name and age.

const toString = ({name, age}) => {

return `${name} is ${age} years old`;

}

toString(person); // Luke is 24 years old

That's it for now. In the next post we'll look at another cool addition to JavaScript, so make sure to subscribe!

**function** **f**(«FORMAL\_PARAMETERS») {

«CODE»

}

f(«ACTUAL\_PARAMETERS»);

# Use of colon in function definition and function call:

function({

incomingProperty: varName = defaultValue

...

})

For an object of parameters, there will be a variable varName for property incomingProperty, with defaultValue by default.

**function** **setLevel**(newLevel = 0) {

light.intensity = newLevel;

}

**function** **setOptions**(options) {

// Missing prop returns undefined => use default

setLevel(options.dimmerLevel);

setMotorSpeed(options.speed);

···

}

setOptions({speed:5});