COURSEWARE

- Professional Skills
- Agile Fundamentals
- Jira
- Git
- Databases Introduction
- Java Beginner
- Maven
- Testing (Foundation)
- Java Intermediate
- HTML
- CSS
- Javascript
  - What is JavaScript
  - Getting started with JS
  - Variables
  - Data types
  - ASI
  - Strict mode
  - Iteration
  - Conditionals with Truthy / Falsey
  - Objects, Arrays + JSON
  - Structuring JS Code
  - Destructuring
  - Scope
  - Functions, function expressions and arrow functions
  - The ECMAScript 6 Specification
  - OOP in JavaScript
  - Best Practices
  - Closures
  - Callbacks and Promises
  - Cookies
  - Hoisting
  - Prototypes
  - Query Parameters
  - Higher Order Functions

Destructuring

Contents

Overview

Destructuring is a JavaScript expression that allows us to extract data from arrays, objects, maps and sets.
Destructuring assignment allows us to assign the properties of an array or object to variables using syntax that looks similar to array or object literals.

Looking at the example below, if we want to access an element within our array we pass in the name followed by the position.
We can see from the last line of the **without destructuring** example that this can get very long without destructuring as we need to repeatedly print multiple elements, specifying the name of the array and the position each time.

Without destructuring

```
const myArr = [1,2,3,4,5];
console.log(myArr[0]); //1
console.log(myArr[1]); //2
console.log(myArr[2],myArr[3],myArr[4]); //3,4,5
```

We can easily split the array down and give it assign it values `const [a,b,c,d,e] = myArray;`. This now means that each value or `myArray` is assigned to each of the values a,b,c,d,e.

With destructuring

```
const myArray =  [1,2,3,4,5];
const [a,b,c,d,e] = myArray;
console.log(a); //1
console.log(b); //2
console.log(c,d,e); //3,4,5
```

Tutorial

Spread operator

We also can use the **spread operator** by ensuring we use **three** dots (...) followed by a name we are able to assign the rest of the array to that variable name in this case we named it `rest`.

```
const myArray =  [1,2,3,4,5,6,7,8,9,];
const [a,b,...rest] = myArray;
console.log(a); //1
console.log(b); //2
console.log(rest); //3,4,5,6,7,8,9
```

skipping values

We can also skip over elements by leaving a blank space by using a comma (,) as a separator.

```
const myArray =  [1,2,3,4,5,6,7,8,9,];
const [a,,c,...rest] = myArray;
console.log(a); //1
console.log(c); //3
console.log(rest); //,4,5,6,7,8,9
```

Objects

We can also use destructing on objects to break them down into similar parts.

```
const person = {
  first: 'Chris',
  last: 'Perrins',
  country: 'UK',
  city: 'Manchester',
};
let {first, last} = person;
console.log( `My name is ${first} ${last}`);
//output: My name is Chris Perrins
```

Default values

Default values for Objects

We can also use default values which can be assigned to a variable as opposed to showing undefined.

```
const person = {
  first: 'Chris',
  last: 'Perrins',
  country: 'UK',
  city: 'Manchester',
};
let {first, last,gender="Female"} = person;
console.log( `My name is ${first} ${last} and I am a ${gender}!`);
//output: "My name is Chris Perrins and I am a Female!"
```

We can then update the `person object` by adding `gender : 'Male'` and this will overwrite the default value of `Female`.

```
 const person = {
  first: 'Chris',
  last: 'perrins',
  country: 'UK',
  city: 'Manchester',
  gender:'Male'
};
let { first,last, gender="Female" } = person;
console.log( `my name is ${first} ${last} and i am a ${gender}!`);
//output: "My name is Chris perrins and i am a Male!"
```

Default values for Arrays

As before we can assign default values. In this case we gave a default value of 255 to red and blue but left green.

```
const rgb = [200];
const [red = 255, green, blue = 255] = rgb;
console.log(`R: ${red}, G: ${green}, B: ${blue}`);
//output:  R: 200, G: undefined, B: 255
```

we also have the ability to an array destructuring assignment. Unlike with object destructuring, you don’t need to enclose the assignment expression in parentheses `[red, green] = rgb;`.

```
let red = 100;
let green = 200;
let blue = 50;
const rgb = [200, 255, 100];
[red, green] = rgb;
console.log(`R: ${red}, G: ${green}, B: ${blue}`);
//output: R: 200, G: 255, B: 50
```

Nested destructuring

Nest Object destructuring

Below we have an example of destructuring nested objects.
Notice that, scores is not defined as a variable. Instead, we use nested destructuring to extract the maths & science values from the nested scores object.

```
const student = {
    name: 'John Doe',
    age: 16,
    scores: {maths: 74,english: 63}
};
const { name, scores: {maths, science = 50} } = student;

console.log(`${name} scored ${maths} in Maths and ${science} in Science.`);
//output: John Doe scored 74 in Maths and 50 in Science.
```

When using nested object destructuring, be careful to avoid using an empty nested object literal.

```
const { scores: {} } = student;
```

Nested Array destructuring

Just like objects, we can also do nested destructuring with arrays. The corresponding item must be an array in order to use a nested destructuring array literal to assign items in it to local variables.

```
const color = ['#FF00FF', [255, 0, 255], 'rgb(255, 0, 255)'];
// Use nested destructuring to assign red, green and blue
const [hex, [red, green, blue]] = color;
console.log(hex, red, green, blue);
//output: #FF00FF 255 0 255
```

Swapping values

Another advantage of Javascript is that during array destructuring we have the ability to swap local variables. For instance if we made a website that does photo manipulation we might want to swap the `height` & `width` when changing the orientation of the image.

The below example shows how we would traditionally do this.

```
let width = 300;
let height = 400;
const landscape = true;

console.log(`${width} x ${height}`);
//output:300 x 400

if (landscape) {
    // An extra variable is needed to copy initial width to
    let initialWidth = width;
    // Swap the variables
    width = height;
    // height is assigned the copied width value
    height = initialWidth;
    console.log(`${width} x ${height}`);
    //output 400 x 300
}
```

The example below shows how we can perform the swap in a single assignment statement.

```
let width = 300;
let height = 400;
const landscape = true;

console.log(`${width} x ${height}`);
//output :300 x 400
if (landscape) {
    // Swap the variables
    [width, height] = [height, width];
    console.log(`${width} x ${height}`);
    //output: 400 x 300
}
```

Using different variable names

```
const person = {
    name: 'Isaac Mewton',
    country: 'UK'
};
// Assign default value of 25 to years if age key is undefined
const { name: fullname, country: place, age: years = 25 } = person;

// Here I am using ES6 template literals
console.log(`I am ${fullname} from ${place} and I am ${years} years old.`);
//output: I am Isaac Mewton from UK and I am 25 years old.'
```

Exercises

1. With the object below write the destructuring assignment that reads:
    - `name` property into the variable `name`.
    - `years` property into the variable name `age`.
    - `isAdmin` property into the variable `isAdmin` (false, if no such property)

    ```
    let user = {
      name: "John",
      years: 30
      };

    // your code to the left side:
    // ... = user
    alert( name ); // John
    alert( age ); // 30
    alert( isAdmin ); // false
    ```

    ▶ Solution