chapter 5 / 8

More ES6 features

Sections in this chapter:

5-1. ES2015 aka ES6

We have in previous chapters seen the use of block-scoped variables with `let` and `const`, functions as `arrow functions` and `template strings` amongst other things. *5-1-1*

Which is some of the handy functions of the EcmaScript 2015 version.

ES2015, also called **ES6** is arguably the most notable update that ECMAScript (and hence, JavaScript) has ever received. *5-1-2*

With that in mind, let's walk through some more of the prominent new syntactic structures.

But first, a short recap of some of them. *5-1-3*

The `let` keyword

5-1-4

Using `let` is now the preferred way to declare variables, instead of using the `var` keyword.

We get smaller scopes and avoid hoisting problems.

Q What output will this program produce? *5-1-5*

```
var radii = [2, 4, 6, 8];
var areas = [];

for (var i in radii) {
    var r = radii[i];
    areas.push(r*r*Math.PI);
}

console.log(r);
```

A Due to hoisting, `r` is accessible from outside. *5-1-6*

The program will log `8` to the console.

Using `let`, we get lexical/block scopes as expected: *5-1-7*

```
let radii = [2, 4, 6, 8];
let areas = [];

for (let i in radii) {
    let r = radii[i];
    areas.push(r*r*Math.PI);
}

console.log(r); // <-- this is now an error
```

We cannot access `r` outside its scope.

Nicer iteration

5-1-8

With ES6 we get the `of` keyword in for loops.

This lets us get a value directly when iterating and not having to use its index to find the value.

Instead of: 5-1-9

```
let values = [1,2,3,4,5,6,7,8,9];

for (let i in values) {
    console.log(values[i]);
}
```

Note the `in` keyword

... we can now type:

```
let values = [1,2,3,4,5,6,7,8,9];

for (let n of values) {
    console.log(n);
}
```

Note the `of` keyword

Arrow functions 5-1-10

Suddenly, we have a much clearer way to write functions. Instead of this:

```
var radii = [2, 4, 6, 8];

var areas = radii.map(function (r) {
    return r*r*Math.PI;
});
```

... we can now write this:

```
let radii = [2, 4, 6, 8];
let areas = radii.map(r => r*r*Math.PI);
```

Template strings 5-1-11

Template strings, or template literals, gives a an easy way to write our strings.

Whether we want to include a value of a variable, an expression, or have a multi-line string, we can do this.

We write template strings using back ticks, ``

```
let word = "From outside"

let str = `Hello World!

A template string can easily becoma mutli-line string :-)

We can also include variables: "${word}",
as well as other inline expressions: ${34 + 33}`
```

5-2. Destructuring Assignment

Destructuring Assignment is a syntax that makes it possible to unpack values from arrays, or properties from objects, into distinct variables. 5-2-1

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

console.log(a); // 10

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

Destructuring - Array Matching 5-2-2

When using destructuring assignment with arrays, also knows as **array matching**, we define the variables within hard brackets, `[]`

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

- `a` equals the first value in the array
- `b` equals the next and so on.

If the amount of variables exceeds the amount of values in the array, that variable will have the value of `undefined`. 5-2-3

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

console.log(c); // undefined
```

We can save ourselves from getting values of undefined, with so called **fail-soft destructuring**. 5-2-4

```
let [a, b, c = 30] = [10, 20];

console.log(c); // 30
```

If the value from the array we are assigning from is `undefined` it will get the fallback value.

But if the value in fact exists that will be used. 5-2-5

```
let [a, b, c = 30] = [10, 20, 100];

console.log(c); // 100
```

We can skip value in the array with just a comma sign 5-2-6

```
let list = [ 1, 2, 3 ]
let [ a, , b ] = list;

console.log(a); // 1
console.log(b); // 3
```

Destructuring - Object matching 5-2-7

We can pick the properties of choice into separate variables by property name, this is called **object matching**.

```
let obj = {one: 1, two: 2, three: 3, four: 4};

let {one, four} = obj;
console.log(one); // 1
console.log(four); // 4
```

If we name a variable that don't match any property name that variable would get the value of `undefined` 5-2-8

```
let obj = {one: 1, two: 2, three: 3, four: 4};

let {one, nine} = obj;
console.log(one); // 1
console.log(nine); // undefined
```