AkiraChix
SHE BUILDS • SHE SERVES • SHE LEADS

Outline

- Destructuring
- Arrow functions
- Modules (export and import)

Destructuring

Destructuring

Destructuring simply implies breaking down a complex structure into simpler parts. In JavaScript, this complex structure is usually an object or an array. With the destructuring syntax, you can extract smaller fragments from arrays and objects.

Destructuring syntax can be used for ***variable declaration*** or ***variable assignment***. You can also handle nested structures by using nested destructuring syntax.

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

Destructuring


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

console.log(a);
// expected output: 10

console.log(b);
// expected output: 20

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(rest);
// expected output: Array [30,40,50]
```

Array Destructuring : Basic variable assignment


```
const foo = ['one', 'two', 'three'];

const [red, yellow, green] = foo;

console.log(red); // "one"

console.log(yellow); // "two"

console.log(green); // "three"
```

Array Destructuring : Assigning the rest of an array to a variable

When destructuring an array, you can unpack and assign the remaining part of it to a variable using the rest pattern:

```
const [a, ...b] = [1, 2, 3];

console.log(a); // 1

console.log(b); // [2, 3]
```

Array Destructuring : Swapping variables

Two variables values can be swapped in one destructuring expression.

Without destructuring assignment, swapping two values requires a temporary variable (or, in some low-level languages, the XOR-swap trick).

```
let a = 1;

let b = 3;

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

console.log(a); // 3

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

Arrow functions

Array Functions

One of the most heralded features in modern JavaScript is the introduction of arrow functions, sometimes called 'fat arrow' functions, utilizing the new token =>.

These functions two major benefits - a very clean concise syntax and more intuitive scoping and this binding.

```
hello = function() {
  return "Hello World!";
}
```

```
hello = () => {
  return "Hello World!";
}
```

Array Functions

What About this?

The handling of this is also different in arrow functions compared to regular functions.In short, with arrow functions there are no binding of this.

In regular functions the this keyword represented the object that called the function, which could be the window, the document, a button or whatever.

With arrow functions the this keyword *always* represents the object that defined the arrow function.

Let us take a look at two examples to understand the difference.

Array Functions : this identifier

With a regular function this represents the object that *calls* the function:

```
//Regular Function:
hello = function() {

document.getElementById("demo").i
nnerHTML += this;
}
```

With an arrow function this represents the *owner* of the function:

```
//Arrow Function:
hello = () => {

document.getElementById("demo").innerH
TML += this;
}
```

Modules (export and import)

Modules

In programming, modules are self-contained units of functionality that can be shared and reused across projects.

They make our lives as developers easier, as we can use them to augment our applications with functionality that we haven't had to write ourselves.

They also allow us to organize and decouple our code, leading to applications that are easier to understand, debug and maintain.

Modules : Export

Now let’s look at how to create our own module and export it for use elsewhere in our program. Start off by creating a user.js file and adding the following:

user.js


```
const getName = () => {
  return 'Jim';
};

const getLocation = () => {
  return 'Munich';
};

const dateOfBirth = '12.01.1982';

exports.getName = getName;
exports.getLocation = getLocation;
exports.dob = dateOfBirth;
```

index.js:


```
const user = require('./user');

console.log(
  `${user.getName()} lives in ${user.getLocation()} and was born on ${user.dob}.`
);
```

The code above produces this:

Jim lives in Munich and was born on 12.01.1982.

Notice how the name we give the exported dateOfBirth variable can be anything we fancy (dob in this case). It doesn’t have to be the same as the original variable name.

Modules : Import

To cover this topic, start off by watching this video.

- Watch this video - https://www.youtube.com/watch?v=s9kNndJLOjg

Ensure you are familiar with using Node js on decoder.

On Dcoder, ensure you have the 2 files mentioned in the exports example i.e users.js and index.js

Class Discussion

- Name ways in which we can declare variable declaration or variable assignment in javascript.
- Run the export module example.
- Run the import module example.

Course Outline

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions

Video of the week - https://www.youtube.com/watch?v=22fyYvxz-do