Bronnen = dag 4

Scope

Eloquent JavaScript - Binding and Scope (book)

Read the except about binding and scope in third chapter of the Eloquent JavaScript book.

Bindings and scopes

```
const hummus = function(factor) {
  const ingredient = function(amount, unit, name) {
    let ingredientAmount = amount * factor;
    if (ingredientAmount > 1) {
      unit += "s";
    }
    console.log(`${ingredientAmount} ${unit} ${name}`);
  };
  ingredient(1, "can", "chickpeas");
  ingredient(0.25, "cup", "tahini");
  ingredient(0.25, "cup", "lemon juice");
  ingredient(1, "clove", "garlic");
  ingredient(2, "tablespoon", "olive oil");
  ingredient(0.5, "teaspoon", "cumin");
};
```

The code inside the `ingredient` function can see the `factor` binding from the outer function. But its local bindings, such as `unit` or `ingredientAmount`, are not visible in the outer function.

You Don't Know JS - Block scope (book)

```
window.something = 42;

let something = "Kyle";

console.log(something);
// Kyle

console.log(window.something);
// 42
```

The `let` declaration adds a `something` global variable but not a global object property. The effect then is that the `something` lexical identifier shadows the `something` global object property.

It's almost certainly a bad idea to create a divergence between the global object and the global scope. Readers of your code will almost certainly be tripped up.

A simple way to avoid this gotcha with global declarations: **always use `var` for globals. Reserve `let` and `const` for block scopes.**

ES Modules (ESM)

Despite being declared at the top level of the (**module**) file, in the outermost obvious scope, `studentName` and `hello` are not global variables. Instead, they are module-wide, or if you prefer, "module-global."

Node

Before processing, Node effectively wraps such code in a function, so that the `var` and `function` declarations are contained in that wrapping function's scope, **not** treated as global variables.

So how do you define actual global variables in Node? The only way to do so is to add properties to another of Node's automatically provided "globals," which is ironically called `global`.

`global` is a reference to the real global scope object, somewhat like using `window` in a browser JS environment.

JS environments

Reviewing the JS environments we've looked at so far, a program may or may not:

- Declare a global variable in the top-level scope with `var` or `function` declarations—or `let`, `const`, and `class`.
- Also add global variables declarations as properties of the global scope object if `var` or `function` are used for the declaration.
- Refer to the global scope object (for adding or retrieving global variables, as properties) with `window`, `self`, or `global`.

Scope in JavaScript - HTTP 203

global scope

- self
- window
- global
- globalThis

Hoisting

Mozilla Developer Network - Hoisting (article)

```
console.log(num); // Returns undefined, as only declaration was hoisted, no
initialization has happened at this stage
var num; // declaration
num = 6; // initialization
```

```
console.log(num); // throws ReferenceError exception
num = 6; // initialization
```

```
// No hoisting, but since initialization also causes declaration (if not
already declared), variables are available.

a = 'Cran'; //Initialize a
b = 'berry'; //Initialize b
console.log(a + "" + b); // 'Cranberry'
```

Closures

Fun Fun Function - Closures (video)

JS functions zijn closures

Dit is te zien bij een functie in een functie. In de inner function kan een variabele gebruikt worden die in de outer function staat.

```
function makeFunc() {
      var name = 'Mozilla';
      function displayName()
            { alert(name);
      }
      return displayName;
}

var myFunc = makeFunc();
myFunc();
```