Bronnen – dag 1

Introduction to JavaScript

Eloquent JavaScript book – Intro (book)

At one point language-based interfaces, such as the BASIC and DOS prompts of the 1980s and 1990s, were the main method of interacting with computers. They have largely been replaced with visual interfaces, which are easier to learn but offer less freedom. Computer languages are still there, if you know where to look. One such language, **JavaScript, is built into every modern web browser and is thus available on almost every device.**

Some programmers believe that this complexity is best managed by using only a small set of well-understood techniques in their programs. They have composed strict rules (“best practices”) prescribing the form programs should have and carefully stay within their safe little zone.

This is not only boring, it is ineffective. New problems often require new solutions. **The field of programming is young and still developing rapidly, and it is varied enough to have room for wildly different approaches**. There are many terrible mistakes to make in program design, and you should go ahead and make them so that you understand them. **A sense of what a good program looks like is developed in practice, not learned from a list of rules.**

In practice, the terms ECMAScript and JavaScript can be used interchangeably—they are two names for the same language.

ECMAScript versions

The Complete ECMAScript 2015-2019 Guide (article)

They are all referring to a **standard**, called ECMAScript.

ECMAScript is the **standard upon which JavaScript is based**, and it’s often abbreviated to **ES**.

JavaScript is the **most popular** and widely used implementation of ES.

You Don't Know JS Yet (GitHub article)

"Java is to JavaScript as ham is to hamster." --Jeremy Keith, 2009

TC39 = the technical steering committee that manages JS. Their primary task is managing the official specification for the language. They meet regularly to vote on any agreed changes, which they then submit to ECMA, the standards organization.

Contrary to some established and frustratingly perpetuated myth, there are *not* multiple versions of JavaScript in the wild. There's just **one JS**, the official standard as maintained by TC39 and ECMA.

JS is backwards compatible (met fouten en al), maar niet forwards compatible

HTML/CSS is forwards comaptible, maar niet backward compatible

ES5 versus ES6

Fun Fun Function - var, let and const (video)

var (ES5)

ES5: één sort variable scope

immediately invoked function expression (IIFI)

variabelen worden hoisted naar de bovenkant van de functie als de variabele declared wordt binnen de functie (binnen de functie)

als de variabele niet declared wordt binnen de functie, maar wel wordt gevuld, dan maakt JS een variabele aan buiten de functie en hoist deze naar de bovenkant van de gehele code. de variabele is dan wel beschikbaar buiten de functie. zo kun je per ongeluk global variables declaren.

gebruik hiertegen: “use strict” statement in de code

let (ES6)

block scope: blijft binnen statement en kun je niet global ophalen

const (ES6)

geeft een error bij reassigning (deels immutable)

property’s kunnen wel aangepast worden

minimize mutable state: zorgt voor het verkleinen van problemen

laat de volgende developer zien dat deze variabele niet veranderd gaat worden

Values, types and operators

Eloquent JavaScript - Values (book)

bits: two valued things, bijv. 0 en 1

values moeten ergens worden opgeslagen om te kunnen worden aangeroepen

oneindige getallen als breuken of speciale getallen (zoals pi) zijn altijd benaderingen, omdat er een limiet zit op de grootte van een JS getal

arithmetic operators: +, -, *, /, % (% geeft na deling het restgetal)
hier gelden de rekenregels

infinity, -infinity (infinity – 1 is nog steeds infinity)
→ leidt vaak tot NaN, zoals infinity – infinity of 0 / 0
NaN is de enige waarde in JS die niet gelijk staat aan zichzelf in boolean (NaN == NaN //false)

undefined, null bijna hetzelfde (null == undefined //true): lege waarden

string = tekst

```
"This is the first line\nAnd this is the second"
→ This is the first line
And this is the second

"A newline character is written like \"\\n\"."
→ A newline character is written like "\n".

`half of 100 is ${100 / 2}`
→ "half of 100 is 50"
```

```
console.log(typeof 4.5)
// → number
console.log(typeof "x")
// → string
```

comparison

=== voor de zekerheid gebruiken ipv ==

Logical operators

&&, **||** werken hetzelfde
&& kijkt of de rechterwaarde false is, in dat geval wordt de linkerwaarde teruggegeven
|| kijkt of de linkerwaarde false is, in dat geval wordt de rechterwaarde teruggegeven
false: 0, NaN, " " (empty string)
als de eerste waarde true is, wordt er nooit gekeken naar de tweede waarde (**short-circuit evaluation)**

Conventions

Mozilla Developer Network - JavaScript guide (article)

Comments altijd vóór de betreffende line (let ook op de spatie tussen // en de comment)

```
// Output the string 'Hello' to the browser's JS console
  console.log('Hello');
```

use strict equality: === en !==

strings

gebruik string literals:

```
let myName = 'Chris';
console.log(`Hi! I'm ${myName}!`);
```

gebruik:

```
para.textContent = text;
```

ipv

```
para.innerHTML = text;
```

Creating objects

Do this:

```
let myObject = { };
```

Not this:

```
let myObject = new Object();
```

Adding to an array

When adding items to an array, use `push()`, not direct assignment. Given the following array:

```
const pets = [];
```

do this:

```
pets.push('cat');
```

not this:

```
pets[pets.length] = 'cat';
```

Error handling

If certain states of your program throw uncaught errors, they will halt execution and potentially reduce the usefulness of the example. You should therefore catch errors using a `try...catch` block:

```
try {
  console.log(results);
}
catch(e) {
  console.error(e);
}
```