

# Webapplicaties II

ECMAScript – formerly known as JavaScript.

**The strength of JavaScript is that you can do anything. The weakness is that you will.**

REG BRAITHWAITE



# Inhoud

- Inleiding.
- De bouwstenen: variabelen, primitieve datatypes.
- Hello world.
- Instructies en operatoren.
- Functies – Hoisting.
- Arrays.

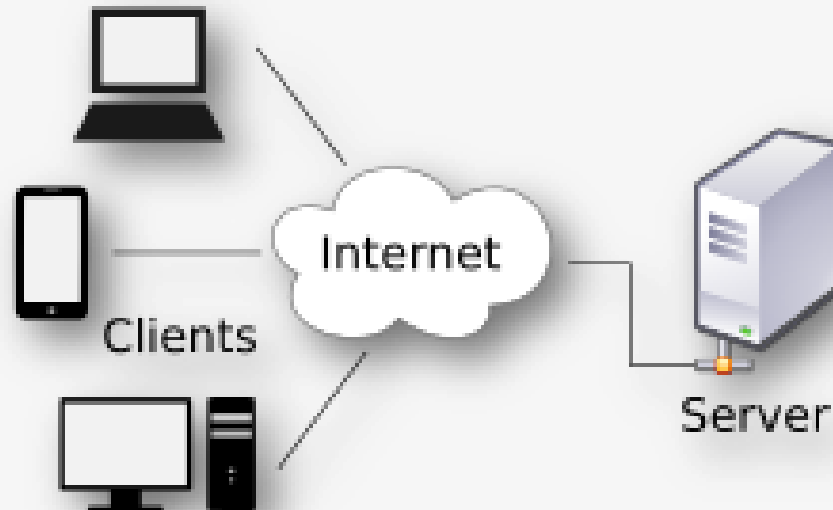


# Surfen op het web

- Als je een webpagina bezoekt, ga je eigenlijk een verzoek doen aan een server (uri = adres van server – [www.deredactie.be](http://www.deredactie.be)) om een pagina naar jouw toestel (laptop – tablet – smartphone) door te sturen.
- Als deze pagina op de server staat, zal deze naar jouw toestel doorgestuurd worden. Je browser (de client) op je toestel zal deze pagina opbouwen en weergeven.
- Surfen op het web maakt dus gebruik van een client- server architectuur. Meer details hierover in het OLOD Netwerken.

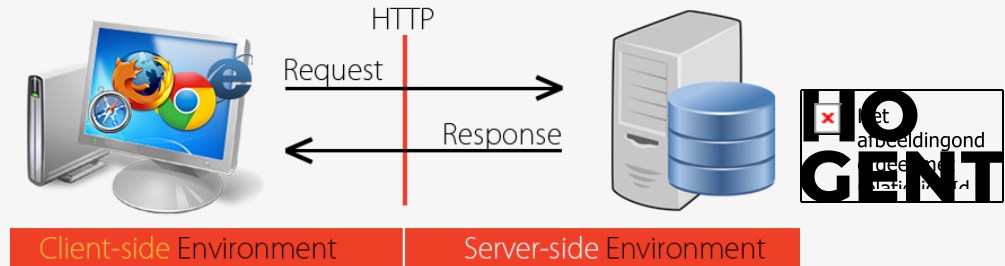


# Client-server architecture



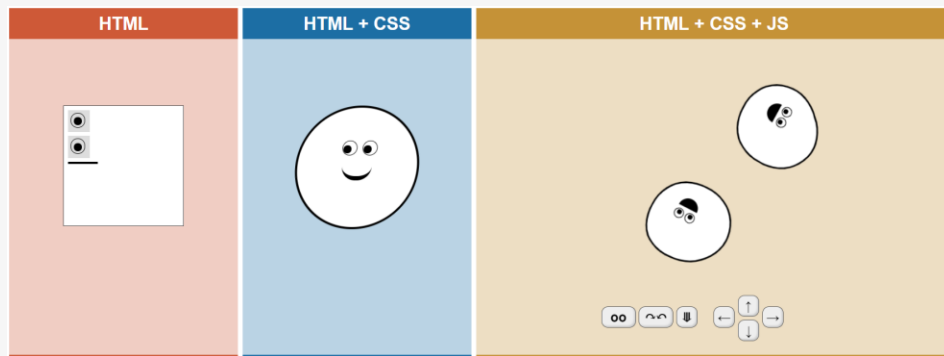
# Client-server architecture

- Het **client-servermodel** is een model in de informatica en computertechnologie voor de samenwerking tussen twee of meer programma's, die zich op verschillende computers kunnen bevinden. Kenmerkend voor de asymmetrie in het model is:
  - de server is permanent beschikbaar en is reactief
  - een client is bij gelegenheid actief en neemt het initiatief tot communicatie met de server
- Een aantal voorbeelden van het client-servermodel zijn e-mail, het www, FTP, Telnet, SSH, ...



# Server – client architecture

- De client (browser) begrijpt drie talen:
  - **HTML** – Hypertext Mark-up Language: geeft aan je pagina **structuur** en **betekenis** door middel van headings, tabellen, afbeeldingen, lijsten, ...
  - **CSS** – Cascading Stylesheet: geeft aan je pagina **opmaak** en **layout** door middel van logische indeling en eigenschappen aan de inhoud zoals lettertype – kleur - ...
  - **ECMAScript** (= JavaScript): geeft interactie aan je pagina zoals het klikken op knoppen, met je cursor bewegen, vraagt data op en geeft deze in de browser weer, kan voor animatie zorgen (games) ...
- Dit zijn client-side technologieën: het enige wat elke browser begrijpt.



# Overzicht OLOD's webapplicaties

- Webapplicaties I behandelt HTML en CSS
- Webapplicaties II behandelt ECMAScript
- Webapplicaties III behandelt de server side.
- Webapplicaties IV behandelt een client side scripting framework (Angular, React, Vue, ...)

# Geschiedenis

- Een uitgebreide geschiedenis van JavaScript vind je op <https://en.wikipedia.org/wiki/JavaScript#History>
- Een video van de bedenker en ontwikkelaar van JavaScript: <https://www.youtube.com/watch?v=3-9fnjzmXWA>
- Voor een samenvatting van de geschiedenis van deze technologie: zie annex.



# JavaScript

- De meeste scriptingtalen zijn ontworpen voor ofwel de client side ofwel de server side te worden uitgevoerd.
- JavaScript is oorspronkelijk ontworpen als client-side(front-end) technologie, maar is sinds enkele jaren verder ontwikkeld als server-side (back-end) technologie: Node.js.



# ECMAScript

ES6 (ook ECMAScript2015 genaamd)

- Een overzicht van de nieuwe features: <http://es6-features.org/>
  - Verbeterde syntax voor features die reeds bestonden via externe libraries
    - [Classes](#)
    - [Modules](#)
  - Nieuwe functionaliteiten in de standard bibliotheek
    - Nieuwe methodes voor [strings](#) en [Arrays](#)
    - [Promises](#)
    - [Maps, Sets](#)
  - Volledig nieuwe features
    - [Generators](#)
    - [Proxies](#)
    - [WeakMaps](#)
- De browser ondersteuning: <https://kangax.github.io/compat-table/es6/>
- Wordt gebruikt in combinatie met een transpiler (bvb Babel) voor compilatie naar ES5



# ECMAScript

- ES Next: code name voor nieuwe versies (totdat het versie nummer gekend is)
  - Probleem met ES6: 6 jaar nodig alvorens standaardisatie
  - Vanaf ECMAScript 2016(ES7): 1 release per jaar, volgens het *TC39 process*
    - Meer op: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/New\\_in\\_JavaScript/ECMAScript\\_Next\\_support\\_in\\_Mozilla#See\\_also](https://developer.mozilla.org/en-US/docs/Web/JavaScript/New_in_JavaScript/ECMAScript_Next_support_in_Mozilla#See_also)
    - Browser ondersteuning: <https://kangax.github.io/compat-table/es2016plus/>



# ECMAScript

- Een ECMAScript engine is een programma dat de broncode uitvoert die is geschreven in een versie van ECMAScript.
- Elke browser heeft zo zijn eigen ECMAScript engine.



**Chakra**



**Nitro**



**V8**

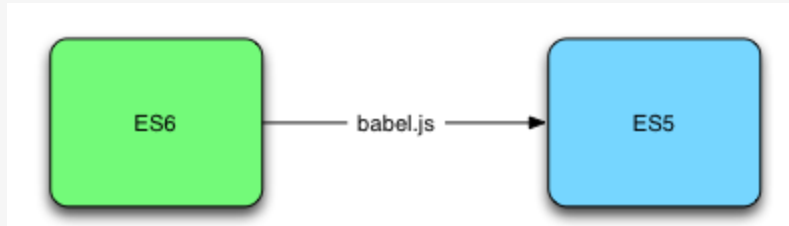


**Carakan**



**SpiderMonkey(s)**

# Transpilers



- Niet alle browsers ondersteunen reeds alle features.
- Transpilers maken het mogelijk om ES versies van morgen reeds vandaag te gebruiken en te runnen in de browser
  - Enkele voorbeelden : [Babel](#), [Traceur](#)
- Ook tal van andere talen zoals TypeScript, Dart ... maken gebruik van transpilers om naar ES5 om te zetten
- Voor de transpilers en features die ze ondersteunen: Browser ondersteuning :  
<https://kangax.github.io/compat-table/es2016plus/>

# Transpilers

## Babel transforms your JavaScript

You put JavaScript in

JavaScript

Try Copy

```
[1,2,3].map(n => n + 1);
```

And get JavaScript out

JavaScript

Try Copy

```
[1,2,3].map(function(n) {  
  return n + 1;  
});
```

# ES5 vs ES6

- In deze cursus komt voornamelijk ES6 aan bod, maar er zijn ook enkele belangrijke features van ES5 opgenomen. Waarom?
  - ES6 is een superset van ES5 - nieuwe JavaScript-versies mogen nooit bestaande code breken.
  - Er zijn verschillende ES6-functies die de ES5-functies vervangen, maar deze toch als hun basis gebruiken. Het is belangrijk om die fundamenteën te begrijpen.
  - Het is belangrijk om oude code te kunnen begrijpen.



# Java vs JavaScript



- Java is een object geörienteerde programmeertaal.
- JavaScript is sterk beïnvloed door functionele programmeertalen.
- Zo zijn functies in JavaScript *first-class*, wat wil zeggen dat functies gewone objecten zijn.



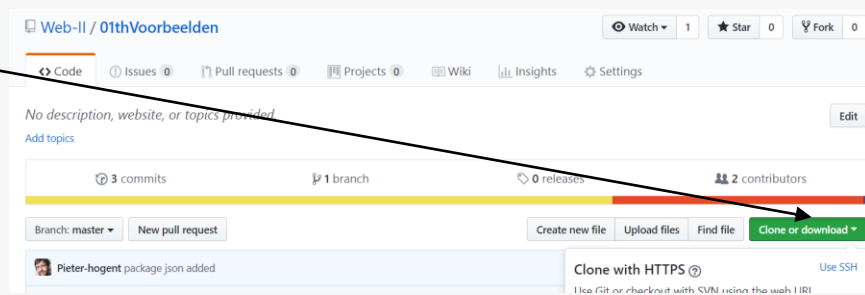
# Java vs JavaScript



Java	JavaScript
Strongly typed	Loosely typed
Static	Dynamic
Classical inheritance	Prototypical
Classes	Functions
Constructors	Functions
Methods	Functions
Compiled	Interpreted
Methods for working with I/O – network / files	Uses the hosting environment


# Github

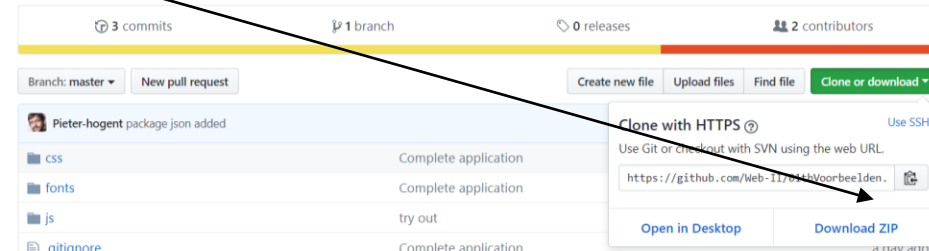
- De **voorbeelden en oefeningen** staan op <https://github.com/web-II>
  - Maak eerst een account aan op github.com
  - Om een voorbeeld/oefening te downloaden
    - Surf naar <https://github.com/web-II>
    - Selecteer de repository, bvb 01thVoorbeelden
    - Klik op clone or download



# Github

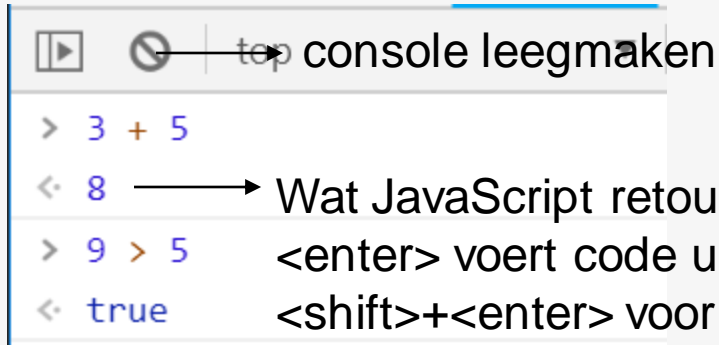
## Om een voorbeeld/oefening te clonen

- Klik vervolgens op  om de applicatie te clonen (dit plaatst de url op het clipboard) of op “download zip” om de applicatie te downloaden
- Open Visual Studio code.
- Kies het **Git: Clone** command in de **Command Palette** (Ctrl+Shift+P).
- Geef vervolgens de URL in van de externe repository en de folder waaronder de lokale repository wordt geplaatst.
- Open dan de lokale repository



# JavaScript console

- Open Chrome
- Ga naar de Developer tools (F12) (Mac: alt-cmd-I)
- Kies de Console tab.  
Hier kan je JavaScript commando's uitvoeren.
- Instructies voor Safari op Mac <https://javascript.info/devtools#safari>



# JavaScript toevoegen aan html pagina

- Via <script>-tag voeg je JavaScript toe aan de html pagina
  - De JavaScript-code staat ofwel in de webpagina zelf.

.html

```
<script>  
...  
</script>
```



# JavaScript toevoegen aan html pagina

- De JavaScript-code kan ook in een afzonderlijk bestand staan. In de script-tag wordt verwezen naar dit bestand (Best practice)

`<script src= "js/hello.js"></script>` .html

`console.log("Hello world");` .js

- Conventies : extensie bestand .js
- Het bestand bevat enkel de JavaScript code, geen script tag
- src = verwijzing naar JavaScript bestand, relatief of absolute adressering
- Voordeel: herbruikbaarheid, onderhoudbaarheid, caching in browser

# JavaScript toevoegen aan html pagina

- De script-tag plaats je in de <body> tag **na** de inhoud (juist voor </body>))
  - Reden: JavaScript wordt pas gedownload, geparsed, en geïnterpreteerd als pagina al is weergegeven in de browser.



# De bouwstenen

- JavaScript gebruikt variabelen om data bij te houden.
- JavaScript catalogeert de data in een aantal basistypes
  - Tekst
  - Getallen
  - Boolean
- JavaScript is een **loosely typed language**: geen type declaraties !!





# De bouwstenen : variabelen

- Een variabele is een geheugenlocatie met een unieke naam (identifier).
  - De namen van variabelen in JavaScript zijn:
    - **Hoofdlettergevoelig**
    - 1 woord, minimaal 1 karakter lang
    - beginnen met een letter, \$ of een underscore
    - Spaties en andere speciale karakters zijn NIET toegelaten.
    - Het is een goede gewoonte om CamelCase toe te passen (elke nieuw woord in de naam start met een hoofdletter).
  - 3 manieren om variabelen te declareren: const, let, var

TIP: Kies altijd **betekenisvolle** namen voor je variabelen!



# De bouwstenen : variabelen

- Declaratie van een variabele waarvan de waarde kan veranderen.

```
let variabeleNaam = initiële waarde;
```

- Naamgeving: zie vorige slide
- Type: afhankelijk van de waarde. Javascript is loosely typed.
- Waarde kan veranderen!!
- Er geldt block scope. Een block wordt gedefinieerd door {} – functies – loops – conditional statements – classes - ....



# De bouwstenen : variabelen

## JavaScript is **loosely typed**!

- Je geeft **geen** type op bij declaratie van een variabele!  
Dit betekent niet dat de variabele geen type heeft. Het type is *undefined*

```
let getal; //type is undefined
```

- Toekennen van een waarde aan een variabe

```
getal = 10; //type is number
```

- Je kan ook een variabele declareren en instantiëren in 1 keer

```
let getal = 10; //type is number
```

- Het type van een variabele kan veranderen en is afhankelijk van de huidige waarde => **impliciete type conversie**

```
let getal; //type is undefined
```

```
getal = 10; //type is number
```

```
getal = 'nu ben ik plots van type string'; //type is string
```

Gebruik ' of " voor strings



# De bouwstenen : variabelen

- Enkele voorbeelden

```
let getal; //declaratie
getal; //>>undefined

getal = 200; //waarde toekennen, getal bevat waarde 200
getal; //>>200 - retourneert de waarde van de variabele getal

getal = 'nu plots een stukje tekst'; //>>"nu plots een stukje tekst"

let naam = 'John'; //declaratie en toekennen van waarde
naam; //>>"John"

let tekst = 'va' + 'kan' + 'tie' + '!!!'; //concatenatie
tekst; //>>"vakantie!!!"

//meerdere declaraties in 1 lijn
let jaar = 2019, maand = 1, dag = 14;

let getal=10; //geeft syntax error Identifier 'getal' has already been declared
```

>> : wat javascript retourneert in de console  
In bouwstenen.js wordt dit voorafgegaan door console.log!!!



# De bouwstenen : variabelen

- Samenvatting
  - JavaScript is **loosely typed**. Er wordt geen type toegekend aan variabelen. Maak gebruik van het keyword **let** om een variabele te declareren en kennen een waarde toe aan de variabele en hierdoor wordt het type toegekend.
  - Je kan een variabele declareren en een waarde geven, of eerst declareren en dan pas een waarde geven.
  - Bij decalaratie wordt het type Undefined toegekend.
  - Indien we daarna de waarde 200 toekennen is het type van de variabele Number. Daarna kennen we aan deze variabele een stuk tekst toe (“Hello World”), dan is het type van de variabele String geworden
  - **Dit betekent niet dat JavaScript geen rekening houdt met het datatype.** Het behandelt getallen anders dan strings, anders dan booleans,...



# De bouwstenen : constanten

- Een constante krijgt bij de creatie onmiddellijk een VASTE waarde die niet meer kan worden gewijzigd

```
const naam = constante waarde;
```

- Naamgeving: idem als bij variabelen.
- Je geeft **geen** type op bij declaratie van een constante.  
Het type is afhankelijk van de toegekende waarde. Er moet onmiddellijk een waarde toegekend worden
- Het toekennen van een nieuwe waarde is niet toegestaan!!!  
Muteren is wel mogelijk (object) – zie later.
- Er is block scope, zoals bij let.

```
const intrest = 0.15;  
const saldo = 2000;  
const intrestPerJaar = saldo * intrest;  
console.log(intrestPerJaar) //>> 300  
intrestPerJaar = 400; //>>throwt een exception : TypeError  
const total; //>>missing initializer in const declaration
```

>> de waarde die je in de Console te zien krijgt



# De bouwstenen : variabelen

- Men kan ook het **var** keyword (ES5) gebruiken voor de declaratie van een variabele.

```
var variabeleNaam = initiële waarde;
```

- Nadelen :
  - var kan opnieuw gedeclareerd worden binnen dezelfde scope
  - Er is **function** scope. Dwz dat de variabele binnen de functie beschikbaar is (ook voor alle blokken in de functie). Als een var niet binnen een functie gedefinieerd is dan is krijgt deze globale scope (in een web applicatie is dit het window object)
- **GEBRUIK NOOIT het "var" KEYWORD**



# De bouwstenen : variabelen

- Best practice
  - Gebruik altijd `const` indien je een vaste waarde aan de variabele wil toekennen.
  - Gebruik `let` als de variabele gewijzigd mag worden
  - Nooit `var` gebruiken, tenzij voor compatibility issues met IE11 (zie <http://kangax.github.io/compat-table/es6/>)
- Oefening
  - Gegeven: de prijs van een product is 200 euro, de korting is 10%.
  - Print de prijs na korting.





# De bouwstenen : de datatypes

- De datatypes
  - JavaScript heeft basistypes
    - “Primitieve typen” : 1 waarde
      - number
      - boolean
      - string
      - null, undefined
    - Objecttypen
      - Object
      - Array
      - Function : bevat uitvoerbare code



# De bouwstenen : de datatypes

## De typeof operator

- type detectie, retourneert het type
- type van variabele wordt afgeleid bij toekennen van een waarde

```
let x; //>>undefined
typeof x; //>>undefined
const y = 0; //>>0
typeof y; //>>number
typeof message = 'hallo'; //>>string
typeof 1; //>>number
typeof 1.0; //>>number
typeof true; //>>boolean
typeof 'hello world'; //>>string
typeof undefined; //>>undefined
typeof null; //>>object
const person = {
  name: 'Frank'
};
typeof person; //>>object
typeof console.log; //>>function
```



# De bouwstenen : undefined

- De speciale waarde undefined
  - Staat voor een onbekende waarde
    - Een variabele waaraan nog geen waarde is toegekend bevat de waarde undefined
    - Een functie zonder return instructie, retourneert undefined
  - Evalueert false in een boolean expressie
    - Gebruik : nagaan of een variabele een waarde heeft

```
let a;  
console.log(a); //>undefined  
if (a) {  
    //deze code wordt niet uitgevoerd, evalueert false  
}  
//OF  
if (a !== undefined) {  
    //deze code wordt niet uitgevoerd, evalueert false  
}
```



# De bouwstenen : null

- De waarde null
  - Staat voor een lege object pointer
  - typeof null retourneert “object”
  - Evalueert false in een boolean expressie
    - Gebruik : testen of een variabele refereert naar een object
    - Tip : bij declaratie van een variabele die later een object zal bevatten, instantieer op null.

```
const x = null;  
console.log(x); //>>null  
console.log(typeof x); //>>object  
console.log(x ? true : false); //>>false
```



# De bouwstenen : number

- Het primitieve datatype number
  - Alle getallen (zowel gehele als floating point getallen) worden bijgehouden als **64 bit floating point getallen**.
    - Er is geen apart integer type <-> Java : verschillende datatypes voor getallen
    - dit formaat is gevoelig voor afrondingsfouten
  - Integer literals
    - Decimaal : 3, 10,...
    - Hexadecimaal : begint met 0X of 0x : 0xff
    - Octal : begint met 0 : 0377
  - Floating point literals : [digits][.digits][(E|e)[(+|-)]digits]
    - 3.14 ; 2345.789 ; .3333333333333333, 6.02e23
  - Operatoren : +, -, \*, /, %



# De bouwstenen : number

- Enkele voorbeelden

```
const a = 1; // geheel getal
console.log(typeof a); //>>number

const b = 1.5; // cijfers na de komma
console.log(typeof b); //>>number

const c = 070; // integer (in octal)
const d = 0xffff; // integer (in hex)
const e = 1.34e6; // Scientific Notation (1340000)
const f = 10.0; // geheel getal (optimization)
```

# De bouwstenen : number

- Alle getallen zijn floating point getallen
  - Cijfers na de komma niet 100% juist
  - Accuraat tot 17 cijfers na de komma

Bouwstenen.js functie doNumber

```
const i = 0.1;  
const j = 0.2;  
let z = i+j;  
console.log('z = ' + z);
```

```
z = 0.30000000000000004
```

- Oplossing 1

```
z = (i * 10 + j * 10)/10;
```

```
z = 0.3
```

- Oplossing 2 (maar nu is z van het type string)

```
z = (i + j).toFixed(1);
```

```
z = 0.3
```



# De bouwstenen : number

- Speciale waarden en bijhorende functions
  - **NaN** (Not a Number)
    - Stelt geen getal voor (iets is geen getal)
    - De waarde die sommige standaardfuncties teruggeven als ze als invoer een getal verwachten en dit niet krijgen
    - **isNaN()** : functie die detecteert of een waarde NaN is
  - **Infinity**
    - stelt overflow voor
    - Een berekening die een getal retourneert buiten het waardenbereik, retourneert **Infinity** (positief) **of** **-Infinity** (negatief).
    - De functie **isFinite()** retourneert true als parameter binnen waardenbereik ligt





# De bouwstenen : number

- Enkele voorbeelden speciale waarden

```
const fail = 10 / "zero";  
console.log(fail); // >> NaN (Not a Number)  
console.log(isNaN(fail)); //>>true  
console.log(isNaN(10)); //>>false  
console.log(isNaN('10')); //>>false  
console.log(isNaN('blabla')); //>>true  
console.log(isNaN(true)); //>>false  
console.log(1.7976931348623157e+308 * 1000); //>> Infinity  
console.log(1.7976931348623157e+308 * -1000); //>>- Infinity  
console.log(10 / 0); //>>Infinity  
console.log(isFinite(123)); //>>true
```

→ Impliciete typeconversie

# De bouwstenen : number

- JavaScript functies : Conversies
  - **parseInt()**: converteert string naar geheel getal (eerste beduidende positie moet +,- of getal zijn)
  - **parseFloat()**: converteert string naar floating point
  - Beide doen dit tot ze het eerste ongeldige karakter tegenkomen
  - **De constructor Number() (type-casting)**: zet een waarde om in een getal (base-10). Bij een ongeldig karakter retourneert hij NaN

```
console.log(parseInt('1234numbers')); //>>1234
console.log(parseInt('$1234')); //>>NaN
console.log(parseInt('22.5')); //>>22
console.log(parseFloat('1234numbers')); //>>1234
console.log(parseFloat('$12.34')); //>>NaN
console.log(parseFloat('22.34.15')); //>>22.34
console.log(parseFloat('22.5')); //>>22.5
```



# De bouwstenen : Number

- Het objecttype Number
  - Prototype voor een object van type number (zie later)
  - Voegt gedrag toe aan de primitieve datatypes
  - JavaScript past impliciete conversie toe : het primitieve getal wordt automatisch omgezet naar een Number object als je een methode van Number oproept
  - Instance methodes
    - **toFixed(x)** : afronden tot x cijfers na de komma
    - **toString(x)** : string representatie, x is de radix, default 10
  - Properties
    - **Number.MIN\_VALUE** : constante voor kleinste getal ( $5e-324$ )
    - **Number.MAX\_VALUE** : grootste getal ( $-1.7976931348623157e+308$ )
    - **Number.NaN** : idem als Nan, maar Property van Number



# De bouwstenen : Number

- Het objecttype Number

```
console.log(12345.6789.toFixed(1)); //>>"12345.7"  
console.log((17).toString(2)); //>>10001  
console.log(Number.MIN_VALUE); //>>5e-324  
console.log(Number.MAX_VALUE); //>>1.7976931348623157e+308  
console.log(Number.POSITIVE_INFINITY); //>>Infinity  
console.log(Number.NEGATIVE_INFINITY); //>>-Infinity
```

# De bouwstenen : Number

- Math object : berekeningen
  - is een ingebouwd object, een JavaScript base class.
  - Bevat enkel **static** properties en methods
  - Math.round(), Math.trunc() : afronden of afkappen
  - Math.max(), Math.min() : grootste/kleinste getal
  - Math.random() : pseudo random getal tussen 0 (incl) en 1(excl)
  - Meer op [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math)

```
console.log(Math.round(200.6)); //>>201  
console.log(Math.max(200, 1000, 4)); //>>1000  
console.log(Math.min(200, 1000, 4)); //>>4  
console.log(Math.random()); //getal tussen 0 en 1
```



# De bouwstenen : Number

- Math object : berekeningen
  - **Oefening :**
    - Genereer een random waarde voor een dobbelsteen
    - Bereken de omtrek van een cirkel met straal 10.
  - Meer info op  
[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math)



# De bouwstenen : boolean

- Het primitieve datatype boolean
  - 2 boolean waarden : true of false (case sensitive!)
  - Alle waarden hebben een boolean equivalent.
    - bij testen in een conditie (if...)
    - De functie Boolean(x) : cast x naar een boolean

<b>datatype</b>	<b>converteert naar true</b>	<b>converteert naar false</b>
boolean	true	false
string	niet lege string	"" (lege string)
number	een niet 0 getal	0, NaN
object	elk object	null
undefined		undefined



# De bouwstenen : boolean

- Het primitieve datatype boolean

```
console.log(typeof (true)); //>>boolean
const b = false;
console.log(typeof (b)); //>>boolean
console.log(Boolean("hello world")); //>>true
console.log(("hello world") ? true : false); //>>true
console.log(("") ? true : false); //>>false
console.log(Boolean(100)); //>>true
console.log((0) ? true : false); //>>false
console.log((100/0) ? true : false); //>>false
const a = null;
console.log((a) ? true : false); //>>false
let c;
console.log((c) ? true : false); //>>false undefined)
```

- Ook hier bestaat het objecttype Boolean (zie later).





# De bouwstenen : string

- Het primitieve type string (een waarde!!)
  - Een sequentie van 0 of meer **Unicode** karakters vervat in ' of "
  - ' of " kunnen in elkaar vervat worden, zoniet moet je escape sequences gebruiken : \" of \'
  - Andere escape characters : \n (newline), \\(backslash),...
  - + operator : concateneren van 2 strings
  - length property : aantal karakters in string

```
const a = "some text"; //Simple Strings
const b = 'some text'; //Either delimiter
let x = a; //Takes a copy of the text
x = "blabla";
console.log(a); //>> "some text"; waarde van a blijft ongewijzigd
const c = 'first text' + 'second text'; //Immutable
console.log(c); //>> "first textsecond text"
console.log(c[3]); //>>s
console.log(b.length); //>>9
```



# De bouwstenen : string

- Omzetting naar een string:
  - toString()

```
const getal = 12;
const b = true;
console.log(getal.toString()); //>> 12;
console.log(b.toString()); //>> true;
console.log(null.toString()); //geeft een exception. >>Uncaught TypeError:
Cannot read property 'toString' of null
```

- String() : Een type-casting met String() roept in feite toString() aan. Eén verschil is dat String() ook een null of undefined kan omzetten zonder fout te genereren.

```
const getal = 12;
console.log(String(getal)); //>> "12";
console.log(String(null)); //>> "null"
```



# De bouwstenen : String

- Het objecttype String : prototype object met heel wat methodes.

Stel `w="Javascript"`

methode	betekenis
<code>w.length</code>	de <b>lengte van de string w</b> , dus 10
<code>w.charAt(7)</code>	geeft het <b>zevende teken van de string</b> , dus r
<code>z = w.concat(p)</code>	Zet de strings <b>w en p achter elkaar</b> , en z wordt die gecombineerde string Dit kan ook m.b.v. de opdracht <code>z = w + p;</code>
<code>w.indexOf("s")</code>	geeft het <b>nummer van de eerste s in de string</b> (ze beginnen te tellen bij 0!), dus 4 Als het teken niet voorkomt wordt de waarde -1. Dus bijvoorbeeld <code>w.indexOf("q") = -1</code>
<code>w.indexOf("a",2)</code>	geeft het <b>nummer van de eerste a gerekend vanaf het derde teken in de string</b> (ze beginnen te tellen bij 0!), dus 3
<code>w.substr(3)</code>	levert een substring, vanaf teken nr. 3 tot het eind (ze beginnen te tellen bij 0!), dus "ascript"
<code>w.substr(3, 4)</code>	levert een substring van 4 tekens, vanaf teken nr. 3 (ze beginnen te tellen bij 0!), dus "ascr"
<code>w.toLowerCase()</code>	levert dezelfde string, maar alles met kleine letters
<code>w.toUpperCase()</code>	levert dezelfde string, maar alles met hoofdletters, dus w = "JAVASCRIPT"

# De bouwstenen : String

- Het objecttype String : prototype object (zie later)

```
const anyString = 'Brave new world';
console.log(anyString.charAt(0)); //>>B
const arrayOfStrings = anyString.split(' ');
console.log(arrayOfStrings); //>>['Brave', 'new', 'world']
console.log(anyString.substr(1, 2)); //>>ra
console.log(anyString.indexOf('re')); //>>-1
console.log(anyString.indexOf('new')); //6
console.log(anyString.includes('new')); //true
```

- Nog veel meer methodes op:  
[https://developer.mozilla.org/en/docs/JavaScript/Reference/Global\\_Objects/String](https://developer.mozilla.org/en/docs/JavaScript/Reference/Global_Objects/String)
- Oefening
  - Ga na of anyString start met het word Brave
  - Herhaal anyString 2 maal



# De bouwstenen : String

- Concatenatie

– +

```
const a = 5;
const b = 10;
console.log('Fifteen is ' + (a + b) + ' and\nnot ' + (2 * a + b) + '.');
// "Fifteen is 15 and
// not 20."
```

– **Template literals:** strings die variabelen/expressies bevatten.

- omsloten door backticks ` ipv ' of "
- Multiline strings (zonder \n) zijn mogelijk
- Expressies : \${expressie}

```
const a = 5;
const b = 10;
console.log(`Fifteen is ${a + b} and
not ${2 * a + b} and not ${b}.`);
// "Fifteen is 15 and
// not 20."
```

ES6



# De bouwstenen : Date

- Houdt datums bij in milliseconden sinds 1/1/1970 UTC
- Kan datums bijhouden 285.616 jaren ervoor en erna
- Creatie datum :
  - `const date = new Date();` //bevat de huidige datum/tijd
  - `const date = new Date(1954,11,14,5,34,0,0);` jaar (4pos), maand (begint vanaf 0, dus waarde tussen 0 en 11), dag, uren, minuten, sec, msec
- Methodes
  - `getDate` : retournt **dag** van de maand
  - `getMonth()` : start vanaf 0!!!!!!! (0-11)
  - `getFullYear()`
  - `getHours()` (0-23), `getMinutes()` (0-59), `getSeconds()` (0-59)
  - `getDay` : dag in de week (0-6)



# Het Date type

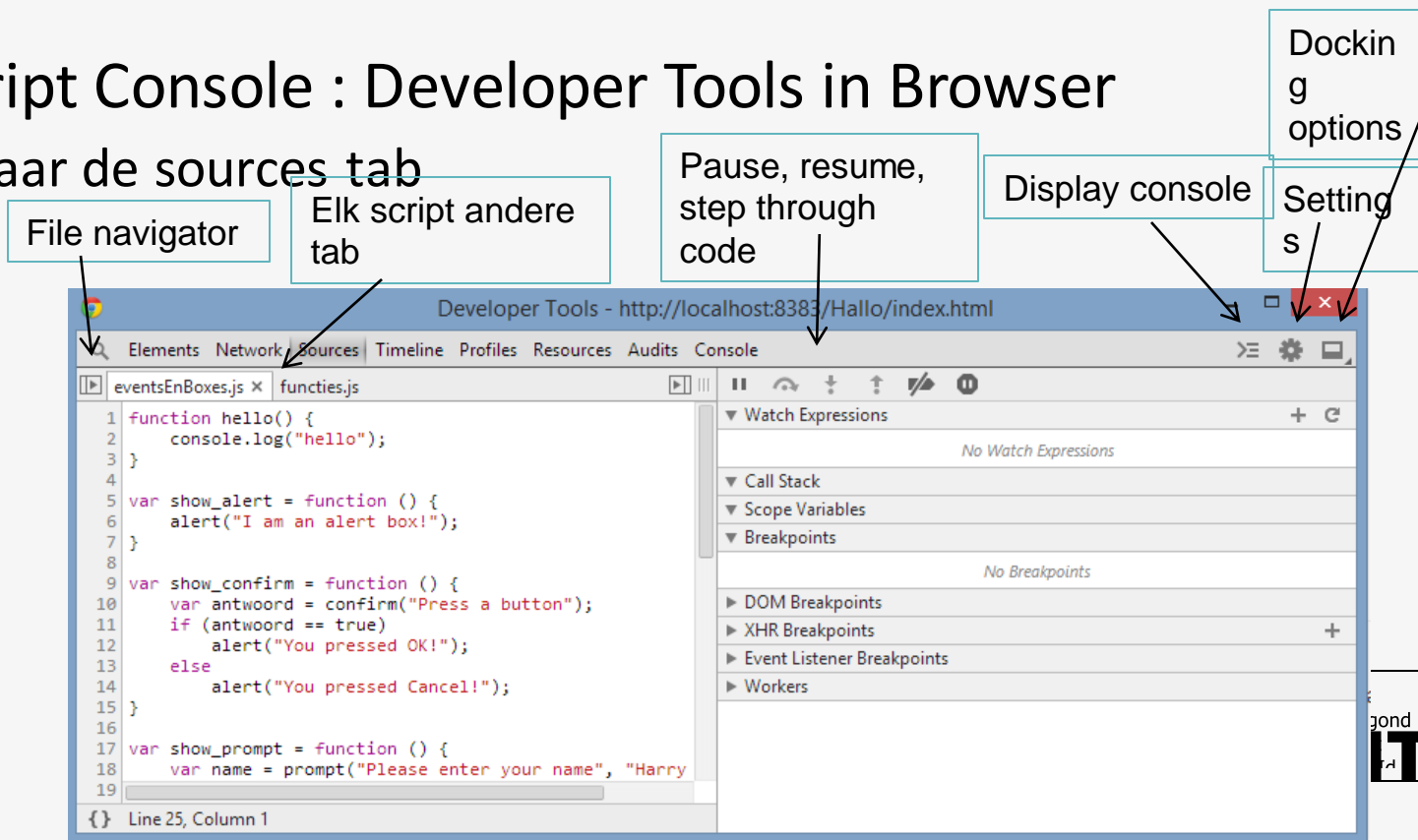
```
const today = new Date();
console.log(today);
console.log(today.getFullYear());
console.log(today.getMonth()); //(start vanaf 0!!!)
console.log(today.getDate());
console.log(today.getHours());
console.log(today.getMinutes());
console.log(today.getSeconds());
console.log(today.getDay());
```

```
> Tue Jan 23 2018 16:32:35 GMT+0100 (Romance Standard Time) {}
2018
0
23
16
32
35
2
```



# JavaScript debuggen

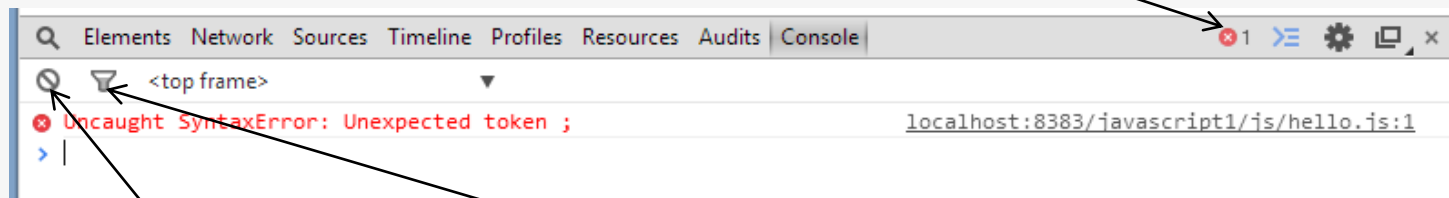
- JavaScript Console : Developer Tools in Browser
  - Ga naar de sources tab





# JavaScript debuggen

- Debugging via de JavaScript console in Chrome
  - Maak een foutje in JavaScript code in hello.js  
`console.log("Hello");`
  - F12: open de Developer Tools
  - Bovenaan zie je een aanduiding van het aantal fouten
  - Ga naar Console. Dit toont de fouten. Klik op de lijn met de fout, brengt je naar de code

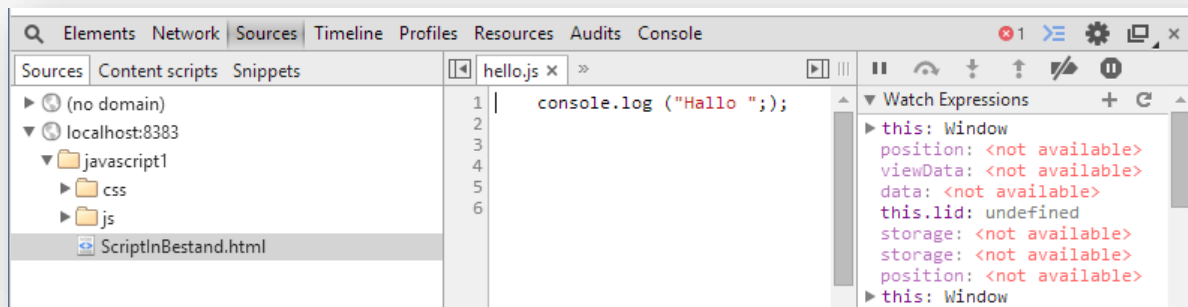


Clear de console

Filter op type fout

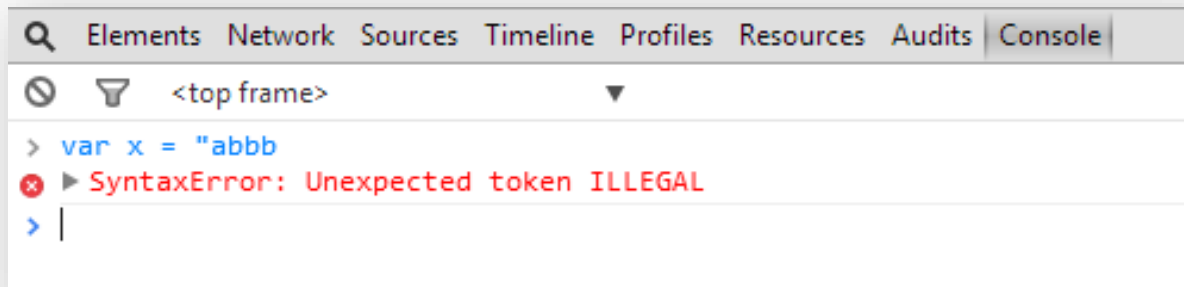
# JavaScript debuggen

- Debugging via de JavaScript console in Chrome
  - Klikken op de fout brengt je naar de source code. Tab Sources in de Developer tools



# JavaScript debuggen

- Je kan ook code ingeven in de Console
  - Typ de code in en druk enter



- Je kan je logs, errors, info schrijven naar de Console vanuit JavaScript code
  - `console.log()`, `console.error()`, `console.info()`

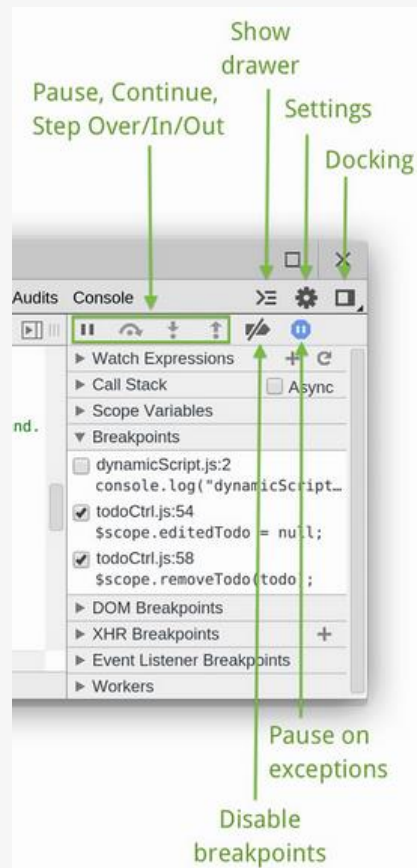
# JavaScript debuggen

- Als JavaScript in een apart bestand zit, kan je de JavaScript debuggen. Open debugging.html in browser
  - Ga naar de Sources tab
  - In de navigator, selecteer Debugging.js in de js folder
  - Nu kan je breakpoints plaatsen : klik in de linkermarge => blauwe marker (Rechtsklik marker > menu waar je breakpoint kan verwijderen,...)
  - Klik op hyperlink start debugging : de uitvoering stopt juist voor uitvoering van die lijn. Die lijn wordt in blauw weergegeven



# JavaScript debuggen

- Knoppen rechts worden actief
  - Continue(F8) - Het script vervolgen tot volgend breakpoint
  - Step over (F10) - De opdracht lijn per lijn uitvoeren (stapt niet in code van een functie)
  - Step in (F11) – Idem als step over, maar bij een functie call wordt naar de eerste lijn code van deze functie gesprongen
  - Step out (Shift F11)- enkel binnen een functie. Voert de rest van de functie uit en springt naar parent functie.
  - Disable breakpoints
  - Pause on exceptions : alle of enkel de niet opgevangen exceptions



# JavaScript debuggen

- Rechts krijg je ook deelvensters
  - Call stack : volgorde van functie aanroepen door script. Je kan async aanvinken als je ook de asynchrone callbacks wenst te zien (zie later ajax)
  - Scope Variables : de variabelen in dit deel van het script. Je kan hier zelf variabelen toevoegen
  - Breakpoints : de aanwezige breakpoints
- Druk Esc (of icoon show daver) om de Console in apart tabblad te openen
- Blackbox JavaScript files : als je een bestand niet wenst te debuggen : rechtsklik bestand > blackbox script (of via Settings)
- Meer op : <https://developer.chrome.com/devtools/docs/JavaScript-debugging>



# De bouwstenen : Heb je commentaar?

- `//` 1 regel commentaar
- `/* ....*/` meerdere regels
  - Opmerking : Vaak voor de leesbaarheid

```
/*  
 * vervolg commentaar  
 * vervolg commentaar  
 */
```

# Instructies

- Blocks
- if
- while/do..while
- for
- break/continue
- <https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Instructies>





# Instructies

- Code blokken

.js

```
{  
  // code  
}
```



# Instructies

- if : als ... dan ...
  - Enkel if .. then

```
if (scoops < 3) {  
    console.log('Ice cream is running low!');  
}
```

.js

I.g.v. 1 instructie  
mag {}  
weggelaten  
worden.

- If .. then .. else

```
if (scoops < 3) {  
    console.log('Ice cream is running low!');  
} else if (scoops > 9) {  
    console.log('Eat faster, the ice cream is going to melt!');  
}
```

.js



# Instructies

- if : als ... dan ...
  - Nog meer keuzes ....

```
if (scoops == 3) {  
  console.log('Ice cream is running low!');  
} else if (scoops > 9) {  
  console.log('Eat faster, the ice cream is going to melt!');  
} else if (scoops == 2) {  
  console.log('Going once!');  
} else if (scoops == 1) {  
  console.log('Going twice!');  
} else if (scoops == 0) {  
  console.log('Gone!');  
} else {  
  console.log('Still lots of ice cream left, come and get it.');
```

.js



# Instructies

- switch
  - Meerdere keuzes op een gestructureerde manier

```
function doSwitch() {  
  const d = new Date();  
  const theDay = d.getDay();  
  switch (theDay)  
  {  
    case 5:  
      console.log('Finally Friday');  
      break;  
    case 6:  
      console.log('Super Saturday');  
      break;  
    case 0:  
      console.log('Sleepy Sunday');  
      break;  
    default:  
      console.log("I'm really looking forward to this weekend!");  
  }  
}
```

*Yeah...Weekend*



# Instructions

- while / do .. while

.js

```
function dowhile() {
  let scoops = 10;
  while (scoops > 0) {
    console.log('More icecream!');
    scoops--;
  }
  console.log("life without ice cream isn't the same");

  do {
    console.log('More icecream!');
    scoops++;
  } while (scoops < 10);
  console.log(':)');
}
```

[illegible]

# Instructies

- for

.js

```
function doFor() {  
  for (let berries = 5; berries > 0; berries--) {  
    console.log('Eating a berry');  
  }  
  console.log('No more berries left');  
}
```

```
Eating a berry  
Eating a berry  
Eating a berry  
Eating a berry  
Eating a berry  
No more berries left
```



# Instructies

- Herhalingen onderbreken
  - break;  
de volledige herhaling wordt gestopt – voorwaarde wordt niet getest.
  - continue;  
stopt de huidige herhaling, test de voorwaarde en voert eventueel de herhaling verder uit.

# Instructies

- Herhalingen onderbreken

.js

```
function doJumpingOut() {  
    //break : jumping out  
    let k = 0;  
    while (true) {  
        k++;  
        if (k > 5)  
            break;  
        console.log(`Waarde voor k : ${k}`);  
    }  
  
    //skipping with continue  
    let l = -3  
    while (l < 3) {  
        l++;  
        if (l < 0)  
            continue;  
        console.log(`Waarde voor l : ${l}`);  
    }  
}
```

```
Waarde voor k : 1  
Waarde voor k : 2  
Waarde voor k : 3  
Waarde voor k : 4  
Waarde voor k : 5
```

```
Waarde voor l : 0  
Waarde voor l : 1  
Waarde voor l : 2  
Waarde voor l : 3
```





# Operatoren

- **Berekeningsoperatoren :**
  - +, -, \*, /, %, ++, --, unary -, unary +
- **Toewijzingsoperatoren :**
  - =, \*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=
- **Vergelijkingsoperatoren :**
  - ==, !=, ===, !==, >, >=, <, <=
- **Logische operatoren :**
  - &&, ||, !
- **String operatoren :**
  - + en +=
- **Conditionele operatoren :** *condition ? ifTrue : ifFalse*
- Meer op <https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Operators>



# Operatoren

- Vergelijkingsoperatoren

- ==

- !=

- <

- >

- <=

- >=

- ▣ Logische operatoren

- && → AND

- || → OF

- ! → NOT

```
function doOperators() {  
    const str = 'ZoekHetWoordVakantie';  
    const zoek1 = 'Examens';  
    const zoek2 = 'Vakantie';  
  
    if (str.indexOf(zoek1) === -1 && str.indexOf(zoek2) === -1) {  
        console.log(`${zoek1} en ${zoek2} komen niet voor in ${str}`);  
    } else {  
        console.log('gevonden');  
    }  
}
```

.js



# Operatoren

- Gelijkheid

- 2 equality operatoren

- == of !=

- Impliciete type conversie (controleert type niet)

```
'hello' == 'hello'; // true  
1 == 1; // true  
1 == '1'; // true  
1 == true; //true
```

- === of !==.

- Komt overeen met de Equals methode in Java
      - Controleert wel het type, geen impliciete type conversie
      - Best Practice : gebruik ===.

```
1 === '1'; // false  
1 !== '1'; // true  
1 === 1.0000000000000001; // false  
1 === 1.0000000000000001; // true
```

# Functies

- efficiëntere code
- herbruikbaar
- elimineert dubbele code
- grote problemen oplossen door kleinere problemen op te lossen.
- grote problemen opdelen in kleinere eenheden.
- Er bestaan built-in functies zoals `alert`, `Math.round()`,...



# Functies

- Definitie

```
function functionname(par1,par2,...,parX) {  
    statements  
};
```

Merk op : ook hier geen datatype voor parameters en geen returntype!

- Uitvoeren

```
functionname(arg1, arg2,...,argx);
```

- Een functie kan een waarde retourneren

```
function getAvatar(points) {  
    let avatar;  
    if (points < 100) {  
        avatar = 'Mouse';  
    } else if (points < 1000) {  
        avatar = 'Cat';  
    } else {  
        avatar = 'Gorilla';  
    }  
    return avatar;  
}  
const myAvatar = getAvatar(335);  
console.log(`my avatar : ${myAvatar}`); //Cat
```



# Hoisting

- “Hoisting is a JavaScript mechanism where variables and function declarations are moved to the top of their scope before code execution.”
- For let and const : The variables are created when their containing Lexical Environment is instantiated but **may not be accessed in any way until the variable’s Lexical Binding is evaluated.**



# Hoisting

- **Hoisting:** Functie en variabele declaraties worden eerst gecompileerd alvorens de browser de script code uitvoert. Na de compilatie fase zijn de variabelen en functies gekend
  - Function declarations

```
sayHi('Bob'); //alert: Hi, my name is Bob

function sayHi(name) {
  alert(`Hi, my name is ${name}`);
}

sayHi('Bob'); //alert: Hi, my name is Bob
```

Een functie op dergelijke manier gedeclareerd, kan overal in de code gebruikt worden, zelfs vóór zijn declaratie!!

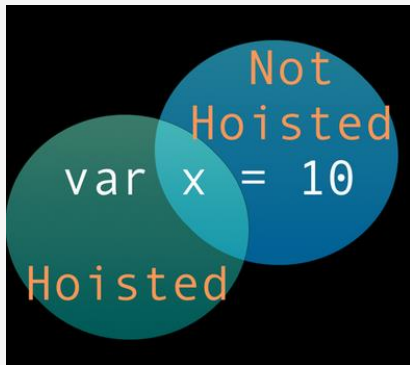


# Hoisting

- var :

```
console.log(x); //>>undefined  
x=10;  
var x;  
console.log(x); //>>10
```

- JavaScript zal enkel declaraties hoisten, geen initialisatie waarden
  - Reden : compilatie proces juist voor interpreter proces. Compilatie proces detecteert alle declaraties en bijhorende scope.





# Hoisting

- Let en const : Hoisting gebeurt ook voor let en const, maar je mag er maar naar refereren als de “actual declaration” geëvalueerd is at runtime (ES6 specificatie) = de Temporal Dead Zone

```
x = 10; //>>ReferenceError  
let x;  
console.log(x); //>>10
```

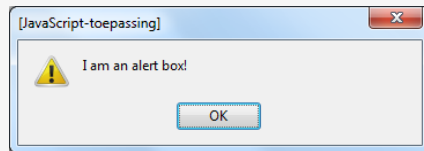
```
22 function doHoisting() {  
23   x= 10; //>>ReferenceError  
      
Exception has occurred: ReferenceError  
ReferenceError: x is not defined  
    at doHoisting (c:\temp\javascript1Voorbeelden\js\functies.js:23:6)  
    at Object.<anonymous> (c:\temp\javascript1Voorbeelden\js\functies.js:2:1)  
    at Module._compile (module.js:638:14)  
    at Object.Module._extensions..js (module.js:652:10)  
    at Module.load (module.js:560:32)  
    at tryModuleLoad (module.js:503:12)  
    at Function.Module._load (module.js:495:3)  
    at Function.Module.runMain (module.js:682:10)  
    at startup (bootstrap_node.js:191:16)  
    at bootstrap_node.js:613:3
```



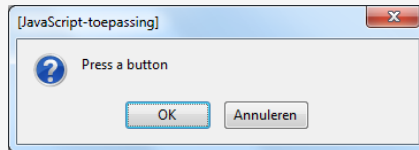
# Funcities

- Gebruik van bestaande functies
  - Bvb: JavaScript Popup Boxes – eventsEnBoxes.js

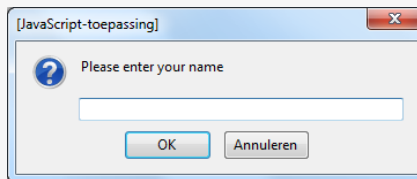
- Alert box: `alert("sometext");`



- Confirm box: `confirm("sometext");`



- Prompt box: `prompt("sometext","defaultvalue");` Als geen defaultValue voorzien wordt null geretourneerd



# Functions

```
function doAlert () {  
    alert('I am an alert box!');  
}  
  
function doConfirm () {  
    const antwoord = confirm('Press a button');  
    if (antwoord === true)  
        alert('You pressed OK!');  
    else  
        alert('You pressed Cancel!');  
}  
  
function doPrompt () {  
    const name = prompt('Please enter your name', 'Harry Potter');  
    if (name !== null && name !== "") {  
        alert(`Hello ${name}! How are you today?`);  
    }  
}
```

# Arrays

- Is een geordende verzameling van elementen. De **elementen** mogen van een **verschillend type** zijn
- Elk element heeft een genummerde positie in de array, **index** genaamd. Het eerste element heeft index 0
- Arrays zijn **dynamisch**! Je geeft geen grootte op bij instantiatie. Ze kunnen groeien/krimpen
- Een array wordt in JavaScript genoteerd met []



# Arrays

- De Ninja Pizzeria maakt pizza's en stopt ze dan in dozen, klaar om te leveren.
  - De gestapelde lege pizzadozen



- Een array is een stack van lege dozen waar je waarden kan in stoppen.



# Arrays

- In JavaScript definieer je een **lege array** als volgt

```
let pizzas = [];
```

```
OF let pizzas = new Array();
```

- Arrays zijn **dynamisch** (je dient het aantal elementen in de array niet op voorhand te definiëren! Een array kan groeien en krimpen!)
- Je dient ook het type van de elementen in de array niet op te geven. De elementen zijn mogelijks van een verschillende type



# Arrays

- Plaatsen van waarden in een array
  - mogelijks van verschillende types

```
pizzas[0] = 'Margherita';  
pizzas[1] = 'Mushroom';  
pizzas[2] = 'Spinach & Rocket';
```



- Je kan dit ook eenvoudig wijzigen

```
pizzas[0] = 'Ham & Pineapple';
```

# Arrays

- Aanmaken van array via **array literal**

```
let pizzas =  
  ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];
```

- De elementen in een array hoeven helemaal niet van hetzelfde type te zijn

```
let mixedArray = [null, 1, 'two', true, undefined ];
```



# Arrays

- Opvragen 1 element adhv de index.
  - Index start vanaf 0. Gebruik []
  - length : aantal elementen in array

```
const pizzas =  
  ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];  
  
console.log(pizzas[2]); //Spinach & Rocket  
  
console.log(`eerste pizza ${pizzas[0]}`); //Margherita  
  
console.log(`laatste pizza ${pizzas[pizzas.length-1]}`); //Pineapple & Sweetcorn
```

- Je kan ook de volledige array afprinten

```
console.log(pizzas); // Array(4) ['Margherita', 'Mushroom', 'Spinach & Rocket',  
  'Pineapple & Sweetcorn'];
```



# Arrays

- Verwijderen van een element in de array
  - Verwijdert de waarde op deze positie, maar de ruimte bestaat nog steeds en bevat dan de waarde undefined.

```
const pizzas =  
  ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];  
  
delete pizzas[2];  
  
console.log(pizzas); // ['Margherita', 'Mushroom', undefined, 'Pineapple & Sweetcorn'];
```



# Arrays

- Overlopen van een array
  - For-loop

```
const pizzas =  
  ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];  
  
for (let i = 0; i < pizzas.length; i++) {  
  console.log(pizzas[i]);  
}
```

Margherita
Mushroom
Spinach & Rocket
Pineapple & Sweetcorn

- For-of loop:

```
const pizzas =  
  ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];  
  
for (let value of pizzas) {  
  console.log(value);  
}
```

ES6



# Arrays

- Overlopen van een array
  - Merk op :

```
const pizzas =  
  ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];  
pizzas[30] = 'Vegetarian';  
console.log(pizzas.length); // 31  
console.log(pizzas[20]); //undefined
```

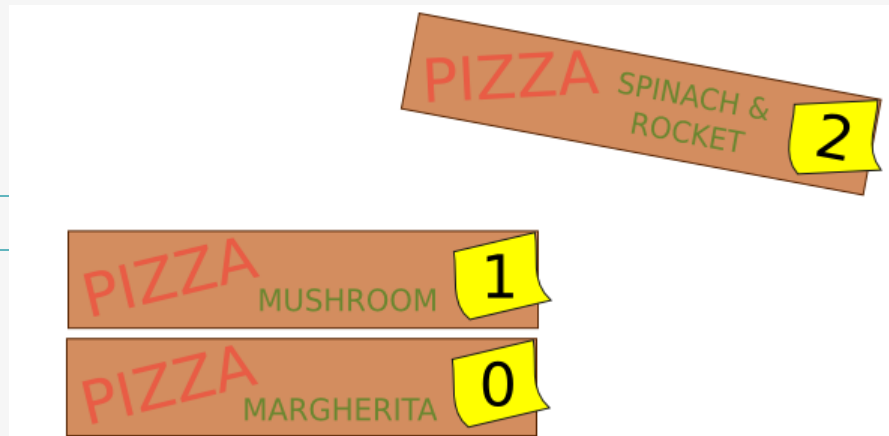
# Arrays

- Pop, push, shift en unshift
  - Gegeven

```
const pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket'];
```

- **pop** : verwijdt het laatste element uit array en retournt deze

```
pizzas.pop(); // << 'Spinach & Rocket'
```



# Arrays

- Pop, push, shift en unshift
  - **push** : voegt één of meerdere waarden toe aan het einde van de array en retournt de nieuwe lengte van de array.

```
pizzas.push('Pepperoni'); // << 3
```



# Arrays

- Pop, push, shift en unshift
  - **shift** : verwijdert de eerste waarde in array en retournt deze

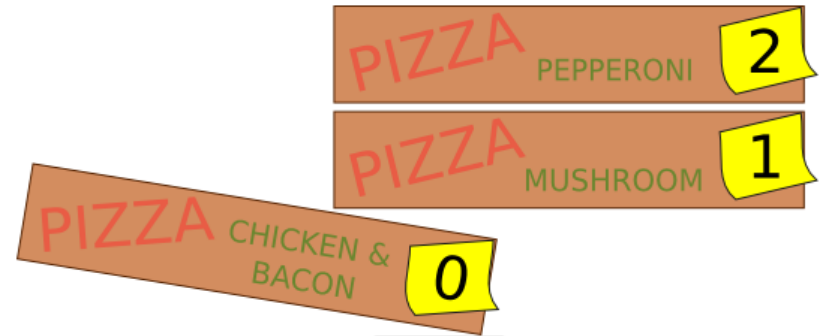
```
pizzas.shift(); //<< 'Margherita'
```



# Arrays

- Pop, push, shift en unshift
  - **unshift** : voegt één of meerdere waarden toe aan het begin van het array en retournt de nieuwe lengte van de array.

```
pizzas.unshift('Chicken & Bacon'); //<< 3
```





# Arrays

- Zoeken of een waarde voorkomt in een array
  - **indexOf** : retourneert index van eerste voorkomen of -1 als waarde niet voorkomt.

```
const pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket'];  
pizzas.indexOf('Spicy Beef'); //<< -1  
pizzas.indexOf('Margherita'); //<< 0
```

# Arrays

- Array methods
  - **concat()** : voegt 2 arrays samen
  - **reverse()** : keert de volgorde van de array elementen om
  - **slice(start\_index, upto\_index)** : retourneert een nieuw array als een stuk van de oorspronkelijke array met als argumenten de begin- en een eindpositie.
  - **splice(start\_index, numberOfItemsToRemove, waarde1,..., waardex)** : verwijdert numberOfItemsToRemove waarden uit de array startend op positie start\_index en voegt dan de nieuwe waarden toe waarde1,... waardex
  - **sort()** : sorteert de elementen in de array
  - **indexOf(searchElement[, fromIndex])** : de index van het eerste voorkomen van het element vanaf fromIndex
  - **lastIndexOf(searchElement[, fromIndex])** : idem indexOf maar begint achteraan
  - **join()** : converteert alle elementen van een array tot 1 lange string
  - Meer op [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Predefined\\_Core\\_Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Predefined_Core_Objects)



# Arrays

```
> var pizzas = ["Chicken & Bacon", "Mushroom", "Pepperoni"];  
pizzas = pizzas.concat(["Spicy Beef", "Chicken and Mushroom"]);  
< ["Chicken & Bacon", "Mushroom", "Pepperoni", "Spicy Beef", "Chicken and Mushroom"]  
> pizzas.join();  
< "Chicken & Bacon,Mushroom,Pepperoni,Spicy Beef,Chicken and Mushroom"  
> pizzas.slice(2,4) ;  
< ["Pepperoni", "Spicy Beef"]  
> pizzas.splice(2, 1, "Chicken and Pepper", "Veggie Deluxe") ;  
< ["Pepperoni"]  
> console.log(pizzas);  
["Chicken & Bacon", "Mushroom", "Chicken and Pepper", "Veggie Deluxe", "Spicy Beef", "Chicken and Mushroom"]  
< undefined  
> pizzas.reverse();  
< ["Chicken and Mushroom", "Spicy Beef", "Veggie Deluxe", "Chicken and Pepper", "Mushroom", "Chicken & Bacon"]  
> pizzas.sort();  
< ["Chicken & Bacon", "Chicken and Mushroom", "Chicken and Pepper", "Mushroom", "Spicy Beef", "Veggie Deluxe"]
```

# Arrays

- Destructuring

- Is een manier om meerdere waarden te extraheren uit een array en toe te kennen aan variabelen

```
//Variabele declaraties
//ophalen van eerste en tweede item uit een array
pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Chicken & Bacon'];
const [eerstePizza, tweedePizza] = pizzas;
console.log(eerstePizza); // Margherita
console.log(tweedePizza); // Mushroom

//ophalen van derde item uit een array
const [, , derdePizza] = pizzas;
console.log(derdePizza); //Spinach & Rocket

//gebruik maken van de rest items: gebruiken de rest operator (...)
const [x, y, ...restpizzas] = pizzas;
console.log(eerstePizza); // Margherita
console.log(tweedePizza); // Mushroom
console.log(restpizzas); //Array(2) ["Spinach & Rocket", "Chicken & Bacon"]
```



# Arrays

- Destructuring

```
//cloning an array
pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket'];
const [ ...clonedPizzas ] = pizzas;
console.log(clonedPizzas); //['Margherita', 'Mushroom', 'Spinach & Rocket']
```

```
//Destructuring Assignment
pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket'];
let pizza1, pizza2;
[pizza1, pizza2] = pizzas;
console.log(pizza1); // Margherita
console.log(pizza2); // Mushroom
```

```
//default values
pizzas = ['Margherita'];
[pizza1, pizza2 = 'Mushrooms' ] = pizzas;
console.log(pizza1); // Margherita
console.log(pizza2); // Mushrooms
```



# Arrays

- Destructuring

```
// Swapping variables in ECMAScript 5
```

```
let a = 1, b = 2, tmp;  
tmp = a;  
a = b;  
b = tmp;  
console.log(a); // 2  
console.log(b); // 1
```

```
// Swapping variables in ECMAScript 6
```

```
a = 1;  
b = 2;  
[ a, b ] = [ b, a ];  
console.log(a); // 2  
console.log(b); // 1
```

```
//push/unshift an element
```

```
pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket'];  
pizzas = [...pizzas, 'Mushrooms'];  
pizzas = ['Chicken & Bacon', ...pizzas];  
console.log(pizzas);
```



# Arrays

- Meerdimensionele arrays
  - Een array van een array  
= 2 dimensionale array.
  - Een array van een array van een array  
= 3 dimensionale array.
- Voorbeeld : een array die de kaarten van 2 poker hands bevat

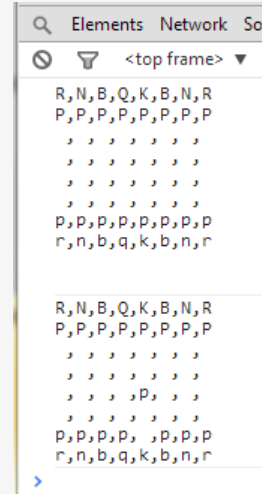
```
const hands = [];  
hands[0] = [5, 'A', 3, 'J', 3];  
hands[1] = [7, 'K', 3, 'J', 3];  
console.log ('2de kaart, 2de hand : ' + hands[1][1]);
```



# De bouwstenen: Arrays

- Voorbeeld 2D array
  - Een schaakbord, voorgesteld als een 2 dimensionele array van strings. De eerste zet verplaatst 'p' van positie (6,4) naar (4,4). 6,4 wordt op blanco geplaatst

```
function doArray() {  
  const board = [  
    ['R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R'],  
    ['P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'],  
    [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
    [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
    [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
    [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],  
    ['p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'],  
    ['r', 'n', 'b', 'q', 'k', 'b', 'n', 'r']];  
  
  console.log(board.join('\n') + '\n\n');  
  board[4][4] = board[6][4]; // Move King's Pawn forward 2  
  board[6][4] = ' ';  
  console.log(board.join('\n'));  
}
```





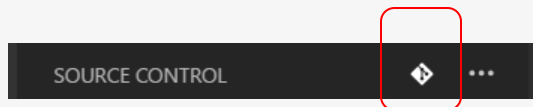
# Github

- VS Code integreert met github.

- Klik source control icon



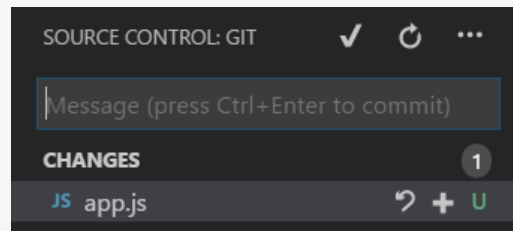
- Klik op Initialise repository



- Selecteer de folder (bvb folder hello) en klik initialize repository

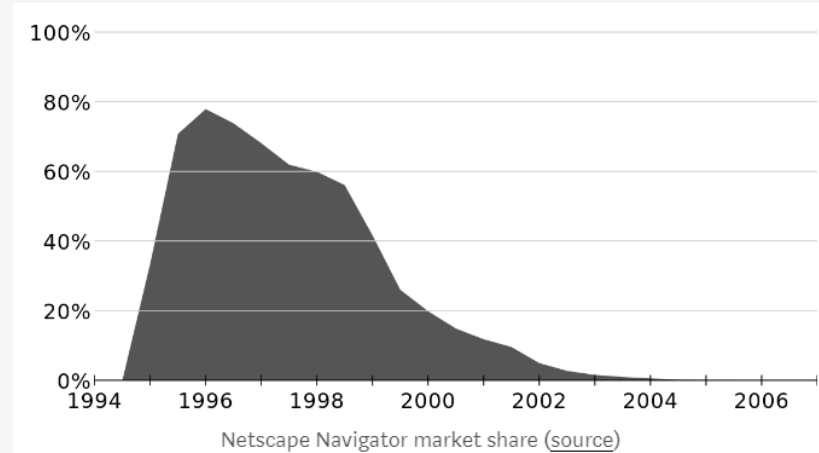
- De locale git repro wordt gecreëerd

- App.js is U(pdated).
    - Geef een commit boodschap in en druk Ctrl+Enter to commit



# Geschiedenis

- JavaScript werd in 1995 gecreëerd door Brendan Eich tijdens zijn tijd bij Netscape Communications. Het was geïnspireerd op Java, Scheme en Self.
- Netscape had in die tijd de beste browser ter wereld en was toen de meest gebruikte webbrowser.
- Eind 1995, startte Microsoft het Internet Explorer-project.



# Geschiedenis

- Door dit te doen, werd Microsoft een dodelijke dreiging en dwong het Netscape om te reageren. Ten eerste startten ze een standaardisatieproces om te voorkomen dat Microsoft de controle over de JavaScript-taal zou krijgen. Ten tweede gingen ze een partnerschap aan met Sun vanuit hun gedeelde interesse in het verbreken van het Microsoft-monopolie.
- Sun begon in 1990 met de ontwikkeling van Java in een poging om een taal te schrijven voor 'slimme apparaten'. Deze aanpak mislukte en in 1994 zette Sun zijn zinnen op het web.



# Geschiedenis

- Het Netscape / Sun-partnerschap betekende dus dat Sun het gebruik van een concurrerende browser heeft overgenomen.
- Netscape daarentegen vond een krachtige bondgenoot tegen Microsoft. Ze wilden ook Microsoft wegmanoeuvreren door de officiële browser te zijn van het toen langverwachte platform dat Java was.
- Brendan Eich heeft gezegd dat ze door met Sun in zee te gaan, besloten om mee te surfen op de toenmalige hype rond Java.



# Geschiedenis

- De naam was eigenlijk niets meer dan een eenvoudige marketingtruc om JavaScript gemakkelijker ingang te laten vinden.
- Het doel van Mocha van Netscape (later JavaScript) was om het web te transformeren tot een volledig toepassingsplatform. Dit was exact wat Sun wou doen met behulp van Java Applets. Sun zag (waarschijnlijk ten onrechte) de taal nooit als concurrent en behield het bondgenootschap.



# Geschiedenis

- Netscape heeft in een poging om JavaScript als een officiële norm erkend te krijgen, de taal laten goedkeuren door de European Computer Manufacturers Association (ECMA), waarbij de naam om redenen betreffende het merkrecht is gewijzigd in ECMAScript. Standaard is gedocumenteerd ECMA-262 specification <http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf>



# ECMAScript

- Geschiedenis ECMAScript
  - ES1 (1997): compatibel met Javascript 1.3
  - ES2(1998)
  - ES3 (2001): eerste versie ondersteund in alle browsers
  - ES4 is er nooit doorgekomen
  - ES5 (2009, later ES5.1 in 2011): ondersteund in alle browsers

