

**HO
GENT**



Web Development II

Hoofdstuk 01: JavaScript, de basis

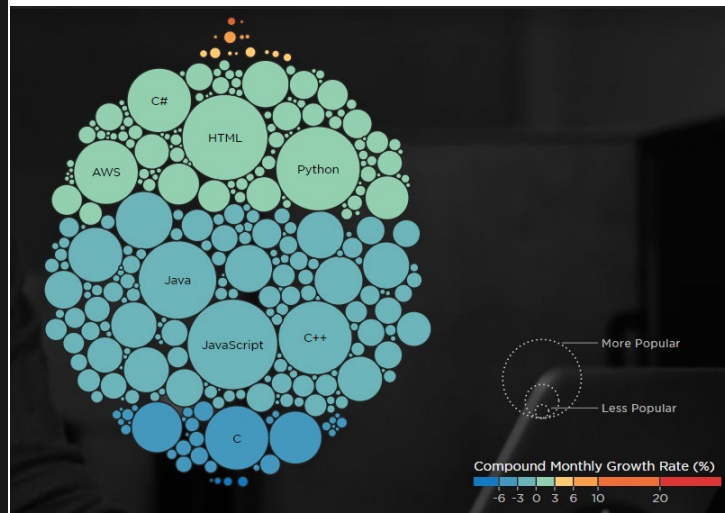
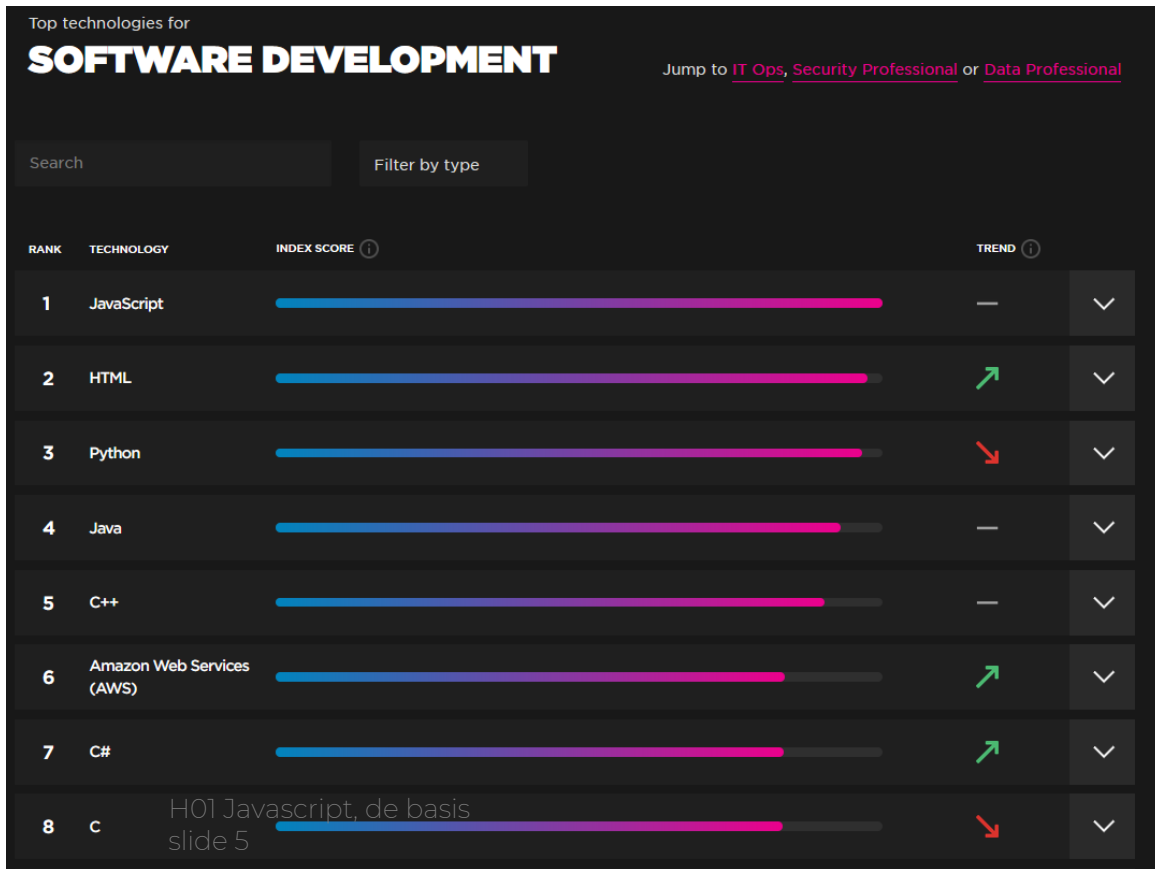
Inhoud

- Situering
- Aan de slag met JS
- Code structuur
- Bouwstenen
 - variabelen
 - datatypes
 - wrapper objecten
 - Date & Math
- Debuggen van JS code
- Controlestructuren & operatoren
- Functies
- Arrays

01 JavaScript, de basis

Situering

JavaScript & software development



**HO
GENT**

bron: <https://www.pluralsight.com/tech-index/software-development>

JavaScript - JS

JavaScript® (often shortened to JS) is a lightweight, interpreted, object-oriented language with first-class functions, and is best known as the scripting language for Web pages, but it's used in many non-browser environments as well.

- oorspronkelijk ontwikkeld als web scripting taal om 'leven te brengen' in web pagina's
- geëvolueerd naar volwaardige general purpose programmeertaal
- JS is gestandaardiseerd in de ECMA – 262 specificatie

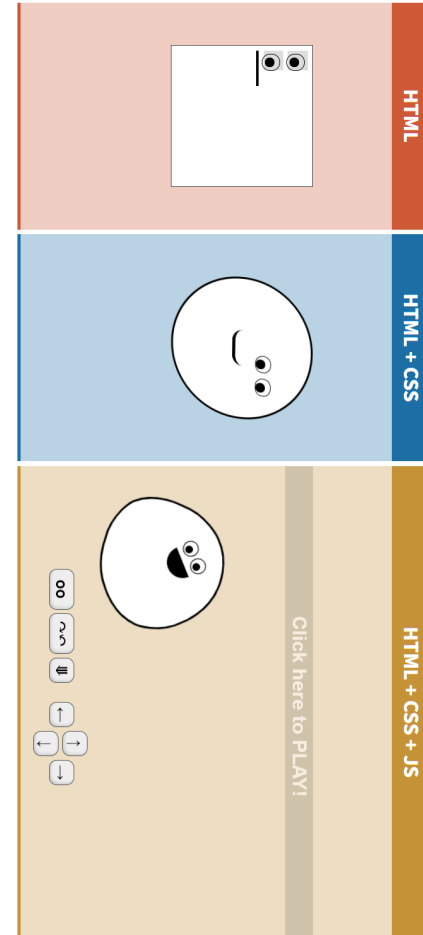
ECMAScript

ECMAScript is an object-oriented programming language for performing computations and manipulating computational objects within a host environment

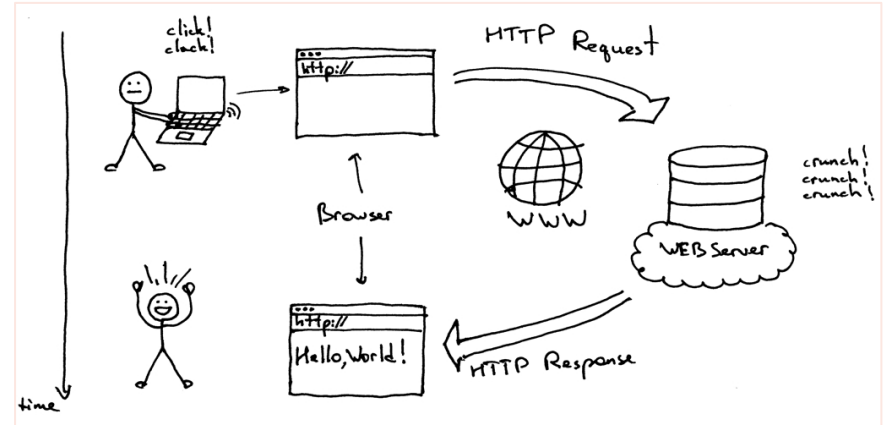
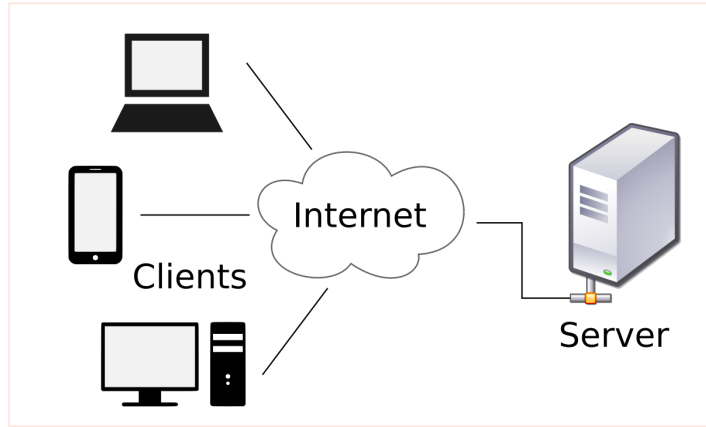
- ECMAScript 2020 Language Specification
 - <https://262.ecma-international.org/11.0/>
- deze standaard evolueert jaar per jaar
 - huidige versie: ECMAScript 2020 Language Specification
 - work in progress: ECMAScript 2021 Language Specification
- wil je weten welke features van een bepaalde versie je browser ondersteunt?
 - <https://kangax.github.io/compat-table/es2016plus/>

HTML – CSS - JavaScript

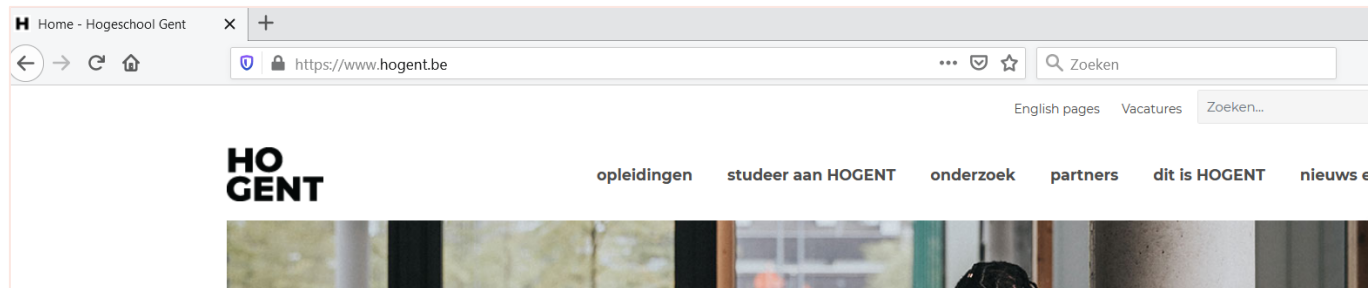
- **HTML**
 - **structuur en betekenis** geven aan onderdelen van een web-pagina
 - section – main – footer – table – ul – small – figure - ...
- **CSS**
 - **opmaak en layout** van de web-pagina verzorgen
 - font-family – max-width – grid – padding – margin - ...
- **JavaScript**
 - **interactie** tussen gebruiker en pagina mogelijk maken
 - reageren op muisbewegingen, klikken, ...
 - **manipulatie** van de web-pagina
 - dynamiek: html & css manipuleren, animaties creëren, ...



Client-server architecture



Client-server architecture



Status	Methode	Domein	Bestand	Initiator	Type	Overgebracht	G
200	POST	hogent-my.sharepoint.com	contextinfo	plt.odsp-common.chunkjs:1 (x...		NS_BINDING_REDIRECTED	
200	GET	www.hogent.be	/	event-loop.js:79 (document)	html	11,42 KB	3.
200	GET	shell.cdn.office.net	suiteux.shell.core.36b1defeed9c64946f1f.js	OneShell:17 (script)	js	gebufferd	2.
200	GET	hogent-my.sharepoint.com	root?\$select=guidedTour_oneDrive.teachingBubbles_oneDrive.surveyStatus_oneDrive	plt.odsp-common.chunkjs:1 (x...	json	3,43 KB	8.
200	GET	hogent-my.sharepoint.com	context?expand=Token,Payload	plt.odsp-common.chunkjs:1 (x...	json	5,40 KB	2.
200	GET	modern.akamai.odsp.cdn.of...	314.chunk.js	odblightspeedwebpackjs:1 (scr...	js	service worker	3.
200	GET	modern.akamai.odsp.cdn.of...	30.chunk.js	odblightspeedwebpackjs:1 (scr...	js	service worker	4.
200	GET	modern.akamai.odsp.cdn.of...	ondemand.resx-ondemand.chunk.js	odblightspeedwebpackjs:1 (scr...	js	service worker	3.
200	GET	modern.akamai.odsp.cdn.of...	222.chunk.js	odblightspeedwebpackjs:1 (scr...	js	service worker	2.
200	GET	amcdn.msftauth.net	me?partner=Sharepoint&version=10.20321.2&market=nl-NL&wrapperId=suiteshell	suiteux.shell.core.36b1defeed9...	js	8,94 KB (raced)	2.
204	OPTIONS	westeurope1-sphomep.svc...	feed?acronyms=true&start=0&count=8	fetch	plain	594 B	0
200	GET	shell.cdn.office.net	suiteux.shell.searchbox.f2413a0f9e03b700ce7.js	OneShell:17 (script)	js	gebufferd	1.
200	GET	shell.cdn.office.net	suiteux.shell.plus.f873362359561fe46493.js	OneShell:17 (script)	js	gebufferd	2.
200	GET	westeurope1-sphomep.svc...	feed?acronyms=true&start=0&count=8	plt.items-view.chunkjs:1 (fetch)	json	3,20 KB	7.
200	GET	ecs.office.com	1.0.0.0?agents=MicrosoftSearch&ecsserviceinformation=true&ClientWorkload=Office	suiteux.shell.searchbox.f2413...		NS_BINDING_ABORTED	
200	GET	www.hogent.be	bootstrap.min.css	stylesheet	css	gebufferd	1.
200	GET	fonts.googleapis.com	css?family=Montserrat:400,500,700,800,900	stylesheet	css	gebufferd	7.
200	GET	use.fontawesome.com	all.css	stylesheet	css	gebufferd	5.
200	GET	www.hogent.be	swiper.min.css	stylesheet	css	gebufferd	1.





de browser op de client
is verantwoordelijk voor
de verwerking van o.a.
HTML, CSS en JS die
van de server
ontvangen werd

HO
GENT

JavaScript uitvoeren

- de **browser** fungeert als host environment voor JS
- de **browser** bevat een **JS-engine** die instaat voor de vertaling en uitvoering van JS
 - interpretation
 - JIT-compilation
- ***non-browser omgevingen*** *(worden niet behandeld in deze cursus)*
 - *Node.js*
 - *Adobe Acrobat*
 - *CouchDB*

Overzicht browser & engines

	Chrome	Firefox	Safari	Edge
				
Rendering engine	Blink	Gecko	WebKit	Blink
JS engine	V8	SpiderMonkey	JavaScriptCore	V8

Geschiedenis

- uitgebreide geschiedenis van JavaScript:
<https://en.wikipedia.org/wiki/JavaScript#History>
- een video van Brendan Eich, bedenker en ontwikkelaar van JavaScript:
<https://www.youtube.com/watch?v=3-9fnjzmXWA>

Java vs JavaScript

- beide zijn het object geörienteerde programmeertalen
- JavaScript is sterk beïnvloed door functionele programmeertalen
- functies zijn first-class citizens in JavaScript
 - functies zijn ook objecten

Java vs JavaScript



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

01 JavaScript, de basis

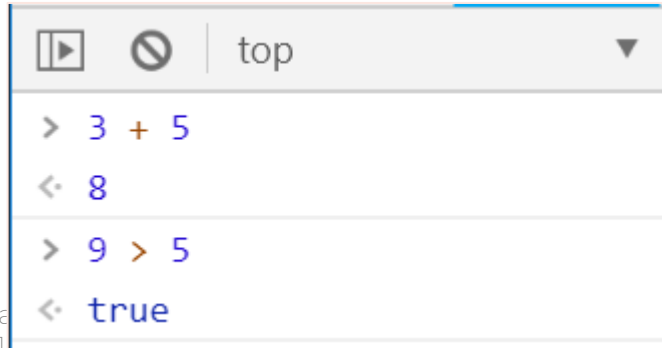
Aan de slag met JS!

GitHub

- Al het materiaal vind je op GitHub via <https://web-ii.github.io/OverViewCourse/>

De browser's JavaScript console

- Open de Developer Tools in de browser
 - Chrome/Firefox/Edge – F12 of <ctrl><shift><i>
 - Safari - <https://javascript.info/devtools#safari>
- Kies de Console tab
 - hier kan je on-the-fly JavaScript commando's uitvoeren.



console leegmaken

resultaat van de uitvoering van JS
<enter> voert code uit.
<shift>+<enter> voor de invoer van de volgende lijn code.

handig:
<enter> voert code uit.
<shift>+<enter> voor de invoer van de
volgende lijn code.

JavaScript toevoegen aan een HTML pagina: het `<script>` element

- het is mogelijk de JS-code in de HTML pagina zelf te zetten...

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Javascript</title>
  </head>

  <body>
    <p>Watch the console...</p>
    <script>
      console.log('Hello JavaScript!')
    </script>
  </body>

</html>
```

index.html

JavaScript toevoegen aan een HTML pagina: het `<script>` element

- het geniet de voorkeur de JS in een apart **.js bestand** op te nemen
 - voordeel voor onderhoudbaarheid, herbruikbaarheid, caching, ...
- via het **src-attribuut** van het script element verwijst je naar dit bestand

```
▼ html
  <> index.html
▼ js
  JS hello.js
```

H01 Javascript, de basis
slide 20

```
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Javascript</title>
  </head>

  <body>
    <p>Watch the console...</p>
    <script src="./js/hello.js"> </script>
  </body>

</html>
```

```
console.log('Hello JavaScript!');
```

hello.js

index.html

**HO
GENT**

JavaScript toevoegen aan een HTML pagina: het `<script>` element

- de script-tag plaats je in de `<body>` tag **na** de inhoud, juist `voor </body>`
 - reden: JS wordt pas gedownload, geparsed, en uitgevoerd als de HTML pagina al is weergegeven in de browser

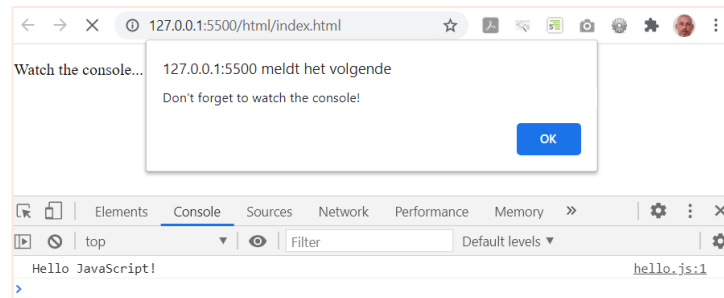
01 JavaScript, de basis

Code structuur

Statements

- maak gebruik van **statements, opdrachten**, om acties uit te voeren
- scheid statements met ;
- eenvoudige statements
 - **console.log**
 - toont uitvoer in de console
 - **alert**
 - toont uitvoer in een simpel dialoogvenster

```
console.log('Hello JavaScript!');  
alert("Don't forget to watch the console!");
```



Comments

- `//` one line comment
- `/*` multi
line
comment `*/`

```
// our first JS program contains two statements separated by semi-colons
console.log('Hello JavaScript!');

/* simple observations about JS strings
   - you can use " " or ' ' to form a string, just make sure the style
     of the opening quote matches the style of the closing quote
   -
     see how we insert a single quote in a string delimited by double quotes
*/

alert("Don't forget to watch the console!");
```


JS strict mode

- de komst van ES5 in 2009 introduceerde 'breaking changes' in de JavaScript taal
 - niet alle oude code kon uitgevoerd worden met de introductie van de nieuwe features
 - daarom werden breaking changes features per default uitgeschakeld
 - het is veiliger en beter om JS te runnen in '**strict mode**'
 - sommige JS silent errors werpen nu exceptions
 - de JS engine kan betere optimalisatie doen
 - verbied het gebruik van syntax die nog niet officieel in de ECMAScript werd opgenomen
 - je kan aangeven dat je script in strict mode moet runnen via een directive
- **"use strict"**
 - deze directive moet **helemaal bovenaan** je script staan
 - wanneer we verderop beginnen werken met **classes** (en **modules**) hoeven we dit niet meer expliciet te vermelden, deze werken **automatisch in strict mode**

01 JavaScript, de basis

Bouwstenen: variabelen

Variabelen

- JS gebruikt **variabelen** om data bij te houden
- een variabele heeft een naam, het is een benoemde plaats in het geheugen
- **let**
 - gebruik het keyword let om een variable te declareren
- **naamgeving**
 - de naam van een variabele mag enkel **letters, cijfers en de symbolen \$ en _** bevatten en de naam mag niet starten met een cijfer
 - gebruik camelCasing
 - ! hoofdlettergevoelig !

```
'use strict';  
  
let message;  
  
let age, _user;
```

Variabelen

- via de **assignment operator =** kan je een waarde toekennen aan een variabele

```
'use strict';  
  
let message;  
let age, user;  
  
message = 'Hello';  
age = 22;  
user = 'Anne-Marie';
```

Variabelen

- je kan declaratie en assignment (~initialisatie) combineren

```
'use strict';  
  
let message = 'Hello';  
let age = 22,  
    user = 'Anne-Marie';  
  
alert(message + ' ' + age + '-year old user called ' + user + '!');
```

127.0.0.1:5500 meldt het volgende
Hello 22-year old user called Anne-Marie!

OK

Variabelen

- nog een voorbeeld

```
'use strict';

let message = 'Hello';
let age = 22,
    user = 'Anne-Marie';

age = age + 1;
message = message + ' ' + age + '-
year old user called ' + user + '!';

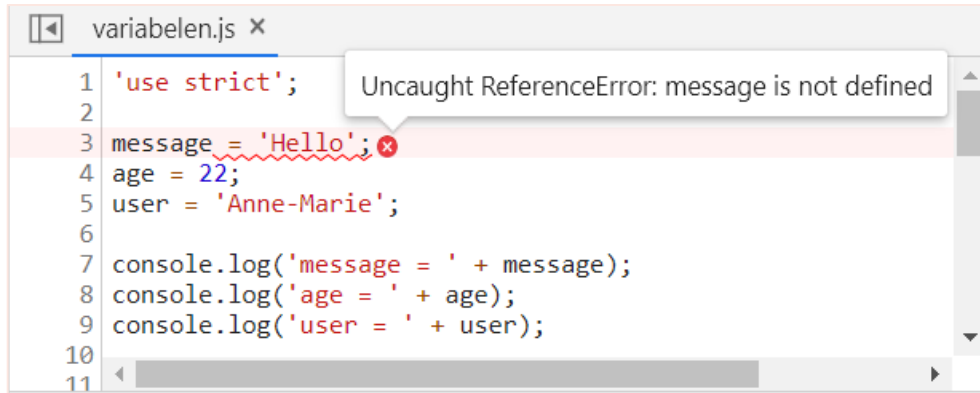
alert(message);
```

127.0.0.1:5500 meldt het volgende
Hello 23-year old user called Anne-Marie!

OK

Variabelen

- voorbeeld 'use strict'



```
1 'use strict';
2
3 message = 'Hello';
4 age = 22;
5 user = 'Anne-Marie';
6
7 console.log('message = ' + message);
8 console.log('age = ' + age);
9 console.log('user = ' + user);
10
11
```

Uncaught ReferenceError: message is not defined

zonder 'use strict' zou dit script, waarbij niet expliciet gedefinieerde variabelen worden gebruikt probleemloos uitgevoerd worden...

Constanten

- **const**
 - variabelen die je moet **initialiseren** bij declaratie en die nadien **niet meer van waarde kunnen veranderen**

```
let message = 'Hello';  
let age = 22;  
const user = 'Anne-Marie';
```



```
let message = 'Hello';  
let age = 22;  
const user;  
user = 'Anne-Marie';
```



```
let message = 'Hello';  
let age = 22;  
const user = 'Anne-Marie';  
user = 'Anna';
```



Variabelen en constanten

- good practice
 - gebruik zoveel mogelijk **const**, beperk het gebruik van **let** tot echte variabelen die van waarde moeten kunnen veranderen
 - naamgeving
 - gebruik zinvolle beschrijvende en precieze namen
 - vermijd afkortingen

01 JavaScript, de basis

Bouwstenen: basis datatypes

Datatypes

een JS variabele heeft steeds een datatype

- JS kent **8 basis datatypes**
 - **number** - getallen
 - **bigint** – hele grote gehele getallen
 - **string** - tekst bestaande uit 0 of meerdere karakters,
 - **boolean** - logische waarden: true/false
 - **null** - voor ongekennde waarden
 - **undefined** - voor niet toegekende waarden
 - **object** - voor meer complexe datastructuren
 - **symbol** - voor unieke identifiers






*maak gebruik van
typeof(x) of **typeof x**
om in string-vorm het type van
de variabele x te achterhalen*

Datatypes

JS is a **dynamically typed** language

- het type van een variabele wordt bepaald **at run-time**
- het type van een variabele **kan wijzigen**

```
3  let dynamisch;  
4  
5  console.log('Type van variabele dynamisch: ' + typeof dynamisch);  
6  dynamisch = 'Hello Javascript';  
7  console.log('Type van variabele dynamisch: ' + typeof dynamisch);  
8  dynamisch = 5;  
9  console.log('Type van variabele dynamisch: ' + typeof dynamisch);  
10 dynamisch = true;  
11 console.log('Type van variabele dynamisch: ' + typeof dynamisch);  
12 dynamisch = 1234567890123456789012345678901234567890n;  
13 console.log('Type van variabele dynamisch: ' + typeof dynamisch);
```

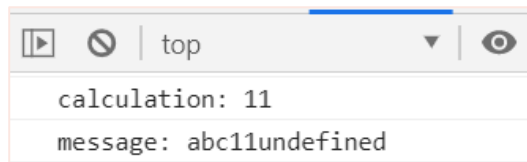
  top			Filter	Default levels ▼	
Type van variabele dynamisch: undefined				variabelen.js:5	
Type van variabele dynamisch: string				variabelen.js:7	
Type van variabele dynamisch: number				variabelen.js:9	
Type van variabele dynamisch: boolean				variabelen.js:11	
Type van variabele dynamisch: bigint				variabelen.js:13	

Datatypes

JS is a **loosely typed** language

- er wordt redelijk los omgesprongen met het type van een variabele
- er gebeuren veel impliciete type conversies

```
const calculation = 10 + true + false;  
const message = 'abc' + calculation + undefined;  
  
console.log('calculation: ' + calculation);  
console.log('message: ' + message);
```



Het datatype **number**

- **gehele én floating point getallen**
- 64-bit double precision floating point representation
 - gehele getallen: $[-(2^{53} - 1), 2^{53} - 1]$
- **integer literals**
 - decimaal: 3, 10, 10000000, 10_000_000, ...
 - prefix **0x** voor hexadecimaal
 - prefix **0o** voor octaal
 - prefix **0b** voor binair
- **floating point literals** [digits][.digits][(E|e)[(+|-)]digits]
 - 3.14, 2345.789, .333333333333333, 6.02e23
- **basis operatoren** : +, -, *, /, %

```
let three = 3;
let minus20 = -20;

let fiveMillion = 5000000;
fiveMillion = 5_000_000;
fiveMillion = 5e6;

let hex255 = 0xff;
let oct255 = 0o377;
let bin255 = 0b11111111;

let quarter = 0.25;
quarter = 25e-2;
```

Het datatype **number**

- afrondingsfouten
 - je hebt precisie tot op 17 cijfers na de komma

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

```
z =0.30000000000000004
```

Het datatype **number**

- speciale waarden
 - **NaN** (**N**ot **a** **N**umber)
 - om iets dat niet kan voorgesteld worden als getal te beduiden
 - bv. resultaat wanneer je een string wil omzetten naar een getal maar de string bevat geen geldige representatie voor een getal
 - bv. resultaat wanneer je de vierkantswortel van -1 berekent
 - **isNaN()** is een boolse functie die detecteert of een waarde NaN is
 - **Infinity**
 - een berekening die een getal retourneert buiten het waardenbereik, retourneert **Infinity** of **-Infinity**
 - bv. resultaat van $3 / 0$, of van 10^{10000}
 - **isFinite()** is een boolse functie die detecteert of een waarde finite is

Het datatype **number**

- enkele voorbeelden NaN & Infinity

```
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*

Het datatype **number**

- **string – number conversie** functies
 - **parseInt(*aString* [, *radix*])**
 - converteert string naar geheel getal
 - leading spaces in de string worden genegeerd
 - eerste beduidende positie moet +,- of cijfer zijn
 - de radix is optioneel
 - **parseFloat(*aString*)**
 - converteert string naar floating point
 - beide doorlopen de string tot ze het eerste ongeldige karakter tegenkomen

```
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
```

Het datatype **boolean**

- kan een logische waarde bevatten
 - **true** of **false** (case sensitive!)
 - alle waarden in JS hebben een bools equivalent, dit wordt gebruikt bij impliciete conversies

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

Het datatype **string**

- bevat een sequentie van 0 of meer Unicode karakters vervat in ‘ **of** “
 - opening en closing quote stijl moet dezelfde zijn
 - gebruik \ voor escape sequences
 - \", \', \n, \\
 - **+** operator: **concatenatie** van 2 strings
 - *merk op: er is geen type char voor één enkel karakter voor te stellen*

```
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
```

Het datatype **string**

- **toString()** conversie naar een string

```
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
```

Het datatype **string**

- **template literals** strings die variabelen & expressies bevatten.
 - worden omsloten door **backticks** ```
 - multiline strings (zonder `\n`) zijn mogelijk
 - expressies in de string evalueren: `${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."
```

Het datatype **undefined**

- kan enkel de speciale waarde **undefined** bevatten
 - staat voor een onbekende waarde
 - een variabele waaraan nog geen waarde is toegekend bevat de waarde undefined
 - een functie in JS retourneert steeds een waarde, indien er niet expliciet een waarde wordt geretourneerd dan wordt er impliciet undefined geretourneerd

Het datatype **null**

- kan enkel de waarde **null** bevatten
 - staat voor een lege object pointer
 - typeof null retourneert “object”
 - dit is een foutje die in JavaScript sloop...
 - evalueert false in een boolean expressie

Het datatype **bigint**

- om gehele getallen voor te stellen die **buiten het bereik van number** vallen
 - $< 2^{53} - 1$
 - $> 2^{53} - 1$

Het datatype **object**

- voorstelling van meer complexe datastructuren
 - worden verderop in deze cursus uitgebreid behandeld

Het datatype **symbol**

- om **unieke identifiers** voor te stellen
 - kan gebruikt worden om bv. unieke namen te hebben voor properties in objecten
 - zullen we niet verder behandelen in deze cursus

01 JavaScript, de basis

Bouwstenen: wrapper objects

Wrapper object

- is een object dat een primitief datatype 'omsluit'
- bestaat voor elk primitief datatype behalve null en undefined
 - **String**
 - **Number**
 - **BigInt**
 - **Boolean**
 - **Symbol**

*maak gebruik van
valueOf() om de primitieve
waarde voorgesteld door het
wrapper object te achterhalen*

Wrapper object

- het wrapper object geeft ons de mogelijkheid **properties** en **methodes** te gebruiken
 - properties – eigenschappen
 - methodes – functies
- *verderop deze cursus gaan we uitgebreid in op properties en methodes, in dit deeltje maak je kennis met enkele handige properties en methodes van wrapper objecten*

Het wrapper object **Number**

- JavaScript past impliciete conversie toe, het primitieve getal wordt automatisch omgezet naar een Number object als je een methode van Number oproept
- voorveelden van methodes
 - **toFixed(x)** : afronden tot x cijfers na de komma
 - **toString(x)** : string representatie, x is de radix, default 10
- voorbeelden van 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
- raadpleeg https://developer.mozilla.org/en-us/docs/Web/JavaScript/Reference/Global_Objects/Number voor meer informatie over de properties en methodes van wrapper objecten

Properties

`Number.EPSILON`

`Number.MAX_SAFE_INTEGER`

`Number.MAX_VALUE`

`Number.MIN_SAFE_INTEGER`

`Number.MIN_VALUE`

`Number.NaN`

`Number.NEGATIVE_INFINITY`

`Number.POSITIVE_INFINITY`

Methods

`Number.isFinite()`

`Number.isInteger()`

`Number.isNaN()`

`Number.isSafeInteger()`

`Number.parseFloat()`

`Number.parseInt()`

`Number.prototype.toExponential()`

`Number.prototype.toFixed()`

`Number.prototype.toLocaleString()`

`Number.prototype.toPrecision()`

Het wrapper object **Number**

- enkele voorbeelden

```
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
```


Het wrapper object **Boolean**

- de functie **Boolean(x)** cast x naar een Boolean
- enkele voorbeelden

```
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)
```

Het wrapper object **String**

- **String()**

- type-casting met String() roept in feite toString() aan maar één 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"
```

Het wrapper object **String**

- het primitief **wrapper object String** bevat heel wat handige methodes
- zie https://developer.mozilla.org/en-us/docs/Web/JavaScript/Reference/Global_Objects/String

Het wrapper object **String**

- voorbeeldje, stel $w = \text{'Javascript'}$;

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

01 JavaScript, de basis

Bouwstenen: Math & Date object

Het **Math**-object

- **Math object**
 - is een **built-in object** dat properties en methodes bevat voor **wiskundige constanten en functies**
 - bevat enkel **static properties en methods**
 - gebruik steeds Math. om te refereren naar een property/methode
 - Meer op https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

Math

Properties

Math.E

Math.LN10

Math.LN2

Math.LOG10E

Math.LOG2E

Math.PI

Math.SQRT1_2

Math.SQRT2

Methods

Math.abs()

Math.acos()

Math.acosh()

Math.asin()

Math.asinh()

Math.atan()

Math.atan2()

Math.atanh()

Math.cbrt()

Het **Math**-object

- **Math object**

- 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)

```
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
```

Het **Math**-object

- **Math object**

- **Oefeningen**

- Genereer een random waarde voor een dobbelsteen
 - Bereken de omtrek van een cirkel met straal 10

- **!!! Maak gebruik van de informatie op MDN !!!**

Het **Date**-object

- 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)

- enkele 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**-object

```
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
```

01 JavaScript, de basis

Controlestructuren

Controle structuren

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

Controle structuren

- Code blokken worden afgebakend met accolades

```
{  
    // code  
}
```

Controle structuren

- **selectie**

- **if (...) {...}**

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

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

- **if (...) {...} 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!');  
}
```

Controle structuren

- **selectie**
 - **if (...) {...} else {...}**
 - 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.');
```



Controle structuren

- **selectie**
 - **switch (...) case ...**

```
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!");  
  }  
}
```

H01 Javascript, de basis
slide 72

Yeah...Weekend



Controle structuren

- **iteratie**

- **while (...) do {...}**
- **do {...} while (...)**

```
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]

Controle structuren

- **iteratie**
 - **for (...) {...}**

```
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
```

Controle structuren

- 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.

Controle structuren

- Herhalingen onderbreken

```
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
```

01 JavaScript, de basis

Operatoren

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');  
  }  
}
```

Operatoren

- !!! er zijn twee operatoren voor gelijkheid !!!

- **== (en !=)**

- er gebeurt impliciete type conversie, m.a.w. het type wordt niet gecontroleerd

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

- **=== (en !==)**

- geen impliciete type conversie
 - best practice: gebruik ===

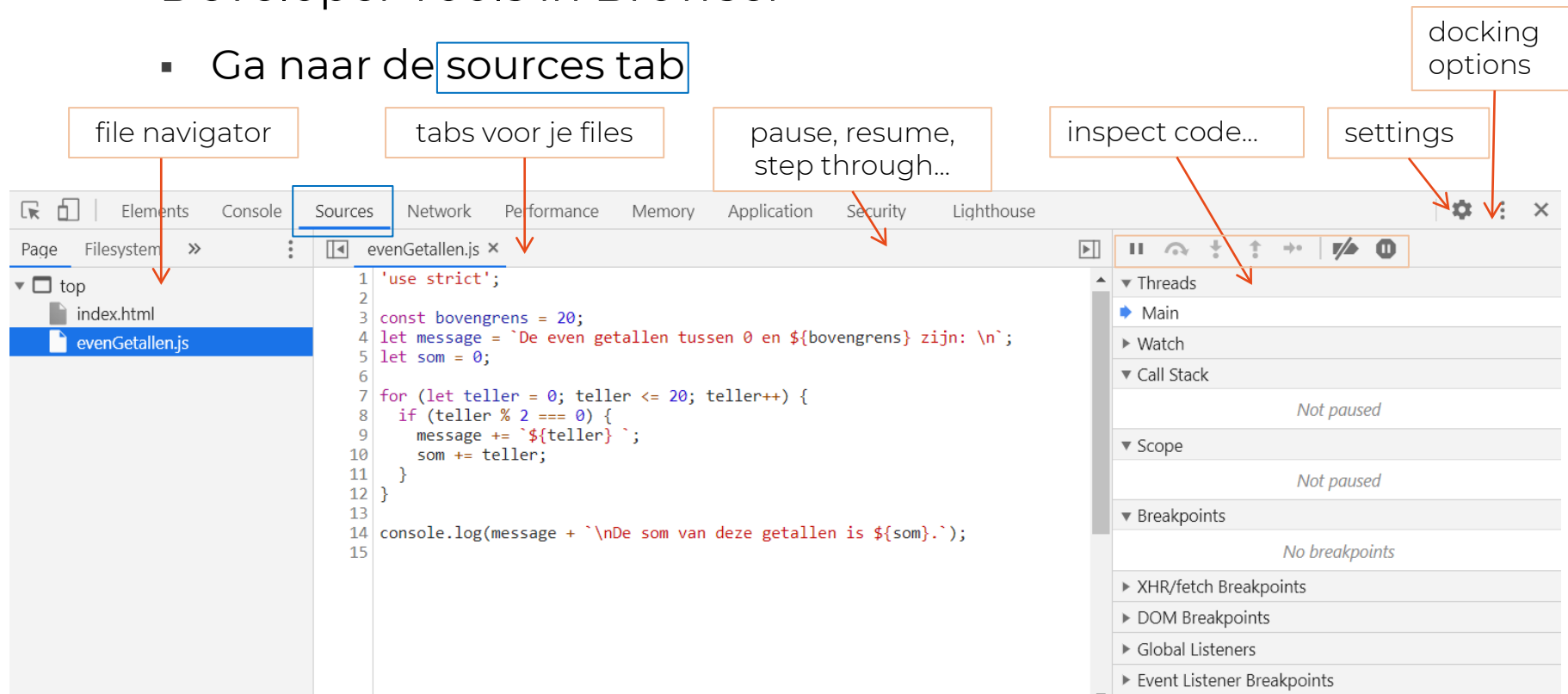
```
1 === '1'; // false  
1 !== '1'; // true  
1 === 1.0000000000000001; // false  
1 === 1.0000000000000001; // true
```


01 JavaScript, de basis

Javascript debuggen

JavaScript debuggen

- Developer Tools in Browser
 - Ga naar de **Sources** tab



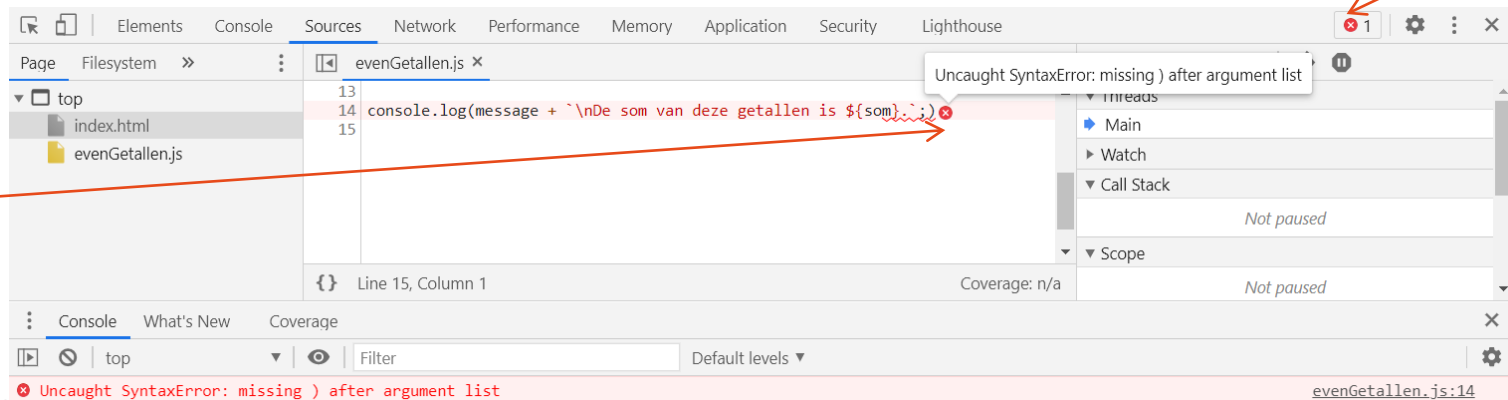
JavaScript debuggen

- Debugging via de JavaScript console in Chrome
 - voorbeeld: syntax-fout in evenGetallen.js

```
console.log(message + `\\nDe som van deze getallen is ${som}.`);
```

- in de console/sources tab zie je de fout

opent de console om de fouten te zien

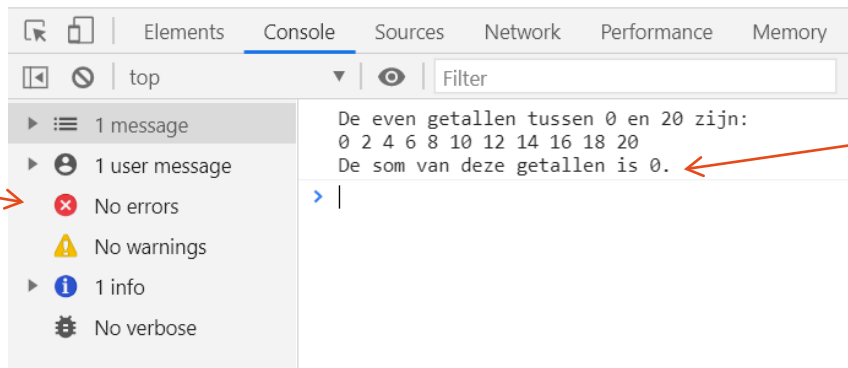


hover
hierover om
de fout te
zien

JavaScript debuggen

- Debugging via de JavaScript console in Chrome
 - voorbeeld: logische fout in evenGetallen.js

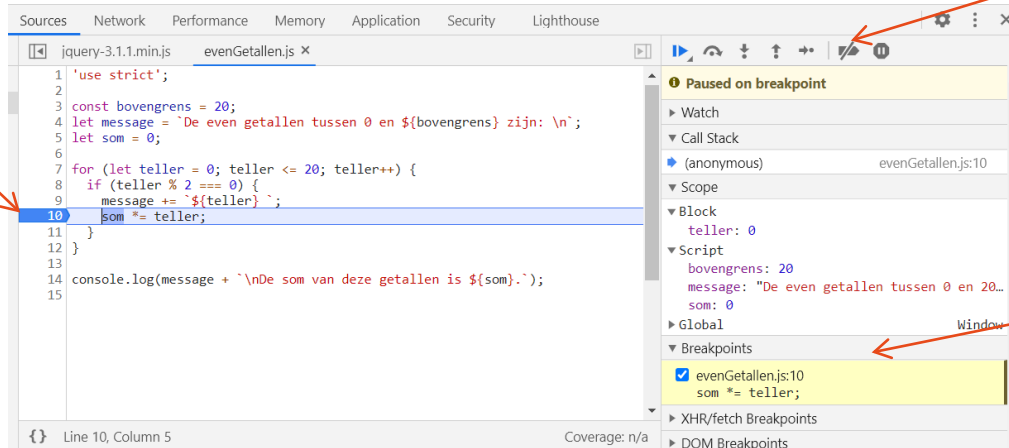
```
som *= teller;
```



JavaScript debuggen

- Debugging via de JavaScript console in Chrome
 - voorbeeld: logische fout in evenGetallen.js
 - we zetten in een **breakpoint** in de lus

klik hier om
een breakpoint
te plaatsen/te
verwijderen



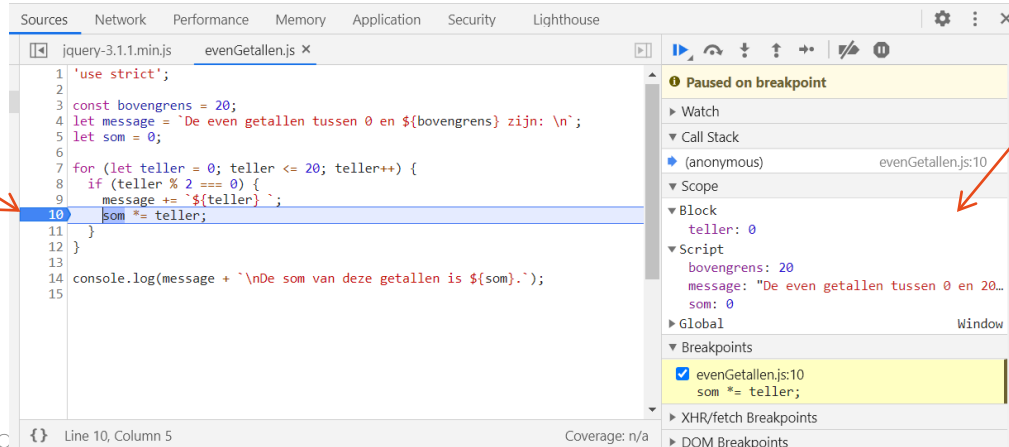
hier kan je alle
breakpoints
(de)activeren

hier kan je
afzonderlijke
breakpoints
(de)activeren

JavaScript debuggen

- Debugging via de JavaScript console in Chrome
 - voorbeeld: logische fout in evenGetallen.js
 - we herladen de pagina
 - we **inspecteren** de variabelen som & teller

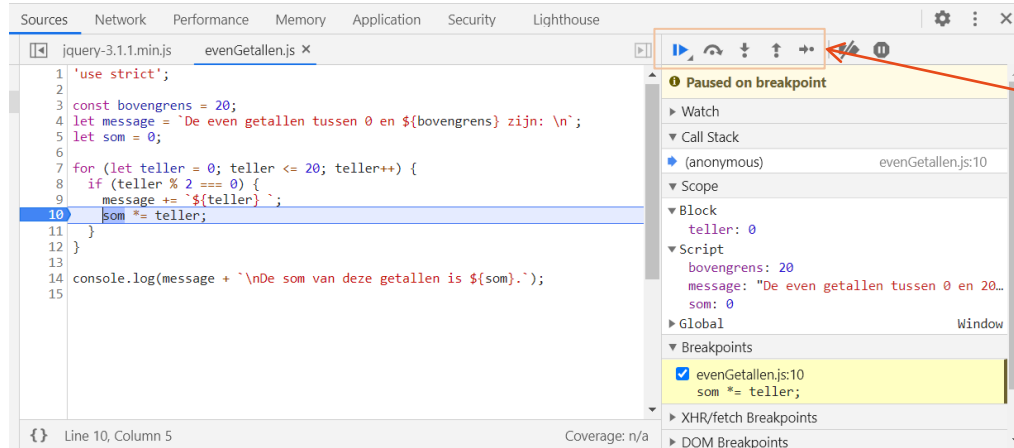
uitvoering
stopt aan
breakpoint



hier kunnen
we de
variabelen
inspecteren

JavaScript debuggen

- Debugging via de JavaScript console in Chrome
 - voorbeeld: logische fout in evenGetallen.js
 - we laten de uitvoering gecontroleerd verder verlopen...

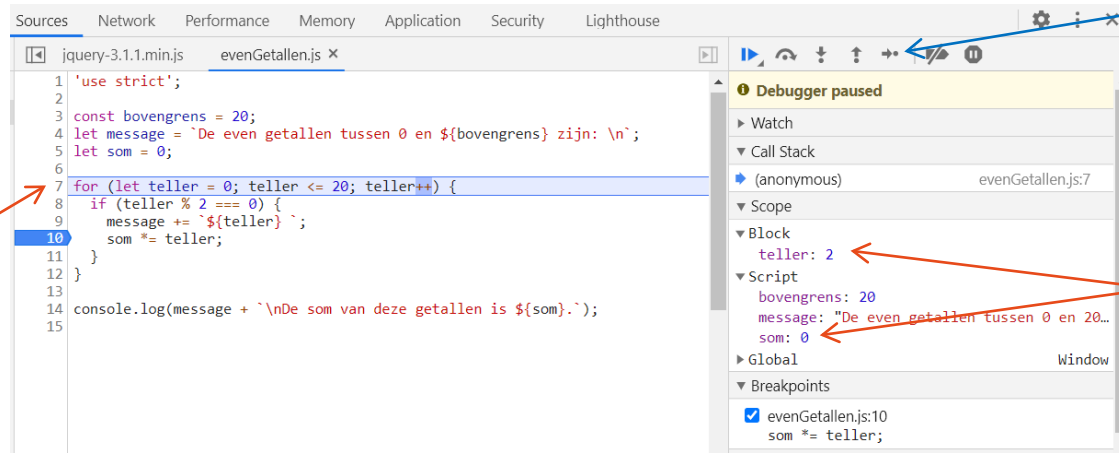


we kunnen nu naar
wens verder doorheen
het script gaan

resume
step over
step into
step out
step

JavaScript debuggen

- Debugging via de JavaScript console in Chrome
 - voorbeeld: logische fout in evenGetallen.js
 - we springen stap per stap verder in de uitvoering van het script via **step**



JavaScript debuggen

- Meer weten over JS debuggen in de developer console?

<https://developers.google.com/web/tools/chrome-devtools/javascript>

01 JavaScript, de basis

Functies

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.”

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
- **Hoisting en 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

- **Hoisting en let/const variables**

- hoisting gebeurt ook **voor de declaratie** van variabelen gedeclareerd met let en const
- hoisting gebeurt **niet voor de initialisatie** van de variabelen!

```
'use strict';
```

```
x = 10;  
console.log(x);
```

✖ ▶ Uncaught ReferenceError: x is not defined
at variabelen.js:3

klassiek: in strict mode moet je je variabelen declareren...

```
'use strict';
```

```
x = 10;  
let x = 20;  
console.log(x);
```

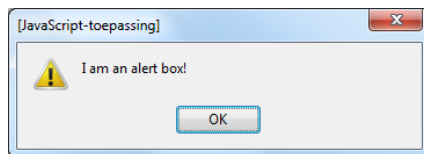
✖ Uncaught ReferenceError: Cannot access 'x' before initialization
at variabelen.js:3

de variabele is gedeclareerd en de declaratie is gehoist, de initialisatie is niet gehoist en je mag niet naar de variabele verwijzen alvorens ze werd geïnitialiseerd

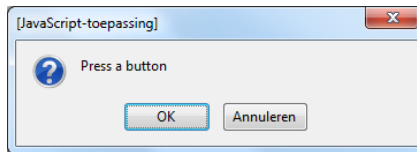
Functies

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

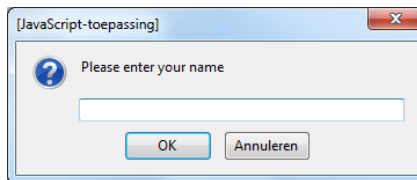
- **alert** box: `alert("sometext");`



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



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



Funcities

```
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?`);  
    }  
}
```

01 JavaScript, de basis

Arrays

Arrays

- een **array** 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
- array notatie: **[]**

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

- declaratie van een **lege array** op twee manieren

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

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

Arrays

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

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



- wijzigen

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

Arrays

- aanmaken van array via **array literal**

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

- voorbeeld met elementen van verschillend type

```
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

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

- verwijdert de waarde op deze positie, maar de ruimte bestaat nog steeds en bevat nu de waarde **undefined**.

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);  
}
```

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

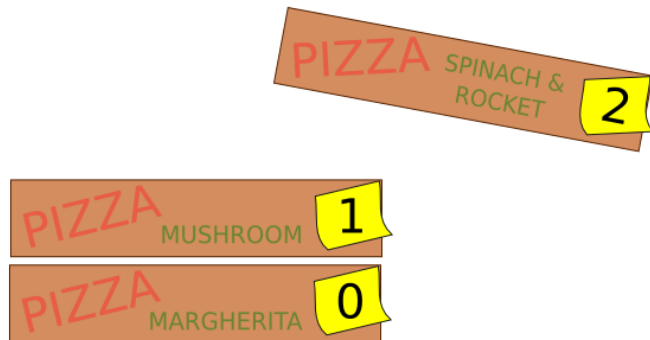
- **pop, push, shift en unshift**

- **stel**

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

- **pop** verwijdert het laatste element uit array en retourneert dit element

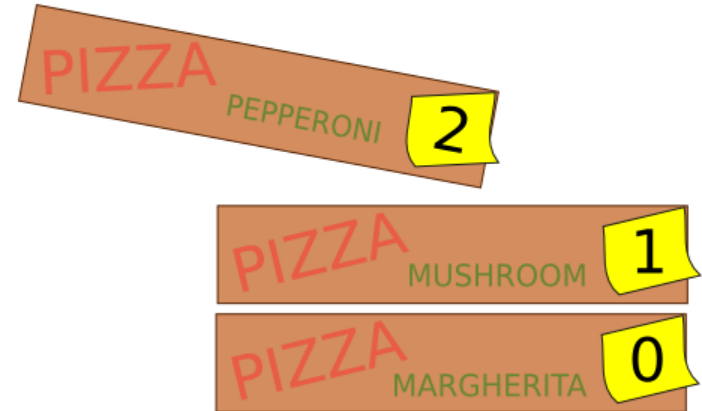
```
pizzas.pop(); // << 'Spinach & Rocket'
```



Arrays

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

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



Arrays

- **pop, push, shift en unshift**
 - **shift** : verwijdert de eerste waarde in array en returnt deze

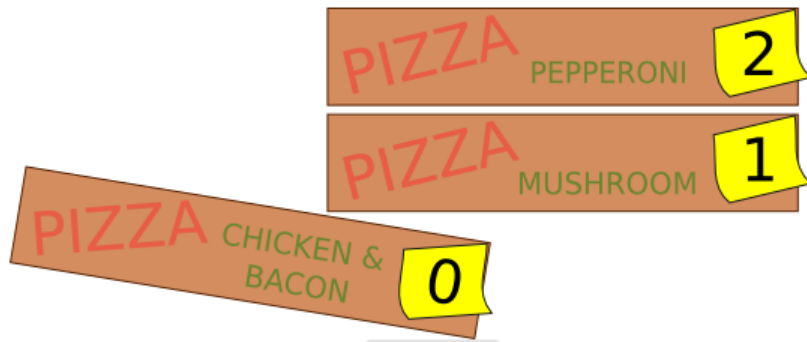
```
pizzas.shift(); //<< 'Margherita'
```



Arrays

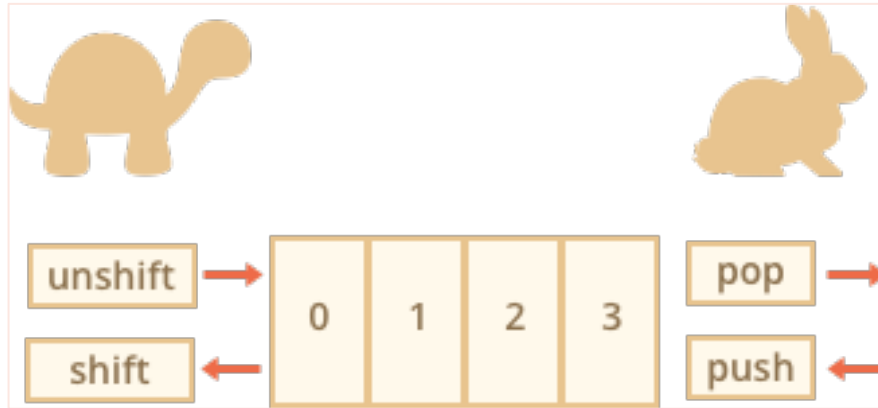
- **pop, push, shift en unshift**
 - **unshift** : voegt één of meerdere waarden toe aan het begin van het array en retournt de nieuwe lengte van de array.

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



Arrays

- **pop, push, shift en unshift**



de array vooraan laten groeien en krimpen is een trager proces omdat overige elementen in de array moeten verplaatst worden...

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

- ander Array methodes
 - **concat()** : voegt 2 arrays samen
 - **reverse()** : keert de volgorde van de array elementen om
 - **slice(start_index, upto_index)** : retournt 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

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
```

Arrays

- destructuring

```
//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
```

Arrays

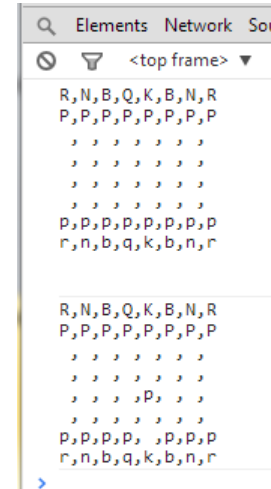
- **meerdimensionele arrays**
 - een array van een array
= 2 dimensionale array.
 - een array van een array van een array
= 3 dimensionale array.
- voorbeeld : een array die de kaarten van 2 poker hands bevat

```
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]);
```

Arrays

- voorbeeld 2-dim 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'));  
}
```



**HO
GENT**