

The background of the slide features a photograph of two individuals with long hair, seen from behind, looking out at a vibrant sunset. The sky is a mix of orange, pink, and blue. The image is partially obscured by a large, white, stylized letter 'H' that serves as a design element.

Web Development II

Hoofdstuk 01: JavaScript, de basis

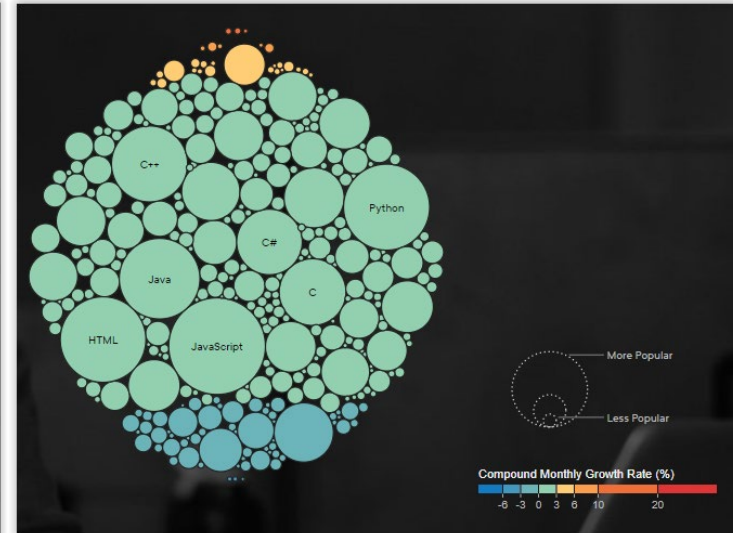
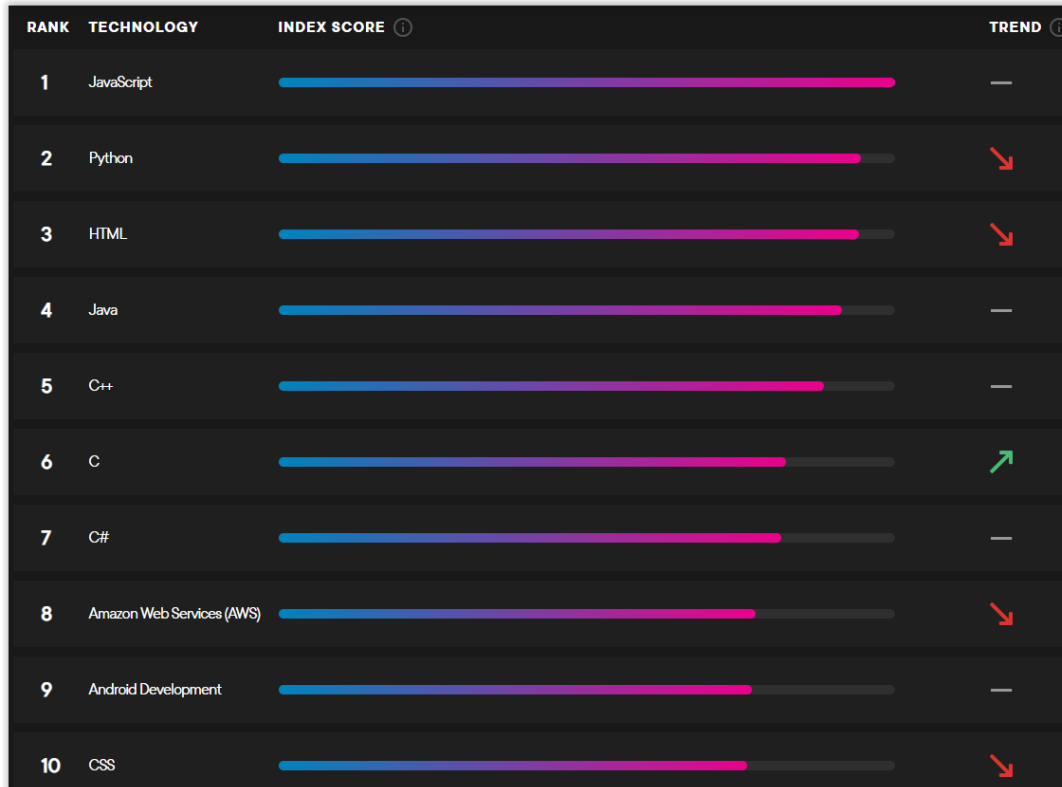
Inhoud

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

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 2022 Language Specification
 - <https://www.eca-international.org/publications-and-standards/standards/ecma-262/>
- deze standaard evolueert jaar per jaar
 - huidige versie: ECMAScript 2022 Language Specification: sinds juni 2022
 - work in progress: ECMAScript 2023 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**

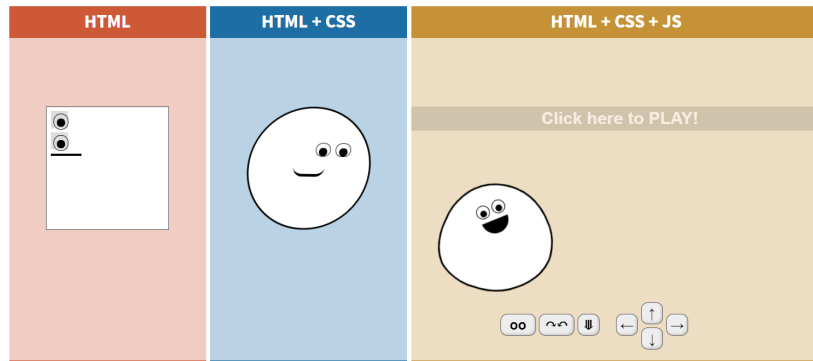
interactie tussen gebruiker en pagina mogelijk maken

- reageren op muisbewegingen, klikken, ...

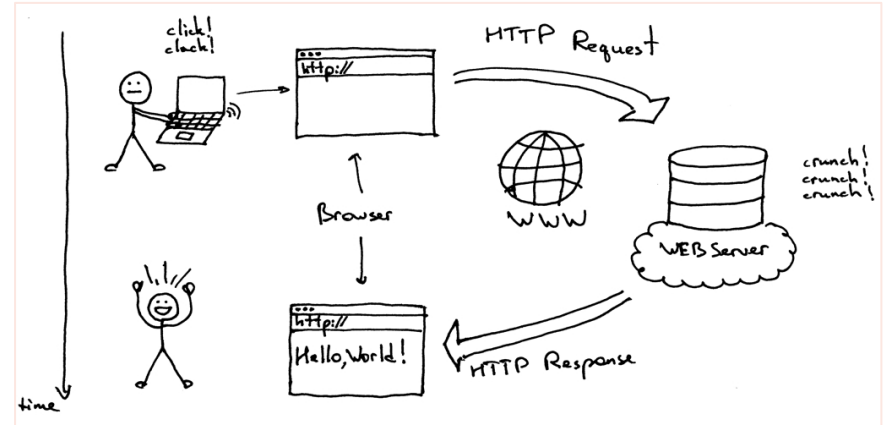
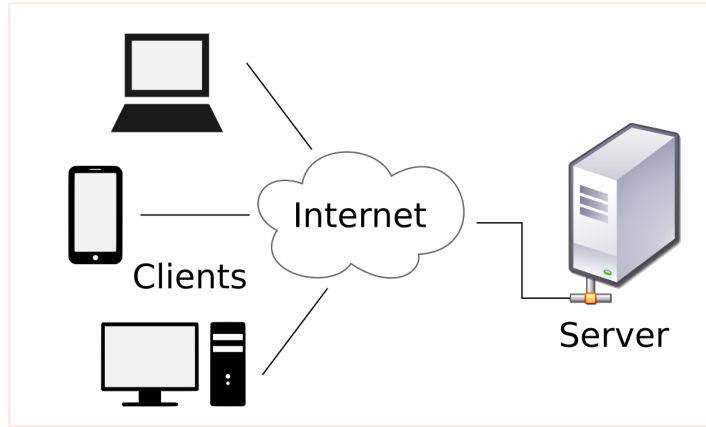
manipulatie van de web-pagina

- dynamiek: html & css manipuleren, animaties creëren, ...

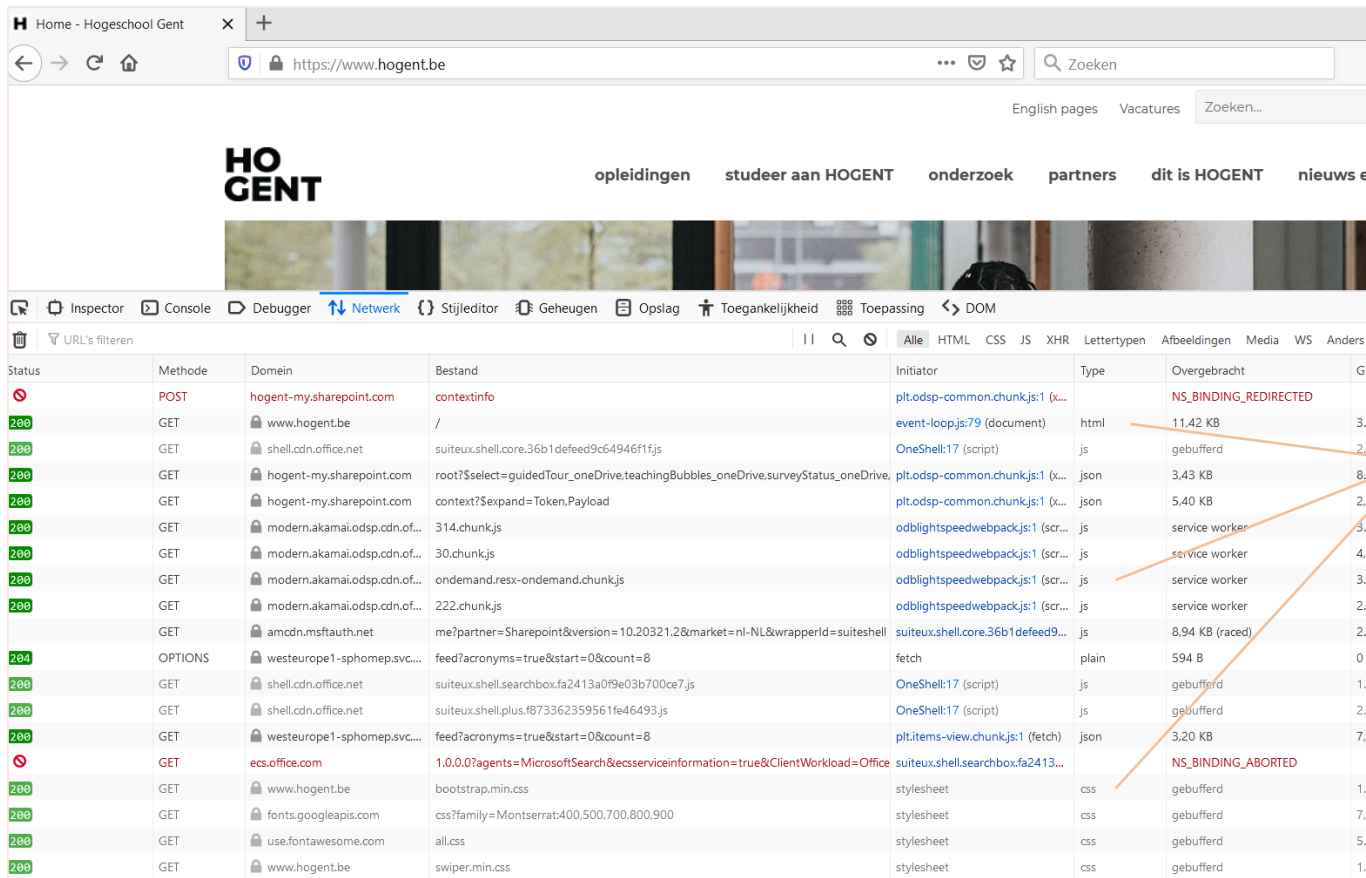
H01 Javascript, de basis
slide 7



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.faf2413a0f9e03b700ce7.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.faf2413...		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	B link	Gecko	WebKit	B link
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 object georiënteerde programmeertalen
- JavaScript is sterk beïnvloed door functionele programmeertalen
- functies zijn first-class citizens in JavaScript
 - spoiler: functies zijn ook objecten



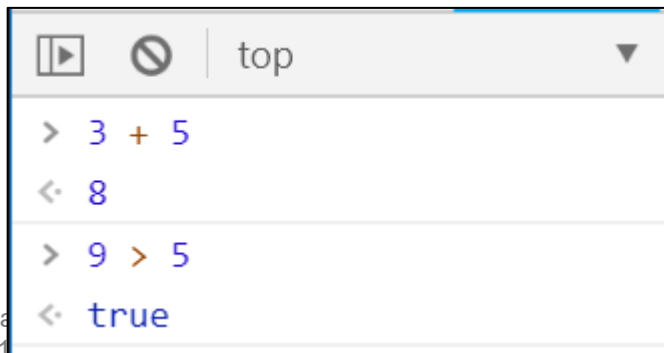
Java	JavaScript
Strongly typed	Loosely typed
Static	Dynamic
Classical inheritance	Prototypal
Classes	Functions
Constructors	Functions
Methods	Functions
Compiled	Interpreted
Methods for working with I/O – network / files	Uses the hosting environment

01 JavaScript, de basis

Aan de slag met JS!

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.

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>
```


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



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Javascript</title>
  </head>
  <body>
    <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, net voor `</body>`

reden: JS wordt pas gedownload, geparsed, en uitgevoerd als de HTML pagina al is weergegeven in de browser

01 JavaScript, de basis

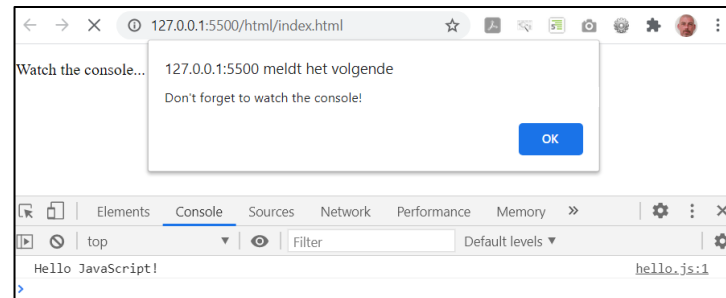
Code structuur

Statements

voorbeeldenSlides

- maak gebruik van **statements** 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

<https://javascript.info/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

JS strict mode

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

<https://javascript.info/variables>

voorbeeldenSlides

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);
```

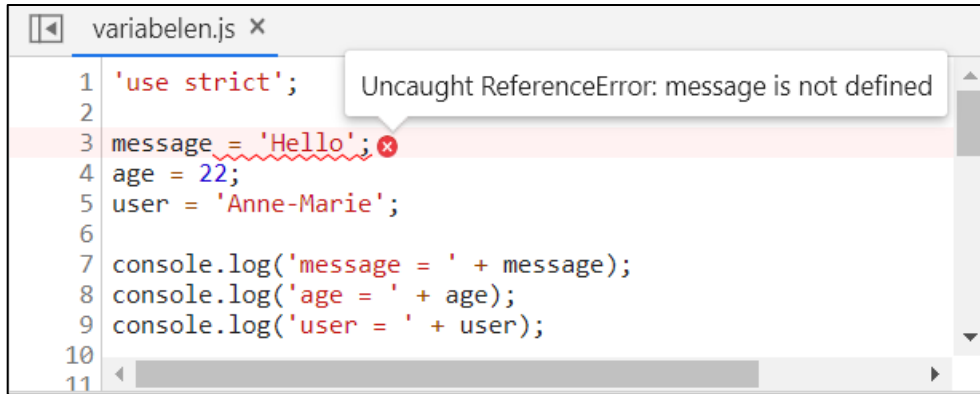
127.0.0.1:5500 meldt het volgende

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
```

*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

<https://javascript.info/types>

een JS variabele heeft steeds een datatype

JS kent **8 basis datatypes**

- **number** - getallen
- **bigint** – hele grote gehele getallen
- **string** - tekst bestaande uit geen 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);
```

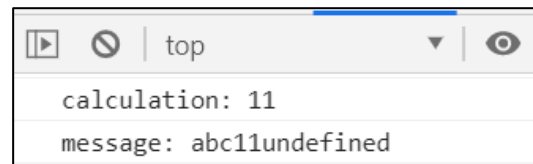
  top	  Filter	Default levels  
Type van variabele dynamisch: undefined	variabelen.js:5	
Type van variabele dynamisch: string	variabelen.js:7	
Type van variabele dynamisch: number	variabelen.js:9	
Type van variabele dynamisch: boolean	variabelen.js:11	
Type van variabele dynamisch: bigint	variabelen.js:13	

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);
```



Datatype **number**

<https://javascript.info/types#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;
```

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
```

Datatype **number**

speciale waarden

NaN (Not a Number)

- 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

Datatype **number**

speciale waarden

- **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

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
```


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
```

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

Datatype string

<https://javascript.info/types#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
```

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
```

Datatype string

template literals strings die variabelen & expressies bevatten.

- worden omsloten door **backticks** ``expressie``
- 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 and not 10."
```

Datatype **undefined**

<https://javascript.info/types#the-undefined-value>

bevat enkel de speciale waarde **undefined**

- 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

Datatype **null**

<https://javascript.info/types#the-null-value>

bevat enkel de waarde **null**

- staat voor een lege object pointer
- typeof null retourneert “object”
 - dit is een foutje dat in JavaScript sloop...
- evalueert false in een boolean expressie

Datatype **bigint**

<https://javascript.info/types#bigint-type>

om gehele getallen voor te stellen die **buiten** het bereik van **number** vallen

$< -2^{53}-1$ of $> 2^{53}-1$

gebruik de **suffix n** voor bigint literals

```
const aVeryBigNumber = 900719925474099199n;
```


Datatype **object** - **symbol**

Datatype **object** is een voorstelling van meer complexe datastructuren: wordt later uitgebreid behandeld

Datatype **symbol** is 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

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

Verder in 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

Wrapper object **Number**

https://developer.mozilla.org/en-us/docs/Web/JavaScript/Reference/Global_Objects/Number

JavaScript past impliciete conversie toe, het primitieve getal wordt automatisch omgezet naar een Number object als je een methode van Number oproept

- voorbeelden 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

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()`

Wrapper object **Number**

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
```

Wrapper object Boolean

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean

de functie **Boolean(x)** cast x naar een Boolean
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 infinity
const a = null;
console.log((a) ? true : false); // false
let c;
console.log((c) ? true : false); // false undefined
```

Wrapper object **String**

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

Wrapper object **String**

```
let w = 'Javascript';
```

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

Wrapper object **BigInt**

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt

je kan de functie **BigInt(x)** gebruiken om een bigint te maken

```
const hugeNumber = BigInt(9007199254740991)
// 9007199254740991n

const anotherHugeNumber = BigInt("9007199254740991")
// 9007199254740991n
```

een **bigint** is gelijkaardig aan een **number**, maar er zijn ook **significante verschillen**.

01 JavaScript, de basis

Bouwstenen: Math & Date object

Math-object

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math

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

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()

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
```

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 !!!

Date-object

houdt datums bij in milliseconden sinds 1/1/1970 UTC

creatie van een date object:

```
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)
```

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

<https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Instructies>

Code blocks

Selectie: if - else / switch -case

Iteratie: while / do..while / for

Iteraties onderbreken: break/continue

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 ...

```
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



Controle structuren

iteratie

while (...) do {...}

do {...} while (...)

```
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(':)');
```

```
More icecream!
More icecream!
More icecream!
More icecream!
More icecream!
More icecream!
More icecream!
More icecream!
More icecream!
More icecream!
More icecream!
life without ice cream isn't the same
```

Controle structuren

iteratie

for (...) {...}

```
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

iteraties 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

```
//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

<https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Operators>

Berekeningsoperatoren :

+, -, *, /, %, ++, --, **, unary -, unary +

Toewijzingsoperatoren :

=, *=, /=, % =, +=, -=

Vergelijkingsoperatoren :

==, !=, ===, !==, >, >=, <, <=

Logische operatoren :

&&, ||, !, ??

String operatoren :

+ en +=

Conditionele operatoren : *condition ? ifTrue : ifFalse*

Operatoren

Vergelijkingsoperatoren

== en ===

!= en !==

<

>

<=

>=

```
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

=== (en !==)

geen impliciete type conversie

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

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

best practice: gebruik ===

Operatoren

Logische operatoren

&& → AND

|| → OF

! → NOT

```
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

Nullish coalescing operator: ??

Is een logische operator die de rechterexpressie retourneert als de linkerexpressie *null* of *undefined* is. In het andere geval wordt de linkerexpressie geretourneerd.

Om te testen of een expressie is gedefinieerd (maw. of een expressie verschillend is van *null* of *undefined*)

Het resultaat van **a ?? b** is:

a indien a is gedefinieerd

b indien a niet is gedefinieerd

(gedefinieerd betekent hier verschillend van null of undefined)

Operatoren

Nullish coalescing operator: ??

Met andere woorden: het is de syntax om de eerste gedefinieerde expressie van twee te gebruiken.

Dit is twee maal hetzelfde:

```
result = a ?? b  
result = (a !== null && a !== undefined) ? a : b;
```

Voorbeelden

```
let user;  
console.log(user ?? "Anonymous"); // Anonymous (user not defined)
```

```
let user = "Johnny";  
console.log(user ?? "Anonymous"); // Johnny (user is defined)
```

Operatoren

Nullish coalescing operator: ??

kan ook gechained worden, waardoor de eerste gedefinieerde waarde wordt toegekend.

```
let firstName = null;  
let lastName = null;  
let nickName = "Supercoder";  
// shows the first defined value:  
console.log(firstName ?? lastName ?? nickName ?? "Anonymous"); // Supercoder
```

Het verschil met || (OR) is dat bij OR geen onderscheid wordt gemaakt tussen alle mogelijke waarden die false geven, namelijk 0, "", false, null en undefined.

```
let height = 0;  
console.log(height || 100); // 100  
console.log(height ?? 100); // 0
```

01 JavaScript, de basis

Javascript debuggen

JavaScript debuggen

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

The screenshot shows the Chrome Developer Tools interface with the Sources tab selected. The file navigator on the left shows a file tree with 'evenGetallen.js' selected. The main pane displays the JavaScript code for 'evenGetallen.js'. The right sidebar shows the 'Call Stack' panel, which is currently empty and shows 'Not paused'. Annotations with arrows point to various UI elements:

- file navigator**: Points to the file tree on the left.
- tabs voor je files**: Points to the tab bar at the top of the Sources panel.
- inspect code...**: Points to the 'inspect' button in the top right of the Sources panel.
- pause, resume, step through ...**: Points to the debugging toolbar at the top right of the Sources panel.
- settings**: Points to the settings gear icon in the top right of the Sources panel.
- docking options**: Points to the docking options menu icon in the top right of the Sources panel.

```
1 'use strict';
2
3 const bovengrens = 20;
4 let message = `De even getallen tussen 0 en ${bovengrens} zijn: \n`;
5 let som = 0;
6
7 for (let teller = 0; teller <= 20; teller++) {
8   if (teller % 2 === 0) {
9     message += `${teller} `;
10    som += teller;
11  }
12 }
13
14 console.log(message + `\nDe som van deze getallen is ${som}.`);
15
```

JavaScript debuggen

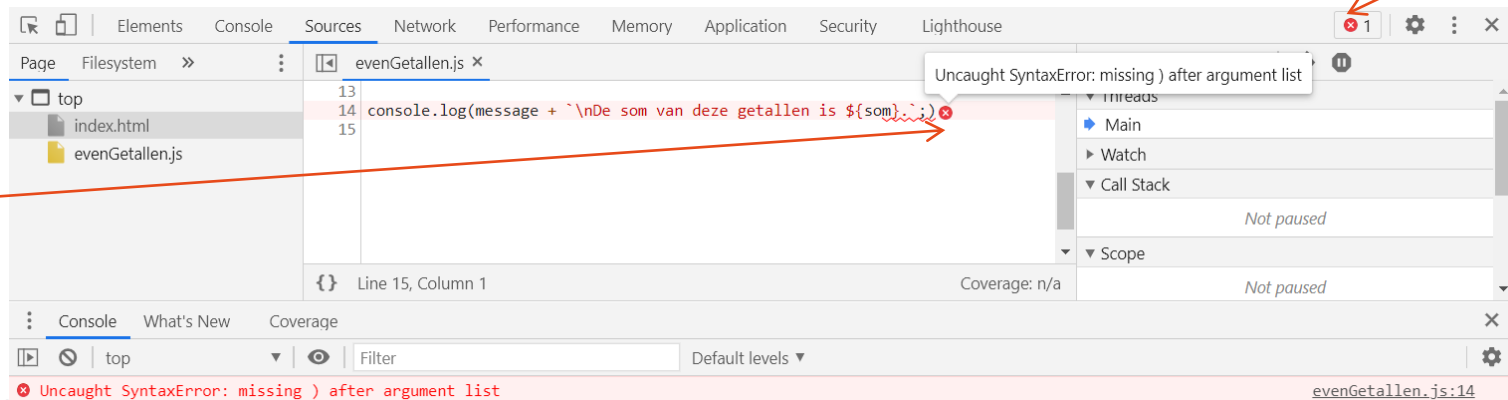
- Debugging via de JavaScript console in Chrome

- 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

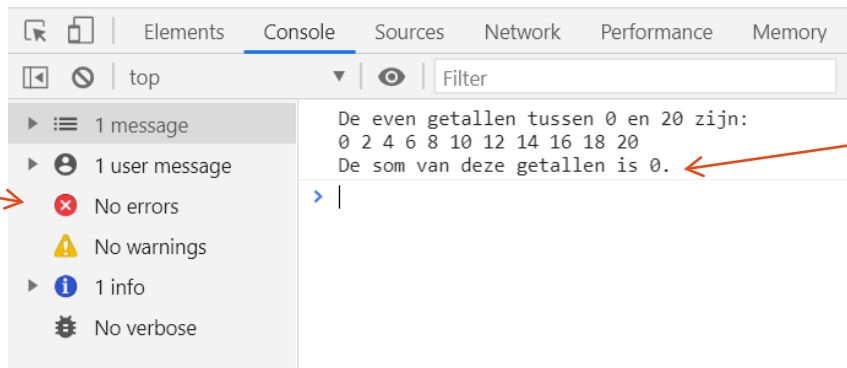


hover hierover om de fout te zien

JavaScript debuggen

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

```
som *= teller;
```



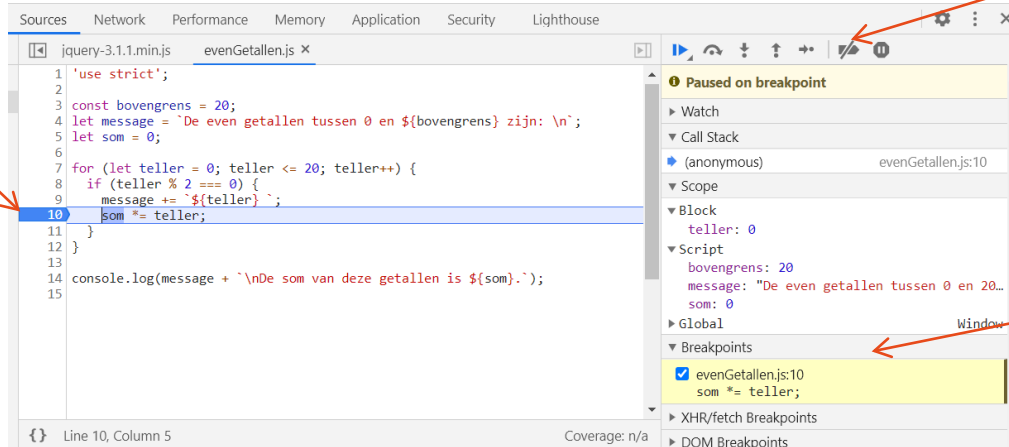
no errors ...



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



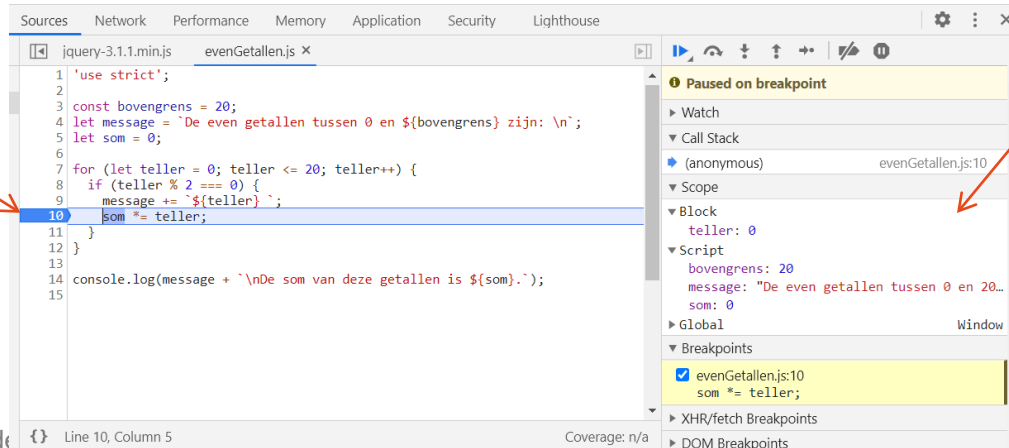
hier kan je alle
breakpoints
(de)activeren

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

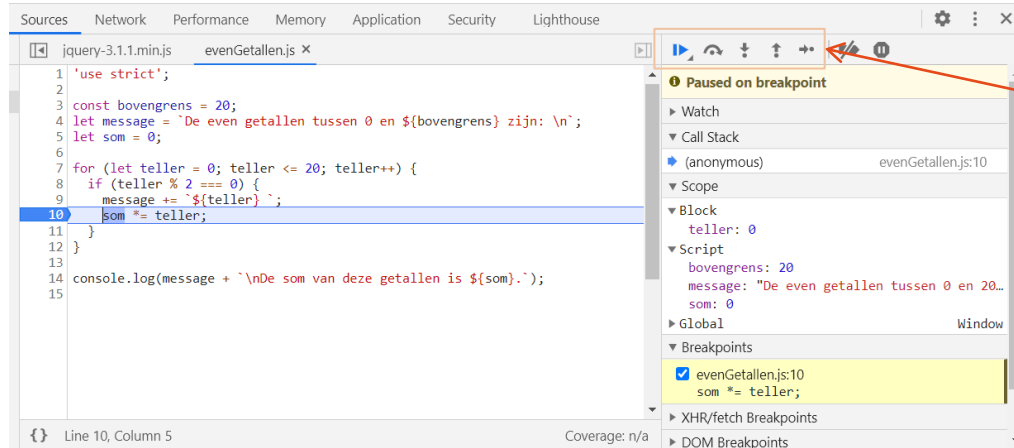
uitvoering
stopt aan
breakpoint



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...

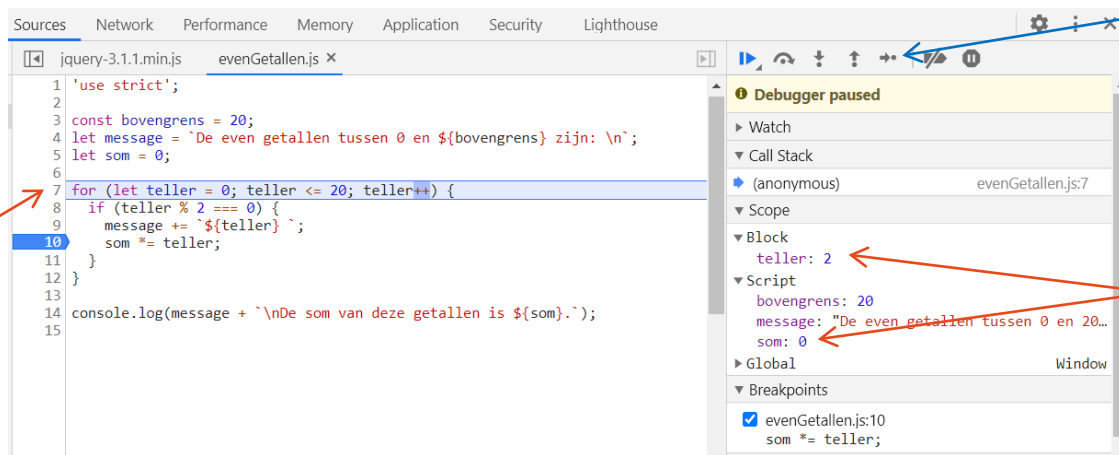


we kunnen nu naar
wens verder doorheen
het script gaan

**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](#)



volgende
stap staat
aangegeven
in
lichtblauw...

na een aantal keer
step te hebben geklikt
zien we dat de teller 2
is maar onze som niet
aangepast werd, we
ruiken onraad en zien
de kans mooi om de
bug plat te slaan...

JavaScript debuggen

Meer weten over JS debuggen in de developer console?

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