Web advanced

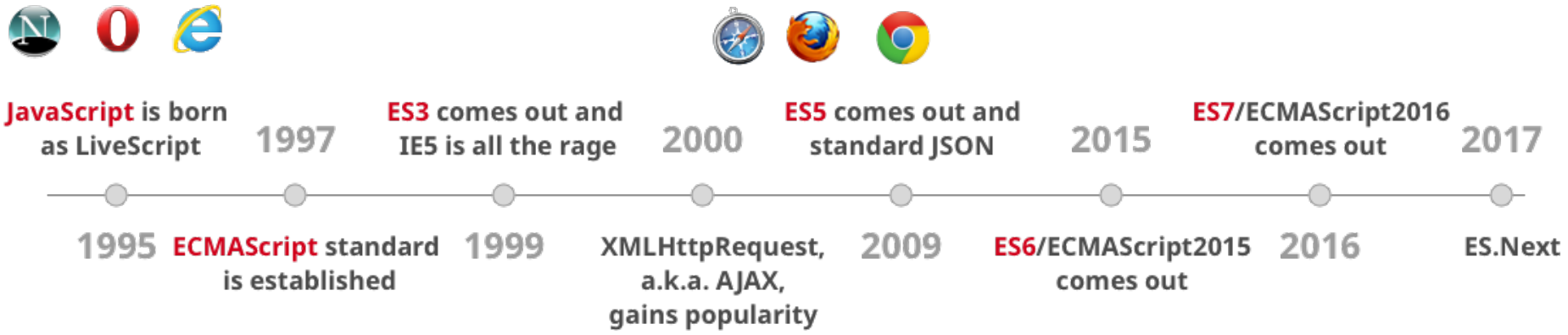
Hoofdstuk 1-2

Javascript: herhaling

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Wat is javascript?



- interpreted
  - ↔ C (compiled)
- dynamic typed
  - type checking at runtime
  - ↔ Java (static typed: type checking tijdens compilation)
- weakly typed
  - datatypes mogen door elkaar gebruikt worden:
  - implicit conversion 1+"a" → "1"+"a"
  - ↔ Python (strong typed, explicit conversion: str(1)+"a")
- ECMAScript standaard

Installatie

Webstorm

https://www.jetbrains.com/webstorm/
(student licence)

Node.js

https://nodejs.org/en/download/
(extra: https://nodejs.org/en/download/package-manager/)

Chrome

https://www.google.com/chrome/

Jetbrains IDE support

https://chrome.google.com/webstore/detail/jetbrains-ide-support/hmhgeddbohgjknpmjagkdomcpobm

JetBrains IDE Support

Offered by: www.jetbrains.com

(1) JS in browser (client side scripting)

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>index.html</title>
</head>
<body>
<script src="demo.js"></script>
</body>
</html>
```

demo.js

```
console.log("demo");
// print demo in de console
```

(1) JS in browser (client side scripting)

What is V8?

V8 is Google's open source high-performance JavaScript and WebAssembly engine, written in C++. It is used in Google Chrome, the open source browser from Google, and in Node.js, among others. It implements ECMAScript and WebAssembly, and runs on Windows 7 or later, macOS 10.12+, and Linux systems that use x64, IA-32, ARM, or MIPS processors. V8 can run standalone, or can be embedded into any C++ application.

https://v8.dev/

(2) Node.js

Cross-platform run-time environment

Gebaseerd op V8-engine 

Javascript buiten de browser
    bv. CLI-application, server-sided scripting

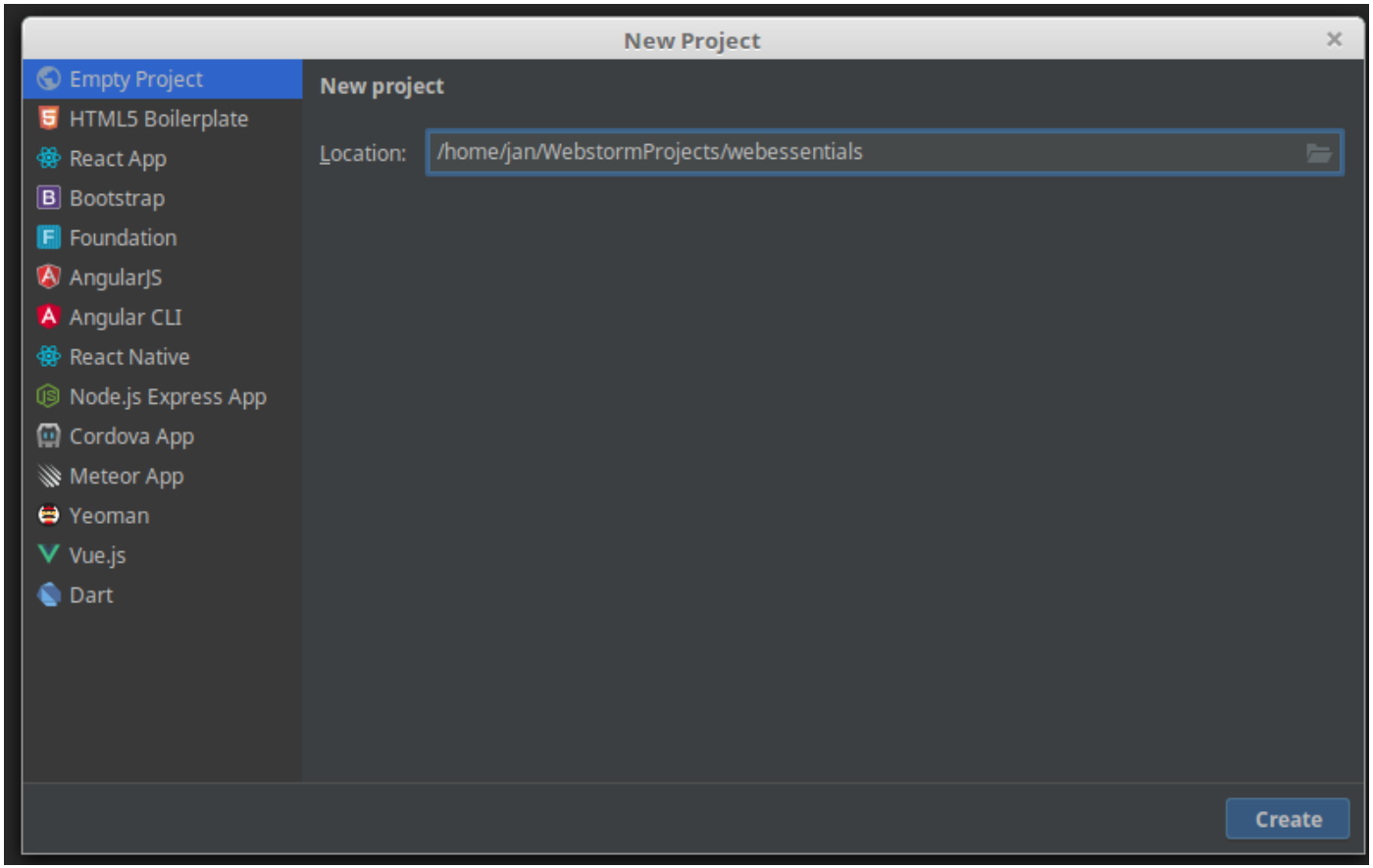
npm: node package manager
    dependencies installeren 

nvm: node version manager

Webstorm

File > New > Project

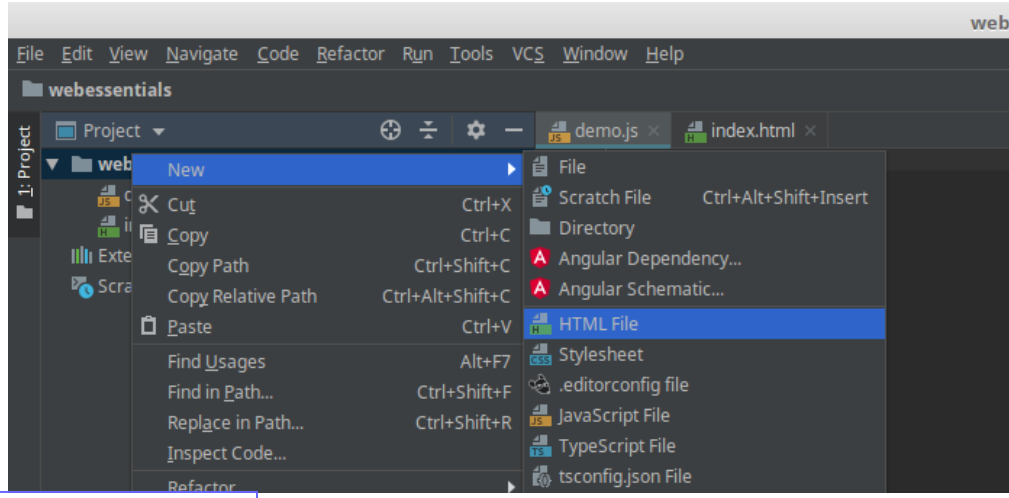
Kies voor Empty project



Webstorm

RMK op project > New > HTML File

(RMK = rechtermuisknop)

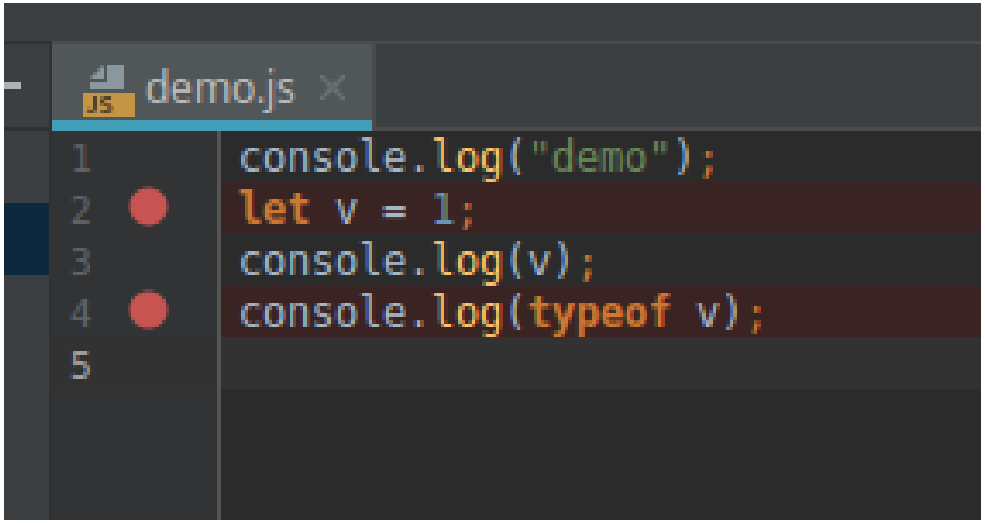


index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>index.html</title>
</head>
<body>
   <script src="demo.js"></script>
</body>
</html>
```

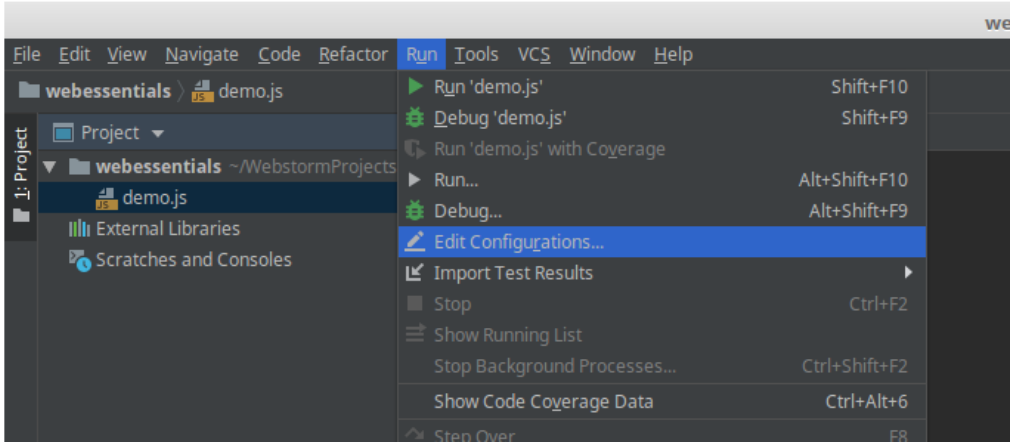
Webstorm

File > New > Javascript File
Plaats breakpoints door te klikken in de marge van demo.js

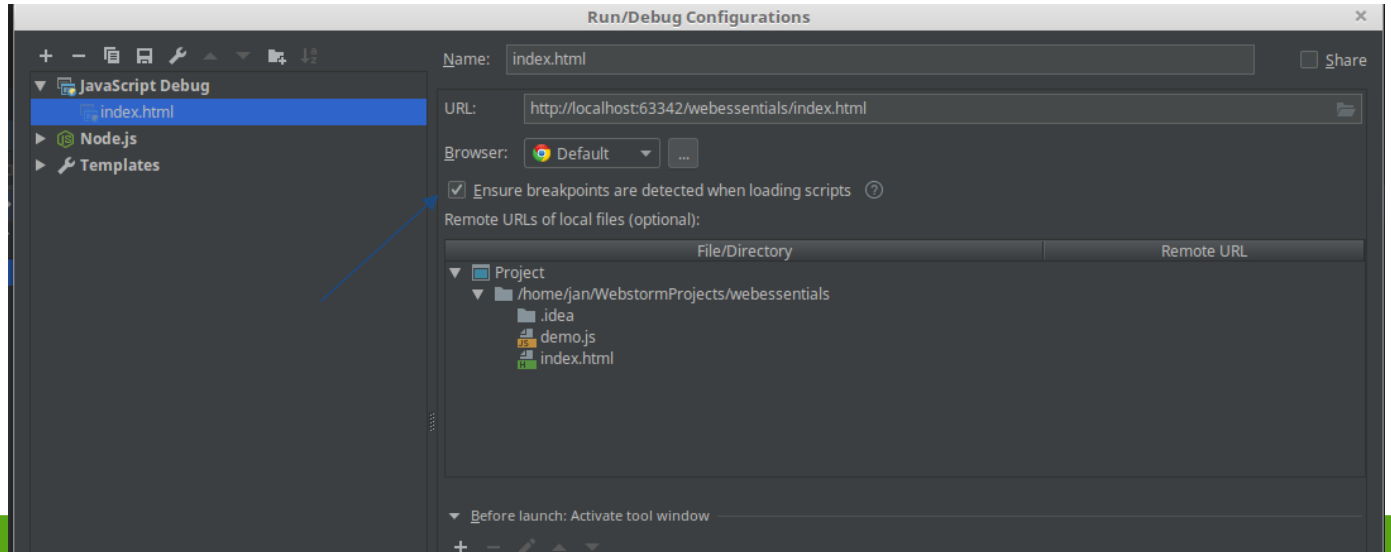


Webstorm (JS in browser)

Run > Edit Configurations



Click op + en kies JavaScript Debug



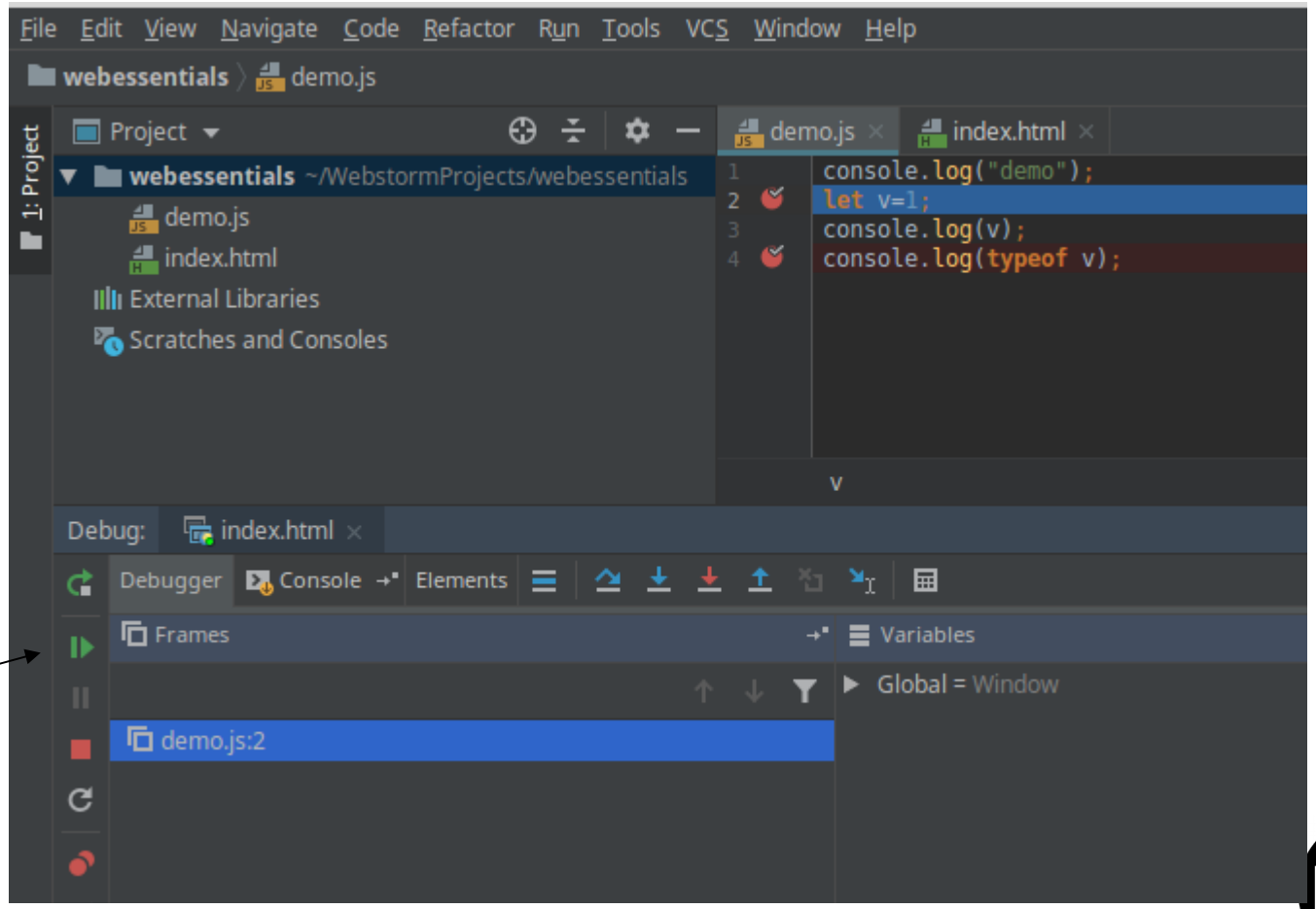
Webstorm (JS in browser)

Run > Run 'index.html'

Console (in Chrome)
- ctrl-shift-i
- klik op Console

Webstorm (JS in browser)

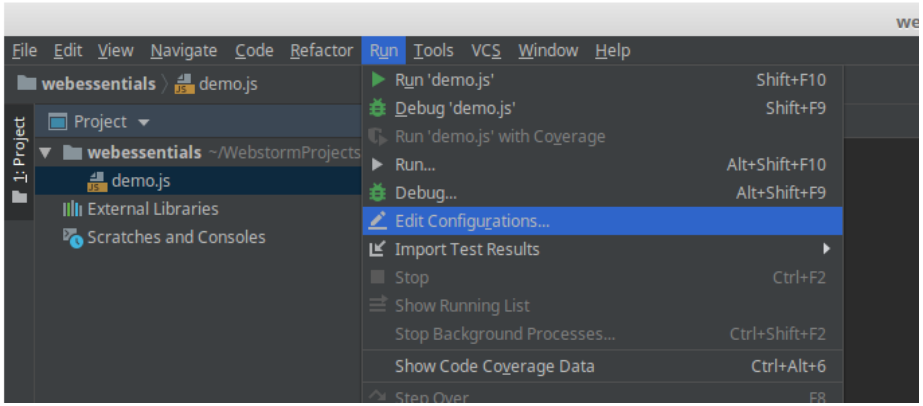
Run > Debug 'index.html'
(Browser wordt geopend)



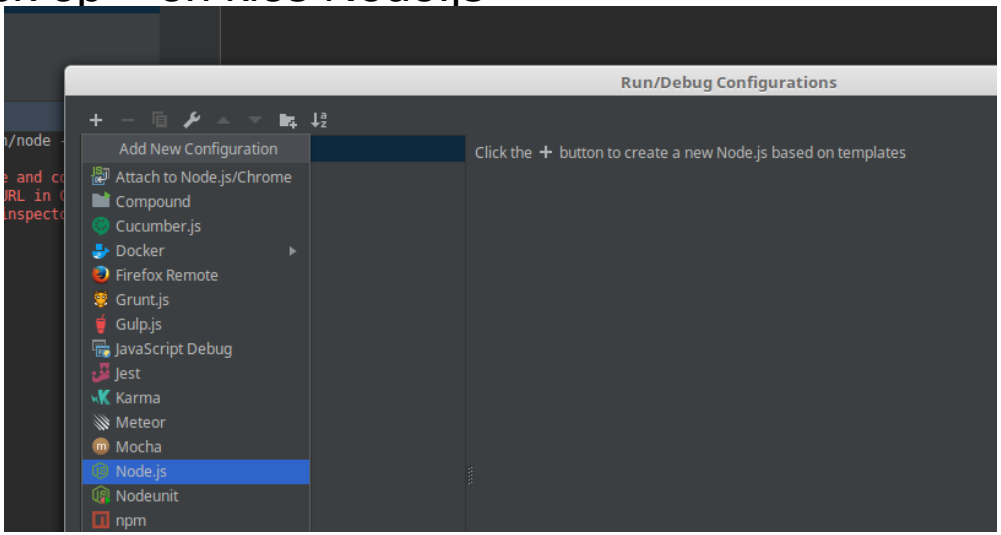
Resume

Webstorm (Node.js)

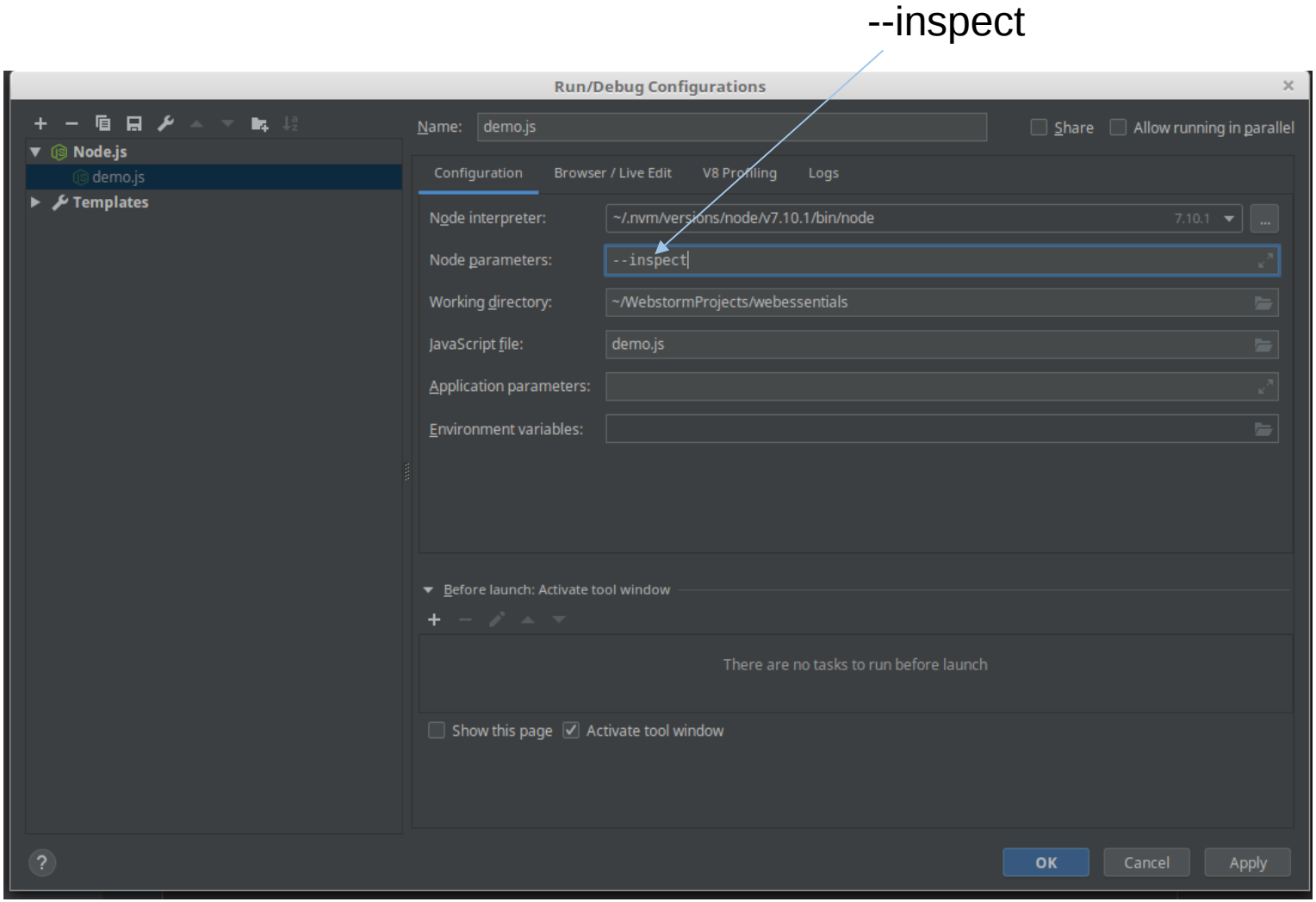
Run > Edit Configurations



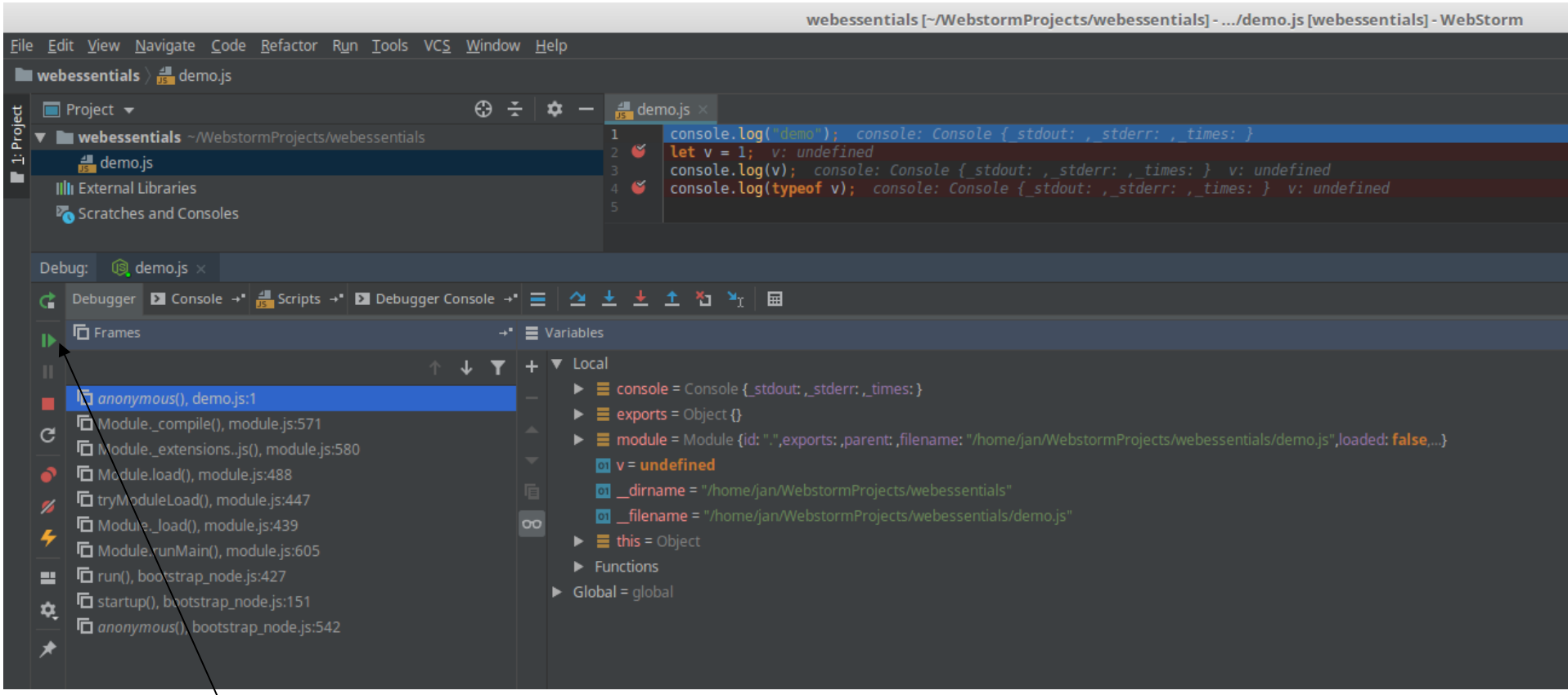
Click op + en kies Node.js



Webstorm (Node.js)



Run > Run demo.js
Run > Debug demo.js



Resume (F9)
Ga naar het volgende breakpoint

Variables / constants

let — variabele (block scoped)

declaratie & initialisatie in 1 keer

```
let aantal = 1;
```

declaratie & initialisatie verspreid

```
let waarde;
waarde = 1;
```

meerdere declaraties / initialisaties op 1 regel

```
let a, b = 1, c;
```

const — constante (block scoped)

declaratie & initialisatie in 1 keer

```
const een = 1;
```

meerdere declaraties /initialisaties

```
const twee = 2, drie = 2;
```

var — variabele (function scoped)

niet gebruiken

Types

Primitive datatypes

- number
- string
- boolean
- null
- undefined

Reference datatypes (later)

- array
- function
- object

Types: number

Geen onderscheid tussen komma- en gehele getallen

```
let number = 13;
let decimalNumber = 3.14;
let avagadroNumber = 6.0221e23;

let teGrootGetal = 1e1000;  //Infinity

if (teGrootGetal === Infinity) {
   console.log("Infinity!");
}

console.log( 1 / teGrootGetal ); // 0

console.log( 0 / 0); // NaN
```

Types: number

Bewerkingen: +, -, *, /, %

```
let number = 13;
console.log( number / 2 );  // 6.5
console.log( number % 2 );  // 1
```

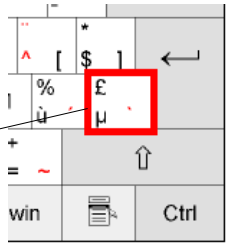
Types: string

Single & double quotes

```
let welcome = "Hello\nWorld";
```

Backticks

```
let getal=12;
console.log(`${getal} + 1 = ${getal + 1} !`);
// 12 + 1 = 13 !
console.log(`half of 100 is ${100 / 2}`);
// half of 100 is 50
```



Geen char

```
let eersteSymbool = welcome[0];
console.log( eersteSymbool);        //H
console.log(typeof eersteSymbool); // string
```

Concatenatie: +

```
console.log("a"+"bc");  //abc
```

Types: boolean

true, false

```
let goedGekeurd = true;
```

negatie: !

```
if ( !goedGekeurd ) {
```

&&, ||

```
if (name == "tim" && age > 2){
```

Types: undefined

waarde nog niet toegekend

```
let getal;
console.log(typeof getal);  // undefined
```

Types: null

```
let a = null;
```

Wrappers

Primitives: string, number

Reference types: String, Number

```
let name = "Sofie";
console.log(typeof name);     // string

let street = new String("Wetstraat");
console.log(typeof street);      // object

console.log(name.toUpperCase());
   // name wordt omgezet naar String-object
   // toUppercase van String-object wordt
   // uitgevoerd
```

Type coercion

Bij een operatie tussen één of meerdere datatypes automatische conversie van datatype

```
console.log(8 * null);  // 0 (number)

console.log("5" - 1);   // 4 (number)

console.log("5" + 1);   // 51 (string)

console.log("five" * 2);  // NaN (number)

console.log(false == 0); // true (boolean)

console.log(false === 0); // false (boolean)
```

https://delapouite.com/ramblings/javascript-coercion-rules.html

Type coercion

Truthy / falsy

```
if ( "a" ){
   console.log("de string a is truthy");
}

if ( !null ){
   console.log("null is not truthy");
}
```

http://adripofjavascript.com/blog/drips/truthy-and-falsy-values-in-javascript.html

Hoofdstuk 3

Functions

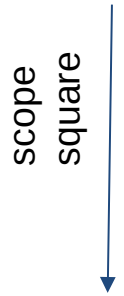
DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Definitie via binding

Binding tussen naam en stuk code

De functie is enkel beschikbaar na de definitie
const: niet wijzigbaar door toekenning
let: wel wijzigbaar




```
const square = function( x ) {
   return x * x;
};
console.log( square( 2 ) );
```

Definitie via binding

Binding tussen naam en stuk code

Functie enkel beschikbaar na de definitie
const: niet wijzigbaar door toekenning
let: wel wijzigbaar

```
let safeMode = true;
let launchMissiles = function() {
   console.log("launching missiles");
};
if ( safeMode ) {
   launchMissiles = function() {
      console.log( "nothing happens" );
   };
}
launchMissiles();
```

Definitie via declaration

Function declaration is 'hoisted' (omhooggebracht)
Je mag de functie aanroepen voor de declaratie:

```
future();
function future() {
    console.log("You'll never have flying cars");
}
```

```
You'll never have flying cars
```

~~future();~~

```
const future = function() {
    console.log("You'll never have flying cars");
}
```

```
ReferenceError: future is not defined
```

Definitie via arrow-notation

```
const power = (base, exponent) => {
    let result = 1;
    for (let i = 0; i < exponent ; i++) {
        result *= base;
    }
    return result;
}
console.log(power(2,3)); // 8
```

```
const square = (x) => {return x*x;}
console.log(square(2)); // 4
```

```
// 1 argument: haakjes mogen weg
const square2 = x => {return x*x;}
```

```
// zonder accolades: expression w. teruggeg.
const square3 = (x) => x*x;
```

Optional arguments

Extra argumenten worden genegeerd

```
function square(x) {
   return x * x;
};
console.log(square(2,"a",12223));      // 4
```

Niet gespecificeerd bij aanroepen: undefined

```
function minus( a, b ) {
   if ( b == undefined ) {
      return -a;
   } else {
      return a - b;
   }
};
console.log(minus(3));     // -3
console.log(minus(3,4));   // -1
```

Scope van bindings

var (niet gebruiken!)

- scope is function indien var in function
  global anders

- hoisting: voor executie worden enkel de declaratie
  omhoog gebracht (niet de initialisatie)

```
console.log(a);
var a = 1;
console.log(a);
```

→

```
var a;
console.log(a); // undefined
a = 1;
console.log(a); // 1
```

Scope van bindings

var (nooit gebruiken!)

- scope is function indien var in function
  global anders

- hoisting: voor executie worden enkel de declaratie
  omhoog gebracht (niet de initialisatie)

```
function test(){

      console.log(a);
      var a = 1;
      console.log(a);
}
test();
```

```
function test(){
   var a;
   console.log(a); //undef.
   a = 1;
   console.log(a); // 1
}
test();
```

```
var x = 1;
function test(){
    console.log(x);
}
test();
```



```
var x;
x = 1;
function test(){
    console.log(x); //1
}
test();
```

```
var x = 1;
function test(){

    console.log(x);
    if (1==2){
        // not executed
        var x = -1;
    }
}
test();
```

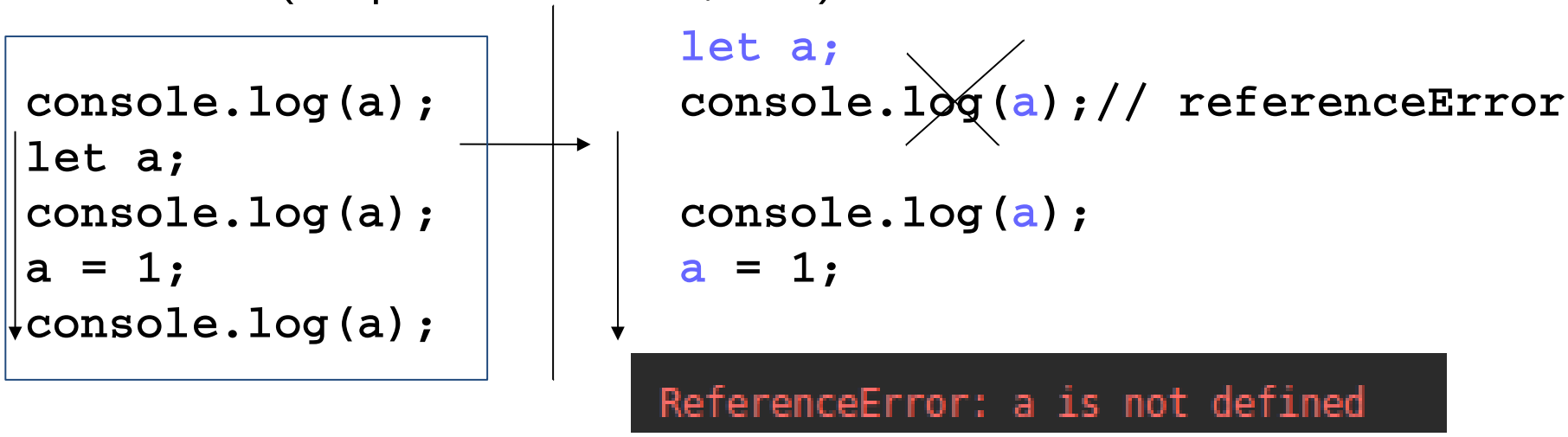


```
var x;
x = 1;
function test(){
    var x;
    console.log(x); //undefined
    if (1==2){
        // not executed
        x = -1
    }
}
test();
```

Scope

let, const:

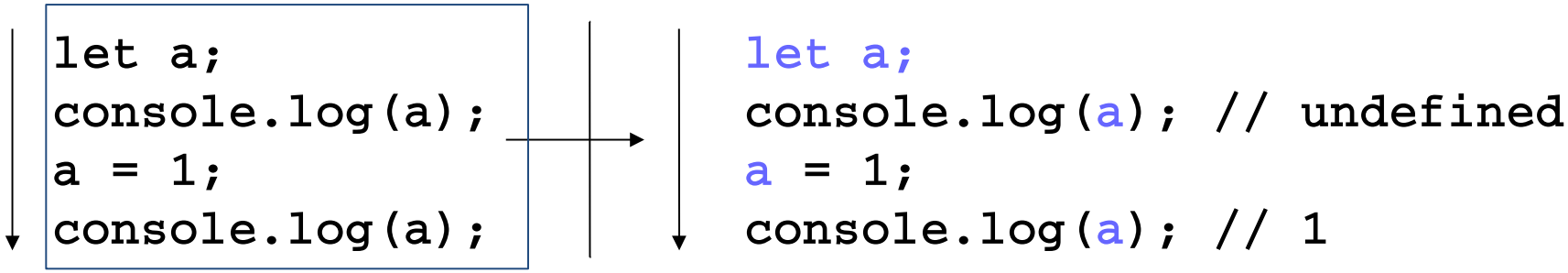
- scope is code block indien let/const in code block
  global anders
- hoisting: voor executie worden enkel de declaraties
  omhoog gebracht
- ReferenceError indien gebruikt voor declaratie
  (temporal dead zone, **TDZ**)



Scope

let, const:

- scope is code block indien let/const in code block
  global anders
- hoisting: voor executie worden enkel de declaraties
  omhoog gebracht
- ReferenceError indien gebruikt voor declaratie



```
let a = 1;
console.log(a);
{
    console.log(a);
}
```

```
let a;
a = 1;

{
    console.log(a); // 1
}
```

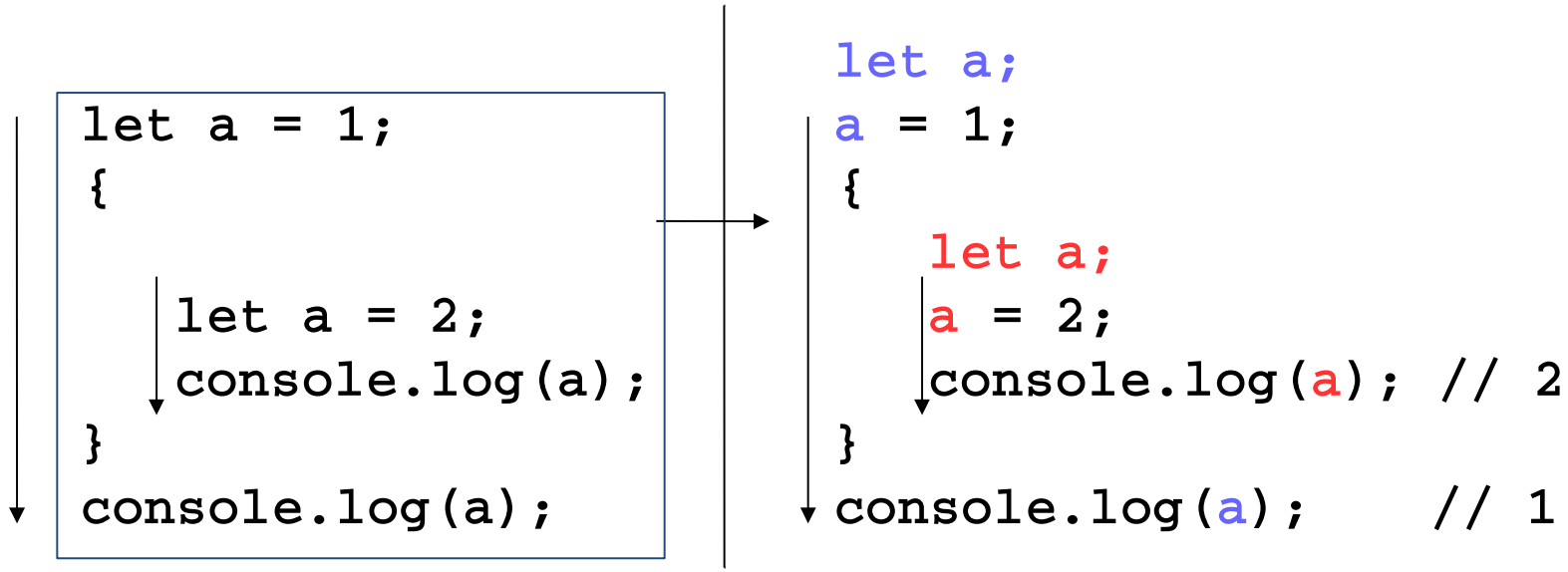
```
let a = 1;
{

    console.log(a);
    let a = 2;
    console.log(a);
}
console.log(a);
```

```
let a;
a = 1;
{
    let a;
    console.log(a);
    a = 2;
    console.log(a);
}
    console.log(a);
```

```
ReferenceError: a is not defined
```

Scope


let a = 1;
{
let a = 2;
console.log(a);
}
console.log(a);
let a;
a = 1;
{
let a;
a = 2;
console.log(a); // 2
}
console.log(a); // 1

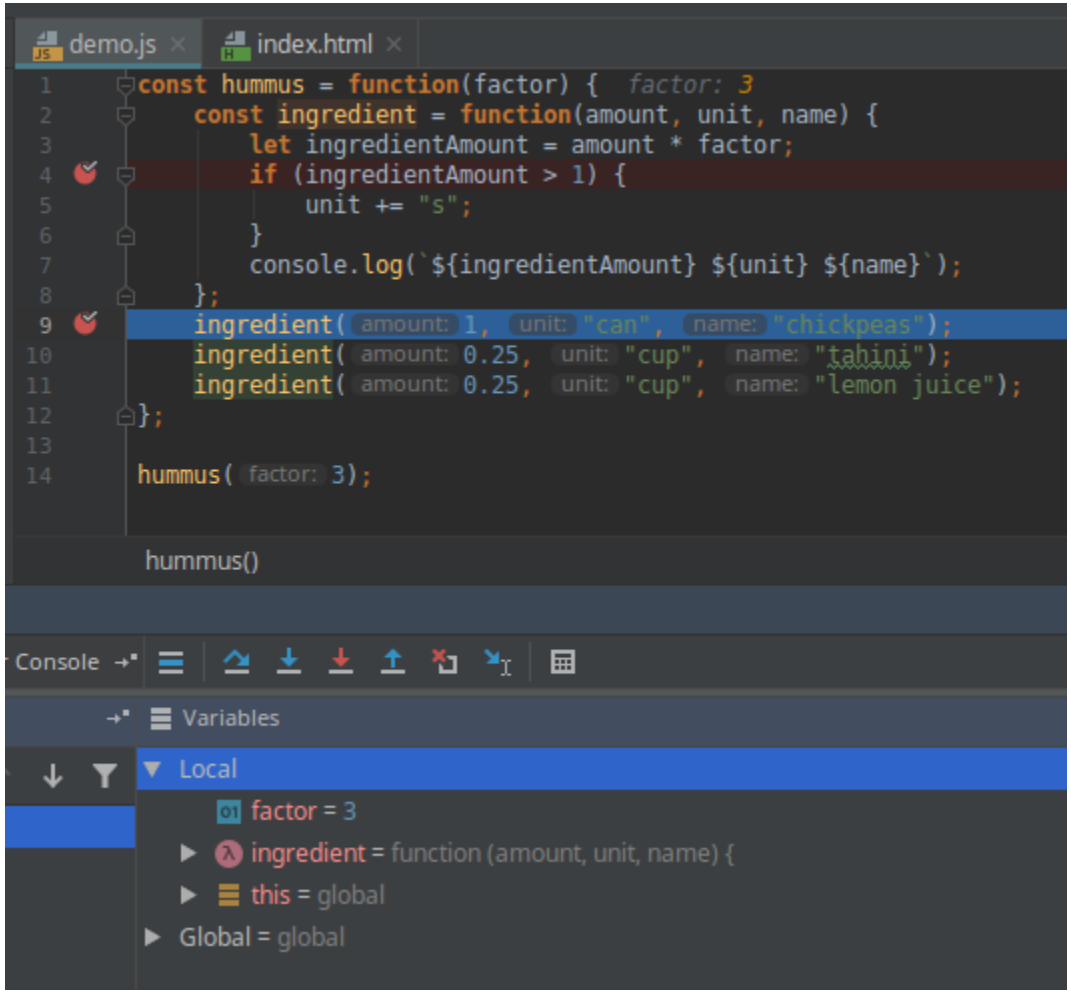
Nested scope

Function ingredient nested in function hummus

```
const hummus = function(factor) {
    const ingredient = function(amount, unit, name) {
        let ingredientAmount = amount * factor;
        if (ingredientAmount > 1) {
            unit += "s";
        }
        console.log(`${ingredientAmount} ${unit} ${name}`);
    };
    ingredient(1, "can", "chickpeas");
    ingredient(0.25, "cup", "tahini");
    ingredient(0.25, "cup", "lemon juice");
};
hummus(3);
```

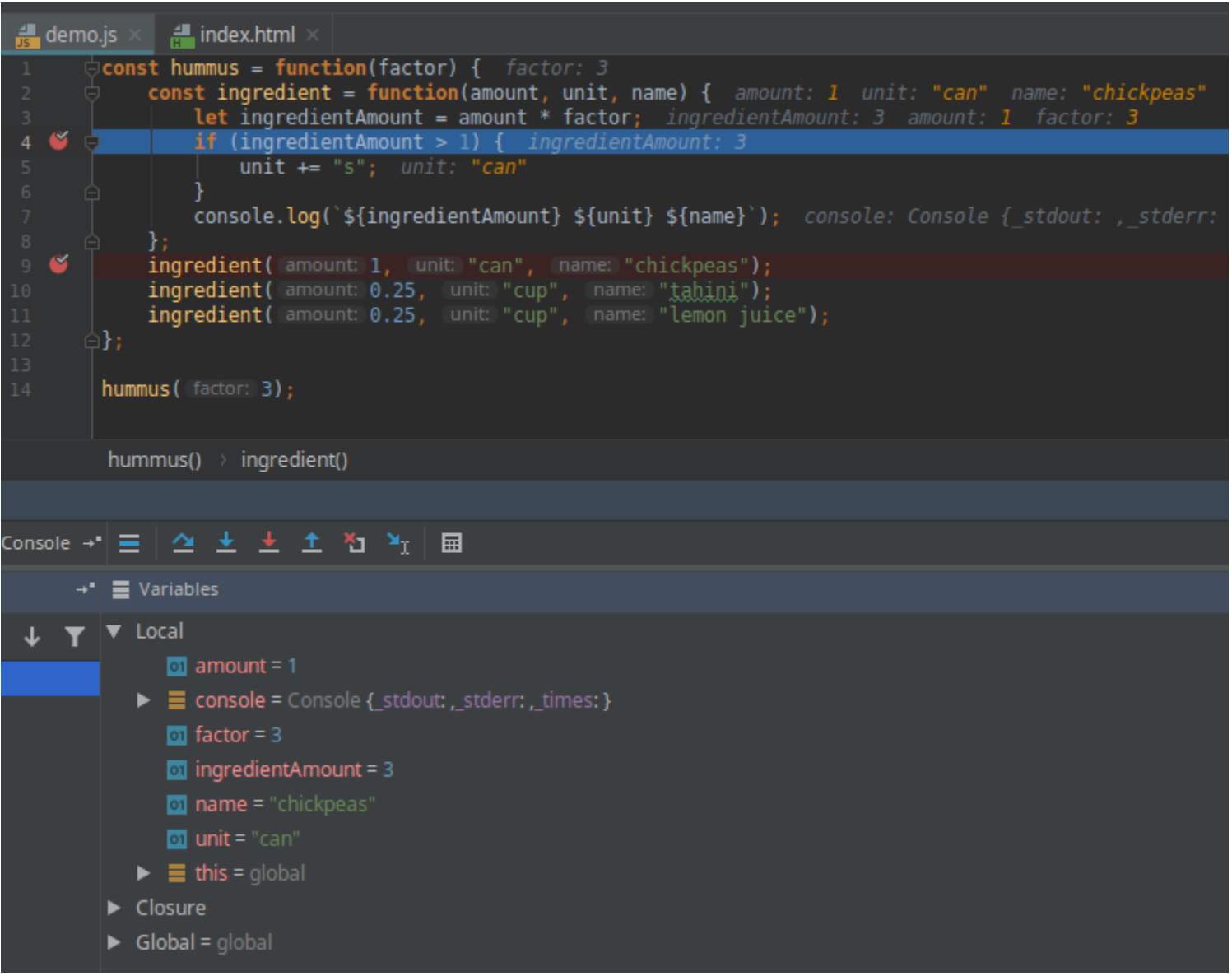
```
3 cans chickpeas
0.75 cup tahini
0.75 cup lemon juice
```

Nested scope



hummus = outer function
2 bindings in hummus:
factor & ingredient

Nested scope



ingredient is inner function

4 bindings in ingredient: amount, unit, name, ingredientAmount & factor

<u>factor afkomstig van outer function!</u>

Closures

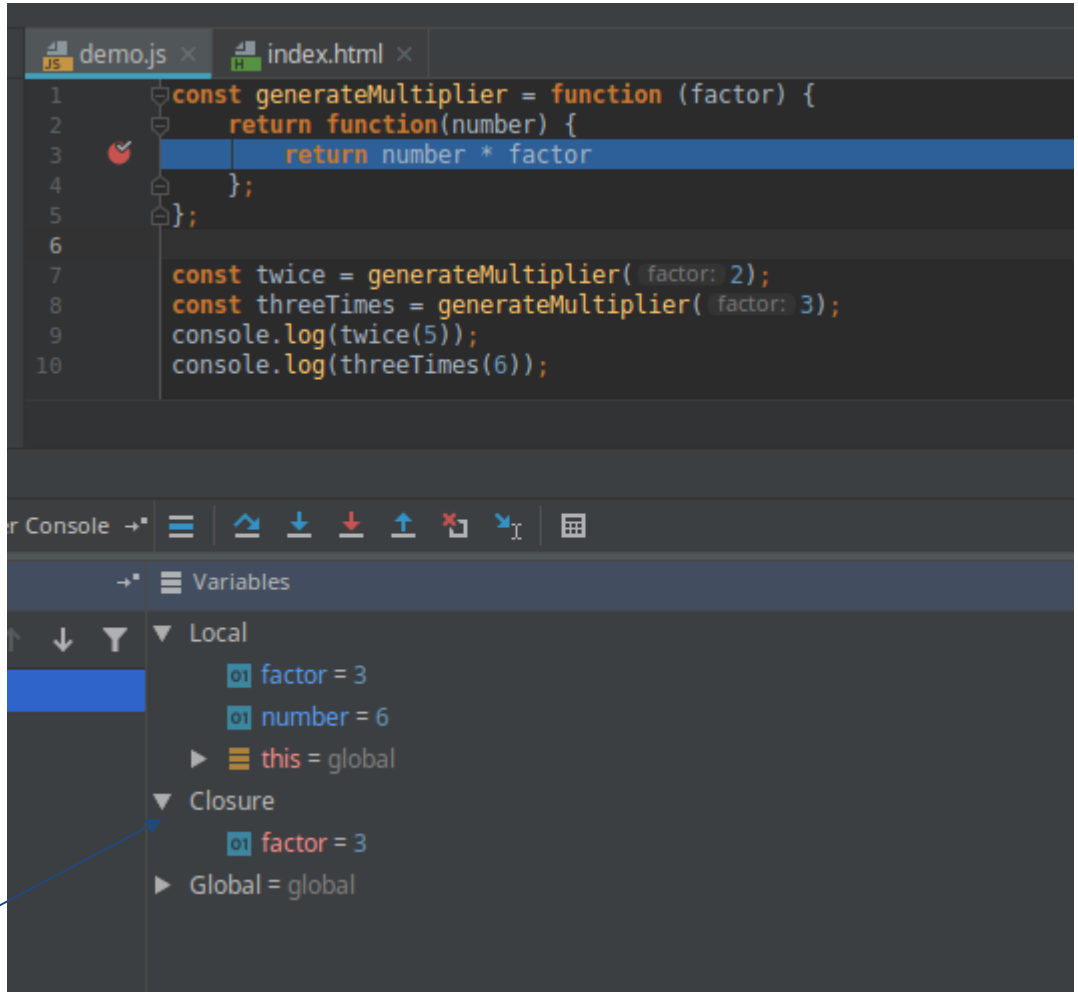
Bij een nested function heeft de inner-function ook toegang tot de variabelen van de outer function.

generateMultiplier	outer function geeft function terug
returned function	inner function heeft toegang tot binding factor

```
const generateMultiplier = function(factor) {
    return function(number) {
        return number * factor
    };
};

const twice = generateMultiplier(2);
const threeTimes = generateMultiplier(3);
console.log(twice(5)); // 10
console.log(threeTimes(6)); // 18
```

Closures


demo.js
index.html
const generateMultiplier = function (factor) {
return function(number) {
return number * factor
};
};
const twice = generateMultiplier(factor: 2);
const threeTimes = generateMultiplier(factor: 3);
console.log(twice(5));
console.log(threeTimes(6));
Console
Variables
Local
factor = 3
number = 6
this = global
Closure
factor = 3
Global = global

Hoofdstuk 4

Datastructures: Objects & arrays

DE HOGESCHOOL MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Arrays

```
let listOfNumbers = [2, 3, 5, 7, 11];
let emptyArray = [];
```

indexed properties: `array[ index ]`

```
console.log(listOfNumbers[0]); // 2
listOfNumbers[1] = 3;
```

named properties: `array.name array["name"]`

```
console.log(listOfNumbers.length);  // 5
console.log(listOfNumbers[ "length" ] );  // 5
```

Array loops

```
let numbers = [1,2,3,4];
for ( let i = 0; i < numbers.length; i++){
   console.log(numbers[i]);
}

// 'foreach'
for ( let number of numbers ){
   console.log(number);
}
```

Array methods

indexOf, lastIndexOf

```
console.log([1, 2, 3, 2, 1].indexOf(2));  // 1
console.log([1, 2, 3, 2, 1].indexOf(5));  // -1
console.log([1, 2, 3, 2, 1].lastIndexOf(2)); // 3
```

pop: verwijdert het laatste element

```
let numbers = [1,2,3];
let last = numbers.pop();  // numbers = [1,2]
                           // last = 3
```

push: plaatst een element acheraan bij

```
let numbers = [1,2,3];
numbers.push(4);     // numbers = [1,2,3,4]
// ook:
numbers[numbers.length] = 5;
// numbers = [1,2,3,4,5]
```

Array methods

slice: copy van de originele array van index1 tot index2

```
let names = ["jan","tim","sofie","geert","nele"];
console.log(names.slice(1,3));   // index 1 tot 3
      // ["tim", "sofie"]
console.log(names.slice(1));     // index 1 tot eind
      // ["tim","sofie","geert","nele"]
```

concat: samenvoegen van 2 arrays of array en element

```
let names = ["jan","tim"];
names = names.concat("sofie");
names = names.concat(["geert", "nele"]);
```

Array: spread / rest (...)

```
// rest-operator
// parameters bij aanroepen v. sum worden in de
// array numbers geplaatst
function sum( ...numbers ) {
   let sum=0;
   for (number of numbers){
      sum += number;
   }
   return sum;
}
console.log(sum(1,2,3));


let a = [1,2,3,4];
// spread-operator
// de array a wordt uitgepakt als parameters
console.log(sum(...a));
```

Objects

Verzameling van eigenschappen (properties)
property-name correspondeert met value
of met function (later)

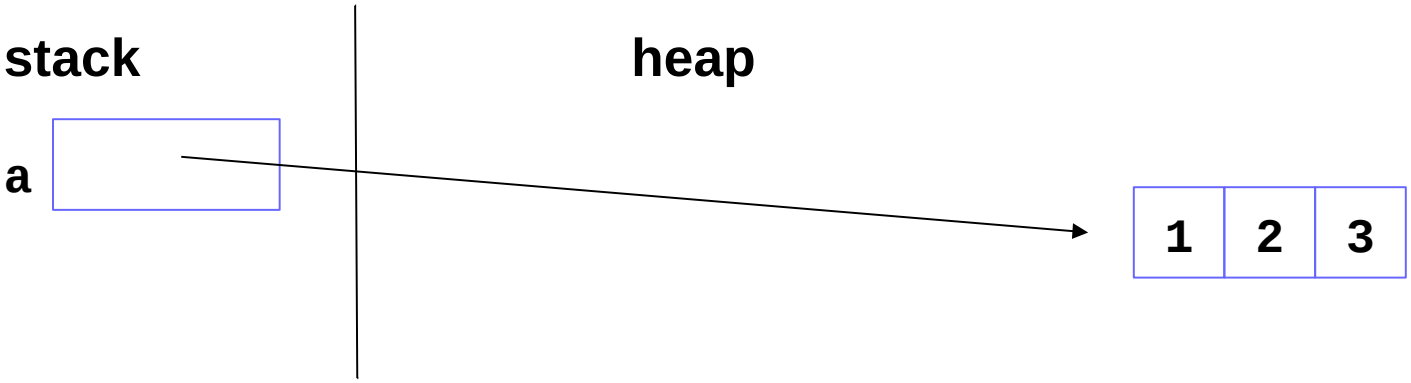
```
let person  = { name:"tim", age:22 };
console.log( person.name ); // tim
console.log( person["name"] ); // tim
person.age = 23;
person.address = "unknown";
console.log( person );
//{ name: 'tim', age: 23, address: 'unknown' }

let keys= Object.keys(person);
for(let key of keys) {
    console.log(key, person[key]);
}
name tim
age 22
address unknown
```

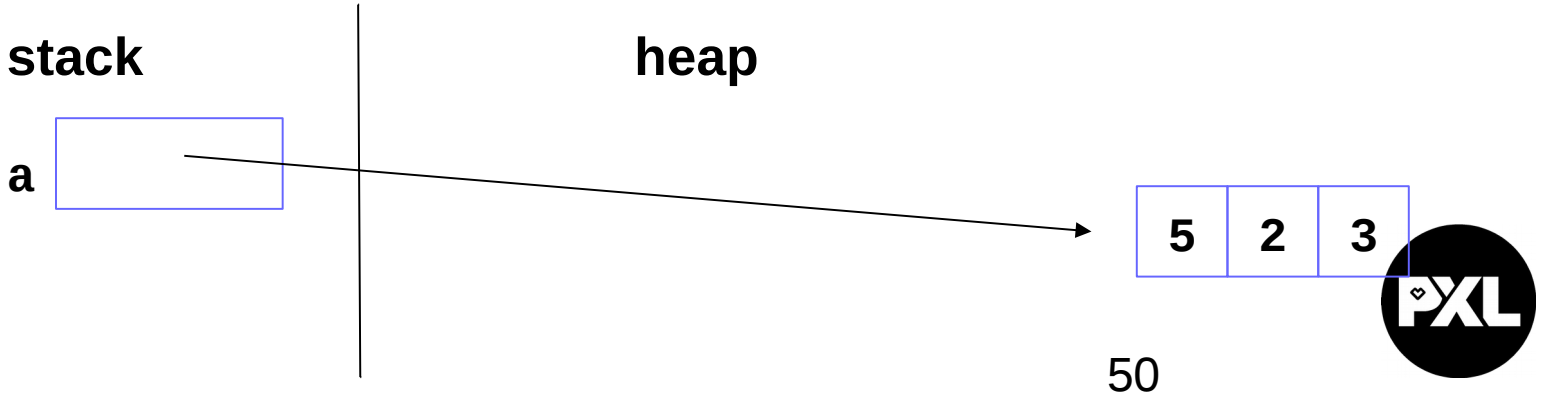
mutable vs immutable

Reference: mutable

```
let a = [1,2,3];
```



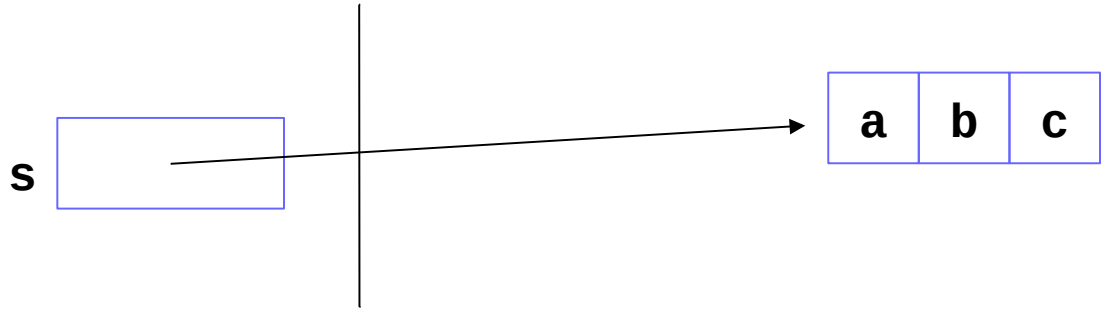

```
a[0] = 5;
```



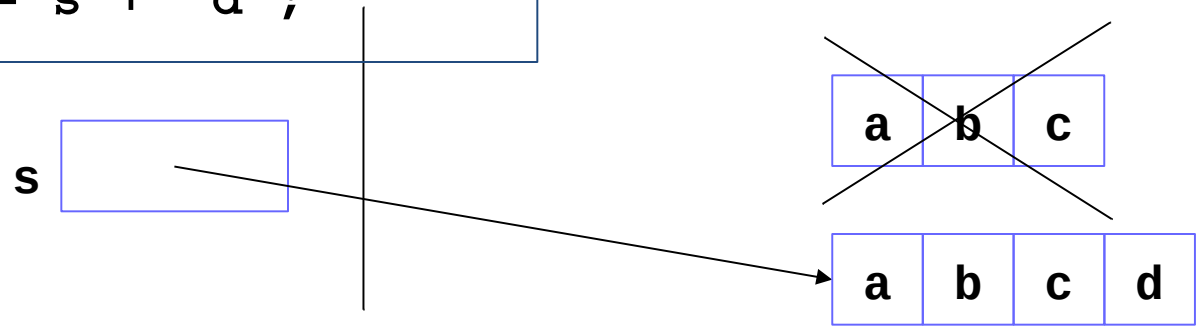
mutable vs immutable

Primitive: immutable

```
let s = "abc";
```




```
s = s + "d";
```



String functions

Primitive datatype string heeft strikt genomen geen functions!

Wanneer we een methode oproepen op een string dan wordt de omzetting naar een String-object gemaakt.

toUpperCase / toLowerCase

```
let s = "abc";
s = s.toUpperCase();
console.log( s );  // ABC
```

indexOf / lastIndexOf

```
let s = "abc";
let index = s.indexOf( "b" );
console.log( index ); // 1
```

String functions

slice: maak een substring

```
let s = "abcdef";
console.log( s.slice( 2,4 ) ); // cd
console.log( s.slice( 2 ) );   // cdef
```

trim: verwijder spaties vooraan & achteraan

```
let s = "   abc   ";
console.log( s.trim( ) ); // abc
```

split: splits een string en een maak een array

```
let s = "dit is een zin";
console.log( s.split( " " ) );
//[ 'dit', 'is', 'een', 'zin' ]
```

join: voeg een array samen in een string

```
let a = [ 1, 2, 3 ];
console.log( a.join( "," ) ); // 1,2,3
```

JSON

JavaScript Object Notation is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse (ontleden) and generate

Lijkt heel sterk op object:

```
{
    "name": "tim",
    "hobbies": ["reading", "running", "tennis"]
}
```

JSON

JSON.stringify: vorm een object om naar een JSON-string
JSON.parse: ontleed een JSON-string, maak er een object van

```
let person = {
   name: "tim",
   hobbies: ["reading", "running", "tennis"]
};
let personJSON = JSON.stringify( person );
console.log( personJSON );
let person2 = JSON.parse( personJSON);
console.log(person2);
```

```
{"name":"tim","hobbies":["reading","running","tennis"]}
{ name: 'tim', hobbies: [ 'reading', 'running', 'tennis' ] }
```

Web advanced

Hoofdstuk 14

The Document Object Model

DE HOGESCHOOL MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Document Object Model (DOM)

pagina wordt geopend in browser:

- parser ontleedt HTML-code

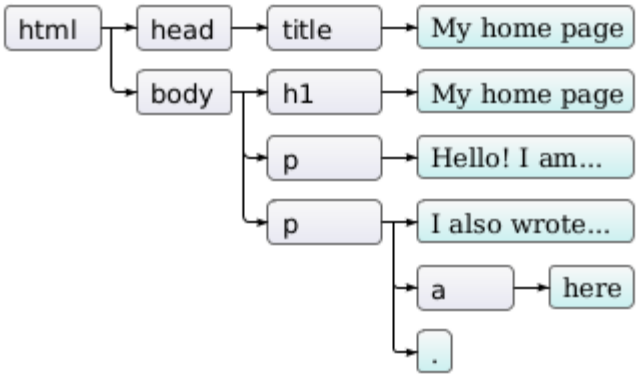
- boomstuctuur van de inhoud wordt gebouwd

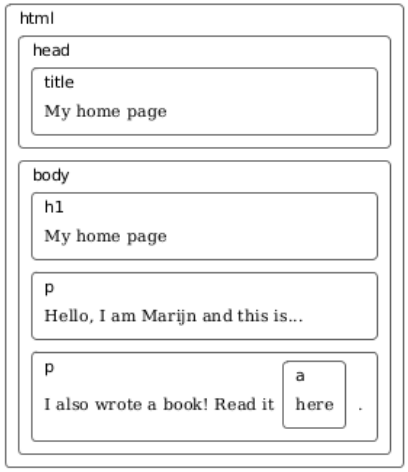
- adhv van de boomstructuur wordt een grafische weergave gegenereerd.

- de boomstructuur kan gewijzigd worden via Javascript

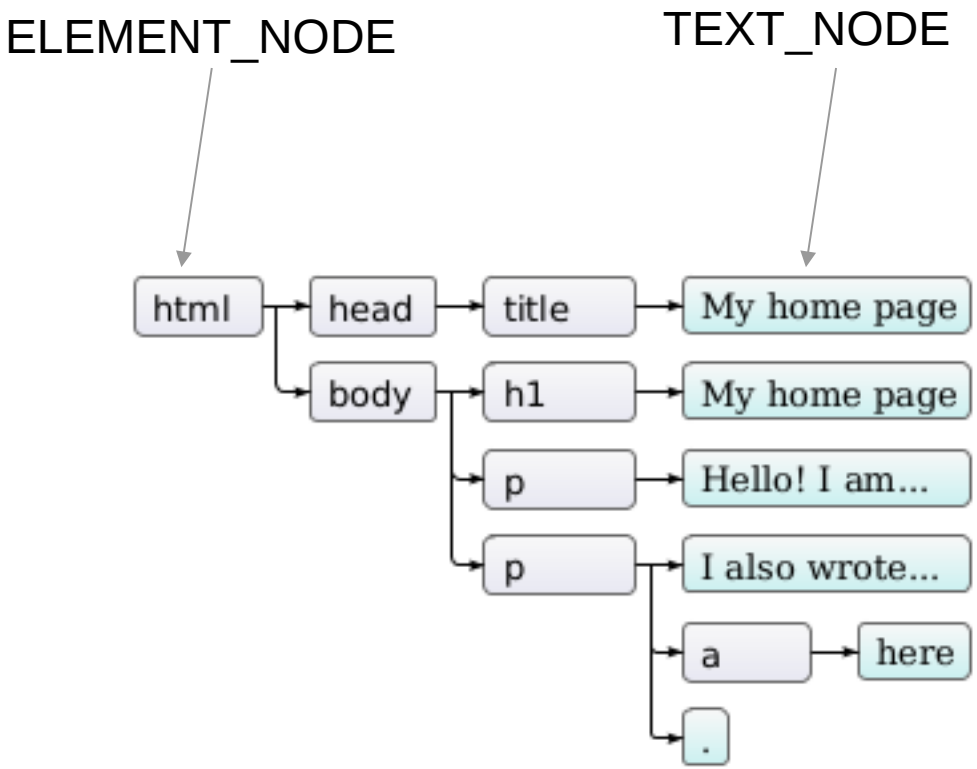
Document Object Model (DOM)

```
<!doctype html>
<html>
    <head>
        <title>My home page</title>
    </head>
    <body>
        <h1>My home page</h1>
        <p>Hello, I am Marijn and this is my home page.</p>
        <p>I also wrote a book! Read it
        <a href="http://eloquentjavascript.net">here</a>.</p>
    </body>
</html>
```





Document Object Model (DOM)



Relaties tussen ekementen:

<html> is parent van <head>, <body>

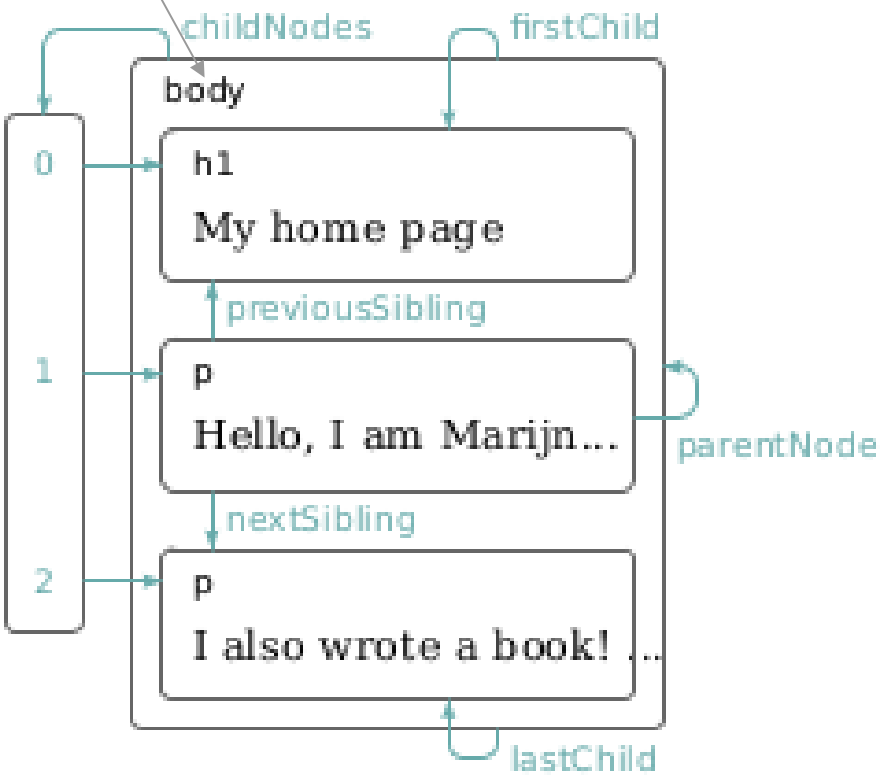
<head> & <body> zijn children van <html>

<head> & <body> zijn siblings (generatiegenoten)

Navigatie

<body> heeft

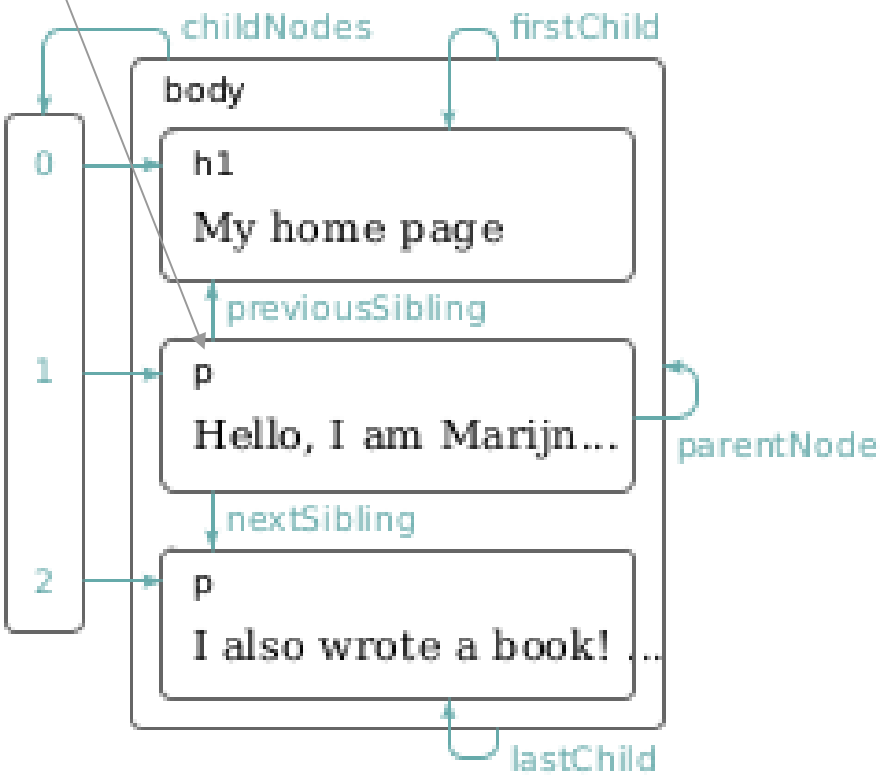
childNodes: <h1>, <p>, <p>
firstChild: <h1>
lastChild: <p>



Navigatie

<p> heeft

previousSibling: <h1>
nextSibling: <p>
parentNode: <body>



DOM: elementen zoeken

getElementById: zoek één element adhv de waarde van het attribuut id

```
<!doctype html>
<html>
    <head>
        <title>My home page</title>
    </head>
    <body>
        <p>My ostrich Gertrude:</p>
        <p><img id="gertrude" src="img/ostrich.png"></p>
        <script>
            let ostrich = document.getElementById( "gertrude" );
            console.log( ostrich.src ); // img/ostrich.png
    </script>
</body>
```

DOM: elementen zoeken

getElementsByTagName: geeft een array terug met elementen die een bepaalde naam hebben

```
<!doctype html>
<html>
    <head>
        <title>My home page</title>
    </head>
    <body>
        <a href="http://destandaard.be">de standaard</a>
        <a href="http://demorgen.be">de morgen</a>
        <script>
            let firstLink =
                document.getElementsByTagName( "a" )[0];
            console.log( firstLink.href ); // http://destandaard.be
        </script>
    </body>
</html>
```

[0]: de eerste uit de rij met <a>'s wordt geselecteerd

DOM: elementen zoeken

getElementsByClassName: geeft een array terug met gezochte elementen, de elementen worden gezocht adhv de waarde van het attribuut class

```
<!doctype html>
<html>
    <head>
        <title>My home page</title>
    </head>
    <body>
        <h1 class="red">My home page</h1>
        <p class="red">Hello</p>
        <p class="red">I wrote a book!</p>
        <script>
            let firstRed =
                document.getElementsByClassName( "red" )[0];
            console.log( firstRed.textContent ); // My home page
        </script>
    </body>
</html>
```

querySelector vs querySelectorAll

querySelector selecteer 1 element adhv van een CSS selector
(eerste element dat voldoet aan de selector)

```
let element = document.querySelector( "p" );
// eerste p-element

let element2 = document.querySelector( ".red" );
// eerste element met attribuut class="red"
```

querySelectorAll selecteer alle elementen adhv van een CSS selector

```
let elements = document.querySelectorAll( "p" );
// alle p-element

let elements2 = document.querySelector( ".red" );
// alle elementen met attribuut class="red"
```

DOM: aanpassingen

appendChild: voeg newElement toe aan de lijst van children van parentElement

```
parentElement.appendChild(newElement);
```

insertBefore: plaats in parentElement newElement voor existingElement

```
parentElement.insertBefore(newElement, existingElement);
```

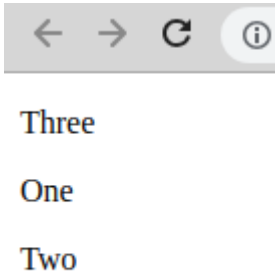
replaceChild: plaats in parentElement newElement op de plaats van existingElement (existingElement verdwijnt)

```
parentElement.replaceChild(newElement, existingElement);
```

Opgepast elke node kan maar één keer voorkomen in de boomstructuur.

DOM: aanpassingen

```
<!doctype html>
<html>
    <head>
        <title>My home page</title>
    </head>
    <body>
     <p>One</p>
     <p>Two</p>
     <p>Three</p>
     <script>
        let paragraphs = document.body.getElementsByTagName("p");
        document.body.insertBefore(paragraphs[2], paragraphs[0]);
     </script>
    </body>
</html>
```



Three wordt voor One geplaatst. Elke node mag maar één keer voorkomen dus de originele Three wordt verwijderd.

Nodes aanmaken

let node = createTextNode(text);

maak de textnode node aan de hand van de string-variabele text

```
<p>The <img src="img/cat.png" alt="Cat"> in the
<img src="img/hat.png" alt="Hat">.</p>
<p><button onclick="replaceImages()">Replace</button></p>
<script>
    function replaceImages() {
        let images = document.getElementsByTagName( "img" );
        for( let i = images.length - 1; i >= 0; i-- ) {
            let image = images[i];
            if( image.alt ) {
                let text = document.createTextNode( image.alt );
                image.parentNode.replaceChild( text, image );
            }
        }
    }
</script>
```

Doorloop alle <img> elementen, als het element een attribuut alt heeft dan wordt een textnode aangemaakt met de waarde van alt. <img> wordt vervolgens vervangen door de textnode

Nodes aanmaken

let node = createElement(name);

maak een leeg element aan met als naam de waarde vd string-variabele name

```
<blockquote id="quote">
No book can ever be finished. While working on it we learn
just enough to find it immature the moment we turn away
from it.
</blockquote>
<script>
    function elt( name, ...children ) {
        let element = document.createElement( name );
        for( let child of children ) {
            if( typeof child == "string" ) {
                element.appendChild( document.createTextNode(child) );
            } else {
                element.appendChild( child );
            }
        }
    return element;
    }
...
```

elt: maak een element met name en een lijst van kinderen.

Nodes aanmaken

```
...
document.getElementById( "quote" ).appendChild(
    elt( "footer",
         "-",
         elt("strong", "Karl Popper"),
         ", preface to the second editon of ",
         elt("em", "The Open Society and Its Enemies"),
         ", 1950"
    )
);
</script>
```

The Open Society and Its Enemies

Karl Popper

<footer> - Karl Popper , preface to the second editon of The Open Society and Its Enemies, 1950

No book can ever be finished. While working on it we learn just enough to find it immature the moment we turn away from it.
-**Karl Popper**, preface to the second editon of *The Open Society and Its Enemies*, 1950

...

Attributen

Een aantal attributen kunnen rechtstreeks aangesproken worden:

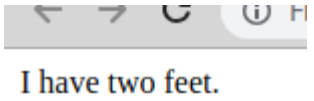
bijvoorbeeld: element.href

Alle attributen kunnen bevraagd worden via getAttribute

Alle attributen kunnen gewijzigd worden via setAttribute

Attributen

```
<!doctype html>
<html>
    <head>
        <title>My home page</title>
    </head>
    <body>
        <p data-classified="secret">The launch code is
            00000000.</p>
        <p data-classified="unclassified">I have two feet.</p>
        <script>
        let paras = document.body.getElementsByTagName( "p" );
        for( let para of paras ) {
            if(para.getAttribute("data-classified") == "secret"){
                para.remove();
            }
        }
        </script>
    </body>
</html>
```

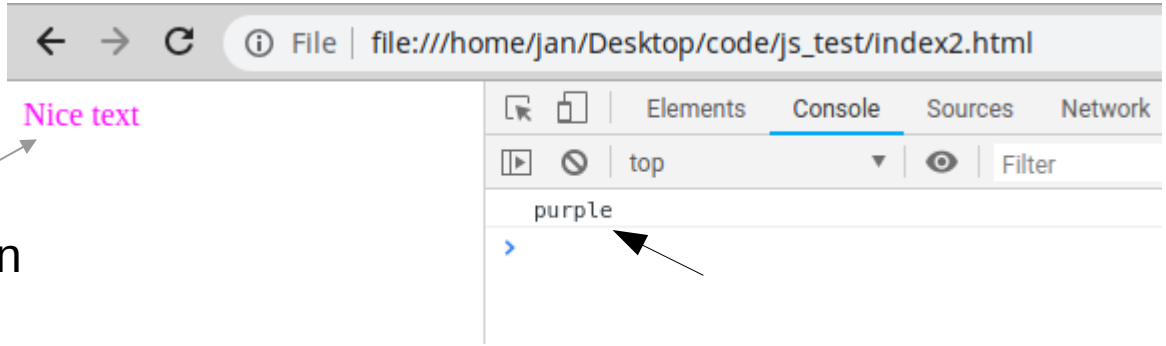


Doorloop alle <p> elementen als het attribuut "data-classified" waarde secret heeft wordt het element verwijderd.

Style

.style ~ CSS-style van een element

```
<p id="para" style="color: purple">Nice text</p>
<script>
    let para = document.getElementById( "para" );
    console.log( para.style.color );
    para.style.color = "magenta";
</script>
```



tekst wordt in magenta getoond