IK WIL

Javascript

vijfhart
IT-OPLEIDINGEN

Introductie

- Voorstelrondje
- Agenda
- Dagindeling
- Cursusmateriaal
- Doel van deze cursus

Agenda

1. Inleiding
2. Basisbegrippen, scope en console.log
3. Programmeerstructuren
4. Functies

Agenda - vervolg

5. Objecten, JSON en AJAX

6. DOM manipulatie, events, formulieren, callback functies

7. Object creatie en overerving

Dagindeling

- 8:45 -…… Begin cursusdag
  - Introductie
  - H1 Algemeen
  - H2 Hoofdstukinhoud
    - theorie
    - opdrachten maken
    - opdrachten bespreken
- 10:30-10:45 Koffiepauze
- 12:00-12:45 Lunchpauze
- 14:30-14:45 Koffiepauze
- …….-16:00 Einde cursusdag

Cursusmateriaal

- Online documentatie: mijn.vijfhart.nl
- Chrome webbrowser
  - Chrome development tools
- RAD tool: codepen.io of JSFiddle

Doel van de cursus

Van een webpagina een webapplicatie kunnen maken

Inleiding

Van web pagina tot applicatie

Client side

- **HTML**: structuur van de pagina
  - vroeger ook: opmaak
  - nu: semantiek (header, footer, article, section, nav, etc..)
- **CSS**: opmaak van de pagina
  - kleur, grootte, positie, achtergrond, etc…
- **Javascript**: gedrag van de pagina
  - validatie, interactie, feedback, etc.

Server side

- **PHP / Java / Javascript / …**
  - Vroeger ook: structuur, opmaak, gedrag
  - Nu: URL routing, database access, rendering data (JSON)

De taal Javascript

- **Variabelen**: geheugenruimte met naam en waarde
- **Operatoren**: bewerkingen (rekenkundig, logisch, etc)
- **Programmeerstructuren**: flow of control (if en else, loops, ...)
- **Arrays**: verzamelingen met index-waarde paren
- **Functies**: programma's met naam en gedrag
- **Objecten**: collectie eigenschappen, naam-waarde paren
- **DOM**: HTML pagina vertaald naar objecten in boomstructuur in geheugen (Document Object Model)
- **Events**: gebeurtenissen op de pagina waarop met Javascript functies kan worden gereageerd

Javascript in de browser

De webbrowser voert Javascript code meteen uit bij het laden van de web-pagina, van boven naar beneden.

Functies worden pas uitgevoerd als deze worden aangeroepen.

Vaak worden functies aangeroepen als reactie op een event (bijvoorbeeld: klikken op een knop).

Voorbeeld flow web applicatie

- Pagina heeft deze **structuur** (HTML)…
- en dit **uiterlijk** (CSS) …
- en deze **data** (Javascript variabelen, objecten, arrays, vaak gevuld aan de server kant uit een database)
- en deze **functies** (Javascript).
- Als veld x en y worden gevuld (**events**), voer dan functie a en b uit voor validatie.
- Als op knop k wordt geklikt, voer dan functie f uit, die deel van pagina wijzigt.

Javascript in HTML pagina

```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>Javascript in HTML pagina</title>
    <script>
       alert('Hello World');
    </script>
  </head>
  <body>
  </body>
</html>
```

Gekoppeld Javascript bestand

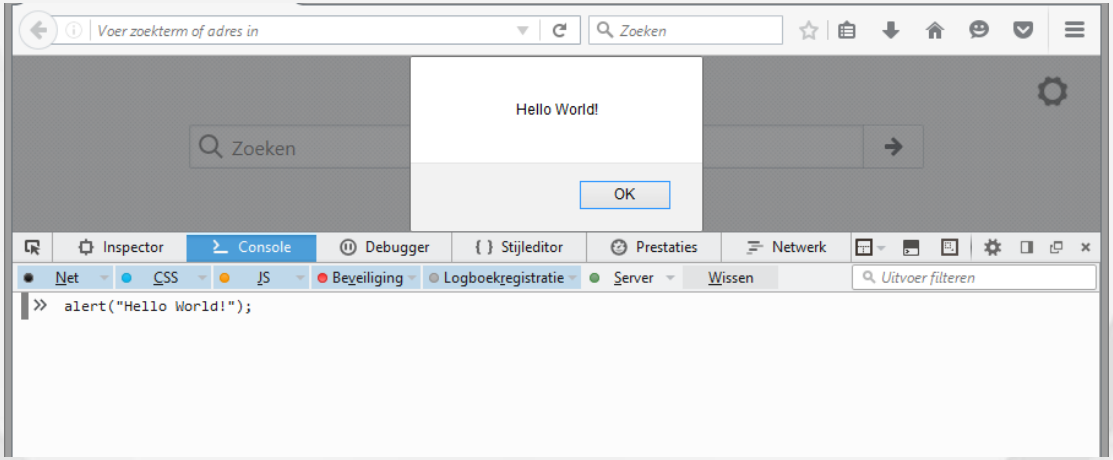
```
<!DOCTYPE HTML>
<html>
  <head>
    <meta charset="utf-8">
    <title>Javascript in extern bestand</title>
    <script src="scripts/hello.js">
    </script>
  </head>
  <body>
  </body>
</html>
```

In scripts/hello.js:

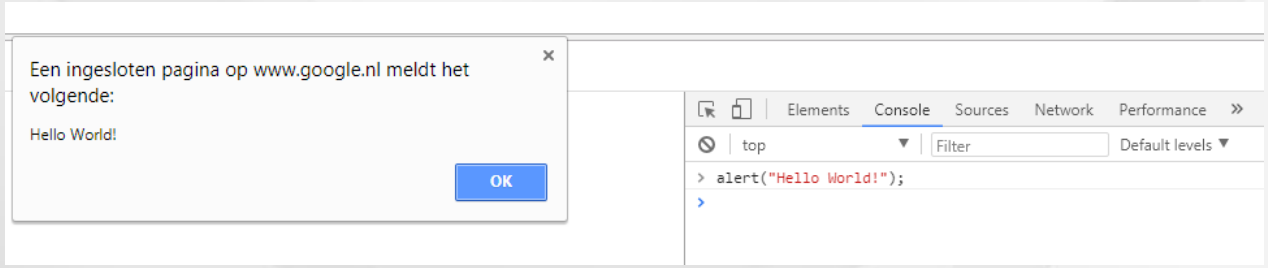
```
alert('Hello World!');
```

In de browser

Firefox Developer Tools

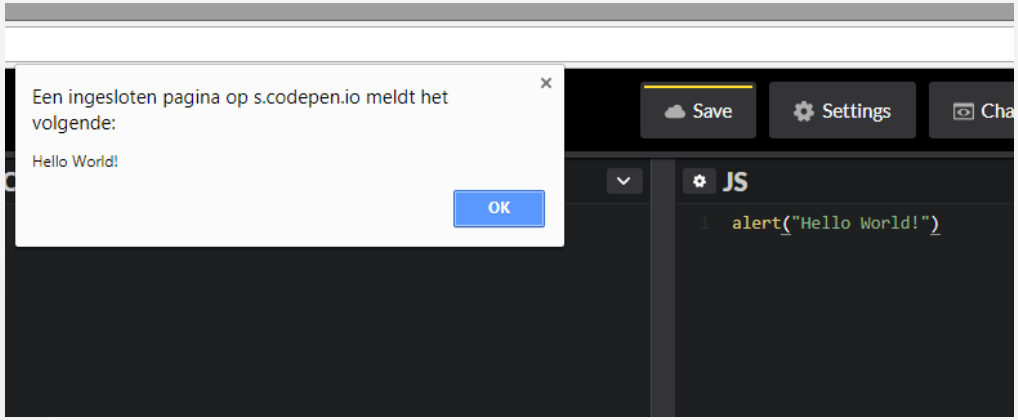


Chrome Developer Tools

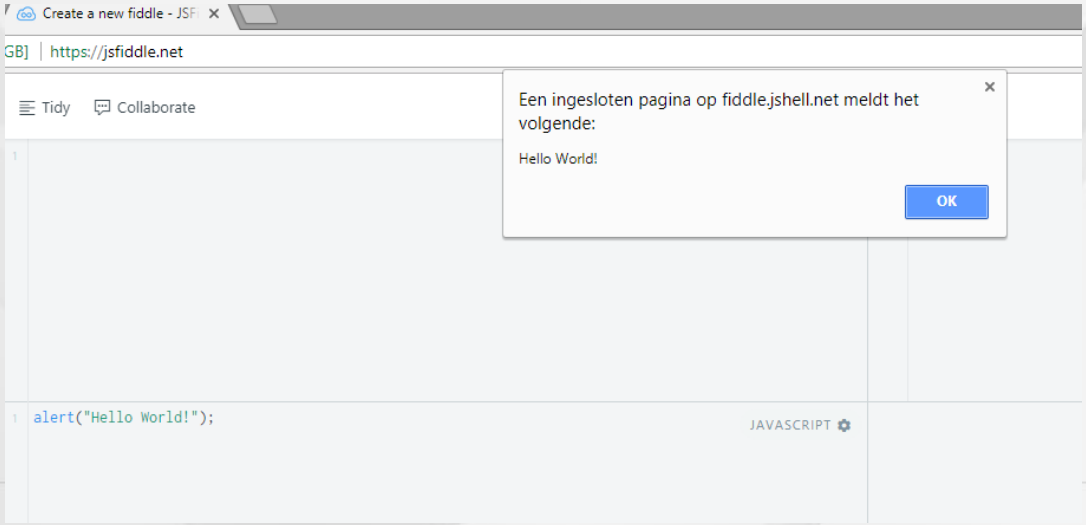


In een online RAD tool

codepen.io

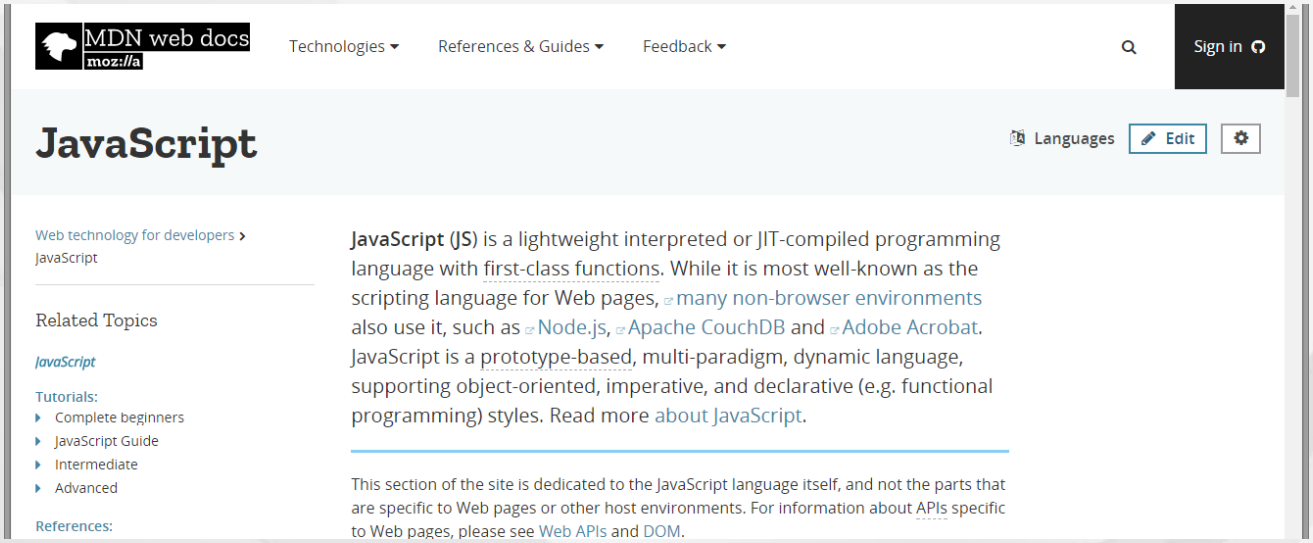


jsfiddle



Documentatie

De officiële documentatie van Javascript is te vinden op MDN (Mozzilla Developer Network)



Basisbegrippen

statements, variabelen, scope, console.log

Statements

- Een statement is een instructie, zoals: een variabele aanmaken, een variabele vullen, een melding op het scherm tonen, een waarde berekenen.
- Elk statement eindig met ;
- Als ; niet wordt toegevoegd, zal de browser deze zelf toevoegen (automatic semi-colon insertion)
- Statements worden van boven naar beneden uitgevoerd

```
var naam = "Marieke";
var som = 3 + 4;
alert("Hello World!");
```

Variabelen

Een variabele is nodig om informatie te bewaren en later te kunnen gebruiken. Een variabele:

- heeft een naam,
- wordt gedeclareerd met var <naam> of let <naam>
- kan een waarde bevatten (tot die tijd: undefined)
- kan ook leeg worden gemaakt (null)
- kan verschillende soorten data bevatten (loosely typed):
    - string: var x = "Test"; x = 'voorbeeld';
    - number: x = 10.3;
    - boolean: var afgerond = false;
    - array: var cursussen = ["Javascript","HTML", "CSS"];
    - object: var artikel = { naam: "Pen", prijs: 1.5 };

console.log

Het browser object "console" kan met de functie log(...) debug informatie naar het developer tools scherm schrijven.

Deze informatie komt niet op de webpagina te staan.

Bijvoorbeeld:

Commentaar

Het is goed gebruik om commentaar toe te voegen in Javascript code, zoals:

- doel van de code
- versie
- auteur

Alles tussen /* en */ wordt als commentaar gezien: de browser zal dat niet proberen uit te voeren

Alles op een regel achter // wordt ook als commentaar gezien

Datatypen

Er bestaan twee groepen datatypen:

- primitieve datatypen (immutable)
  - **boolean** (true of false)
  - **null** type (intentionele afwezigheid van een waarde)
  - **undefined** (nog geen waarde toegekend)
  - **number** (64 bits floating point), zoals:
    - +0, -0, NaN (not a number), -Infinity, +Infinity, 2, 20.3
  - **string**: tekstwaarden tussen enkele of tussen dubbele quotes
- complexe datatypes:
  - **arrays**
  - **objecten**
  - **functies**

Number

```
var x = 3;
var y;
console.log(y);          //  undefined
console.log(x/y);        //  NaN
console.log(isNaN(x/y)) // true
y=null;
console.log(x/y);        //  Infinity
y=-0;
console.log(x/y);        // -Infinity
y = "test";
console.log(x/y);        //  NaN
Number.isInteger(x);     //  true
y = Number.parseFloat("2.5.9");
console.log(y);          // 2.5
y = Number.parseInt("2.5.9");
console.log(y);          // 2
```

String

Waarde tussen " en ", of tussen ' en ', of tussen ` en ` (template literal, EcmaScript 2015)

```
var naam = window.prompt("Geef uw naam op: ");
var tekst = "Wij hopen dat het u " +
"goed zal bevallen.";
var alternatief = "Wij hopen dat het u \
goed zal bevallen.";

var boodschap = `Welkom op deze cursus, ${naam}.
${tekst}`;
console.log(boodschap);
```

Uitvoer, bijvoorbeeld:

```
Welkom op deze cursus, Heleen.
Wij hopen dat het u goed zal bevallen.
```

Array

Verzamelingen met index

Eerste element heeft index 0

```
var fruit = ["appels", "peren", "bananen"];
console.log(fruit[1]); // uitvoer: peren

fruit[0]="aardbeien";
fruit[3]="kersen"; // nieuwe waarde

console.log(fruit);
// uitvoer: ["aardbeien", "peren", "bananen", "kersen"]
```

Object

Verzameling naam-waarde paren

```
var artikel = {
    naam: "pen",
    prijs: 1.5
};

console.log(artikel.naam); // uitvoer: pen
artikel.prijs = 1.6;
console.log(artikel.prijs); // uitvoer: 1.6
```

Functies

Programma met naam, eventuele invoer, eventuele uitvoer.

```
function som(a, b) {
   var uitkomst = a + b;
   return uitkomst;
}

var x = som(3,4);
console.log(x); // uitkomst: 7
```

Operatoren

Operatoren maken het mogelijk om:

- variabelen te bewerken
- variabelen te vergelijken
- waarden toe te kennen aan variabelen

Bij meerdere operatoren in een statement:

operator precedence (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence)

Rekenkundige operatoren

+ (plus)
- (min)
* (keer)
/ (gedeeld door)
% (modulus, rest van een deling)
** (machtsverheffen)

%, * en / hebben voorrang op + en -

** heeft voorrang op de rest

```
console.log(3 + 4 * 5);      // uitvoer 23
console.log( (3 + 4) * 5 ); // uitvoer 35
```

Strings aan elkaar plakken

Met + kunnen strings aan elkaar worden geplakt.

Eventuele getallen worden daarbij omgezet naar String.

```
console.log(3 + " + " + 4 + " = " + 3 + 4);
// 3 + 4 = 34

console.log(3 + " + " + 4 + " = " + (3 + 4));
// 3 + 4 = 7
```

Assignment operatoren

Operator	Voorbeeld	Betekent
= (toekenning)	var x = 10; var y = 3;	x is nu 10 y is nu 3
+= (ophoging)	x+=2;	x = x + 2; (x is nu dus 12)
-= (vermindering)	x-=y;	x = x - y; (x is nu dus 9)
*= (vermenigvuldiging)	y*=x;	y = y * x; (y is nu dus 27)
%= (modulus)	y%=5;	y = y % 5; (y is nu dus 2)
/= (deling)	x/=y;	x = x / y; (x is nu dus 4.5)

++ en --

```
var x = 1;
x++; // postfixed increment
console.log(x);   // uitvoer: 2;
++x; // prefixed increment
console.log(x);   // uitvoer: 3;
console.log(++x); // uitvoer: 4;
console.log(x++); // uitvoer: 4 ???
console.log(x);   // uitvoer: 5 !
```

Bij x++ in statement: x wordt pas opgehoogd **na** gebruik oude waarde in het statement.

Hetzelfde geldt voor x-- en --x maar dan 1 eraf

Vergelijkingen

Operator	Voorbeeld	Uitleg
== (gelijk)	var x = 3, y = '3'; console.log(x==y); // true	inhoud gelijk, type mag verschillen
=== (exact gelijk)	console.log(x===y); // false	inhoud wel, maar type niet gelijk
!= (ongelijk)	console.log(x!=y); // false	inhoud niet ongelijk
!== (exact ongelijk)	console.log(x!==y); // true	inhoud niet ongelijk, maar type wel, dus true
< (kleiner dan)	console.log(x<y); // false	x niet kleiner dan y
> (groter dan)	console.log(x>y); // false	x niet groter dan y
<= (kleiner dan of gelijk aan)	console.log(x<=y); // true	x wel kleiner of gelijk aan y
>= (groter dan of gelijk aan)	console.log(x>=y); // true	x wel groter of gelijk aan y

Condities combineren

- && (logical AND)
- || (logical OR)
- ! (logical NOT)

```
var x = 3;
var y = 4;

console.log(x < 4 && y < 4);  // false
console.log(x < 4 || y < 4);  // true
console.log(!(x == 4));       // true
```

Conditional / ternary expression ? :

Een conditionele waarde.
Als de conditie voor ? true is, geef dan de waarde na ?
en anders de waarde na :

```
var x = 3, y = 4;
var max = x > y ? x : y;
// als x > y, dan max = x, anders max = y
console.log(max); // 4;
```

Genest kan ook (niet te diep voor leesbaarheid):

```
var melding = x < y ? "kleiner" :
              x > y ? "groter" : "gelijk";
console.log(melding); // kleiner
```

Booleans: truthy en falsey

Het boolean type kent maar 2 waarden: true en false.

Bij gebruik van boolean expressies (zoals &&, ||, !, en if en while e.d.) kunnen ook andere waarden zich als boolean gedragen.

falsey: **null, undefined, NaN, +0, -0, "" (empty string)**

truthy: alle andere waarden.

truc om te voorkomen dat een variabele wordt overschreven:

```
x = x || 4; // x wordt nu 4, als x nog geen waarde had
            // andere manier: x = x ? x : 4;
```

truc om te controleren of een waarde als truthy of fasey kan worden gebruikt:

x = [ ] ; // lege array

```
console.log(!!x); // niet niet x: true (dus truthy)
```

Opdracht

Vraag met window.prompt om twee getallen en tel deze bij elkaar op. Toon de som op het scherm.

tip: maak gebruik van parseFloat("...") om de invoer naar getallen om te zetten.

Opdracht

Probeer de volgende oefeningen uit:

- Vul een variabele x met een getal;
- Verhoog de waarde van x met 1;
- Toon x in een boodschap, bijvoorbeeld:

De waarde van x is nu: 6

Kunnen het verhogen en tonen ook in één keer?

- Wijzig x: tel er 1 bij op als het getal kleiner is dan 10, maak 'm anders gelijk aan 10
- Maak een array met namen
- Toon de eerste en de laatste naam

Programmastructuren

if en else, while, do while, for

Flow of control

Sommige statements moeten conditioneel worden uitgevoerd.

Keuze:

- **als** iets waar is (eenmalig): if (en else), switch
- **zolang** iets waar is: while, do while, for loop
- **voor alle waarden** in een array of object: varianten op de for loop

if

Alleen iets uitvoeren als een conditie waar is:

```
if (conditie)
  statement;
```

of beter:

```
if (conditie) {
  een of meer statements
}
```

if - voorbeeld

```
var x = parseFloat(prompt("geef een getal"));
// vraagt om invoer en zet de tekst om in een getal

if (x < 10) {
  console.log("x is kleiner dan tien!");
}

// uitvoer mogelijk: x is kleiner dan 10
```

if en else

```
var x = parseFloat(prompt("geef een getal"));
// vraagt om invoer en zet de tekst om in een getal

if (x < 10) {
  console.log("x is kleiner dan tien!");
} else {
  console.log("x is groter of gelijk aan tien!");
}

// uitvoer mogelijk: x is kleiner dan tien!
// anders: x is groter of gelijk aan tien!
```

if en else - genest

```
var x = parseFloat(prompt("geef een getal"));
// vraagt om invoer en zet de tekst om in een getal

if (x < 10) {
  console.log("x is kleiner dan tien!");
} else {
  if(x > 10) {
    console.log("x is groter dan tien!");
  } else {
    console.log("x is gelijk aan tien!");
  }
}

// uitvoer mogelijk: x is kleiner dan tien!
// of: x is groter dan tien!
// of: x is gelijk aan tien!
```

if - veel voorkomende fouten

```
var x = 3;
if (x=2){
  console.log("x is gelijk aan twee");
}
// uitvoer: x is gelijk aan twee


if(x==4);
  console.log("x is gelijk aan vier");
// uitvoer: x is gelijk aan vier
```

switch

Vergelijkt een een variabele met opgegeven waarden.

```
var x = 3;
switch(x){
  case 1: console.log(1);
  case 2: console.log(2);
  case 3: console.log(3);
  case 4: console.log(4);
}

// uitvoer: 3 EN 4!
```

Switch voert alle code uit vanaf de gevonden waarde, **ook de statements die bij volgende waarden horen**.

switch met break

Vergelijkt een een variabele met opgegeven waarden.

```
var x = 3;
switch(x){
  case 1: console.log(1); break;
  case 2: console.log(2); break;
  case 3: console.log(3); break;
  case 4: console.log(4); break;
}
// uitvoer: 3
```

Switch voert alle code uit vanaf de gevonden waarde, ook de statements die bij volgende waarden horen!

Na een break; statement wordt het switch blok verlaten

switch met default

Vergelijkt een een variabele met opgegeven waarden.

```
var x = 8;
switch(x){
  case 1: console.log(1); break;
  case 2: console.log(2); break;
  case 3: console.log(3); break;
  case 4: console.log(4); break;
  default: console.log("anders");
}
// uitvoer: anders
```

Default mag worden toegevoegd, meestal onderaan, maar mag op een andere plek. Als geen van de case waarden voldoet, wordt naar default gesprongen.

while

Voert code uit als, en zolang, de conditie true is.

Binnen blok moet conditie uiteindelijk false worden.

Waarom?

```
x = 1;
while(x<=10){
  console.log(x++);
}
// uitvoer: 1
            2
            3
            ...
            10
```

do ... while

Voert code minimaal één keer uit, en daarna als en zolang de conditie true is.

voorbeeld:

```
x = 1;
do {
    console.log(x++);
} while (x==2);
console.log(x);
```

for loop

De for loop begint met de gehele "loop administratie":

- welke variabele wordt gebruikt?
- hoe lang gaat de loop door?
- wat gebeurt na elke loop?

Veel gebruikt bij het doorlopen van een array.

```
let namen = ["Jan", "Pier", "Joris", "Corneel"];
for(let i=0;i<namen.length;i++){
  console.log(namen[i] + " heeft een baard.");
}
```

for ... of

Nieuwe manier om een array te doorlopen.

```
let namen = ["Jan", "Pier", "Joris", "Corneel"];
for(let naam of namen){
  console.log(`${naam} heeft een baard.`);
}
```

for ... in

Om de eigenschappen van een object te doorlopen.

```
var artikel = { naam: "pen", prijs: 1.5 };
for(let eigenschap in artikel){
  console.log(eigenschap, artikel[eigenschap]);
}
```

Uitvoer:

```
naam pen
prijs 1.5
```

Opdracht

Gegeven is de volgende array:

```
var prijzen = [10, 3.5, 2.7, 8];
```

Bereken de totale prijs en toon deze op de console.
Gebruik eerst de for loop.
Doe het daarna nog eens met de for ... of loop.

Opdracht

De volgende code geeft een geheel getal tussen 1 en 10:

```
var getal = Math.floor(Math.random()*10)+1;
```

Maak een programma die met
`var poging = parseInt(window.prompt("geef een getal: "))`
blijft vragen om het getal te raden, totdat het getal geraden is. Toon "hoger" als de poging te laag was, "lager" als de poging te hoog was, en anders "getal geraden in ... pogingen".

Gebruik een while loop met if statements.

Opdracht (optioneel)

Met de volgende code wordt de dag van de week opgehaald, waarbij 0 gelijk is aan zondag, en 6 aan zaterdag.

```
var date = new Date();
var dag = date.getDay();
```

Toon met behulp van switch of het weekend is of niet.

functies

Wat is een functie?

Een functie is:

- een blok van 0 of meer statements
- met een naam
- en 0 of meer parameters (invoer)
- en optionele uitvoer (return waarde)

Zo kan de groep statements vaker worden uitgevoerd.

Voorbeeld

```
function telop(a, b){
  // a en b zijn parameters, vergelijkbaar met variabelen
  // deze zijn alleen binnen de functie bekend
  return a + b;
}

console.log(telop(3,4)); // 7
console.log(telop(6,7)); // 13
```

Een functie is ook een object

Dit mag ook:

```
var telop = function(a, b){
    return a + b;
}

console.log(telop(3,6)); // 9
```

parameters zijn optioneel

```
var telop = function(a, b) {
   return a + b;
}

console.log(telop(2)); // NaN

telop = function(a, b) {
  a = a || 0;  // als a niet gevuld is: 0
  b = b || 0;  // als b niet gevuld is: 0
  return a + b;
}

console.log(telop(2)); // 2
```

Mogelijk meer invoer? arguments

```
console.log(telop(3,4,5)); // 7

var telop = function(a, b) {
  a = a || 0;  // als a niet gevuld is: 0
  b = b || 0;  // als b niet gevuld is: 0
  let som = a + b;
  let args = arguments;  // alle argumenten
  for(let i=2;i<args.length;i++){ // vanaf 3e positie
    som+=args[i];
  }
  return som;
}

console.log(telop(3,4,5)); // 12
```

alternatief: rest parameter

Nieuwe feature: de rest parameter, als laatste gedefinieerd, met ... voor de naam, vangt alle overige waarden op:

```
var telop = function(a, b, ...overige) {
  a = a || 0;  // als a niet gevuld is: 0
  b = b || 0;  // als b niet gevuld is: 0
  let som = a + b;
  let args = overige;  // overige argumenten
  for(let i=0;i<args.length;i++){ // vanaf 1e positie
    som+=args[i];
  }
  return som;
}

console.log(telop(3,4,5)); // 12
```

typeof en instanceof

Vaak is controle van datatypen van argumenten gewenst.

- Controle primitieve typen: typeof
- Controle complexe typen: instanceof

```
console.log(typeof 'test' == 'string');   //true
console.log(typeof true == 'boolean');    //true
console.log(typeof 123 == 'number');      //true
console.log(function() {} instanceof Function); //true
console.log({ x:1 } instanceof Object);   //true
console.log([1,2,3,4] instanceof Array);  //true
```

foutsituatie: throw

```
function telop(a, b){
   if(typeof a !== 'number' || typeof b!== 'number'){
     throw 'foute invoer, gebruik getallen';
   }
   return a + b;
}
console.log(telop('test', 4));
// Uncaught foute invoer, gebruik getallen
```

foutsituatie afvangen: try en catch

```
function telop(a, b){
   if(typeof a !== 'number' || typeof b!== 'number'){
     throw 'foute invoer, gebruik getallen';
   }
   return a + b;
}
try {  // iets wat mis kan gaan:
   console.log(telop(3,4));        // gaat goed
   console.log(telop('test', 4)); // gaat mis, naar catch
   console.log(telop(4,5));        // wordt niet uitgevoerd
}
catch (fout){ // als er iets mis gaat:
   console.log("Er ging iets mis: " + fout);
}
// 7
// Er ging iets mis: foute invoer, gebruik getallen
```

Meer waarden retourneren?

Een functie kan een return statement hebben.
Daarin kan maar één waarde worden geretourneerd.
Als toch meer informatie gewenst is: object retourneren.

```
function minMax(a,b){
  let minWaarde = a<b ? a : b;
  let maxWaarde = a>b ? a : b;
  return {
    min: minWaarde,
    max: maxWaarde
  };
}

let test = minMax(3,4);
console.log(test.min); // 3
console.log(test.max); // 4
```

anonieme functies

Als een functie maar op één plek wordt gebruikt, hoeft deze geen naam te hebben.
Voorbeeld: doe iets met elke waarde in een array.

```
var namen = [ "Jan", "Pier", "Joris", "Corneel" ];
function toonNaam(naam){
  console.log(naam.toUpperCase());
}
namen.forEach(toonNaam);
// toonNaam wordt uitgevoerd op elke naam
// en wordt alleen daar gebruikt

// uitvoer:

JAN
PIER
JORIS
CORNEEL
```

anonieme functies - vervolg

Als een functie maar op één plek wordt gebruikt, hoeft deze geen naam te hebben.
Voorbeeld: doe iets met elke waarde in een array.

```
var namen = [ "Jan", "Pier", "Joris", "Corneel" ];
namen.forEach(function(naam){
    console.log(naam.toUpperCase());
  });
// een anonieme functie wordt uitgevoerd op elke naam

// uitvoer:

JAN
PIER
JORIS
CORNEEL
```

arrow functies

Als een functie maar op één plek wordt gebruikt, hoeft deze geen naam te hebben.
Voorbeeld: doe iets met elke waarde in een array.

```
var namen = [ "Jan", "Pier", "Joris", "Corneel" ];
namen.forEach(naam => console.log(naam.toUpperCase()));
// een arrow functie wordt uitgevoerd op elke naam

// uitvoer:

JAN
PIER
JORIS
CORNEEL
```

closures

Een functie kan ook een functie retourneren.
Deze houdt toegang tot de context waarin de geretourneerde functie is aangemaakt.

```
function counter(startWith, step){
   var result;
   return function(){
     result= result==undefined? // bestaat result nog niet?
             startWith:         // begin dan met startwaarde
             result+step;       // hoog anders op met step
     return result;
   };
}
var teller = counter(0,2); // teller is een functie
                           // die bij elke aanroep de
                           // variabele result kan lezen
console.log(teller()); // 0
console.log(teller()); // 2
console.log(teller()); // 4
```

IIFE

IIFE: Immediately Invoked Function Expression
Anonieme functie die meteen eenmalig wordt aangeroepen.
Onder meer om global scope schoon te houden.
Vaak ook een closure.

```
var teller = (function (startWith, step){
   var started = false;
   return function(){
     if(started) return startWith+=step;
     started=true;
     return startWith;
   };
})(1,2);
// nu bestaat alleen de functie teller in global scope
console.log(teller()); // 1
console.log(teller()); // 3
console.log(teller()); // 5
```

scope en hoisting - global scope

```
<script>
  console.log(x);  // uitvoer: undefined, geen foutmelding
  var x = 3;       // x heeft hier "global scope"
</script>
```

Dit wordt door de browser vertaald naar:

```
<script>
  var x;  // gedeclareerde variabele wordt
          // "omhooggetakeld" naar boven
  console.log(x);
  x = 3;
</script>
```

scope en hoisting - global scope

```
<script>
  toonGroet(); // uitvoer: Hallo!
  function toonGroet(){
    console.log("Hallo!");
  }
<script>
```

Nu wordt de hele functie opgetakeld:

```
<script>
  function toonGroet(){
    console.log("Hallo!");
  }
  toonGroet(); // uitvoer: Hallo!
<script>
```

scope en hoisting - local scope

```
<script>
  console.log(x);     // foutmelding: Uncaught ReferenceError:
  function test(){    //               x is not defined
    console.log(x);   // geen foutmelding: undefined
    var x = 3;        // x heeft hier "function scope"
  }
</script>

<script>
  console.log(x);
  function test(){
    var x;  // gedeclareerde variabele wordt
            // "omhooggetakeld" naar bovenin de functie
    console.log(x);
    x = 3;
  }
</script>
```

scope en hoisting: block scope?

```
<script>
  console.log(x);  // uitvoer: undefined, geen foutmelding
  var y = 1;
  if(y==3){
     var x = 3;       // x heeft hier "global scope"
  }
</script>
```

Dit wordt door de browser vertaald naar:

```
<script>
  var y;
  var x;
  console.log(x);  // uitvoer: undefined, geen foutmelding
  y = 1;
  if(y==3){
    x = 3;       // x heeft hier "global scope"
  }
</script>
```

scope en hoisting: block scope

```
<script>
  console.log(x);      // foutmelding
  var y = 1;
  if(y==3){
     console.log(x);   // foutmelding
     let x = 3;        // x heeft hier "block scope"
  }
</script>
```

Nieuw sinds EcmaScript 6: **let** en **const**
Nu wordt x niet opgetakeld,
en is x alleen binnen het if blok bekend.

Alternatief met block scope: **const** x = 3;
(dan kan aan x geen andere waarde worden toegekend).

Opdracht

Maak een functie maal(a, b) die a en b met elkaar vermenigvuldigt.

Zorg ervoor dat het ook werkt als b niet is opgegeven (welke waarde krijgt b als b niet wordt opgegeven?)

Optioneel: maak het ook mogelijk om meer waarden op te geven, bijvoorbeeld:

```
var result = maal(3,4,5);

console.log(result); // 60
```

Opdracht

Gegeven is deze array:

var getallen = [1,3,5,7,9];

Doorloop deze array met getallen.forEach(...) en toon elk getal verdubbeld. Doe dit met (naar keuze):

- een zelf gedefinieerde functie
- een anonieme functie
- een arrow functie

Opdracht

Maak een functie maal(a, b) die het volgende teruggeeft:

- maal(3,4) : 12
- maal(3,'a'): aaa
- maal('test', 2): testtest
- maal('a', 'b'): foutmelding: geef minstens één getal op.

Maak hierbij gebruik van typeof en throw. Voor de herhaling van tekst kan repeat() van String worden gebruikt (zie documentatie op MDN).

Opdracht (optioneel)

Maak een IIFE die een functie oplevert die een totaal bijhoudt, zoals in het volgende voorbeeld:

```
var totaal = (...)();

console.log(totaal(3)); // 3
console.log(totaal(1)); // 4
console.log(totaal(2)); // 6
```

Objecten, JSON en AJAX

Objecten

Objecten zijn verzamelingen eigenschappen met een naam, en waarden.
De waarden kunnen primitieve waarden, arrays, functies en objecten zijn.

Oude manier:

```
var artikel = new Object();
artikel.naam = "pen";
artikel.prijs=1.5;
```

Nieuwe manier, de JSON notatie:

```
var artikel = {
   naam:"pen",
   prijs: 1.5
}

console.log(artikel.naam);// pen
```

Objecten met functies

```
var artikel = {
  naam: "pen",
  prijs: 1.5,
  totaal: function(aantal){
    return this.prijs * aantal;
  }
}
// this.prijs: de prijs van het object
// waar de functie totaal toe behoort

console.log(artikel.totaal(3)); // 4.5
```

eigenschappen benaderen

Een eigenschap wordt direct benaderd via punt notatie:

```
var artikel = {
  naam: "pen",
  prijs: 1.5,
  totaal: function(aantal){
    return this.prijs * aantal;
  },
  type: { punt: "fijn",
          kleur: "zwart" }
}
console.log(artikel.type.kleur); // zwart
```

eigenschappen indirect benaderen

```
var artikel = {
  naam: "pen",
  prijs: 1.5,
}
console.log(artikel["naam"]);              // pen

function getWaarde(ding, eigenschap){
  return ding[eigenschap];
}
console.log(getWaarde(artikel, "prijs")); // 1.5

for(let prop in artikel){
  console.log(artikel[prop]);
}
// pen
// 1.5
```

Constructor function

Een functie kan met **new** <functie> worden gebruikt om een object aan te maken.

Deze functie hoort geen return waarde te hebben.

Wat in de functie aan this wordt toegekend, wordt een eigenschap van het nieuwe object.

Conventie: de naam begint met een hoofdletter.

```
function Artikel(naam, prijs){
  this.naam=naam;
  this.prijs=prijs;
}

var a = new Artikel("pen", 1.5);
console.log(a.naam, a.prijs); // pen 1.5
var b = new Artikel("potlood", 0.5);
console.log(b.naam, b.prijs); // potlood 0.5
```

prototype

Objecten gemaakt met een functie krijgen een eigen prototype object.

Eigenschappen (met name functies) die voor elk object van dat type hetzelfde zijn kunnen aan het prototype worden toegekend.

```
Artikel.prototype.totaal = function(aantal){
  return this.prijs * aantal;
};
var c = new Artikel("gum", 0.8);
console.log(c.totaal(2));  // 1.6
```

Nieuw: class

Met class kan automatisch een constructor function worden aangemaakt, met functies die in het prototype terecht komen:

```
class Artikel {
  constructor(naam, prijs){ // geen keyword function
    this.naam = naam;
    this.prijs = prijs;
  }
  totaal(aantal){             // geen keyword function
     return this.prijs * aantal;
  }
}
```

Date

Met de constructor functie Date kan met datums en tijden worden gewerkt.

```
var nu = new Date();
console.log(nu.getFullYear()); // 2017
console.log(nu.getMonth());    // 11 (0-based)
console.log(nu.getDate());     // 28
console.log(nu.getHours());    // 12
console.log(nu.getMinutes());  // 53

var nieuwjaar = new Date(2018,0,1);
```

Date heeft ook set-functies om een Date object te wijzigen.

String properties en functies

String heeft behalve de property length ook veel functies.

Deze zijn terug te vinden om MDN (zoek op "MDN javascript String")

```
var tekst = "Dit is een voorbeeld.";
console.log(tekst.length);         // 21
console.log(tekst.toUpperCase()); // DIT IS EEN VOORBEELD.
console.log(tekst.charAt(2));      // t
console.log(tekst.split(" "));
   // ["Dit", "is", "een", "voorbeeld."]
console.log(tekst.substr(4,2));    // is
console.log(tekst.substring(4,6));// is
console.log(tekst.slice(-6,-1));   // beeld
```

arrays

Een array is een verzameling waarden. Dit kunnen verschillende typen waarden zijn, bijvoorbeeld:

```
var test = [1, "test", {min: 1, max: 4}, [2,3,4]];

console.log(test[0]);    // 1
console.log(test[3][0]); // 2
console.log(test[2].max);// 4
```

Een array is ook aanpasbaar, bijvoorbeeld:

```
test[9]=10;  // test.length is nu 10,
// waarden tussen test[3] en test[9] zijn undefined
test.length=2; // test is nu [1, "test"];
```

array methods

```
var fruit = ["appels", "peren", "mandarijnen", "bessen"];
fruit.push("bananen"); // voegt toe aan het eind
fruit.unshift("kersen"); // voegt toe aan het begin
let laatste = fruit.pop(); // verwijdert laatste
let eerste = fruit.shift(); // verwijdert eerste
fruit.splice(2,1); // verwijdert element op 3e positie
fruit.splice(1,0,"aardbeien"); //voegt op 2e positie toe
fruit.splice(1,1,"perziken"); //vervangt op 2e positie
console.log(fruit.slice(1,-1)); // 2e tot voorlaatste
var copy = fruit.slice(); // geen argumenten:
                          // copy van hele array
console.log(fruit.indexOf("perziken")); // 1
```

array arrow functions

```
var fruit = ["appels", "peren"];
fruit = fruit.map(naam => naam.toUpperCase());
fruit.forEach((naam, i) => console.log(`${i}: ${naam}`));
// uitvoer
0: APPELS
1: PEREN
var getallen = [1,3,5,2,7,9,10,6];
var somEven = getallen.filter(n => n%2==0)
                      .reduce((a,b) => a+b);
console.log(somEven); // 18
getallen.sort(); // [1,10,2,3,5,6,7,9]
getallen.sort((a,b) => a-b); // [1,2,3,5,6,7,9,10]
```

arrays - string - array

```
fruit = ["ananas", "mango", "pruim"];
console.log(fruit.join(" - ")); // ananas - mango - pruim
console.log(fruit.join());      // ananas,mango,pruim
console.log(fruit.join(""));    // ananasmangopruim
console.log(fruit.toString());  // ananas,mango,pruim

var tekst = "ananas";
console.log(tekst.split());   // ["ananas"];
console.log(tekst.split("")); // ["a","n","a","n","a","s"]
console.log(tekst.split("a")); // ["","n","n","s"]
console.log(tekst.split("an"));// ["","","as"]
console.log(tekst.split("na"));// ["a","","s"]
```

JSON

Formaat voor het versturen/ontvangen van gegevens.

Vergelijkbaar met Javascript object formaat:

- key-value paren
- met beperkingen:
    - key altijd tussen dubbele quotes
    - value:
        - number
        - string tussen dubbele quotes
        - array
        - object (volgens zelfde beperkingen)
        - niet: functie

In bestand: extensie .json

Uitwisseling met server: MIME-type is application/json

JSON voorbeeld

```
var artikel = {
  naam: "pen",
  prijs: 1.5,
  totaal: function(aantal){
    return this.prijs * aantal;
  },
  type: { punt: "fijn",
          kleur: "zwart" }
}

var json = JSON.stringify(artikel);
console.log(json); // {"naam":"pen","prijs":1.5,
                   // "type":{"punt":"fijn","kleur":"zwart"}}

artikel = JSON.parse(json);
console.log(artikel.type.kleur); // zwart
```

AJAX

Asynchronous Javascript And XML

Nu vooral gebruikt om objecten in JSON formaat met server uit te wisselen.

```
var request = new XMLHttpRequest(); // leeg request object
request.onreadystatechange = function(response) { // bij response:
  if (request.readyState === XMLHttpRequest.DONE) {//klaar (4)
    if (request.status === 200) {                  // gelukt (200)
      var list = JSON.parse(request.responseText);// lees response
      list.filter( item => item.postId<5)          // update DOM
          .forEach(item => console.log(item.email));
    } else {
       console.log("Couldn't load comments :(");
    }
  }
};
request.open('GET',                                  // verzend request
'https://jsonplaceholder.typicode.com/comments', true);
request.send();
```

Opdracht

Maak een object met willekeurige eigenschappen, waaronder:

- een string
- een array
- een functie

Probeer vervolgens de eigenschappen te gebruiken / tonen / aan te passen.

Opdracht

Maak een array met vier namen.

Verwissel de eerste twee namen van plek met behulp van array methods.

Vervang de derde naam met behulp van een array method.

Toon alle namen aan elkaar, door van de array een string te maken.

Opdracht

Maak een class Boek waarmee boek-objecten kunnen worden aangemaakt. Elk boek-object heeft een auteur en een titel. Geef ook een functie toString(): deze geeft een boek als tekst terug.

Maak vervolgens een array van boeken. Probeer daarin diverse functies van een array uit, zoals:

- alle boeken tonen met forEach
- sorteren op titel
- zoeken op auteur

Zie documentatie bij Array

Opdracht (optioneel)

Een datum wordt standaard als volgt weergegeven:

Wed Jan 31 2018 13:38:41 GMT+0100 (West-Europa (standaardtijd))

Geef Date.prototype een extra eigenschap maanden: een array met de namen van de maanden van het jaar.

Geef Date.prototype ook een extra functie formatDate(). Deze geeft bovenstaande datum als volgt terug:

31 Januari 2018.

Maak in formatDate() o.a. gebruik van this.getFullYear(), this.getMonth() en this.maanden.

DOM en events

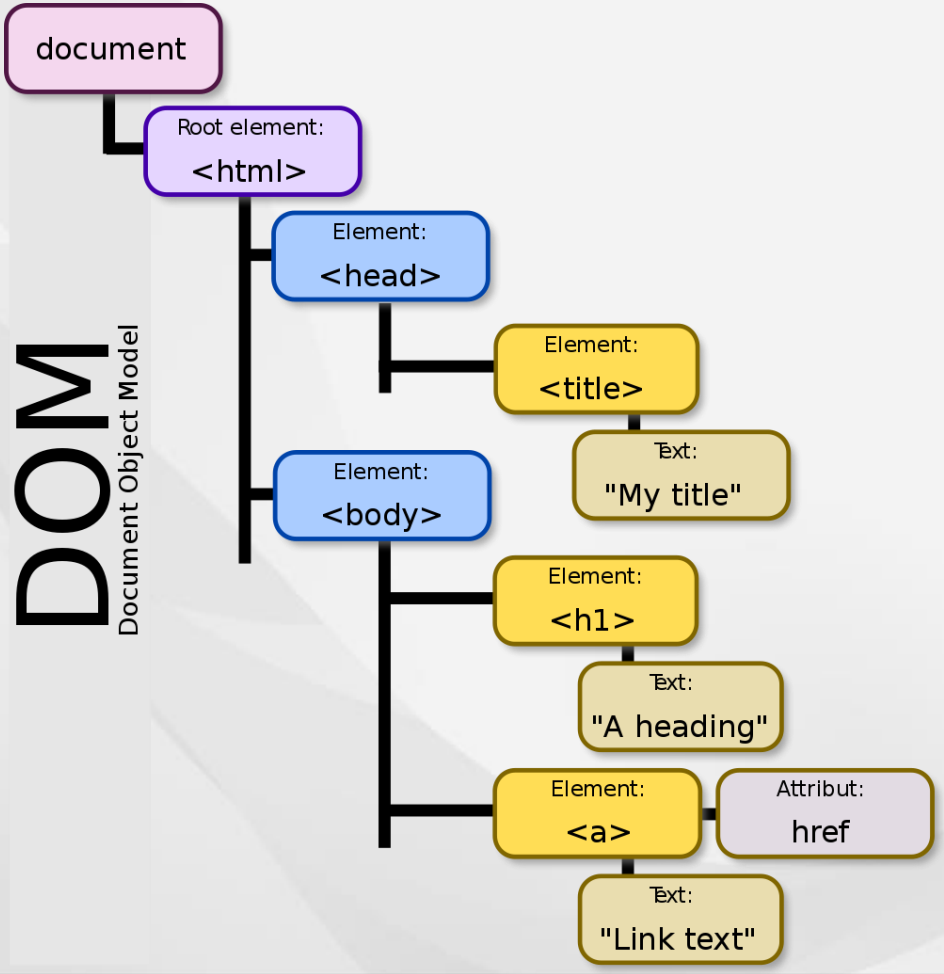
bewerk de pagina met Javascript

Het Document Object Model

Het Document Object Model (DOM) is:

- een programmeer interface voor HTML en XML documenten,
- een geheugenpresentatie van de webpagina,
- vormt een boomstructuur in het geheugen,
- kan worden gelezen en aangepast,
- kan weer naar het scherm worden weggeschreven.

DOM boomstructuur



DOM interfaces

interface	representeert
window	tabblad in browser
document	geladen html document
element	een element, zoals div, table, td ...
nodeList	verzameling elementen, als array te benaderen
attribute	attribuut van een element, zoals href van a
namedNodeMap	naam-waarde paren: de property attributes van een element geeft een namedNodeMap

DOM uitlezen

Om elementen in document te vinden: document.

functie	geeft
getElementsByTagName(...)	HTMLCollection (elementen)
getElementsByClassName(...)	HTMLCollection
getElementsByName(...)	HTMLCollection
getElementById(...)	element
querySelector(...)	element (gebruikt CSS selector)
querySelectorAll(...)	HTMLCollection (gebruikt CSS selector)

HTMLCollection omzetten naar array: Array.from(collection);
Ook vanaf element te gebruiken.

DOM elementen vinden

```
<div id="voorbeeld">
  <a class="test" href="http://www.5hart.nl">Vijfhart</a>
  <a href="http://www.atcomputing.nl">AT Computing</a>
</div>
<span class="test">Voorbeeld</span>

var voorbeeld = document.getElementById("voorbeeld");
console.log(voorbeeld.childElementCount); // 2
var links = document.getElementsByTagName("a");
console.log(links[1].textContent);     // AT Computing
var test = document.getElementsByClassName("test");
console.log(test.item(1).textContent); // Voorbeeld
var test2 = document.querySelector("#voorbeeld .test");
console.log(test2.textContent);        // Vijfhart
```

DOM elementen uitlezen

Veel gebruikte eigenschappen van een element:

- **textContent**: tekst inhoud van het element
- **innerHTML:** de HTML inhoud van het element
- **children**: de elementen binnen het element
- **parentNode**: het bovenliggende element

DOM attributen uitlezen

```
<div id="voorbeeld">
  <a class="test" href="http://www.5hart.nl">Vijfhart</a>
  <a href="http://www.atcomputing.nl">AT Computing</a>
</div>
<span class="test">Voorbeeld</span>

console.log(links[0].getAttribute("href"));
console.log(links[0].href);
```

DOM wijzigen

```
<div id="result"></div>

function createLink(href, tekst){
 var link = document.createElement("a");     // maak element
 var text = document.createTextNode(tekst); // maak tekst
 link.appendChild(text);  // voeg tekst toe aan element
 link.setAttribute("href", href);
                          // geef attribuut aan element
 return link;
}

var mijnLink = createLink("http://www.vijfhart.nl", "Vijfhart");
var result = document.getElementById("result");
result.appendChild(mijnLink);

<div id="result"><a href="http://www.vijfhart.nl">Vijfhart</a><div>
```

Alternatief: innerHTML

```
<div id="result"></div>

function createLink(href, tekst){
  return `<a href="${href}">${tekst}</a>`;
}

var mijnLink =
createLink("http://www.vijfhart.nl", "Vijfhart");
var result = document.getElementById("result");
result.innerHTML=mijnLink;
```

class aanpassen

```
<style>
  .rood { color:red;}
  .sterk { font-weight: bold; }
  .onderstreept { text-decoration: underline;}
</style>
<div id="test" class="rood">test</div>


var test = document.getElementById("test");

test.classList.add("sterk");
test.classList.remove("rood");
if(test.classList.contains("sterk")){
  test.classList.replace("sterk", "onderstreept");
}
test.classList.toggle("onderstreept"); // zet aan/uit
```

HTML5 data attributes gebruiken

```
<ul id="lijst"></ul>

var artikelen = ["pen", "potlood"];
var lijst = document.getElementById("lijst");

// genereer li-elementen uit artikelen
lijst.innerHTML =
artikelen.map ((a,i) => `<li data-nr="${i}">${a}</li>`)
         .join("");
// <li data-nr="0">pen</li><li data-nr="1">potlood</li>

var li = document.querySelector("li[data-nr='1']");
console.log(li.dataset.nr,li.textContent);

// 1 potlood
```

Opdracht

Maak een array met artikelen (namen).

Genereer hieruit een lijst op het scherm (li-elementen binnen een leeg ul-element).

Probeer dit eerst met een String template, zoals eerder voorbeeld. Probeer het ook met createElement en appendChild.

Optioneel: geef elk element ook een data attribuut.

Events afhandelen

- Events zijn gebeurtenissen op de pagina, zoals:
  - pagina is geladen
  - er is op een knop gedrukt
  - een veld is ingevuld
- Het reageren op zo'n event wordt **event handling** genoemd
- Daarvoor zijn twee ingrediënten nodig:
  - een **event listener**: wordt wakker als een opgegeven event optreedt
  - een **callback functie**: wordt aangeroepen door de event listener als het event optreedt

Veel gebruikte events

- **window**
  - **load**: pagina is geladen
  - **unload**: pagina is verlaten
  - **beforeunload**: voordat de pagina wordt verlaten
- **element**
  - **focus**: cursor komt in element
  - **blur**: cursor verlaat element
  - **change**: element is gewijzigd
- **form**
  - **submit:** formulier wordt verzonden
  - **reset:** verzenden formulier wordt gecanceld

Keyboard en mouse events

- **keyboard**
  - **keypress**: op toets gedrukt
  - **keydown**: toets is ingedrukt
  - **keyup**: toets is losgelaten
- **mouse**
  - **click**: in element op muis geklikt
  - **mouseover**: muis pointer op element
  - **mousemove**: muis wordt verplaatst

Reageren op een event

Een veel gebruikte event is **het load event** van window: pas dan is het hele document in de DOM geladen.

Dit gaat bijvoorbeeld fout:

```
<html>
  <head><title>lukt niet</title>
   <script>
     var mijnDiv = document.getElementById("test");
     // div met id="test" is hier nog niet bekend
     console.log(mijnDiv.textContent);
   </script>
 </head>
  <body><div id="test">Voorbeeld</div></body>
</html>
```

Wacht tot document geladen is

```
<html>
  <head><title>lukt wel</title>
   <script>
     window.onload=function(){
       var mijnDiv = document.getElementById("test");
       // div met id="test" is nu wel bekend
       console.log(mijnDiv.textContent);
     }
   </script>
 </head>
  <body><div id="test">Voorbeeld</div></body>
</html>
```

on<event> of addEventListener

- Met on<event> kunnen we een functie koppelen aan een event.
- Een volgend **on<event>** van het zelfde type overschrijft de eerste.
- Niet overschrijven? Dan **addEventListener**(<event>, <functie>, <propagatie>).

```
window.addEventListener("load", mijnFunctie, false);
```

Propagatie: wordt het event vervolgens naar parents doorgegeven (false), of naar children (true).

propagatie

```
<div id="buiten">
  buiten
  <div id="binnen">binnen</div>
</div>

binnen.addEventListener("click",
  function(){ console.log("op binnen geklikt" );
  }, false);
buiten.addEventListener("click",
  function(){ console.log("op buiten geklikt");
  }
  , false);

// klik binnen: dan eerst "op binnen geklikt",
                daarna "op buiten geklikt"
// bij true:    dan eerst "op buiten geklikt",
                daarna "op binnen geklikt"
```

callback functies

De functie die wordt aangeroepen bij een event wordt "callback functie" genoemd.

Deze krijgt het event zelf als argument mee.

```
<input id="test" type="text">

var test = document.getElementById("test");
test.addEventListener("keydown",
        function(e){ console.log(e.key);}, false);
```

Properties van events

- **target**: het element dat het event heeft afgevuurd
- keyboard events:
  - **key** (de toets zelf, als string)
  - **ctrlKey, metaKey, shiftKey, altKey** (boolean)
- mouse events:
  - **clientX, clientY, screenX, screenY, movementX, movementY**
  - **ctrlKey, metaKey, shiftKey, altKey**

Voorbeeld aanpassen DOM

```
window.addEventListener("click",function(e){
  var newDiv = document.createElement("div");
  var newText = document.createTextNode(
   `positie: [${e.clientX}, ${e.clientY}]`);
  newDiv.appendChild(newText);
  document.body.appendChild(newDiv);
});

// nu worden steeds nieuwe DIV elementen
// toegevoegd aan de body
// bijvoorbeeld
// <div>positie: [18, 27]</div>
```

Element vervangen in DOM

```
window.addEventListener("click",function(e){
  var newDiv = document.createElement("div");
  var newText = document.createTextNode(
 `positie: [${e.clientX}, ${e.clientY}]`);
  newDiv.appendChild(newText);
  newDiv.id="test";
  var test=document.getElementById('test');
  if(test) // als <div id="test"> al bestaat:
    test.parentNode.replaceChild(newDiv,test);
  else     // anders nieuw aanmaken
    document.body.appendChild(newDiv);
});

// nu wordt steeds één div vervangen
```

Validatie met reguliere expressie

```
<input id="postcode" type="text"
pattern="^[1-9][0-9]{3} [A-Z]{2}$"><br>
<label for="postcode">geef een geldige postcode</label>

var postcode = document.getElementById("postcode");
postcode.addEventListener("change", function(e){
  var re = new RegExp(e.target.pattern);
  var text = e.target.value;
  var label = document.querySelector("[for='postcode']");
  if(re.test(text)){
     label.textContent="geldige postcode";
     label.style.color="inherit";
   }
   else{
     label.textContent="ongeldige postcode";
     e.target.focus(); e.target.value=""; label.style.color="red";
  }
});
```

Opdracht

Maak een div met tekst, en een knop.

Als op de knop wordt geklikt verdwijnt de tekst.

Nog een druk op de knop toont de tekst weer.

Maak gebruik van deze css classes:

```
.zichtbaar{visibility:visible;}
.verborgen{visibility:hidden;}
```

en gebruik in Javascript classList.replace(...);

Opdracht

Maak een input en een output element in HTML:

```
<input id="invoer" type="text"
placeholder="invoer">
<br><output id="uitvoer"></output>
```

Als het input element is gewijzigd (het "change" event gaat af), toon dan de waarde van het element in hoofdletters in het uitvoer element.

(De tekst-waarde van het input element zit in het value attribuut: invoer.value. De tekst-waarde van het output element zit in uitvoer.textContent).

Opdracht - lijst vullen met data

Maak een object met:

- een array namen
- een functie add(...) om een naam toe te voegen: this.namen.push(...)
- een functie toList() die de array sorteert, en daaruit LI-elementen maakt (zie eerdere opdracht).

Test het object eerst met console.log()

Maak in HTML een input veld, een label en een leeg UL element aan:

```
<label for="namen">voeg naam toe</label>
<input type="text" id="namen"/>
<ul></ul>
```

Als het change event op het input veld afgaat, gebruik dan add() om de waarde toe te voegen aan de array, en toList() om de innerHTML van het UL element te vervangen.

Opdracht - validatie

Maak een webpagina die vraagt om twee getallen (twee inputvelden) en die daarna de som ervan op het scherm toont.

Controleer of elk veld numeriek is. Geef anders een duidelijke foutmelding.

```
var getal1 = parseFloat(...);
if(isNaN(getal1)){
   ....
}
```

Objecten creatie en overerving

call en apply

Een functie kan op een ander object worden uitgevoerd, dan waar het vandaan komt.

<functie>**.call**(**object**, argumenten).

of

<functie>**.apply**(**array[]**, argumenten)

Bijvoorbeeld, maak van tekst een array:

```
var array = Array.prototype.slice.call("test");
// "test".slice(), terwijl slice() niet in String voorkomt
console.log(array);
//  ["t", "e", "s", "t"]
var tekst = String.prototype.toUpperCase.apply(array);
//  ["t", "e", "s", "t"].toUpperCase()
console.log(tekst);
// T,E,S,T
```

bind

Met bind kan worden aangegeven welk object als ***this*** moet worden beschouwd bij het aanroepen van een functie.

```
var test = {values:[]}
var add = Array.prototype.push.bind(test.values);
add(1,2,3);
add(4,5);
console.log(test.values); // 1,2,3,4,5
```

overerving oud: Object.create()

```
function Artikel(naam, prijs){
   this.naam=naam;
   this.prijs=prijs;
}
Artikel.prototype.totaal=function(aantal){
  return this.prijs*aantal;
}
function ArtikelMetBTW(naam, prijs, BTW){
   Artikel.call(this,naam,prijs);
   this.BTW=BTW;
}
ArtikelMetBTW.prototype = Object.create(Artikel.prototype);
ArtikelMetBTW.prototype.totaalMetBTW=function(aantal){
  return this.prijs*(1+this.BTW/100.0)*aantal;
}
var test = new ArtikelMetBTW("pen",1,5);
console.log(test.totaal(3)); // 3
console.log(test.totaalMetBTW(3)); // 3.15
```

overerving nieuw: class

```
class Artikel{
  constructor (naam, prijs){
   this.naam=naam;
   this.prijs=prijs;
  }
  totaal(aantal){          // geen keyword function
    return this.prijs * aantal;
  }
}
 class ArtikelMetBTW extends Artikel {
   constructor(naam, prijs, BTW){
     super(naam, prijs); // roept constructor van Artikel aan
     this.BTW=BTW;
   }
   totaalMetBTW(aantal){
     return this.prijs*(1+this.BTW/100.0)*aantal;
   }
}
```

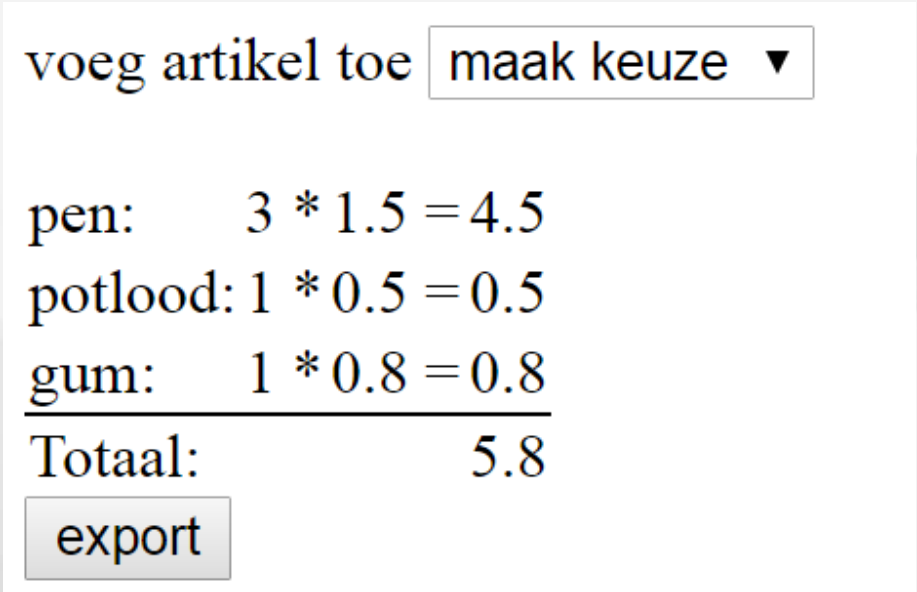
Casus (optioneel)

Maak een applicatie met:

- een class Artikel met een naam, een prijs en een aantal. De naam en de prijs worden aan de constructor meegegeven. Artikel heeft ook een functie totaal(): die geeft de prijs * aantal terug.
- Een lijst artikelen (array).
- De HTML pagina heeft een select item, waarin alle artikelen als option elementen vermeld staan (via Javascript).

Casus - vervolg

Als een gebruiker een artikel selecteert, wordt deze toegevoegd aan het "boodschappenmandje": een rij van een tabel, waarin naam, aantal, prijs en prijs * aantal vermeld staan. Bijvoorbeeld:



Casus - vervolg

Maak gebruik van een module object (var module = { ... }) waarin

- de bestellingen als array worden bijgehouden,
- met een add(artikel) functie die een bestelling toevoegt, of als het artikel al besteld is, het aantal ophoogt.
- optioneel: een remove(index) functie, die wordt aangeroepen als wordt dubbelgeklikt op een rij. Dan wordt het aantal met één verminderd, of als het aantal op 1 staat wordt de bestelling verwijderd uit de array. (gebruik slice() met index)
- Een sum() functie berekent het totaal van alle bestellingen.
- Een toRows() functie maakt table rows (tr en td elementen) van alle artikelen die in bestelling zijn. Deze wordt na elke wijziging (dubbel klik of gekozen artikel) gebruikt om de innerHTML van de table aan te passen.

Optioneel: geef ook een toJSON functie aan het object: deze geeft de bestellingen als JSON string terug. Gebruik console.log om deze te testen. (Daar is de export knop van de vorige sheet voor bedoeld)