# JavaScript Fundamentals

Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt door middel van druk, fotokopie of op welke wijze dan ook, zonder voorafgaande schriftelijke toestemming van de uitgever.

© Computrain

Postbus 447
1200 AK Hilversum
Telefoon: 030 - 23 48 500
E-mail: info@computrain.nl

www.computrain.nl

september 2018

# INHOUDSOPGAVE

# OVER DEZE CURSUS

## Inleiding

JavaScript behoort samen met HTML en CSS (Cascading Style Sheets) tot de meest basale en gangbare technieken om webpagina's te maken. Daar waar HTML bedoeld is om de inhoud van een webpagina te tonen en CSS om de details van het uiterlijk te bepalen, kunt u met behulp van JavaScript dynamiek en interactie aan een webpagina toevoegen. Met behulp van JavaScript kan een webpagina reageren op gebeurtenissen, zoals een klik of beweging met de muis, en kan de inhoud veranderen, bijvoorbeeld door het tonen van een ander plaatje of het uitklappen van een submenu. JavaScript wordt veel gebruikt voor het valideren van formulieren. Webdesigners gebruiken JavaScript in combinatie met HTML en CSS om dynamisch de opmaak van webpagina's te veranderen. De combinatie van deze technieken wordt ook wel DHTML (Dynamic HTML) genoemd.

JavaScript is de meest gebruikte, meest populaire internet scriptingtaal. Een scriptingtaal bestaat uit regels code met opdrachten (statements) die een voor een worden uitgevoerd. In het geval van JavaScript worden de statements uitgevoerd door de browser.

### Wat wordt in deze cursus behandeld?

In deze cursus wordt de basis van JavaScript behandeld. Aan de orde komen onder andere

- De basiselementen van JavaScript
- het gebruik van variabelen
- beslissingen
- lussen
- functies
- arrays
- objecten
- interactie met het Document Object Model (DOM)
- events
- interactie met formulieren

Dit zijn de hoofdonderwerpen. Daarnaast worden nog enkele kleinere onderwerpen behandeld.

Diverse meer complexe onderwerpen in JavaScript, zoals het maken van classes, overerving, closures, gebruik van AJAX en toepassen van promises, komen aan bod in de cursus *Advanced JavaScript Programming*. Frameworks zoals Angular komen niet aan de orde. jQuery zal kort worden aangestipt.

**JavaScript en andere technieken**

Er bestaan een aantal talen en technieken die lijken op JavaScript, maar die soms met JavaScript worden verward. U kunt dan denken aan Java, VBScript en server scriptingtalen.
Java is een zelfstandige programmeertaal die ondanks de naam weinig met JavaScript te maken heeft. Oppervlakkig gezien kunnen de talen op elkaar lijken, maar wie iets dieper kijkt zal grote verschillen zien.
JavaScript en VBScript bezitten een zelfde functionaliteit. Het zijn beide scriptingtalen. JavaScript wordt ondersteund door alle browsers. VBScript wordt alleen ondersteund door Internet Explorer.

Naast browser scripting bestaat ook server scripting, waarbij de statements in een pagina worden uitgevoerd door de webserver voordat de pagina naar de browser wordt verstuurd. Bekende server scriptingtalen zijn PHP en het oudere ASP. De taal nodeJS gebruikt JavaScript als taal op de server.

**De versie van JavaScript**

Versies van JavaScript zijn in feite implementaties van het alleen op papier bestaande ECMAScript. De meest relevante versies zijn momenteel ECMAScript 5 (ES5) uit 2012 en ECMAScript 6 (ES6) uit 2015. Medio 2018 worden nog juist niet alle features van ES6 door alle browsers ondersteund. De stof in dit boek is daarom gebaseerd op ES5. Een overzicht van van de diverse aanvullende features van ES6 wordt behandeld in de cursus *Advanced JavaScript Programming*.

**Oefenbestanden**

Bij deze cursus horen oefenbestanden die u kunt downloaden (`www.computrain.nl/oefenbestanden`). U treft hierin niet alleen de opgave-

bestanden maar ook de oplossingen aan. Uiteraard kunt u de door u zelf gemaakte oefeningen mee naar huis nemen via een usb-stick (of u kunt ze aan uzelf mailen).

Bij de oefenbestanden vindt u ook een document waarin de oplossingen stap voor stap worden beschreven. Als u op een bepaald punt vast loopt bij een oefening kunt u daar vinden wat er precies gevraagd wordt.

# 1 JAVASCRIPT BASISBEGRIPPEN

## 1.1 JavaScript en HTML

JavaScript statements bevinden zich doorgaans in webpagina's tussen de HTML-tags `<script>` en `</script>`.

```
<script>
  JavaScript statement
  JavaScript statement
</script>
```

In oudere browsers moest u ook het soort script aangeven. De bovenste regel werd dan als volgt:

```
<script type="text/javascript">
```

Met het attribuut `type` wordt aangegeven dat u met JavaScript te maken heeft, en niet met een andere scriptingtaal zoals VBScript. In moderne browsers is dit tegenwoordig de standaard, en dan kunt u dit deel weglaten.
Net als bij CSS kan JavaScript ook in een apart document worden gezet. Dit document dient dan de extensie `.js` te hebben. U zet in dit bestand alléén JavaScript code. U plaatst dus geen `<script>` tag.
In de HTML-webpagina waarin de code moet worden gebruikt dient u een verwijzing op te nemen naar dit document. De JavaScript-code zal door de browser op de plaats van de link worden ingevoegd.

```
<script src="myjavascriptlib.js"></script>
```

Merk op dat u op deze wijze meer externe JavaScript-bestanden kunt gebruiken binnen één webpagina. Deze kunnen ook van een externe lokatie komen. Voor ieder bestand is een aparte `<script>` tag nodig.

Let er op dat de verkorte wijze van het afsluiten van een tag hier niet werkt. Wanneer u het bovenstaande neerzet als hieronder werkt dit niet.

```
<!--  Foutieve versie van include !! -->
<script src="javascriptdocument.js" />
```

JavaScript kan niet zomaar de harde schijf benaderen. JavaScript kan dus nooit uw bestanden lezen of verwijderen.

## 1.2 Commentaar

U kunt uw code van commentaar voorzien. door dit commentaar vooraf te laten gaan door `//`. Tussen `<script>` tags wordt alle tekst na `//` op dezelfde regel door de browser genegeerd. U kunt commentaar ook tussen `/*` en `*/` zetten. Het commentaar mag dan over meer regels doorlopen.

```
<script type="text/javascript">
  // These 2 lines are comments
  // and will not be executed.
  JavaScript statement // This is a comment too
  JavaScript statements
  /* This is a
     multiple line
     comment
  */
  JavaScript statements
</script>
```

U kunt de JavaScript-code van een willekeurige pagina bekijken net zoals u de HTML-code kunt bekijken. Dit doet u met de optie *View Source* van de browser. Soms is de code 'minified', dat wil zeggen tot een absoluut minimum teruggebracht. Dan is de code nagenoeg onleesbaar.

## 1.3 JavaScript en browser compatibility

Doordat JavaScript in de browser draait kunt u er als auteur van een niet 100% zeker van zijn dat het script in iedere browser op dezelfde manier wordt uitgevoerd. Veel elementen van JavaScript zijn standaard, en in grote lijnen zijn de belangrijkste browsers, zoals FireFox, Chrome, Edge / Internet Explorer (IE) en Safari onderling compatibel. Maar er bestaan verschillen tussen de browsers onderling op bepaalde punten, en daar zult rekening mee moeten houden. Om te beginnen is het zeer raadzaam om uw scripts *altijd* te testen in de diverse browsers. Sommige browsers ondersteunen bepaalde zaken niet of zijn minder tolerant voor fouten. Zorg ervoor dat uw script tenminste in de browsers FF,Chrome en Edge/IE correct draait.

In sommige gevallen kan testen bewerkelijk zijn. Bij FireFox en Chrome worden updates van de browsers geheel automatisch verwerkt. U mag er dus van uitgaan dat gebruikers de laatste versie van deze browsers op hun systemen hebben. Dat geldt inmiddels ook voor Edge.

Internet Explorer kent echter geen automatische updates. Media 2018 gebruikt nog ongeveer 3 procent van de internetgebruikers IE. De meeste daarvan draaien IE11. Eerdere versies van IE veroorzaken meestal meer problemen. De verwachting is dat IE gebruik uiteindelijk helemaal zal verdwijnen, tot dan moet rekening worden gehouden met de gebruikers hiervan.

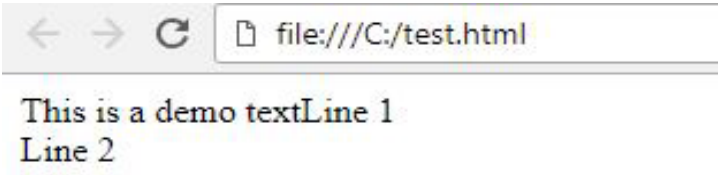
## 1.4 JavaScript statements

Een eenvoudige manier om tekst in de browser te laten afdrukken is via `document.write`. De tekst die u meegeeft zet u tussen aanhalingstekens. U kunt zowel enkele als dubbele aanhalingstekens gebruiken. JavaScript maakt tussen deze twee geen onderscheid, zolang ze maar wel met elkaar corresponderen.

```
document.write('This is a demo text');
document.write("Line 1<br>Line 2");
```

Let op de code `<br>` die in de tweede regel staat. Deze HTML-code zorgt er voor dat de tekst erna op de volgende regel begint.

Als deze regels in een browser worden uitgevoerd, dan ziet u dit:




Via `alert` kunt u ook tekst in een dialoogvenster krijgen:

```
window.alert("This appears in a dialog");
```

Dit levert in Chrome de volgende dialoog op:


Indien u in een dialoogvenster uw tekst wilt verdelen over meer regels, dient u in de tekst `\n` te gebruiken. De backslash (`\`) wordt gebruikt om aan te geven dat het volgende karakter op een bijzondere manier moet worden uitgevoerd. In `\n` staat de `n` voor *newline*.

```
window.alert("Line 1\nLine 2");
```


Merk op dat `\n` niet werkt in een `document.write()` statement.

U kunt met behulp van JavaScript ook gegevens over de gebruikte browser tonen. Aangezien u bepaalde informatie opvraagt en geen letterlijke tekst afdrukt zet u deze tekst dan *niet* tussen aanhalingstekens.

```
document.write(navigator.appName);
document.write(navigator.appVersion);
```

In Chrome zou dit (afhankelijk van de versie) het volgende opleveren:

```
Netscape
  5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36
KHTML, like Gecko) Chrome/66.0.3359.181 Safari/537.36
```

Met het plusteken kunt u teksten aan elkaar plakken tot één stuk tekst.

```
document.write('You are using ' + navigator.appName);
```

Deze regel levert in Chrome het volgende op:

You are using Netscape

## 1.5 Puntkomma's en hoofdlettergevoeligheid

In de voorbeelden zag u dat de opdrachten steeds afgesloten worden met een puntkomma. Tegenwoordig is dit in moderne browsers aan het einde van een regel niet meer verplicht. Toch worden de puntkomma's vaak toegevoegd 'voor alle zekerheid'. Kwaad kan het in elk geval nooit.

Verder moet u ervan uitgaan dat JavaScript meestal (maar niet overal) hoofdletter-gevoelig is. Als u in bovenstaand voorbeeld het woordje 'appName' niet met een kleine letter 'a' en een hoofletter 'N' schrijft dan werkt het niet.

## 1.6 camelCase

JavaScript is hoofdlettergevoelig, maar hoe namen precies gespeld worden is gelukkig voorspelbaar. JavaScript gebruikt vrijwel altijd de conventie van 'camelCase'. Hierbij krijgt elk woord binnen een naam een hoofdletter, behalve het eerste woord. Dus bijvoorbeeld

- numberOfPeople
- firstChild
- getElementById

Het is niet verplicht om voor de namen van functies die u zelf schrijft deze conventie te gebruiken, maar als u het doet verhoogt u de voorspelbaarheid van uw eigen namen.

## 1.7 Fouten opsporen

Als u een fout maakt in JavaScript dan werkt het script niet, maar u krijgt meestal geen foutmelding. In de meeste browsers gaat u dan naar de *Developer Tools* (meestal via <F12>). Daar kunt u vaak snel zien wat er mis is. Neem bijvoorbeeld dit script:

```
document.write(navigator.appName);
document.write(navigator.appName);
document.Write(navigator.appName);
document.write(navigator.appName);
```

Wanneer u scherp kijkt dan ziet u een foutieve hoofdletter in de derde regel. Op de console in Chrome ziet u dan:


Via dit soort hulpmiddelen kunt u de fout vaak snel opsporen.

# 2 VARIABELEN

## 2.1 Wat is een variabele?

In vrijwel elke programmeertaal is er al snel behoefte om te kunnen werken met variabelen. U kunt een variabele zien als een stukje geheugen waaraan een naam gegeven is, zodat u ernaar kunt verwijzen. Een andere analogie is die van een doosje met een label.

```
number = 42;
river = 'Mississippi';
```

Hier ziet u hoe variabelen worden gevuld met waarden. De naam van de eerste variabele is 'number'. U kunt dit zien als een doosje met het label 'number', dat als inhoud het getal 42 bevat. De tweede variabele heet 'river' en bevat de tekst 'Missisippi'.

U kunt de inhoud van variabelen oproepen door hun naam te gebruiken. In dat geval moet de naam van de variabele *niet* tussen aanhalingstekens staan, want dan wordt het gezien als letterlijke tekst.

```
result1 = number;
result2 = 'number';
```

De eerste variabele heet 'result1' en krijgt als waarde de *inhoud* van de variabele 'number' (dus 42). Maar de variabele 'result2' krijgt als inhoud gewoon de letterlijke tekst 'number'.

## 2.2 Het gebruik van window.prompt

Eerder zag u dat u met `window.alert()` een boodschap in een dialoogvenster kunt laten verschijnen. Het is ook mogelijk om een dialoogvenster te laten verschijnen waarin de gebruiker iets kan invoeren. Dit kan met `window.prompt()`. Door het resultaat van deze opdracht (de invoer van de gebruiker) in een variabele op te slaan kunt u deze waarde vervolgens in het script gebruiken. Neem het onderstaande voorbeeld:

```
fruit = window.prompt("What fruit would you like?");
document.write("You have chosen : " + fruit);
```

Als u dit script in een browser laat uitvoeren verschijnt er een dialoogvenster zoals hieronder, waarin u gegevens kunt invoeren. Vervolgens wordt de ingegeven waarde in de browser getoond.


## 2.3 Namen van variabelen

Over het algemeen is het raadzaam om voor variabelen leesbare namen te gebruiken. U mag een variabele ook simpelweg `a` of `b` noemen, maar dan is he minder duidelijk wat de inhoud precies voorstelt.
Een naam van een variabele moet een woord zijn dat bestaat uit letters, cijfers en underscores (_). Daarnaast geldt:

- Een naam mag geen spaties bevatten.
- Een naam mag niet met een cijfer beginnen.
- Hoofdletters en kleine letters zijn verschillend (dus `myname` is een andere variabele dan `myName`).
- Een naam mag geen keyword van de taal JavaScript zelf zijn (zoals bijvoorbeeld `if, else, for, function`).

## 2.4 Declareren van variabelen

Normaal gesproken maakt u een variabele in JavaScript bekend met het woord `var:.`

```
var age = 24;
```

In veel programmeertalen is het declareren van variabelen verplicht. In JavaScript hoeven variabelen niet verplicht te worden gedeclareerd voor gebruik, maar het wordt wel bijzonder sterk aangeraden. De reden hiervoor komt later aan de orde.
Merk op dat u dezelfde variabele meer dan eens nieuw mag aanmaken. Het volgende gaat dus niet fout:

```
var age = 24;
var age = 28;
```

In JavaScript worden namen meestal geschreven volgens de conventie van *(lower) camel case*. Hierbij wordt elk woord binnen de naam geschreven met een hoofdletter, behalve het eerste woord. Een paar voorbeelden:

```
var myAge = 24;
var numberOfPeople = 300;
var currentLocation = 'London';
```

Als u een variabele bekend maakt (declareert), maar geen waarde geeft, dan is de inhoud van die variabele `undefined`. Deze waarde kunt u ook als `null` beschouwen . .

```
var profession;  // value is undefined, or null
```

## 2.5 Soorten variabelen

Javascript onderscheidt enkele types varabelen qua inhoud. Er zijn maar drie basistypes:

- Numbers; dit zijn getallen, geheel (17) of met een decimale punt (4.15).
- Strings; dit zijn teksten, zoals 'hallo'.
- Booleans; deze kunnen uitsluitend `true` of `false` bevatten.

```
var num1 = 200;         // a number
var num2  = 18.55;      // another number
var myName = 'John';    // a string
var isMarried = false; // a boolean
```

Er bestaan nog enkele andere types variabelen, namelijk arrays en objecten. Deze komen later in de cursus aan de orde.

## 2.6 Operatoren

Voor het rekenen kent JavaScript een groot aantal operatoren. Een aantal zult u ongetwijfeld al kennen omdat ze dagelijks gebruikt worden voor optellen, aftrekken, vermenigvuldigen en delen. Daarbij worden de normale voorrangsregels gebruikt; vermenigvuldigen en delen hebben dus voorrang boven optellen en aftrekken. Door haakjes te gebruiken kunt u de voorrang zelf bepalen.
Minder bekend is de modulus operator, in JavaScript genoteerd als een procentteken (%). De modulus is de rest die u overhoudt na een deling. Als u 17 appels moet verdelen over vijf kinderen houdt u er aan het einde twee over. Als u ze verdeelt over zes kinderen houdt u er vijf over. Met andere woorden: 17 % 5 levert 2 op, en 17 % 6 levert 5 op.

Hieronder ziet u enkele voorbeelden:

```
var x = 4;
var y = 9;
var z;
z = y - x;   // z will be 5
z = x * y;   // z will be 36
z = x + y;   // z will be 13
z = y / x;   // z will be 2.25
  // modulus operator
z = y % x;   // z will be 1 (remainder after division)
z = 83 % 10; // z will be 3
  // preference rules
z = 8 + 2 * 3     // z will be 14
z = (8 + 2) * 3;  // z will be 30
```

Het komt regelmatig voor dat de inhoud van een variabele moet worden gewijzigd. Stelt u zich voor dat een doosje (variabele) een bepaalde waarde bevat en dat er nog eens 10 bij moet. Als er eerst 40 inzat dan moet dat dus 50 worden. De inhoud van de variabele wordt dan met 10 verhoogd. Dit doet u als volgt:

```
number = number + 10;  // add 10 to the value
```

Op dezelfde wijze kunt u de inhoud van een variabele met 10 verlagen of met drie vermenigvuldigen. Zie hieronder.

```
number = number - 10;
number = number * 3;
```

Dergelijke operaties, waarbij dezelfde variabele zowel links als rechts van het '=' teken staat, kunt u in JavaScript op een verkorte manier weergeven. Zie hieronder.

```
number += 10;   // same as number = number + 10
number -= 8;    // same as number = number - 8
number *= 3;    // same as number = number * 3
number /= 10;   // same as number = number / 10
number += x;    // same as number = number + x
```

Een speciaal geval is het optellen van 1 bij een variabele. Dit komt zo vaak voor dat hiervoor een nog kortere schrijfwijze bestaat, namelijk '++' (twee plusjes aan elkaar). Deze operator mag zowel voor als achter de variabele staan. Behalve '++' is er ook '- -' voor het verlagen.

```
var number = 20;
number++; // number will be 21
++number; // number wil be 22
number--; // number will be 21
--number; // number wil be 20
```

Voor strings is er eigenlijk maar één operator, namelijk het plusteken. Dit wordt gebruikt om variabelen aan elkaar te plakken (concatenatie).

```
var firstName = "John";
var lastName = "Smith";
  // create full name: 'John Smith''
var fullName = firstName + " " + lastName;
```

## 2.7 Onderscheid maken tussen tekst en getallen

In JavaScript heeft het plusteken een dubbele betekenis. Bij getallen wordt het gebruikt voor optellen, bij strings voor concatenatie. JavaScript moet zelf beslissen welke van de twee gebruikt moet worden. Bij twee getallen wordt automatisch opgeteld.

```
var age = 45;
var ageInTenYears = age + 10;  // will be 55
```

Als de gegevens echter uit andere bron komen, zoals een tekstvak of een invoerdialoog, dan ziet JavaScript de gegevens als tekst, ook als de invoer toevallig een getal bevat. Het plusteken wordt dan gebruikt om de teksten aan elkaar te plakken.

```
var age = window.prompt("Enter your age");
  // assume the user enters 45 - age is now '45'
var ageInTenYears = age + 10;  // will be 4510
```

In dergelijke gevallen gebruikt u een speciale functie om duidelijk te maken dat deze tekst in een getal moet worden omgezet. Er zijn twee van deze functies, namelijk `parseInt` om een tekst om te zetten in een geheel getal, en `parseFloat` om een getal met eventuele decimale punt te krijgen.

```
var age = window.prompt("Enter your age");
  // assume the user enters 45 - age is now '45'
var ageInTenYears = parseInt(age) + 10;  // will be 55
```

Het is overigens vaak veiliger om de invoer direct en altijd om te zetten in een (al dan niet geheel) getal. Daarmee voorkomt u dat u enkele regels later alsnog

de omzeting moet doen (en het misschien vergeet). Die omzetting kan zonder extra tussenstap gebeuren:

```
var age = parseInt(window.prompt("Enter your age"));
  // assume the user enters 45 - age is now 45
var ageInTenYears = age + 10;  // will be 55

  // use parseFloat for input with decimals
var price = parseFloat(window.prompt("Enter price"));
```

# 3 CONTROLESTRUCTUREN

Wanneer u een aantal programmaregels achter elkaar of onder elkaar plaatst, worden deze één voor één uitgevoerd. Maar het is ook mogelijk andere structuren aan te brengen. U kunt gedeeltes van een programma een aantal keren laten herhalen en u kunt de uitvoering van de code afhankelijk maken van bepaalde voorwaarden. Zoiets gebeurt meestal via een zogenaamd if-statement. Een herhaling staat bekend als een lus of een loop.

## 3.1 Condities

Met een `if` statement kunt u de code laten beslissen of bepaalde zaken wel of niet moeten worden uitgevoerd. In het meest eenvoudige geval ziet dit er als volgt uit:

```
if ( <some condition> )
   <some code>
```

Het blokje code op de tweede regel wordt uitsluitend uitgevoerd als de conditie 'waar' (t`rue`) is. Als de conditie 'onwaar' (`false)` is wordt deze code niet uitgevoerd. Een eenvoudig voorbeeld

```
var age;
age = window.prompt("Enter your age");
age = parseInt(age);
if ( age >= 18 )
  document.write("You are over 18!");
```

De conditie achter het woordje `if` moet altijd tussen haakjes staan. U kunt voor het opstellen van de conditie een groot aantal operatoren gebruiken, zoals u uit onderstaande tabelletje kunt aflezen.

| | |
|---|---|
| > | Groter dan |
| < | Kleiner dan |
| >= | Groter dan of gelijk aan |
| <= | Kleiner dan of gelijk aan |

| | |
|---|---|
| `==` | Gelijk aan |
| `!=` | Ongelijk aan |

Een paar voorbeelden van condities in een `if` ziet u hieronder.

```
if( age >= 100 )
if( name == "John" )
if( age < 12 )
if( name != "admin" )
```

Het resultaat van een conditie is altijd `true` of `false`. Dit wordt een *Boolean* waarde genoemd.

## 3.2 Condities combineren

U wilt soms op meer voorwaarden kunnen controleren. Denk aan een geval waarbij u controleert of een waarde niet alleen minstens 1 is, maar ook hoogstens 10. Voor dergelijke gevallen zijn er drie operatoren:

| | |
|---|---|
| `&&` | And. Hiermee eist u dat aan beide condities wordt voldaan. |
| `\|\|` | Or. Hiermee eist u dat aan tenminste een van beide condities wordt voldaan. |
| `!` | Not. Hiermee vraagt u om de tegenovergestelde conditie |

**Een paar voorbeelden:**

```
  // check if age is between 10 and 19
if( age >= 10 && age < 20 )
  // check if name 'admin' or 'Admin'
if( name == "admin" || name == "Admin" )
  // check if name is not equal to 'John'
if( ! (name == "John") )
```

## 3.3 De code onder een if-statement

Een `if`-statement dient om te controleren of aan een bepaalde voorwaarde wordt voldaan. Indien dit het geval is moet bepaalde code worden uitgevoerd. Deze code zet u onder de `if`-regel. Houd u daarbij rekening met het volgende:

- U mag slechts één statement (opdracht) achter een `if` zetten.
- Maar: u mag een aantal opdrachten tussen accolades zetten en ze daarmee combineren tot een enkel super-statement. Zoiets geldt dan als een enkele opdracht en staat bekend als een *block statement*.
- Het is *zeer* gebruikelijk om de code onder een `if` in te springen, zodat de structuur van de code duidelijk is.

**Enkele voorbeelden:**

```
var age = window.prompt("enter your age");
age = parseInt(age);
if( age >= 10 && age < 20 )
  alert( "You are a teenager");

  // Assume user has logged in somehow
if( username == "admin" ) {
  alert("OK, you have full rights");
  var isAdmin = true;
}
```

Er bestaat discussie over hoeveel spaties er moet worden ingesprongen. De meest gebruikte aantallen zijn 2 en 4. Als u in een 'slimme' omgeving werkt kunt u dit vaak instellen en wordt het inspringen automatisch verwerkt.
Ook over de plaats van de accolades verschilt men nog wel eens van mening. Erg belangrijk is het niet. Ook dit kan in slimme editors worden ingesteld, zodat u in elk geval consequent kunt zijn.
Merk tenslotte op dat accolades verplicht zijn als u onder de `if` meer regels code kwijt wilt. Hebt u daar slechts één regel staan dan kunt u de accolades weglaten. Ook hier zijn de meningen overigens vaak verdeeld. Veel mensen vinden dat het de voorkeur verdient de acoolades *altijd* neer te zetten, ook als er maar één regel staat. Dus:

```
var age = window.prompt("enter your age");
age = parseInt(age);
if( age >= 10 && age < 20 ) {
  alert( "You are a teenager");
}
```

Daar is in elk geval niets tegen, maar noodzakelijk is het dus niet.

**De else en de else if**

In veel gevallen hebt u een bepaalde controle gedaan. U wilt bepaalde code laten uitvoeren als die controle klopt. Maar u wilt óók bepaalde code uitvoeren als die controle niet klopt. In dergelijke gevallen hebt u een `else`-tak nodig, zoals u hieronder ziet.

```
  // Assume user has logged in somehow
if( username == "admin" ) {
  alert("OK, you have full rights");
  var isAdmin = true;
} else {
  alert("Sorry, you have limited rights");
  var isAdmin = false;
}
```

De `else`-tak wordt dus alleen uitgevoerd als de controle **niet** waar is.
De `else` wordt vaak direct achter de sluitaccolade gezet, maar u mag die sluitaccolade natuurlijk ook op een losse regel zetten.
Indien u nog meer dan twee mogelijkheden hebt kunt u ook nog een `else if` gebruiken. Hieronder wordt op drie categorieën getest.

```
var number = window.prompt("Enter a number");
number = parseFloat(number);
if( number < 0 ) {
  alert("A negative number");
} else if ( number > 0 ) {
  alert("A positive number");
} else {
  alert("You entered zero");
}
```

U mag overigens net zo vaak `else if` gebruiken als u wilt. U kunt dus ook testen op tien of twintig verschillende gevallen. In sommige gevallen kan dat nog iets handiger, zoals u heronder nog zult zien.

**Bevestiging met windows.confirm**

Indien u een gebruiker interactief wilt laten kiezen of iets wel of niet moet gebeuren dan kunt u gebruik maken van `windows.confirm`. Hiermee biedt u de gebruiker een dialoog waarin een vraag kan worden bevestigd. Of juist niet natuurlijk. Dat werkt als volgt:

```
var choice = window.confirm("Are you sure?");
if( choice ) {
  alert("OK");
} else {
  alert("Never mind");
}
```

In de variabele `choice` komt `true` te staan als de gebruiker in de dialoog op 'OK' klikt, anders `false`. Deze waarde wordt vervolgens gebruikt in een controle, waarna de betreffende actie kan worden ondernomen.

**Het switch-statement**

Bekijk als inleiding van dit gedeelte eerst onderstaande code.

```
var mark = window.prompt("Enter mark (1-10)");
mark = parseInt(mark);
if( mark == 10 ) {
  var message = "Excellent";
} else if ( mark == 9 ) {
  var message = "Very good";
} else if ( mark == 8 ) {
  var message = "Good";
} else {
  var message = "Illegal mark";
}
alert (message);
```

U ziet hier een if-statment dat een groot aantal takken heeft. In al deze takken wordt de inhoud van dezelfde variabele gecontroleerd. In dergelijke gevallen kunt u ook gebruik maken van een `switch`.

In bovenstaand geval ziet de code er dan als volgt uit:

```
var mark = window.prompt("Enter mark (1-10)");
mark = parseInt(mark);
switch( mark ) {
  case 10:
    var message = "Excellent";
    break;
  case 9:
    var message = "Very good";
    break;
  case 8:
    var message = "Good";
    break;
  default:
    var message = "Illegal mark";
}
alert (message);
```

Na elke `case` dient u het woord `break` te zetten, anders loopt de code door naar de volgende `case`. Hier kunt u een enkele keer gebruik van maken, maar meestal is dit niet de bedoeling.
De `default` dient voor alle overige gevallen, en is dus in feite equivalent aan een `else`.
Overigens worden `switch` statements veel en veel minder gebruikt dan `if` statements.

**Herhalingen**

Een herhaling van uw programma-opdrachten kunt u bewerkstelligen met een zogenaamde lus (loop). Voor eenvoudige gevallen zijn er drie soorten, namelijk de `while`, de `do while` en de `for`.

### De while lus

De meest eenvoudige lus maakt u met een `while`. Schematisch gezien ziet die er als volgt uit.

```
while( <some condition> )
  <some code>
```

De code die achter de `while` staat wordt steeds opnieuw uitgevoerd zolang de conditie waar is. Net als bij de `if` mag u achter een `while` maar één opdracht zetten, maar kunt u via accolades meer code kwijt. Natuurlijk moet u er wel voor zorgen dat de conditie ooit onwaar wordt, anders hebt u een oneindige loop!

```
var count = 0;
while( count < 10 ) {   //  endless loop !!
  document.write("Hello<br>");
}
```

Bovenstaande code loopt oneindig door, omdat de variabele *count* nooit verandert en dus altijd kleiner dan 10 zal zijn. Bij een lus moet u er dus *altijd* voor zorgen dat de conditie ooit onwaar wordt. Bijvoorbeeld zo:

```
var count = 0;
while (count < 10 ) {
  document.write("Hello!<br>");
  count++;
}
```

Nu wordt de variabele *count* in elke lusdoorgang verhoogd. Deze code zal dus 10 regels uitvoer geven en daarna stoppen.
In bovenstaand voorbeeld werd de variabele *count* alleen maar gebruikt om te tellen. Maar vaak wordt de lusvariabele ook in de code onder de lus gebruikt. Een simpel voorbeeld is om de waarde af te laten drukken:

```
var count = 0;
while (count < 10 ) {
  document.write("Count = " + count + "<br>");
  count++;
}
```

In de browser ziet u dan:

Count = 0
Count = 1
Count = 2
Count = 3
Count = 4
Count = 5
Count = 6
Count = 7
Count = 8
Count = 9

Een interessanter versie van een lus krijgt u wanneer de variabele die voor de lus wordt gebruikt niet wordt opgehoogd, maar op een andere wijze wordt veranderd..

```
var power = 1;
while ( power < 1000 ) {
  power *= 2;
}
document.write(power);    // 1024
```

In dit voorbeeld wordt de variabele *power* steeds verdubbeld, totdat deze groter is dan 1000. De *power* neemt achtereenvolgens de waarden 1, 2, 4, 8 enzovoorts aan, totdat de waarde 1024 (twee tot de tiende macht) wordt bereikt.
Merk op dat een `while` loop mogelijk nul keer kan worden uitgevoerd. Indien de conditie bovenin al direct onwaar is zal de code binnen de loop in het geheel niet worden uitgevoerd.

```
var count = 100;
while (count <= 10 ) {
  document.write("Hello!<br>");
  count++;
}
```

Hier is de waarde van de variabele *a* al direct groter dan 10, zodat de code onder de while nooit wordt uigevoerd en er dus geen uitvoer verschijnt.

### De do ... while lus

Een tweede vorm van herhaling is de `do ... while`. Deze zie er schematisch als volgt uit:

```
do {
  <some code>
} while( <some condition> )
```

Alhoewel de `do ... while` sterk lijkt op de `while` is er toch een essentieel verschil. Omdat bij deze vorm de conditie *onderaan* de lus staat wordt de code in een `do ... while` altijd tenminste één maal uitgevoerd.
In het volgende voorbeeld wordt gevraagd een wachtwoord te typen. Deze vraag wordt net zolang herhaald tot het juiste antwoord (`"secret"`) wordt ingegeven.

```
var password = "";
do {
  password = window.prompt("Enter password");
} while (password != "secret");
window.alert("Password accepted");
```

Het voorbeeld kan worden verbeterd door ook te testen of de gebruiker op [Annuleren] ([Cancel]) heeft geklikt. In dat geval zal `window.prompt()` de waarde `null` opleveren. Na de loop moet dan nog wel even worden getest of de loop was gestopt door het juiste wachtwoord en niet doordat op [Annuleren] was geklikt.

```
var password = "";
do {
  password = window.prompt("Enter password");
} while (password != "secret" && password != null);
if (password != null ) {
  window.alert("Password accepted ");
}
```

De `do ... while` wordt vaak gebruikt bij gevallen waar iets moet gebeuren totdat het gelukt is. Daarbij is het natuurlijk vaak nodig dat bepaalde code tenmisnte één maal wordt uitgevoerd.

### De for lus

Bekijk u nog eens een eerder voorbeeld van de `while` lus:

```
var count = 0;
while (count < 10 ) {
  document.write("Hello!<br>");
  < more code... >
  count++;
}
```

De lus wordt aangestuurd met behulp van de variabele *count*, die op drie plaatsen wordt gebruikt. De eerste maal krijgt de variabele een startwaarde (de *initialisatie*), de tweede maal wordt de waarde gecontroleerd in de *conditie* en de derde maal wordt de variabele gewijzigd, in dit geval opgehoogd, de zogenaamde *iteratie*.
Het kenmerkende van een `for` lus is dat die drie zaken allemaal bovenin bijeen staan in de lus. Schematisch:

```
for(<initialization> ; <condition> ; <iteration>) {
  <some code>
}
```

De drie termen staan tussen haakjes achter de `for`, gescheiden door (verplichte) puntkomma's. De voorbeeldcode van de `while` zou er dus met een `for` als volgt uit zien:

```
for( var count = 0; count < 10; count++ ) {
  document.write("Count = " + count + "<br>");
}
```

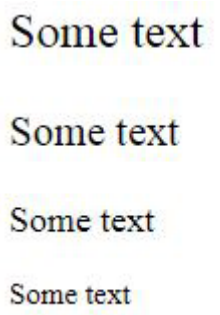
De for lus is in dot soort gevallen qua code wat compacter dan een while, en mede daarom wordt de for erg vaak gebruikt.

### Enkele praktische voorbeelden van lussen

Met lussen, of herhaling, kunnen heel veel effecten worden bereikt. Als eerste extra voorbeeld een geval waarin de lusvariabele wordt verwerkt in de uitgevoerde HTML:

```
for( var size = 16; size > 8; size-=2 ) {
  document.write('<p style="font-size:'
      + size + 'px">Some text');
}
```

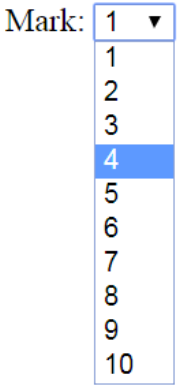
De lusvariabele loopt hier van 16 tot 8, en moet dus niet verhoogd maar verlaagd worden. In dit geval gebeurt dit zelfs met 2 tegelijk. De waarde wordt verwerkt in de uitvoer en opgevat als een fontgrootte in pixels. De uitvoer ziet er dan zo uit:




In een ander voorbeeld ziet u hoe met behulp van een lus een HTML *select* element wordt gevuld met de waarden 1 tot en met 10:

```
document.write('Mark: <select>');
for( var mark = 1; mark <= 10; mark++ ) {
  document.write('<option>' + mark);
}
document.write('</select>');
```

Het resultaat in de browser ziet er nu als volgt uit:




Lussen komen heel veel voor in de praktijk. Overal waar u dezelfde code een aantal malen terugziet kunt u dit waarschijnlijk inkorten door een lus te gebruiken. Lussen komen met name heel vaak voor zodra u gaat werken met arrays. Dit komt later in dit boek nog uitgebreid aan de orde.

**Extra: break en continue**

Met behulp van de opdracht `break` kan een lus worden afgebroken. Dit is handig als er iets mis gaat of als er duidelijke redenen zijn om de natuurlijke loop van de code af te breken.
In onderstaand voorbeeld wordt de lus afgebroken als de gebruiker op [Annuleren]([Cancel]) heeft gedrukt.

```
var invoer = "";
do {
  invoer = window.prompt("Geef het wachtwoord");
  if( invoer == null )
    break;
  alert("Verkeerd wachtwoord");
} while (invoer != "geheim") ;
if ( invoer != null ) {
  window.alert("Het wachtwoord is correct");
}
```

Met behulp van `continue` kan een enkele lusdoorgang worden overgeslagen. De lus loopt vervolgens gewoon door. In het volgende (overigens niet heel zinnige) voorbeeld worden alle getallen afgedrukt tussen 1 en 20 behalve het getal 13.

```
for( var teller = 1; teller <= 20; teller ++) {
  if( teller == 13 )
    continue;
  document.write teller + "<br>";
}
```

De `continue` wordt in de praktijk slechts zelden gebruikt. De `break` komt wat vaker voor, met name als met arrays wordt gewerkt.

# 4 FUNCTIES

## 4.1 Wat is een functie?

Vaak ziet u in uw code een aantal regels die tezamen een taak uitvoeren, in dat soort gevallen is het vaak raadzaam om deze regels code te bundelen tot een *functie*. Een functie is als het ware een recept – een aantal opdrachten wordt gebundeld tot een soort super-opdracht. Een functie heeft altijd een naam, en u kunt de code die in de functie staat laten uitvoeren door de functie via die naam 'aan te roepen'. De code 'springt' dan naar het begin van de functiecode, voert de regels die daar staan uit, en springt weer terug naar de functie-aanroep.

Het gebruik van functies heeft een groot aantal voordelen, zoals:

- U geeft een aantal regels code een duidelijke, verklarende naam.
- U vermijdt dubbele code, want u kunt de functie op meer plaatsen aanroepen.
- U kunt functies onderbrengen in een bibliotheek. Zo kunt u dezelfde code gebruiken op meer webpagina's.

Functies hoeven niet altijd precies hetzelfde te doen. Als u een recept hebt voor 10 oliebollen, dan is het niet moeilijk om te verzinnen hoe u er 20 moet maken. Zoiets kunt u meegeven als een zogenaamde *parameter* van een functie. U maakt dan niet een functie die 10 oliebollen maakt, maar een functie die N oliebollen maakt, waarbij u N pas specificeert op het moment dat dat nodig is.

JavaScript bevat zelf ook een groot aantal functies. Een aantal daarvan hebt u al gebruikt, zoals `parseInt`, `parseFloat`, `document.write`. Een bekende bibliotheek als jQuery bevat honderden functies.

## 4.2 Hoe schrijft u zelf een functie?

In JavaScript schrijft u een functie met behulp van het woord `function`. Achter dat woord zet u de naam van de functie, gevolgd door twee haakjes. Tussen die haakjes kunt u de variabelen kwijt die aan de functie kunnen worden meegeven. De code die bij de functie hoort zet u tussen accolades. Schematisch:

```
function myexample() {
    // code to do something
}

// call the function
myexample();
```

En dan een praktisch voorbeeld.

```
function writeparagraphs(n) {
  for( var i=1; i<=n; i++ ) {
    document.write("<p>Paragraph " + i);
  }
}
writeparagraphs(5);
writeparagraphs(3);
```

Hier ziet u een functie die met behulp van een lusje een x aantal paragrafen naar de webpagina schrijft. De waarde van x hangt af van de aanroep. U ziet de variabele die daarvoor gebruikt wordt ('n') tussen de haakjes van de functie-definitie staan. Zoiets wordt het *argument* van een functie genoemd. De variabele wordt nuttig gebruikt in de lus, om het maximum aantal aan te geven. Daaronder wordt de functie aangeroepen. De eerste aanroep levert vijf paragrafen op, de tweede aanroep voegt er nog drie aan toe.
Nog een ander voorbeeld:

```
function showInDialog(text) {
  if( text == "" || text == null )
    text = "<no text specified...>"
  window.alert(text);
}
showInDialog ("Show this on screen!");
showInDialog ("");
showInDialog ();
```

De functie 'showInDialog' toont in principe de tekst in een dialoog. Maar eerst wordt nog gecontroleerd of de variabele iets zinnigs bevat. De eerste aanroep levert de meegegeven tekst in een dialoog op, maar bij de andere twee komt er géén lege dialoog tevoorschijn, maar de melding dat er geen tekst is meegegeven.

## 4.3 Returnwaardes

Tot nu toe zag u functies die een opdracht uitvoeren. Maar er bestaan ook functies die antwoord geven op een vraag. Denkt u bijvoorbeeld aan `parseInt`: die functie accepteert een waarde (vaak een tekst) en stuurt als antwoord het gehele getal terug dat er in staat.

Om een functie een dergelijke *returnwaarde* te laten produceren maakt u gebruik van het woord `return`. Eigenlijk doet deze opdracht twee dingen: niet alleen wordt het antwoord dat er achter staat teruggestuurd, ook wordt de functie daarna direct verlaten.

```
function multiply(a,b) {
  return a * b;
}
document.write("5 * 3 = " + multiply(5,3));
```

In dit eenvoudige voorbeeld ziet u een functie die twee getallen accepteert als argumenten. Vervolgens wordt het product als antwoord teruggestuurd via een `return`.

Zoals u kunt zien in het bovenstaande voorbeeld, kunnen functies meer dan één argument hebben. Het voorbeeld hieronder accepteert zelfs drie parameters en geeft als antwoord de grootste van de drie terug. Zoals u in het voorbeeld ziet kan een functie meer dan één `return` statement bevatten. Zodra een `return` wordt uitgevoerd wordt direct teruggesprongen naar de aanroepende code.

```
function calculateMax(num1, num2, num3) {
  if( num1 >= num2 && num1 >= num3 )
     return num1;
  if( num2 >= num3 )
     return num2;
  return num3;
}
    var a = 17, b = 42, c = 39
    var maxnum = calculateMax(a,b,c);
        // maxnum will be 42
```

## 4.4 Scope (zichtbaarheid) van variabelen

U kunt niet elke variabele die u gebruikt overal binnen uw script gebruiken. Welke variabele op welk moment 'zichtbaar' is hangt af van de zogenaamde *scope* van de variabele. De basisregels binnen JavaScript zijn op dat gebied vrij eenvoudig:

- Elke variabele die u met `var` declareert binnen een functie is uitsluitend binnen die functie zichtbaar. Een parameter van een functie geldt ook als binnen die functie gedeclareerd. Dit staat ook wel bekend als *local scope*.
- Elke variabele die u buiten een functie declareert is door het gehele script zichtbaar. Dit staat bekend als *global scope*.
- Elke variabele die u *niet* declareert belandt ook in de *global scope*. Dus ook niet-gedefinieerde variabelen binnen functies.

Oppervlakkig gezien lijkt het misschien makkelijk om variabelen binnen de *global scope* te declareren. Maar niets is minder waar. Elke variabele binnen de *global scope* kunt u in feite vanuit elke functie overschrijven. Dit betekent dat uw code zeer kwetsbaar wordt voor namen die 'toevallig' hetzelfde zijn. Houdt u zich dus vooral aan de volgende richtlijnen voor wat betreft declaraties en scopes:

- Declareer **elke** variabele die u nodig hebt met `var`.
- Declareer een variabele die u alleen binnen een bepaalde functie nodig hebt ook *binnen* die functie.
- Als u waarden binnen meer functies nodig heeft, geef deze dan door via de parameters
- Gebruik in principe binnen een functie **nooit** variabelen uit de *global scope*.

Deze laatste regel begrijpt u vooral goed als u bedenkt dat u een functie ook wel eens zou willen kopiëren naar een andere webpagina (via een extern bestand is het waarschijnlijk handiger, maar het principe blijft hetzelfde). Als die functie een variabele uit de *global scope* gebruikt dan werkt de functie op een andere pagina dus niet!
Er valt nog meer te vertellen over scopes (zoals over zogenaamde *closures*) maar dat valt buiten het bestek van deze cursus.

## 4.5 Anonieme functies

Een functie die u vanaf elders wilt aanroepen heeft uiteraard een naam nodig. Maar een functie die slechts op één plek gebruikt wordt heeft eigenlijk geen naam nodig. Die naam mag dan ook weggelaten worden. Een dergelijke functie noemt men een *anonymous function*. Het voordeel van een anonymous function is dat u geen naam hoeft te bedenken, maar belangrijker is dat er daardoor ook nooit een naamconflict kan ontstaan. Anonymous functions worden vooral veel gebruikt in samenhang met events, die later in dit boek aan de orde komen. Ook in jQuery gebruikt men veel anonymous function.
De syntax is gelijk aan die van een functie met een naam, behalve dat u uiteraard de naam weglaat. Dus bijvoorbeeld zo:

```
button1.click = function() {
                    alert('you clicked me!');
                }
```

Hier wordt aangenomen dat *button1* naar een knop verwijst. Als er op geklikt wordt treedt de functie in actie. Aan dit onderwerp wordt later in deze cursus ruim aandacht besteed.

# 5 ARRAYS

## 5.1 Wat is een array?

Een variabele kunnen wij ons voorstellen als een doosje met een inhoud. Als u veel van deze doosjes hebt dan wordt het lastig om ze uit elkaar te houden. Wie honderd variabelen heeft moet ook honderd namen verzinnen. In zulke gevallen kunt u beter een zogenaamd *array* gebruiken. Bij een array hebben namelijk alle elementen dezelfde naam - maar een ander nummer. Als u dat niet meteen bekend voorkomt, denk dan aan een lange straat. Alle huizen hebben dezelfde naam (*Kerkstraat*, bijvoorbeeld), maar elk huis heeft een ander nummer. Dat is precies hoe een array werkt.
Er is een klein verschil met een straat. Het eerste element staat bekend als nummer 0. Een array met tien elementen is dus genummerd van 0 tot en met 9, nummer 10 bestaat niet.

## 5.2 Declaratie en gebruik

U kunt een array aanmaken via een `new Array`. U zet het aantal gewenste elementen daar tussen vierkante haken achter. Maar het kan ook korter door alleen maar vierkante haken te gebruiken, zoals hieronder.

```
var studentnames = new Array(20);
// alternatively
var studentnames = [ ];
```

In het eerste geval kunt u direct aangeven dat het array initieel 20 elementen zal bevatten. Maar u kunt arrays zonder enig probleem uitbreiden, dus het is niet nodig dit aantal aan te geven.
U kunt naar de elementen in een array verwijzen door het rangnummer van een element tussen vierkante haken te zetten. Denk eraan dat de nummering bij 0 begint. De waarde van het eerste element van het array `studentnames` krijgt u dus via `studentnames[0]`. `studentnames[1]` verwijst naar het volgende (tweede) element, enzovoort.
Na creatie van een array met het sleutelwoord `new`, hebben de eventueel aanwezige elementen de waarde `null`. Het array is leeg. De elementen kunnen als volgt worden gevuld:

```
studentnames [0] = "John";
studentnames [1] = "Patrick";
```

Onderstaand voorbeeld toont hoe een element op meer interactieve wijze kan worden gevuld.

```
var nr = window.prompt("Student number?");
studentnames[nr] = window.prompt("Name student?");
```

U kunt ook met getallen werken.

```
var test = new Array(2);
test[0] = 3;
test[1] = 5;
var sum = test[0] + test[1]; // sum will be 8
```

## 5.3 Arrays en for-loops

Arrays worden vrijwel altijd gebruikt in combinatie met een `for` loop. Hiermee kunt u met weinig code veel handelingen verrichten. In het volgende voorbeeld worden tien getallen gevraagd, die in omgekeerde volgorde in het document worden geschreven.

```
var numbers = [];
for (var i=0; i<10; i++) {
  numbers[i] = window.prompt("Enter number", 0);
}
for (var i=9; i>=0; i--) {
  document.write(numbers[i] + "<br>");
}
```

Hieronder een iets uitgebreider voorbeeld, met een functie die het gemiddelde van een array berekent. De functie controleert of de waarde in het array numeriek is. Zo niet, dan wordt de waarde genegeerd.

```
function average(arr) {
  var sum = 0, count = 0;
  for( var i=0; i<arr.length; i++) {
    if( ! isNaN(arr[i] ) ) {
      sum += parseFloat(arr[i]);
      count++;
    }
  }
  return sum / count;
}
```

Deze functie zou nu als volgt gebruikt kunnen worden:

```
  var arr = [];
  for (var i = 0; i < 5; i++) {
    arr[i] = i + 10.5;
  }
  arr[5] = "five";    // will be ignored in function
  alert(average(arr));  // will be 12.5
```

## 5.4 De length eigenschap

Elk array heeft een lengte. Deze lengte is op te vragen via <arraynaam>.`length`. Merk op dat het array automatisch de hoogst bekende index, min één, zal geven. Voorgaand voorbeeld kun je als volgt herschrijven:

```
var numbers = [];
for (var i=0; i<10; i++) {
  numbers[i] = window.prompt("Enter number", 0);
}
for (var i=numbers.length-1; i>=0; i--) {
  document.write(numbers[i] + "<br>");
}
```

De `length` is van belang als het array wordt doorgegeven aan een functie. Als het tweede deel van het voorbeeld wordt herschreven met een functie is de `length` onmisbaar, zie het volgende voorbeeld:

```
function writeArrayReversed ( arr ) {
  for (var i=arr.length-1; i>=0; i--) {
  document.write(arr[i] + "<br>");
}

var numbers = [];
for (var i=0; i<10; i++) {
  numbers[i] = window.prompt("Enter number", 0);
}
writeArrayReversed(numbers);
```

## 5.5 Initialisatie van arrays

Het is mogelijk een array direct te vullen met waarden bij de initialisatie. Beide vormen van declaraties ondersteunen deze mogelijkheid. In onderstaande voorbeelden ziet u dat een array gevuld wordt met de weekdagen. Een tweede array wordt gevuld met de eerste vijf priemgetallen.

```
var days = ["Sunday", "Monday", "Tuesday",
   "Wednesday","Thursday","Friday","Saturday"];
var primes = new Array(2,3,5,7,11);
```

## 5.6 Arrays met 'gaten' (sparse arrays)

Het is toegestaan om elementen te gebruiken zonder dat alle voorgaande elementen reeds gevuld zijn. Een dergelijk array met 'gaten' noemt men een 'sparse array'. Alle elementen die u niet expliciet vult bevatten niets (null). U blijft er zelf verantwoordelijk voor om te testen of een bepaald element een waardevolle waarde bevat.

```
var numbers = [];
numbers[6] = 300;
// the length is now 7. Elements 0-5 are empty.
```

Dergelijke arrays zijn zinvol wanneer het vullen van een element lang duurt. Denk hierbij aan het dynamisch ophalen van afbeeldingen, zoals dat met AJAX zou kunnen gebeuren. De precieze invulling hiervan valt buiten het bestek van deze cursus.

## 5.7 Arrays in de praktijk

U zult zich afvragen of het vaak voorkomt dat u zo veel gegevens in JavaScript gebruikt, dat u een array nodig hebt. Als het gaat om gegevens die u zelf aanlevert komt dat inderdaad niet vaak voor. Maar arrays komen op allerlei plekken voor zodra u gegevens over de pagina opvraagt. U kunt bijvoorbeeld:

- Alle images (plaatjes) op een pagina opvragen.
- Alle `div`-elementen opvragen.
- De rijen en kolommen van een tabel benaderen.

In dit soort gevallen krijgt u de gegevens aangeleverd als een array. U moet dan kunnen inzien hoeveel elementen er zijn, en wat u met de elementen kunt doen. Dan zult u ervaren waarom het gebruik van arrays zinvol is.

## 5.8 Extra : Meerdimensionale arrays

U heeft ze waarschijnlijk niet vaak nodig, maar u kunt in JavaScript ook meerdimensionale arrays gebruiken. In feite maakt u een array, van arrays. Elk element van het array is op zijn beurt ook weer een array. Een tweedimensionaal array kunt u zien als een tabel, een driedimensionaal array als een data-kubus. Meer dimensies mogen ook, maar het voorstellen ervan wordt lastiger.
In het volgende voorbeeld ziet u hoe u een tabel maakt van 10 bij 20 waarden. Na de code zitten er dus in totaal 200 waarden in het array. De eerste 'rij' bevat de waarden 1, 2, 3 enzovoorts, tot en met 20. De volgende rij bevat de waarden 2, 4, 6 en zo verder. De laatste rij bevat de waarden 10, 20 tot en met 200. In de laatste regel worden ook nog alle 200 waarden getoond via de handige methode `toString`();

```
var arr = [];
for (var i = 0; i < 10; i++) {
  arr[i] = [];
  for (var j = 0; j < 20; j++) {
    arr[i][j] = (i+1) * (j+1);
  }
}
alert(arr.toString());  // show all values
```

# 6 OBJECTEN EN OBJECTMODELLEN

## 6.1 Inleiding objecten

In veel programmeertalen wordt met objecten gewerkt. Een object is een voorwerp dat bepaalde eigenschappen heeft, en bepaalde acties kan uitvoeren. De eigenschappen en de acties zijn specifiek voor dat type object. De technische termen voor acties en eigenschappen zijn *methods* en *properties*.
Een object van het (fictieve) type hond kan properties (eigenschappen) hebben zoals naam, ras, gewicht, geboortedatum, enzovoort. Properties zijn meestal waardes. Ze zijn meestal op te vragen, en vaak ook in te stellen.
De methods (acties) van een hond kunnen bijvoorbeeld zijn: blaf, bijt, eet, slaap, apport enzovoort.
In de meeste programmeertalen bereikt men de methods en de properties door een punt (.) achter het object te zetten. In slimme editors verschijnt er dan een uitklaplijstje met de opties. Dit maakt het veel eenvoudiger om met objecten te werken.
Enkele voorbeelden van methods en properties zijn eerder in de cursus al gebruikt: bij *document.write* gebruikt u de *write* methode van het *document* object. Een *array* heeft een *length* property. En het *window* object heeft methodes genaamd *alert* en *prompt*.

## 6.2 JavaScript en objecten

Hoewel JavaScript geen echt objectgeoriënteerde taal is spelen objecten toch een belangrijke rol. In een browser wordt gebruik gemaakt van een JavaScript interpreter. U kunt dit zien als een aanvullend programma op de browser. Dit programma is in staat een aantal zaken die met de browser en het geladen document te maken hebben te onderscheiden. Het programma heeft weet van de browser, het venster, het document en een aantal elementen in het document. Al deze zaken, die door het programma onderscheiden kunnen worden, zien we in JavaScript terug als objecten.

## 6.3 Objecten zijn dynamisch

In JavaScript krijgt u al veel objecten cadeau. Maar u kunt ook zelf nieuwe objecten maken met nieuwe properties. U kunt een nieuw object maken door een `new Object` op te vragen. De nieuwe properties kunt u zelf bepalen. Als u vindt dat een bepaald object een bepaalde property moet hebben, dan kunt u deze gewoon instellen. Vanaf dat moment heeft dat object ook inderdaad die property. Zie onderstaand voorbeeld waar een persoon wordt gedefinieerd.

```
var person = new Object;
person.firstname = "John";
person.lastname ="Smith";
alert (person.firstname);   //  "John"
```

U kunt aan een object vragen of dit een bepaalde eigenschap bezit. U doet dit eenvoudig via een `if`-statement waarbij u vraagt om de bepaalde eigenschap. Als deze niet bestaat zal dit `undefined` opleveren, hetgeen JavaScript interpreteert als `false`. Zie onderstaand voorbeeld waar de achternaam en het adres worden getoond, uitsluitend als deze bestaan.

```
    if( person.lastname )
      alert(person.lastname);
    if( person.address )
      alert(person.address);
```

Deze manier om te controleren of een eigenschap bestaat (de techniek werkt overigens ook voor methodes) is zeer belangrijk voor het oplossen van browser-specifieke problemen. Als u twijfelt of een bepaalde browser een bepaalde methode ondersteunt kunt u dat eerst controleren.

## 6.4 Associatieve arrays

Veel talen kennen een soort array waarbij elementen niet via een index worden opgevraagd, maar via een naam. Dit is in het dagelijks leven een gebruikelijke manier van doen. Als u tien collega's hebt, dan weet u vast wel wie collega 'Margriet' is. Maar als men u vraagt wie collega nummer zes is, dan weet u vermoedelijk niet wie er wordt bedoeld. Een array waarbij de elementen via een naam worden benaderd noemt men een associatief array.

JavaScript ondersteunt deze manier van werken. U kunt als volgt met een willekeurig nieuw object aan de slag:

```
var person = new Object;
person["firstname"] = "John";
person["lastname"] ="Smith";
```

Wellicht vindt u deze manier van werken nogal lijken op de vorige. In dat geval heeft u gelijk. In JavaScript is een object eigenlijk hetzelfde als een associatief array. U kunt dus zowel deze manier van werken gebruiken, als de hierboven geschetste wijze. De werkwijze via een property is eenvoudiger in te tikken, maar de werkwijze via het associatief array heeft ook een voordeel. U kunt namelijk tussen de vierkante haken een variabele gebruiken. Bekijk bijvoorbeeld de code hieronder:.

```
var person = new Object;
person.firstname = "John";
person["lastname"] = "Smith";

var text = "firstname";
alert (person.text);    // will not work
alert (person[text]);   // works fine
```

Als u de eigenschap van een object wilt opvragen via een variabele, dan bent u verplicht dit via de schrijfwijze met vierkante haken te doen.

## 6.5 De for ... in lus

Als u wilt weten of een object een bepaalde eigenschap heeft kunt u dit controleren. Als u wilt weten welke zaken een object zoal ondersteunt kent JavaScript de `for ... in` lus. Met deze lus kunt u alle properties en methodes opvragen van een specifiek object. Merk op dat hierbij de array-syntax benodigd is. In onderstaand voorbeeld wordt aan een zojuist gemaakt object gevraagd welke eigenschappen het heeft. Er zullen twee boodschappen verschijnen. De eerste keer wordt de voornaam getoond, de tweede keer de achternaam:

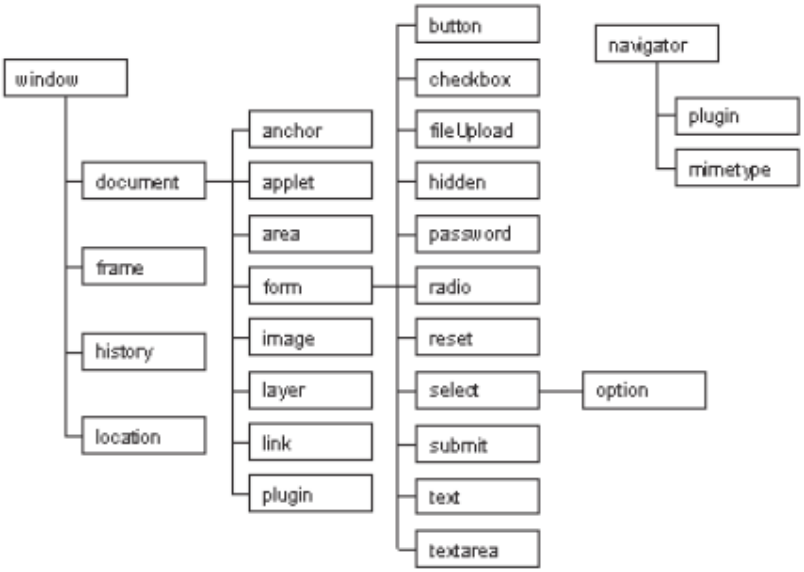
```
var person = new Object;
person.firstname = "John";
person["lastname"] ="Smith";

for( var key in person )
  alert( key + ": " + person[key]);
```

De `for ... in` lus is een zeer krachtig middel om te onderzoeken welke eigenschappen een bepaald object in een browser ondersteunt.

## 6.6 Het Windows Object Model

Het Windows Object Model vormt de basis voor de indeling van objecten. U ziet in de figuur de indeling van de boom van objecten. Merk op dat het belangrijkste deel van de objecten onder het document object te vinden is. Dit model wordt dan ook in een aparte sectie besproken.




Hoewel u diverse objecten moet aanmaken, geldt voor de objecten in dit model dat ze allemaal door de browser worden aangeleverd. Zodra u een

pagina geladen heeft zijn al deze objecten, mits aanwezig op de pagina, dus benaderbaar.

## 6.7 Enkele objecten nader beschouwd

Het object `window` verwijst naar het venster waarin een document getoond wordt. Het object `window` vormt de absolute top van de boomstructuur van objecten, en staat ook bekend als de *global scope*. Wanneer u een variabele buiten een functie declareert wordt dat bijvoorbeeld een property van het `window` object. Het heeft enkele veel gebruikte methods. U maakte al eerder kennis met twee van deze methods, namelijk `alert()` en `prompt()`.

| | |
|---|---|
| `window.open(parameters)` | Opent een nieuw browservenster. U kunt hierbij allerlei zaken aangeven zoals de URL, afmetingen en aanwezigheid van menubalken. |
| `window.alert(text)` | Geeft een simpel dialoogvenster met de door u opgegeven tekst en een knop [OK]. |
| `window.prompt(text, [default])` | Geeft een dialoogvenster met de tekst (een vraag bijvoorbeeld) en een invoerveld waarin de bezoeker iets kan invullen. U kunt desgewenst al een standaardtekst opgeven. Deze method geeft ook een waarde terug: de ingevulde tekst. |
| `window.confirm(text)` | Het dialoogvenster Confirm biedt twee knoppen: [OK] en [Annuleer]. Deze method retourneert de waarde `true` (OK) of `false` (Annuleer). |

**Hieronder enkele voorbeelden van deze methods:**

```
var answer = window.confirm("Do you want to continue?");
window.alert(answer);
```

```
window.open("http://www.google.nl", "_blank", "status=no,s
crollbars=no,resizable=no," +                "width=400,hei
ght=350,left=200,top=150");
```

Het `window` object heeft een aantal properties beschikbaar, zoals:

| | |
|---|---|
| `window.navigator` | Toegang tot het navigator object met diverse browser-specifieke informatie. |
| `window.location` | Levert info over de url en de host. |
| `window.status` | Benadert de statusbalk. Dit is achterhaald en wordt in bv Chrome niet meer ondersteund. |
| `window.screen` | Informatie over het scherm (o.a. grootte). |

**Enkele voorbeelden van het gebruik:**

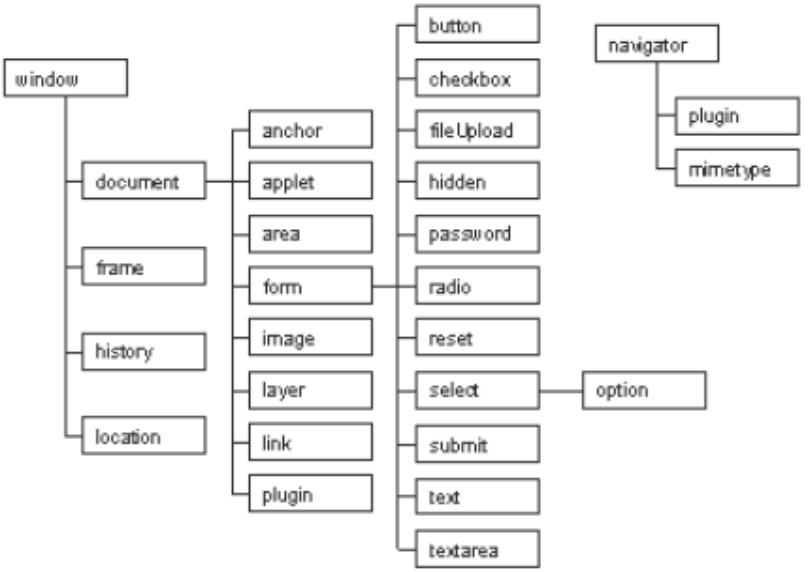
```
// Show url of this page
// For instance: "file:///C:/JavaScript/test.html"
   alert(window.location.href);
// show browser language, for instance "en-US"
   alert(window.navigator.language);
// display screen width and height
   alert(window.screen.width);  // for instance 1024
   alert(window.screen.height); // for instance 768
```

# 7 HET DOCUMENT OBJECT MODEL

Zoals u in de figuur van het window object model gezien hebt vallen zeer veel objecten onder het document object. De indeling van al die objecten in een boomstructuur staat bekend als het *Document Object Model*, of kortweg de DOM. Merk op dat de DOM geen onderdeel is van JavaScript, maar wel bijzonder veel wordt gebruikt door JavaScript. De combinatie hiervan staat ook wel bekend als Dynamic HTML (DHTML). Wij zullen nu ingaan op de relaties tussen de verschillende objecten van de DOM en een aantal ervan nader bespreken.

## 7.1 Objecthiërarchie

Wanneer u naar de objecten gaat kijken, zult u zien dat ze een bepaalde samenhang hebben. Een *textbox* bij voorbeeld is een onderdeel van een formulier (*form*), dat op zijn beurt weer een onderdeel is van het *document*, dat in een *window* wordt getoond. Sommige objecten zitten in een array, bijvoorbeeld forms en images. Hieronder ziet u een schema van de hiërarchie van de objecten.



Wanneer u een bepaald object wilt aanspreken, moet u de hiërarchie aflopen en alle objecten die u op uw weg tegenkomt benoemen, zodat u zeker weet dat u het juiste object benadert. De (namen van de) objecten scheidt u, zoals eerder genoemd, door middel van een punt.

```
<html>
<body>
  <form name="data">
  Address: <input name="address" value="Abbey Road 42">
  </form>
  <script>
    window.alert(window.document.data.address.value);
  </script>
</body>
</html>
```

Dit voorbeeld toont aan hoe u gegevens uit een formulier kunt halen. Als dit script wordt uitgevoerd in een browser, zal in de browser een formulier met één invoerveld worden getoond met daarin de waarde `"Abbey Road 42"`. Deze waarde wordt ook in een dialoog getoond.

De constructie met het `name` attribuut werkt overigens alleen voor form elementen (buttons, textboxen, listboxen, textarea's en dergelijke). Sommige andere browsers staan het gebruik van deze techniek ook toe bij andere elementen, maar volgens de standaard is dat eigenlijk niet toegestaan, en in veel browsers wordt dat niet ondersteund.

In voorafgaand script wordt naar het tekstvakobject `adres` van het formulier 'data' verwezen met `window.document.data.address`. Merk op dat `window` kan worden weggelaten, waardoor de verwijzing wat korter wordt.

```
alert(document.data.address.value);
```

Soms is er aan formulieren geen naam gegeven. In dat geval kan het DOM-object `forms` worden gebruikt. Dit object (een array) bevat verwijzingen naar alle formulieren in het document. Het eerste formulier wordt dan geïdentificeerd met `forms[0]`.

```
alert(forms[0].adres.value);
```

Op dezelfde manier is er ook een array `images`, met verwijzingen naar alle plaatjes in het document.

## 7.2 Het document object

Het object `document` verwijst naar het document dat in een venster getoond wordt. Hieronder enkele methods die tot dit object behoren.

| | |
|---|---|
| `document.clear()` | Wist de inhoud van het document. |
| `document.close()` | Sluit het document. |
| `document.open(parameters)` | Opent een nieuw document in het venster. |
| `document.write(tekst)` | Schrijft de meegegeven tekst in het document. |
| `document.writeln(tekst)` | Zoals de vorige, maar dan met een carriage return achter de tekst. |

Een aantal belangrijke properties van `document` zijn:

| | |
|---|---|
| `document.bgColor` | Instellen achtergrondkleur. |
| `document.title` | Titel document. |
| `document.location` | Stel URL in van weer te geven pagina. |

```
<html>
<body>
  <script>
    document.title = "My new title";
    if (window.confirm("Browse to Google?")) {
      document.location = "http://www.google.nl";
    }
  </script>
</body>
</html>
```

## 7.3 Opvragen van elementen binnen de DOM

Wat nu als u een element wilt lokaliseren maar u niet precies weet waar dit zich bevindt? Of wat moet u doen als u een bepaald type element wilt lokaliseren? Er zijn enkele manieren om een bepaald element of een bepaald soort element op te vragen.

U kunt form-elementen binnen een formulier altijd via hun naam opvragen. Zo kunt u de inhoud van een textbox genaamd 'phone' binnen een formulier 'userdata' opvragen op onderstaande manier.

```
var content = document.userdata.phone.value;
```

Maar het opvragen van elementen via hun naam ('name') werkt alleen voor form-elementen binnen formulieren. Daarbuiten werkt het officieel niet. Voor andere elementen kunt u het beste een element een `id` geven. De waarde van het `id` moet uniek zijn binnen de pagina.
U kunt vervolgens dit element opvragen via de methode `getElementById`. Deze methode doorzoekt de gehele boom van elementen en zal (indien het bestaat) het juiste element opleveren. Met dit element kunt u dan verder werken. U kunt bijvoorbeeld de `innerHTML` van het element instellen. Deze eigenschap bevat alle tekst binnen het element, inclusief HTML tags.
Hieronder volgt een complexer voorbeeld. Hier zal een prompt verschijnen om tekst in te voeren, welke vervolgens boven in de pagina wordt getoond.

```
<body>
  <h1>Your input: <span id="spn"></span></h1>

  <script>
    var spanobject = document.getElementById("spn");
    var input = window.prompt("Enter a text");
    spanobject.innerHTML = input;
  </script>
</body>
```

Het `span` element heeft hier een id ('`spn`') en dus kan het element worden opgevraagd via `getElementById()`. Vervolgens kan het element worden gemanipuleerd. Zo wordt hier de tekst in de `span` tag aangepast door de waarde van de property `innerHTML` te vervangen door de nieuwe invoer.
De property `innerHTML` werkt niet alleen met `span`, maar bij nog veel meer elementen, zoals headers en lijsten. Merk op dat u op deze manier tussen HTML-tags niet alleen tekst kunt invoegen, veranderen of weghalen, maar ook HTML-code. U kan nieuwe elementen voor een lijst `<ul id="mylist">` tonen met onderstaande code.

```
var mylist=document.getElementById('mylist');
mylist.innerHTML = "<li>Horse<li>Cow<li>Sheep";
```

U kunt elementen weer laten verdwijnen met onderstaande code.

```
mylist.innerHTML = "";
```

Als u elementen wilt opvragen van een bepaalde soort (tag) dan gebruikt u `getElementsByTagName`. U geeft aan deze methode een tagnaam mee en krijgt een array terug met al die elementen er in. Vervolgens kunt u met een lus al deze elementen aflopen, zoals in dit voorbeeld, waar alle `div`-elementen een gele achtergrondkleur krijgen:

```
var divs = document.getElementsByTagName("div");     for(
var i = 0; i < divs.length; i++) {
    divs[i].style.backgroundColor = "yellow";
}
```

Merk op dat u met de property `innerHTML` de inhoud tussen de begin- en eindtag van een HTML-element aanpast. Voor het aanpassen van de inhoud van een `form` element moet de property `value` worden gebruikt en voor het veranderen van de inhoud van een image element (het plaatje) de property `src`.
Voor het wijzigen van de stijl van een HTML-element gebruikt u de `style`. Zo is de `style.color` de kleur van de tekst, enzovoorts. U kunt ook nog de class van een element wijzigen via de `className`. Het element zal dan een geheel andere stijl krijgen. Daarmee kunt u soms met heel weinig code veel effect bereiken.

## 7.4 Toevoegen van elementen aan de DOM

Een van de krachtigste technieken in JavaScript is het toevoegen van elementen aan een pagina. Op deze wijze kunt u in feite een complete pagina opbouwen, los van de HTML in de oorspronkelijke pagina die van de server kwam. Door deze techniek kunt u dus applicaties maken die geheel op de client werken en de server niet meer nodig hebben als de pagina eenmaal geladen is.
Elementen maakt u nieuw aan via `document.createElement`. U geeft daarbij in de vorm van een string aan welk element u wilt maken. Vervolgens vult u het element met HTML of tekst. Ook kunt u een id toekennen, de stijl aanpassen enzovoorts. Tenslotte voegt u het nieuwe element toe aan een ander element via de `appendChild` methode. Een eenvoudig voorbeeld ziet u hieronder.

```
var list = document.getElementById('multitable');
for (var i = 1; i <= 10; i++) {
  var p = document.createElement('p');
  p.innerHTML = '7 * ' + i + ' = <b>' + (7 * i) + '</b>';
  p.style.color = 'blue';
  list.appendChild(p);
}
```

In de eerste regel wordt een element opgehaald dat 'multitable' heet. Dit element moet dus wel al bestaan (hoewel u ook gewoon elementen kunt toevoegen aan de body van een pagina). Daarna wordt tienmaal een nieuwe paragraaf ('p') gemaakt. De paragraaf wordt gevuld met de uitkomst van een vermenigvuldiging, waarbij de uitkomst vet gemaakt wordt. Daarna wordt de hele paragraaf blauw gemaakt, waarna de paragraaf uiteindelijk wordt toegevoegd aan het 'multitable' element. Zo ontstaat de tafel van zeven op de webpagina.

Als u nu bedenkt dat u ook een paar tekstboxjes op de pagina zou kunnen plaatsen waarin u de waarden '10' en '7' invult, en die waarden in de code uitleest en gebruikt, dan ziet u dat u heel flexibel kunt zijn. U kunt dan ook de tafel van 29 tot en met 100 laten afdrukken.

Overigens, als u bovenstaande code tweemaal zou uitvoeren krijgt u de uitvoer ook tweemaal achter elkaar - de vorige uitvoer wordt dus in dit voorbeeld niet gewist. Maar dat is wel degelijk mogelijk. Voordat u elementen toevoegt kunt u alle `childNodes` weggooien. U test in een lusje of er nog een (laatste) `childNode` bestaat. Zo ja, dan wordt deze gewist. Uiteindelijk blijft dus alleen het lege hoofdelement over:

```
var list = document.getElementById('multitable');
while (list.lastChild) {
  list.removeChild(list.lastChild);
}
for (var i = 1; i <= 10; i++) {
  var p = document.createElement('p');
  p.innerHTML = '7 * ' + i + ' = <b>' + (7 * i) + '</b>';
  p.style.color = 'blue';
  list.appendChild(p);
}
```

U kunt dus niet alleen elementen toevoegen, maar u kunt ze ook naar believen weer verwijderen. Dat is wat JavaScript in combinatie met de DOM zo krachtig maakt.

# 8 EVENTS

## 8.1 Wat is een event?

De kracht van het gebruik van JavaScript in combinatie met de DOM komt pas goed tot uiting bij de toepassing van events. Een event (of ook wel gebeurtenis) is datgene wat plaatsvindt als er iets verandert aan de pagina. Zodra een dergelijke verandering wordt waargenomen wordt dit verstuurd als een *event*. Op dit event kunt u reageren via JavaScript. Bijvoorbeeld het klikken op een button op de pagina, maar ook het bewegen van de muis en het gebruik van het toetsenbord leiden tot het zogenaamde 'afvuren' van een event. Ook zijn er events die plaatsvinden zonder dat de gebruiker daar de hand in heeft, zoals bijvoorbeeld het event dat plaatsvindt als de pagina compleet geladen is.

## 8.2 Soorten events

De naam van een event is in JavaScript altijd het woordje 'on' gevolgd door het soort event. Dus *onclick*, *onchange*, enzovoort. Vreemd genoeg zijn de namen van events *niet* case-sensitive. U mag dus `onclick` gebruiken, maar ook `onClick`, `OnClick` of zelfs `ONCLICK` (maar dat staat niet erg fraai). De meest gebruikte vormen zijn die waarin alles met kleine letters wordt geschreven of de vorm waar het woord in *camelCase* staat. Dus `onclick`, of `onClick`.
Er zijn tientallen soorten events, maar niet alle events komen bij alle soorten elementen voor. Zo kan een textbox de invoerfocus hebben en dus is er een event dat aangeeft dat dit gebeurd is. Maar een stukje tekst op een pagina kan geen focus hebben, dus kan daar nooit een dergelijk event optreden. Hoewel er tientallen soorten events zijn blijft het gebruik in de praktijk meestal beperkt tot een vrij klein aantal, dat u hieronder in een tabelletje aantreft.

| | |
|---|---|
| onload | Vindt alleen plaats bij de pagina zelf, als de pagina compleet geladen is |
| onclick | Treedt bij veel elementen op, wanneer op het element geklikt wordt met de muis |
| onfocus | Treedt op bij een invoerelement (zoals een textbox) dat de focus ontvangt |

| | |
|---|---|
| onblur | Treedt op bij een invoerelement wanneer het element de focus weer verliest |
| onchange | Treedt op bij invoerelementen (textbox, textarea) wanneer de inhoud verandert |
| onselect | Treedt op bij een listbox als de selectie verandert |
| onmousover | Treedt op als de muis boven een element zweeft |
| onmouseout | Treedt op als de muis een element verlaat |
| onsubmit | Treedt op wanneer de inhoud van een formulier verstuurd gaat worden naar de server |

## 8.3 Event handlers

U kunt een event koppelen aan een stukje JavaScript. Een dergelijk stukje script noemt men doorgaans een *event handler*. U kunt voor ieder event een aparte handler maken, alhoewel u later zult zien dat handlers ook events van diverse elementen kunnen afhandelen.
Een event handler voegt u het eenvoudigst toe als een attribuut aan een HTML-element. Daarbij geeft u dan direct aan welke actie er moet worden ondernomen als het event optreedt. In onderstaand voorbeeld ziet u dat een knop voorzien is van een event handler `onclick`, hetgeen aangeeft dat de code wordt afgevuurd als er op de knop wordt geklikt. Zodra de gebruiker op de knop klikt komt er dus een dialoog naar voren.

```
<input type="button" value="click here"
       onclick="window.alert('Success!')">
```

Merk op dat de code achter onclick tussen aanhalingstekens moet worden geplaatst, maar dat in dit geval in die code weer een andere string zit, die dus weer tussen andere aanhalingstekens komt. U mag zelf kiezen welke soort u eerst gebruikt, dus het mag zo:

```
onclick="window.alert('Success!')"
```

of ook zo:

```
onclick='window.alert("Success!")'
```

U kunt naar aanleiding van een event meer opdrachten laten uitvoeren door deze te scheiden met puntkomma's.

```
onclick="document.bgColor='red';
         window.alert("Success!')"
```

## 8.4 Het woordje 'this'

In veel gevallen komt het voor dat u iets wilt wijzigen zodra er iets mee gebeurt. Een veelgebruikt effect is bijvoorbeeld om de lay-out van een stuk tekst te wijzigen als de gebruiker er met de muis overheen beweegt. Hierbij gebruikt men de muis events `onmouseover` en `onmouseout`. Hoe geeft u nu aan dat u de stijl van het betreffende element wilt wijzigen? U kunt natuurlijk een id gebruiken en daarmee dit element weer opzoeken, maar er is gelukkig een kortere manier. Met het woordje `this` kunt u aangeven dat u 'dit element' bedoelt, het element waar het event is opgetreden. Zie onderstaand voorbeeld, waar de tekst groen wordt als de muis erboven hangt.

```
<div onmouseover="this.style.color='green'"
     onmouseout="this.style.color='black'">
  Hover the mouse over this text to turn it green
</div>
```

## 8.5 Koppelen van functies aan events

In simpele gevallen wilt u op een event reageren met een enkele opdracht. In dergelijke gevallen is er weinig op tegen om die regel code gewoon binnen de HTML te zetten zoals in het vorige voorbeeld. Als de code wat langer wordt, en ook als de code vaker voorkomt, wordt het handiger om die code in een functie te zetten en de functie aan te roepen vanuit de event handler. Zo is het vrij gebruikelijk om bij het starten van een pagina een aantal zaken af te handelen. Het is onhandig om al dit script in de body-tag te verstoppen. Veel beter is het dan om een functie te maken, bijvoorbeeld genaamd 'start'. In de body-tag vindt u dan het volgende:

```
 <body onload="start()">
```

Nu kunt u alle opstartcode gewoon in de functie kwijt.
Een andere mogelijkheid is om dezelfde functie te koppelen aan meer events. Stel bijvoorbeeld dat u wilt dat bepaalde gedeeltes van kleur veranderen als

de gebruiker erop klikt. Bij een eerste klik moet de tekst rood worden, bij een tweede klik weer zwart. U maakt nu eerst een functie, als volgt:

```
function modifyColor(element)
{
  if (element.style.color == "red")
    element.style.color = "black";
  else
    element.style.color = "red";
}
```

U ziet dat de functie een argument heeft (element). U kunt er nu eenvoudig voor zorgen dat dit element de juiste waarde krijgt door het woordje 'this' door te geven aan de functie:

```
<div onclick="modifyColor(this)">
     Line one</div>
 <div onclick="modifyColor(this)">
     Line two</div>
```

Niet alleen is de functie hier bijzonder handig om de code in de HTML veel korter te maken, ook zorgt het gebruik van een functie er voor dat u deze code op meer plaatsen kunt toepassen.

**Dynamisch veranderen van class**

Natuurlijk kunt u in de functies die u schrijft diverse aspecten van de stijl van het element wijzigen. Maar nog efficiënter kan het zijn om die stijlen apart vast te leggen op CSS-niveau en simpelweg de klasse van het element te veranderen. Laten we aannemen dat u in de pagina al een paar stijlen hebt gedefinieerd, bijvoorbeeld:

```
<style type="text/css">
    div.larger
    {
      font-size: 15pt;
      color: Red;
    }
    div.smaller
    {
      font-size: 9pt;
      color: Gray;
    }
  </style>
```

U kunt nu eenvoudig van een element de stijl veranderen door van dit element de `className` te wijzigen. Dus als volgt:

```
  <div onmouseover="this.className='larger'"
       onmouseout="this.className='smaller'">
    Hover the mouse over to enlarge</div>
```

Op deze manier kunt u zeer veel effecten tegelijk bereiken. Door het gebruik van de CSS kunt u bovendien de stijl van veel elementen tegelijk wijzigen zonder dat u de JavaScript gedeeltes hoeft aan te passen.

**DHTML en images**

De combinatie van JavaScript met images komt bijzonder vaak voor. Hiermee maakt u bijvoorbeeld dia-shows en dergelijke. De meeste van deze toepassingen komen ongeveer op hetzelfde neer: de gebruiker klikt op een klein plaatje en de vergrote versie van dit plaatje verschijnt. Wat ook vaak voorkomt zijn een paar knoppen waarmee u van voor naar achter door een reeks plaatjes kunt lopen. In vrijwel alle gevallen komt het er op neer dat u de `src` van een image wijzigt. In onderstaand voorbeeld kunt u een foto in vol formaat zien door op een miniversie te klikken

```
<img src="picture1.jpg" width="40" height="40"
  onclick="document.images.bigpic.src=this.src">
<img src="picture2.jpg" width="40" height="40"
  onclick="document.images.bigpic.src=this.src">
<p>
<img id="bigpic" src="">
```

Let weer op het gebruik van `this` waarmee u dus hier aangeeft: deze image. U kunt de laatste image `document.images.bigpic` noemen, want voor images bestaat een speciale collectie, genaamd `document.images`. Vanuit images kun u elke image benaderen met als naam het `id` van dat image.
Als de gebruiker dus op de bovenste image klikt wordt de bron daarvan doorgegeven aan de 'bigpic'. Aangezien daar de `width` en `height` niet staan ingesteld zal het oorspronkelijke formaat van het plaatje worden gebruikt. De eerste twee plaatjes zullen slechts op het formaat 40 bij 40 worden weergegeven.

## 8.6 Dynamisch koppelen van events

Stel u dat u in bovenstaand voorbeeld 20 plaatjes hebt. Of dat u heel veel `div`-elementen hebt waarbij u steeds hetzelfde effect wilt laten optreden. Zoals u eerder zag is het een goed idee om de code in een functie te zetten. Maar dan nog is het een vervelend werkje om bij elk HTML-element de code te koppelen. Kan dat eenvoudiger? Zeker, dat kan. U kunt de event handlers zelf ook weer vanuit JavaScript koppelen. In de praktijk is dat zelfs heel gebruikelijk.
Stel bijvoorbeeld dat u een groot aantal `div`-elementen hebt, waarbij een bepaalde functie (laten we zeggen: 'dostuff') gekoppeld moet worden aan het `click` event. U kunt dit oplossen door voor elk element de methode `addEventListener` te gebruiken. U kunt dit doen bij het starten van de pagina, dus code die start bij het 'onload' event. Eerst haalt u bijvoorbeeld alle `div`-elementen op via de methode `getElementsByTagName`. Dan krijgt u een array met alle `div`'s. Vervolgens doorloopt u dat array met een lusje, en voegt aan elk element de event listener toe.

```
var divs = document.getElementsByTagName("div");
for (var i = 0; i < divs.length; i++) {
    divs[i].addEventListener("click", dostuff);
}
```

Het event dat u wilt koppelen geeft u in dit geval aan **zonder** het woordje "on" ervoor. Let er verder op dat u in dit geval de functienaam gebruikt **zonder** er haakjes achter te zetten. U wilt de functie op dit moment, vanuit deze regel code nog *niet* aanroepen. U wilt de code van de functie als geheel koppelen aan het event. Pas als het event optreedt, *dan* moet de functie worden aangeroepen.
Nu wordt bij elk `div`-element de functie 'dostuff' uitgevoerd als u er op klikt. Probleem opgelost? Nog niet helemaal. Want als u de elementen stuk voor stuk

in de HTML zou koppelen dan gaf u meestal het woordje `this` mee. Dat gebeurt hier niet. Hoe weet de functie nu op welk element er geklikt is?

## 8.7 Bepalen van de source van een event

Als u een functie wilt gebruiken om te koppelen aan een event dan geeft u die functie altijd een parameter mee. Hoewel het niet verplicht is wordt deze parameter doorgaans 'e' genoemd. Deze parameter bevat informatie over het event dat door de browser gegenereerd is. Met name bevat het een eigenschap genaamd `target`, die verwijst naar het element uit de DOM waar het event vandaan komt. In het volgende voorbeeld ziet u hoe het element waarop geklikt is een groene tekst krijgt.

```
function dostuff(e) {
    e.target.style.color = "forestgreen";
}
```

## 8.8 Events en anonymous functies

Veel functies worden slechts eenmalig toegepast in de code. In dergelijke gevallen is het aan te raden om een *anonymous function* te gebruiken. Een dergelijke functie heeft geen naam en kan dus ook geen naamconflict opleveren.

```
var button1 = document.getElementById('button1');
button1.addEventListener('click', function () {
      alert('you clicked button 1');
    });

document.getElementById('button2').
    addEventListener ('click', function () {
      alert('you clicked button 2');
    });
```

In het eerste voorbeeld wordt eerst het buttonelement opgehaald waarna er eenanonieme functie aan wordt gekoppeld. In het tweede voorbeeld wordt ook voor de button geen variabele mer gebruikt. Het resultaat van de aanroep naar `getElementById` wordt direct gebruikt om de anonieme functie aan

het event te koppelen. Hoewel deze code wellicht complex lijkt wordt deze techniek vaak gebruikt.

## 8.9 Extra: compatibility met oude versies van IE

Helaas was er vroeger geen eenduidige manier om uit te vinden waar nu precies het event vandaan komt. Dat wil zeggen, toen dit alles nog in ontwikkeling was heeft men bij Microsoft een bepaalde oplossing gekozen, daarmee op de zaken vooruitlopend. Later is de standaard anders geformuleerd. Als uw pagina's per se op oudere versies van IE moeten draaien (versie 8 of eerder) dan zult u even wat listige code nodig hebben, want daar werkt `e.target` niet. Bovendien kent IE8 nog geen `addEventListener`.
Om te beginnen zult u dus moeten testen of de browser `addEventListener` ondersteunt. Indien dit niet het geval is dan kunt u terugvallen op de oudere `onclick` methode. Om dat te bereiken wijzigt u de code voor het instellen van de event handlers als volgt:

```
for( var i=0; i < divs.length; i++ ) {
  if( divs[i].addEventListener )
     divs[i].addEventListener("click",dostuff);
  else
     divs[i].onclick = dostuff;
}
```

Merk op dat u met het if-statement alleen maar test of de methode bestaat. Als dit zo is dan kunt u deze aanroepen, anders niet.
Daarmee bent u er nog niet, want in eerdere versies van IE werd het event niet automatisch doorgegeven. In plaats daarvan had u `window.event` nodig. Niet alleen dat, maar ook kende IE geen `target`. In plaats van `target` hebt u dan `srcElement` nodig.
In onderstaand voorbeeld is dit uitgewerkt.

```
function dostuff(e)
{
  if ( !e )  // Microsoft IE
    e = window.event;
  if (e.target)
    var element = e.target;
  else  // IE8 or older
    var element = e.srcElement;
  element.style.color == "red";
}
```

Deze versie van de functie werkt dus echt in alle versies van alle browsers (zelfs in IE6!).

Zolang er nog oudere versies van Internet Explorer in gebruik zijn zult u in JavaScript helaas af en toe dergelijke technieken moeten gebruiken om te zorgen dat uw pagina's in echt álle browsers werken. Gelukkig neemt het gebruik van oude browsers snel af. Medio 2018 was het gebruik van IE8 al gedaald tot 0.5%. Over enkele jaren zal bovenstaande code definitief overbodig worden.

# 9 WERKEN MET GEGEVENSTYPES

## 9.1 Manipuleren van tekst

Wanneer u in JavaScript wilt werken met teksten, dan gebruikt u hiervoor bijvoorbeeld een variabele of een tekstveld in een HTML-formulier.
Hoewel u niet hoeft aan te geven tot welk gegevenstype een variabele behoort, speelt het gegevenstype wel degelijk een rol. Hieronder ziet u twee toekenningen van een waarde aan de variabele `voorbeeld`.

```
voorbeeld = 3;
voorbeeld = "3";
```

In het eerste geval heeft `voorbeeld` een numerieke waarde, in het tweede geval een alfanumerieke (tekst)waarde. Een of meer tekens tussen aanhalingstekens noemen we een *string* (tekenreeks).
JavaScript ziet strings als objecten van het type `String`. Voor strings zijn er, net als voor andere objecten, diverse methods en properties.
Er is een verschil tussen een method en een functie. U maakte al eerder kennis met de speciale JavaScript-functies `parseInt()` en `parseFloat()`. Dit zijn echter geen methods! Het verschil is duidelijk te zien in de wijze van gebruik.

| | |
|---|---|
| **Functie**<br>`parseInt(tekenreeks)` | `var a = "3";`<br>`var b = parseInt(a);` |
| **Methode**<br>`toUpperCase()` | `var x = "letters";`<br>`var y = x.toUpperCase();` |

Wanneer gebruik gemaakt wordt van een functie die iets met een object moet doen, wordt het object meegegeven als parameter. Wanneer een method van een object wordt aangesproken, wordt de method (voorafgegaan door een punt) achter het object geplaatst en wordt er iets gedaan met de inhoud van het object. Hierbij wordt over het algemeen het resultaat geretourneerd, waarbij de inhoud van het object zelf niet verandert. Zo zal hierboven de tekst in de variabele `x` nog steeds in kleine letters staan, terwijl `y` dezelfde tekst in hoofdletters zal bevatten.

Een veelgebruikte method voor strings is de method `substring(startpositie,eindpositie)`. U kunt hiermee een gedeelte uit een bestaande string kopiëren. Er worden hierbij twee parameters meegestuurd, waarmee u aangeeft bij welke letter de substring moet beginnen en waar deze moet eindigen. De eerste letter heeft hierbij de waarde `0` (de zogenaamde indexwaarde).

```
|g|r|a|s|m|a|a|i|e|r|
0 1 2 3 4 5 6 7 8 9 10
```

Met

```
var geheel = "grasmaaier";
var deel = geheel.substring(0,4);
```

krijgt `deel` de waarde `"gras"`.
U kunt de voor het object `String` gedefinieerde methods gebruiken in combinatie met variabelen, maar ook direct toepassen op een tekst.

```
var y = "letters".toUpperCase();
```

Een handige property van het object `String` is de property `length`, waarmee u de lengte van de tekenreeks kunt opvragen. Bijvoorbeeld: `"zeven".length` levert als resultaat `5`.

| | |
|---|---|
| `"Hallo".length` | Lengte van de string (5) |
| `"Morgen".substring(2,4)` | Deel van de tekst vanaf letter 2 tot 4 (telt vanaf nul!), dus 'rg' |
| `"Morgen".substr(2,4)` | De tekst vanaf letter 2 met lengte 4 (telt vanaf nul!), dus 'rgen' |
| `"Hallo".toUpperCase()` | HALLO |
| `"Hallo".toLowerCase()` | hallo |
| `"Hallo".charAt(1)` | De tweede letter, dus een 'a' |
| `"Dit is een test".split(' ')` | Levert een array van 4 strings op, gesplitst op spaties. |

**Voorbeeld gebruik String methoden**
In dit voorbeeld wordt een tekst omgedraaid en afgebeeld.

```
<html>
<head>
  <title>Keer tekst om</title>
</head>
<body>
<script type="text/javascript">
  var tekst = window.prompt("Typ een tekst:");
  document.write(tekst + "<br><br>");

  // Schrijf de tekst in omgekeerde volgorde:
  for (var i= tekst.length-1; i>=0; i--) {
    document.write(tekst.charAt(i));
  }
</script>
</body>
</html>
```

Een andere handige method van een string object is `indexOf(`*`substring ,startpositie`*`)`. Deze retourneert de positie (index) van de meegegeven substring vanaf een bepaalde positie. Met

```
var tekst = "Mien, waar is mijn feestneus?";
var pos = tekst.indexOf("feest", 0);
```

krijgt `pos` de waarde `19`. Als de substring niet wordt gevonden in het string object, dan zal de waarde `-1` worden geretourneerd.

In het volgende voorbeeld wordt geteld hoeveel keer een woord voorkomt in een stuk tekst. Hiervoor is een aparte functie `telAantal()` geschreven. Met behulp van de method `indexOf()` wordt hierin steeds gezocht naar de positie van de volgende keer dat het woord voorkomt in de tekst. Steeds als het woord wordt gevonden wordt de hulpvariabele `aantal` opgehoogd. Om te voorkomen dat niet steeds dezelfde wordt gevonden, wordt steeds gezocht vanaf de huidige positie plus één.

**Voorbeeld gebruik String methoden**

```
<html>
<head>
<title>Tel woorden met indexOf</title>
<script type="text/javascript">

  function telAantal(txt, wrd) {
    var aantal = 0;
    var pos = txt.indexOf(wrd, 0)
    while (pos >= 0) {
      // wrd is (nog) een keer gevonden, dus hoog op:
      aantal++;
      // zoek positie van volgende:
      pos = txt.indexOf(wrd, pos+1)
    }
    return aantal;
  }

</script>
</head>
<body>
<script type="text/javascript">

  var tekst = window.prompt("Typ een tekst:");
  document.write(tekst + "<br><br>");

  var woord = window.prompt("Typ een woord:");
  document.write("Het aantal keer dat " + woord
                 + " voorkomt in de tekst is: "
                 + telAantal(tekst, woord));
</script>
</body>
</html>
```

## 9.2 Berekeningen

Net als bij strings, ziet JavaScript getallen ook als objecten en wel als objecten van het type `Number`. Een handige method van `Number` is `toFixed()`, die bijvoorbeeld kan worden gebruikt voor het weergeven van een vast aantal decimalen bij niet-gehele getallen.

```
var x = 3.1415;
document.write("x: " + x.toFixed(2));
```

Naast het object `Number`, dat gebruikt wordt om getallen in op te slaan, bestaat er een ander JavaScript-object, het object `Math`, dat u kunt gebruiken voor (complexe) wiskundige bewerkingen. (Merk op dat ingebouwde JavaScript-objecten altijd beginnen met een hoofdletter). Door middel van dit object hebt u de beschikking over wiskundige constanten en handige functies. Hieronder ziet u een overzicht van een aantal hiervan:

| | |
|---|---|
| `var x = Math.PI;` | x wordt gelijk aan pi (3.14159...). |
| `var x = Math.sqrt(16);` | Vierkantswortel, x wordt 4. |
| `var x = Math.random();` | Een random (willekeurig) getal tussen 0 en 1. |
| `var x = Math.round(3.8);`<br>`var y = Math.round(3.2);` | Rondt getallen af tot een geheel getal. Dus x wordt 4 en y wordt 3. |
| `var x = Math.floor(3.8);`<br>`var y = Math.floor(3.2);` | Rondt het getal naar beneden af. Zowel x als y wordt 3. |
| `var x = Math.ceil(3.8);`<br>`var y = Math.ceil(3.2);` | Rondt het getal naar boven af. Zowel x als y wordt 4.. |
| `var x = Math.max(3, 4, 6);`<br>`var y = Math.min(a, b);` | Levert de hoogste en de laagste waarde op. Dus x wordt 6, y wordt de kleinste van a en b. |
| `var x = Math.log(10);`<br>`var y = Math.exp(3);` | Logaritme en e-macht.<br>Hier wordt x ≈ 2.3 en y ≈ 20. |
| `var x = Math.sin(Math.PI/2);`<br>`var y = Math.cos(Math.PI/4);` | Goniometrische functies. Hier wordt x=1, y ≈ 0.7 (cos 45°). |

Het volgende voorbeeld simuleert het 10 keer gooien met een dobbelsteen. Hierbij is de functionaliteit verdeeld over twee functies: de functie `worp()` die een random getal tussen 1 en 6 retourneert en de functie `toonWorpen(`*`aantal`*`)`

die het resultaat van een aantal worpen toont in de body van het document. De functie `toonWorpen()` wordt aangeroepen naar aanleiding van het event `onload`, dus zodra het document geladen is.

**Voorbeeld gebruik Math methoden**

```
<html>
<head>
<title>Simulatie worpen met dobbelsteen</title>
<script type="text/javascript">

  function worp() {
    var x = Math.random();
    x = x * 6;
    return Math.ceil(x);
  }

  function toonWorpen(aantal) {
    var tekst = "Resultaat " + aantal + " worpen: ";
    for (var i=1; i<=aantal; i++) {
      tekst += worp() + " ";
    }
    document.body.innerHTML = tekst;
  }
</script>
</head>
<body onload="toonWorpen(10)">
</body>
</html>
```

## 9.3 Werken met datum en tijd

Het werken met datums levert bij het programmeren vaak problemen op door de grote verscheidenheid in de opmaak van datums. Op internet komen hier nog twee complicaties bij, namelijk

- De tijd wordt op de client-pc bepaald, **niet** op de server
- De tijd gebruikt de tijdzone die op de client staat ingesteld.

JavaScript beschikt over een object van het type `Date` met een groot aantal methods en properties voor het werken met datum en tijd.

Om met een object te kunnen werken wordt dit toegekend aan een variabele. De variabele is dan een handvat om te kunnen werken met het object. Voor het in het leven roepen van een object gebruikt JavaScript het sleutelwoord `new`. U kunt als volgt een datum-object maken.

```
var huidigeDatum = new Date();
```

Het object `huidigeDatum` bevat nu de datum en tijd volgens Greenwich Mean Time van het moment dat het object werd gemaakt. Voor de bepaling wordt gebruik gemaakt van de klok- en tijdzone-instellingen van de lokale pc.
Een `Date` object heeft diverse methods en properties. Hieronder wordt steeds aangenomen dat de variabele d een Date bevat.

| | |
|---|---|
| `var x = d.getMilliseconds();`<br>`var x = d.getSeconds();`<br>`var x = d.getMinutes();`<br>`var x = d.getHours();`<br>`var x = d.getDate();`<br>`var x = d.getMonth();`<br>`var x = d.getFullYear();` | Methods om elementen uit een datum te halen. Merk op dat getMonth op 0 begint. De maand maart levert dus 2 op! Voor jaren is er ook een oude functie getYear, maar die levert het aantal jaren sinds 1900! |
| `var x = d.getDay();` | De weekdag. Zondag is 0. |
| `var x = d.toLocaleString();`<br>`var x = d.toUTCString();`<br>`var x = d.toDateString();`<br>`var x = d.toTimeString();` | Uitvoerfuncties. De UTC-tijd kent u ook als Greenwichtijd. De laatste twee functies leveren alleen de datum of de tijd. |
| `var x = d.getTime();` | Verstreken tijd sinds 1-1-1970. |

U kunt datum- en tijdweergave natuurlijk ook zelf samenstellen, bij voorbeeld met behulp van `getHours()` en `getMonth()`.

```
document.write("De tijd is " + huidigeDatum.getHours()
              + ":" + huidigeDatum.getMinutes());
```

Bij het maken van een nieuw `Date` object kunnen ook parameters worden meegegeven om een bepaalde datum in het object op te nemen (in plaats van de huidige datum). Een van de manieren om dit te doen is:

```
var geboorteTijd = new Date(1997,9,28,14,40,0);
```

Hiermee staat de geboortetijd op 28 oktober 1997, 10 over half 3 's middags. Let weer op de telling van de maanden: 0 staat voor januari en met 9 wordt dus oktober bedoeld. Wanneer u geen tijd opgeeft, wordt deze door JavaScript op 00:00:00 gezet.

De method `getTime()` geeft het aantal milliseconden op sinds 1 januari 1970 0 uur 0 (GMT). Intern slaat JavaScript de datum op deze wijze op. De method `getTime()` lijkt op het eerste gezicht vrij nutteloos, maar kan goed worden gebruikt voor het rekenen met tijden. In onderstaand voorbeeld wordt deze method gebruikt om het aantal dagen sinds de geboortedatum te bepalen.

**Voorbeeld gebruik Date methoden**

```
<html>
<head><title>Rekenen met tijden</title></head>
<body>
<script type="text/javascript">
  var huidigeDatum = new Date();
  var geboorteTijd = new Date(1997,9,28,14,40,0);

  // bereken verschil in milliseconden:
  var tijdsVerschil = huidigeDatum.getTime()
                      - geboorteTijd.getTime();

  // bereken verschil in dagen:
  var dagInMsec = 1000 * 60 * 60 * 24;
  tijdsVerschil = tijdsVerschil / dagInMsec;

  document.write("Aantal dagen sinds geboorte: "
                 + tijdsVerschil.toFixed(2));
</script>
</body>
</html>
```

Tenslotte kunt u ook en nieuwe datum maken met als basiswaarde het aantal milliseconden sinds 1970. Dat is handig als u met tijden wilt rekenen. Hier ziet u een voorbeeld waarin u de datum en tijd krijgt die precies 60 dagen verder ligt. De huidige tijd wordt opgevraagd en bij het aantal milliseconden wordt precies het equivalent van 60 dagen opgeteld:

```
var nu = new Date();
var dagInMsec = 1000 * 60 * 60 * 24;
var nu2 = nu.getTime() + (60 *dagInMsec);
var nuPlus60Dagen = new Date(nu2);
alert(nuPlus60Dagen.toLocaleString());
```

# 10 FORMULIEREN EN VALIDATIE

## 10.1 Waarom validatie?

Eén van de meest gebruikte technieken in JavaScript is het controleren van de invoer van een formulier. Deze controle kent overigens ook beperkingen. Controle van gegevens via JavaScript is beslist **niet** geschikt als vorm van beveiliging tegen ongeldige gegevens. Kwaadwillende gebruikers kunnen altijd JavaScript uitzetten op uw pagina om zodoende de controles te omzeilen. Ook kan een hacker uw webpagina aanroepen vanuit een ander formulier waarin geen controles zijn verwerkt. Op de server mag een webontwikkelaar er dus nooit van uitgaan dat controle al op de client is gebeurd.

Maar controle met JavaScript heeft wel twee andere grote voordelen. Ten eerste gebeurt controle met JavaScript op de lokale machine, waardoor deze veel sneller wordt uitgevoerd dan controle op de server. Bij controle op de server moet ook de complete pagina weer in de browser worden geladen. Voor de eindgebruiker is controle met JavaScript dus veel prettiger, omdat de responsetijd veel beter is.

Het tweede voordeel is dan ook meteen dat alles wat op de client kan gebeuren de server ontlast. Indien een gebruiker een ingewikkeld formulier moet invullen en tot vier maal toe constateert de code in JavaScript dat er nog iets mis is met de invoer, dan zijn dat dus ook vier keren dat de server *niet* in actie hoefde te komen. Controle van gegevens via JavaScript verhoogt dus de snelheid op zowel client als server.

## 10.2 Waarden bepalen van formulierelementen

Als u elementen van een formulier wilt valideren dan geeft u zowel het formulier als de elementen in het formulier een naam met het HTML-attribuut `name`. Deze namen kunnen in JavaScript gebruikt worden voor het benaderen van de elementen. De `value` van de elementen bevat de ingevoerde waarde.
Een invoerveld van het type `text` met als naam *woonplaats* in een formulier met als naam *invoer* kunt u dus als volgt vanuit een script een waarde geven.

**Voorbeeld**

```
window.document.invoer.woonplaats.value = "Utrecht";
```

Als volgt kunt u vanuit JavaScript lezen wat er is ingevuld.

**Voorbeeld**

```
var stad = window.document.invoer.woonplaats.value;
```

Het gedeelte `window.document` is niet verplicht, zodat u het ook korter neer kunt zetten.

**Voorbeeld**

```
var stad = invoer.woonplaats.value;
```

Er kan ook op een andere wijze naar een `form`-element verwezen worden. JavaScript maakt intern een array met de naam `forms` van alle in een document aanwezige formulieren. In een document met één formulier kunt u dus ook de volgende constructie gebruiken.

**Voorbeeld**

```
var stad = document.forms[0].woonplaats.value
```

## 10.3 Waar controleert u de gegevens?

Het controleren van invoer kunt u per element doen of per formulier als geheel. Als u elk formulierelement apart controleert, dus per tekstveld, checkbox, combobox enzovoort, houdt dat in dat de invoer meteen wordt getest. U kunt zelfs het veld actief laten totdat een juiste invoer is ingevuld.
U kunt hiervoor het event `onblur` gebruiken, dat optreedt zodra het veld wordt verlaten.

**Voorbeeld**

```
<input name="postcode" onblur="testPostcode(value)">
```

Deze methode heeft echter wel enkele nadelen. Zo vinden gebruikers het soms vervelend om de velden in een bepaalde verplichte volgorde te moeten

invullen. Indien iemand bijvoorbeeld de invoer van een bepaald veld even wil overslaan is dit niet mogelijk. Het controleren van elk individueel veld tijdens het invullen is bovendien meer werk qua scriptcode. Deze methode is daarom niet aan te bevelen.

## 10.4 Het submit event

Veel gebruikelijker is het dan ook om de invoerwaarden pas te controleren als de gebruiker op de verzendknop klikt. Bij het verzenden van een formulier treedt het `onsubmit` event van het formulier op. U kunt een functie aan dit event koppelen die alle invoer controleert op geldige waarden. Van belang hierbij is verder dat het verzenden van het formulier wordt geannuleerd als het resultaat van de `onsubmit` als resultaat `false` oplevert. De webpagina blijft dan dus in de browser en de gegevens worden nog niet naar de server verzonden. Gebruikelijk is dan ook om een functie te maken die alle gegevens controleert en de waarde `true` teruggeeft als de gegevens in orde zijn maar `false` als de gegevens (deels) foutief zijn. Wanneer deze functie bijvoorbeeld `validate`() heet dan ziet de koppeling met `onsubmit` er zo uit:

**Voorbeeld**

```
<form name="gegevens" action="eval.php" onsubmit="return
   validate()">
```

In dit voorbeeld zal dus bij het klikken op de verzendknop van het formulier de functie `validate()` worden aangeroepen. Deze functie zal, afhankelijk van of alles juist is ingevuld, de waarde `true` of `false` retourneren. Als er iets mis was dan geeft de functie `false` terug en wordt de verzending geannuleerd. Uiteraard kan de functie ook aan de gebruiker te kennen geven wat er precies mis is, bijvoorbeeld via een alertbox of door in een veld de betreffende meldingen neer te zetten. Als alles correct is ingevuld dan wordt `true` geretourneerd en zal het formulier worden verzonden, in dit geval naar de pagina 'eval.php'. (asp, aspx of php zijn gebruikelijke formuliertypes op de server).

## 10.5 Controle of een tekstveld is ingevuld

Op de volgende manier kunt u controleren of een veld wel is ingevuld.

**Voorbeeld**

```
function isIngevuld(element) {
  if (element.value.length == 0) {
    alert("Veld " + element.name + " niet ingevuld");
  }
}
```

Deze functie zal moeten worden aangeroepen met als parameter het gehele invoerveld. In deze functie wordt van dit object zowel de property `value` als de property `name` gebruikt. Zie ook het voorbeeld aan het einde van dit hoofdstuk.

## 10.6 Controle of invoer een getal is

Met behulp van de functie `isNaN(`*`getal`*`)` kunt u controleren of een variabele een getal is. `NaN` staat voor 'Not a Number'. De functie retourneert een boolean: `true` of `false`.

**Voorbeeld**

```
if (isNaN(leeftijd)) {
  alert("Leeftijd bevat geen getal");
}
```

## 10.7 Controle van tekstvelden

Als u een tekstveld wilt laten invullen dan kunt u controleren of alle karakters wel geldig zijn. Zo mag een voornaam bijvoorbeeld hoofdletters en kleine letters bevatten, maar ook spaties ('Gert Jan') of streepjes ('Gert-Jan'). Andere karakters zijn echter waarschijnlijk gevolg van verkeerde invoer.

U kunt hiervoor een aparte functie maken die controleert of elke letter van de invoer wel een correct karakter is. Een tweede functie die controleert of een karakter geldig is maakt de code wat eenvoudiger.
Deze eerste functie zou er zo uit kunnen zien:

**Voorbeeld**

```
function isValidLetter(ch)
{
  ch = ch.toLowerCase();
  if (ch >= 'a' && ch <= 'z')
    return true;
  if (ch == '-' || ch == ' ')
    return true;
  return false;
}
```

Deze functie ontvangt een karakter ('ch') en zet dit eerst om in een kleine letter indien mogelijk. Daarna wordt gecontroleerd of het tussen 'a' en 'z' ligt. Zo ja, dan is het een geldig karakter en wordt `true` teruggegeven. Dat gebeurt vervolgens ook als de letter een streepje of een spatie is. Als het geen van deze karakters is wordt `false` teruggestuurd.

Met behulp van deze functie kunt u nu een tweede functie maken die de inhoud van een naam controleert:

**Voorbeeld**

```
function isValidName(name)
{
  for (var i = 0; i < name.length; i++) {
    var c = name.charAt(i);
    if (!isValidLetter(c)) {
      return false;
    }
  }
  return true;
}
```

Deze functie loopt met behulp van een lusje alle karakters af van de naam. Zodra er een ongeldige letter wordt aangetroffen stuurt de functie direct als resultaat `false` terug. Als alle letters gecontroleerd zijn wordt als resultaat `true` teruggegeven.

Ingewikkelder gevallen van controle waarbij op zeer specifieke invoer wordt gecontroleerd kunnen worden opgelost met een zogenaamde regular expression. Te denken valt bijvoorbeeld aan telefoonnummers, postcodes, e-mailadressen, enzovoort. Regular expressions worden in een volgend hoofdstuk behandeld.

## 10.8 Controle per veld of per formulier

Zelfs als u de velden van het gehele formulier controleert dan kunt u nog kiezen op welke wijze u deze controle precies afhandelt. Ruwweg hebt u hier twee mogelijkheden.

U kunt de velden één voor één controleren en de controle afbreken zodra u een fout constateert. Dit heeft als voordeel dat de foutmelding heel eenvoudig wordt (u constateert slechts één probleem). En u kunt gemakkelijk teurgspringen naar het specifieke element waar de fout zit. Nadeel is wel dat de gebruiker geen inzicht heeft in wat er nog meer mis zou kunnen zijn. Voor de gebruiker is het een kwestie van 'proberen en hopen'. Elke keer kan er weer een andere fout naar voren komen. Dit kan vrij vervelend werken.

Anderzijds kunt u ook alle velden in één keer controleren. Het kan nu echter gebeuren dat er drie of vier velden verkeerd (of niet) zijn ingevuld. U moet dan dus een manier vinden om die diverse meldingen te verwerken. Ook kunt u niet direct naar het ene element gaan waar de fout zit - er zijn er immers meer waar iets verkeerd zit. Ondanks deze kleine problemen is deze laatste manier toch vaak te verkiezen boven de 'één fout tegelijk'-controle.

## 10.9 Focus op een element zetten

Indien een formulier twintig velden bevat en alleen het veertiende veld bevat een verkeerde invoer dan is het niet erg vriendelijk voor de gebruiker als deze op het formulier het betreffende veld moet opzoeken. Beter is het dan om dit veld direct de zogenaamde 'focus' te geven. Met andere woorden, om het veld actief te maken en de invoer direct op dat veld te richten. Dit kunt u doen met de methode `focus()`. In onderstaand voorbeeldje wordt het veld 'postcode' in het formulier met de naam 'invoer' actief gemaakt.

**Voorbeeld**

```
invoer.postcode.focus();
```

## 10.10 Gecombineerd voorbeeld

In dit voorbeeld worden de voorafgaande onderdelen toegepast. In dit voorbeeld wordt de controle per veld verricht, en wordt de controle dus afgebroken zodra er een fout is gevonden. Afhankelijk van de waarde hiervan, `true` of `false`, zal het formulier worden verzonden.

**Voorbeeld**

```
<html>
<head>
<script type="text/javascript">

function isIngevuld(element) {
  if (element.value.length <= 0) {
    alert("Veld " + element.name + " niet ingevuld");
    element.focus();
    return false;
  }
  return true;
}
```

```
function isGeldigeLetter(ch)
{
  ch = ch.toLowerCase();
  if (ch >= 'a' && ch <= 'z')
    return true;
  if (ch == '-' || ch == ' ')
    return true;
  return false;
}
```

```
function isGeldigeNaam(element)
{
  for (var i = 0; i < element.value.length; i++) {
    var c = element.value.charAt(i);
    if (!isGeldigeLetter(c)) {
      alert("Ongeldige karakters in " + element.name);
      element.focus();
      return false;
    }
  }
 return true;
}
```

```
function testForm() {
  if (    isIngevuld(formulier.naam)
       && isGeldigeNaam(formulier.naam)
       && isIngevuld(formulier.woonplaats)
       && isGeldigeNaam(formulier.woonplaats) ) {
    return true;
  }
  else {
    return false;
  }
}
</script>
</head>
```

```
<body>
<form action="eval.asp" name="formulier"
      onsubmit="return testForm()">
   Naam: <input type="text" name="naam">
   Woonplaats: <input type="text" name="woonplaats">
   <input type="submit" value="Verstuur">
</form>
</body>
</html>
```

In dit voorbeeld is de functionaliteit voor het testen opgesplitst in de functies `isIngevuld`, `isGeldigeLetter` en `isGeldigeNaam`. Dit zijn handige hulpfuncties, die u zou kunnen toepassen in vele webpagina's. U kunt dit soort

functies gemakkelijk hergebruiken door ze in een apart JavaScript-bestand te zetten, bijvoorbeeld in 'testfuncties.js'. U hoeft dan in een HTML-pagina alleen maar te verwijzen naar dit bestand.

**Voorbeeld**

```
<script src="testfuncties.js"></script>
```

# 11 REGULAR EXPRESSIONS

## 11.1 Wat zijn Regular Expressions?

Iedereen gebruikt wel eens 'zoek-en-vervang' om een woord door een ander woord te vervangen. Meestal betreft het dan een vrij eenvoudig geval, waarin gewoon een bepaald woord moet worden verbeterd of vervangen door iets anders. Maar in sommige gevallen ligt de zaak ingewikkelder. Als u bijvoorbeeld in een financieel overzicht alle negatieve getallen zoudt willen vervangen door het bedrag tussen haakjes (dus -100 moet (100) worden) dan is dat nog niet zo eenvoudig.

Voor dit soort gevallen, waarin een bepaald patroon wordt gezocht, zijn al in de jaren 1970 zogenaamde 'regular expressions' (RE's) bedacht. Een RE bestaat uit een patroon waarin bepaalde codes bepaalde betekenissen hebben. Zo betekent '\d' bijvoorbeeld een cijfer (de d van digit). En het patroon '\d{9}' betekent een reeks van negen cijfers. Een dergelijk patroon kunt u dus gebruiken om bepaalde codes te zoeken en ook eventueel te vervangen door een aanpassing van dat patroon.

Een buitengewoon prettige eigenschap van RE's is dat ze al tientallen jaren universeel en ongewijzigd zijn. RE's bestonden al lang voordat pc's of Microsoft bestonden, en de codes die worden gebruikt waren al snel openbaar. Daardoor kunnen RE's in feite over de hele wereld op dezelfde manier gebruikt worden.

## 11.2 Regular Expressions op webpagina's

Met een RE kan een bepaald patroon worden vastgelegd waarop kan worden gecontroleerd. En dat is dan weer iets dat bij de validatie van invoer op een formulier kan worden toegepast. Het is bijvoorbeeld nog niet zo eenvoudig om helder en eenduidig vast te leggen wat een geldig e-mailadres is. Er moet bijvoorbeeld een '@' in zitten, maar dat is lang niet genoeg. Twee punten achter elkaar in het domein mag bijvoorbeeld niet, en een uitroepteken in de domeinnaam is ook verboden. Zelf programmeren is vrijwel onbegonnen werk en voor zoiets is een RE dan ook onmisbaar. In eenvoudige gevallen kunt u de RE zelf opstellen, ingewikkelder patronen kunt u vaak op internet vinden. Er

zijn tegenwoordig namelijk diverse websites waarop u veel nuttige RE's kunt vinden. Een RE voor een specifieke bedrijfscode zult u daar niet vinden, maar een RE voor een Nederlandse postcode of telefoonnummer vermoedelijk wel, en een RE voor e-mail, een website of een bankpasnummer zeker wel.

**De codes voor een regular expression**

Hieronder vindt u een kort overzicht met enkele veelgebruikte patronen voor het opstellen van een regular expression. De complete documentatie is gemakkelijk op internet te vinden.

| | |
|---|---|
| \d | Een cijfer |
| jan \| piet | jan of piet |
| [4-9] | Een cijfer uit de reeks 4 tot en met 9 |
| {7} | Het voorgaande patroon 7 keer |
| ? | Het voorgaande patroon 0 of 1 keer |
| * | Het voorgaande patroon 0 of meer keer |
| + | Het voorgaande patroon 1 of meer keer |
| ^ | Het begin van de reeks |
| $ | Het einde van de reeks |

Stel dat u een oud Nederlands nummerbord wilt vastleggen. De oudste codes waren van de vorm 'ER-19-56'. Een dergelijk patroon kunt u als volgt vastleggen:

**Voorbeeld**

```
^[A-Z]{2}-?\d{2}-?\d{2}$
```

Dit patroon houdt het volgende in: een letter uit de reeks A-Z, en dat tweemaal. Dan een streepje. Het vraagteken erachter betekent dat het streepje mag, maar niet verplicht is. Een code als 'XW1973' is dus ook toegestaan. Daarna komt er tweemaal een cijfer (\d). Weer een facultatief streepje. Weer twee cijfers. Om te voorkomen dat een patroon als ABC1234567 ook wordt geaccepteerd staat bovendien aan het begin een ^ en aan het einde een $. Dat betekent dat er niets voor of achter het patroon mag staan.

## 11.3 Regular Expressions in JavaScript

Er zijn twee manieren om in JavaScript een RE te definiëren. Het kan via een zogenaamde `new RegExp` , of via een manier met forward slashes. Beide manieren vindt u hieronder.

```
var re1= new RegExp("^[A-Z]{2}-?\\d{2}-?\\d{2}$",
   "i");
var re2 = /^[A-Z]{2}-?\d{2}-?\d{2}$/i;
```

Bij de eerste wijze gebruikt u een `RegExp` object. U geeft de RE mee als argument, maar past u daarbij wel op dat een backslash een speciale betekenis heeft in een string. Alle backslashes in de RE moeten daarom beschermd worden met een extra backslash. U kunt een extra "i" meegeven om de string ongevoelig te maken voor hoofdletters en kleine letters. Ook kunt u een 'g' (global) toevoegen om er voor te zorgen dat niet alleen de eerste maar *alle* gevonden strings worden vervangen.

Bij de tweede methode staat de RE tussen forward slashes. Ook hier kunt u een extra 'i' meegeven voor hoofdletters/kleine letters.

De prettigste methode van een RE om mee te werken is `test`. Met deze methode kunt u testen of een string het patroon wel of niet bevat. De methode geeft via `true` of `false` aan of het patroon gevonden is.

```
var x1 = re1.test("ab-34-56");  // x1 wordt true
var x2 = re2.test("ab34567");   // x2 wordt false
```

Hieronder vindt u het compleet uitgewerkte voorbeeld voor het valideren van een Nederlands kenteken (oudste stijl). Voor de preciezen onder u valt nog op te merken dat sommige lettertekens, zoals de 'Q' en de 'C' in de praktijk niet gebruikt werden. Hiermee is in dit voorbeeld geen rekening gehouden.

```
<html>
<head>
  <title>regular expression demo</title>
  <script type="text/javascript">
    function validate()
    {
      var re = /^[A-Z]{2}-?\d{2}-?\d{2}$/i;
      var result = re.test(auto.kenteken.value);
      if (!result)
        alert("Geen geldig kenteken");
      return result;
    }
  </script>
</head>
<body>
  <form name="auto" action="invoer.php"
        onsubmit="return validate()" >
    Kenteken: <input type="text" name="kenteken" />
  <input type="submit" value="verzend" />
  </form>
</body>
</html>
```

# 12 TIMERS EN JAVASCRIPT

Op sommige webpagina's wilt u bepaalde code misschien niet slechts één maal laten uitvoeren, maar een aantal keren. Of u wilt een bepaalde functie elke twee seconden aanroepen. In JavaScript zijn er twee technieken om dit te bereiken.

## 12.1 setTimeout en clearTimeout

De eerste techniek is via een methode van het `window` object genaamd `setTimeout`. Deze kunt u aanroepen met twee parameters. De eerste parameter moet een statement bevatten, een opdracht die na een bepaald interval wordt uitgevoerd. De tweede parameter bepaalt de lengte van dit interval, uitgedrukt in milliseconden.

**Voorbeeld**

```
window.setTimeout("alert('Hallo!')", 60000);
```

Deze opdracht zal na 60 seconden een melding vertonen.
Uiteraard kan men als opdracht ook de naam van een functie meegeven. Dus bijvoorbeeld:

**Voorbeeld**

```
function showmessage()
{
  alert('Hallo!');
}
setTimeout(showmessage, 60000);
```

Interessant wordt het nu als de functie zelf ook weer setTimeout aanroept en zijn eigen functienaam meegeeft. De functie roept in feite zichzelf weer aan. Deze techniek staat bekend als recursie:

**Voorbeeld**

```
function showmessage()
{
  alert('Hallo!');
  setTimeout(showmessage, 60000);
}

setTimeout(showmessage, 60000);
```

Deze code zal elke 60 seconden weer een boodschap laten zien.
Met deze techniek kan men aardige effecten bereiken. Het laten lopen van een klok, een diavoorstelling met elke 10 seconden een nieuwe afbeelding. Een ander idee is om een soort lichtkrant te maken. Het concept er achter is om een bepaalde tekst te vervangen door bijna dezelfde tekst, maar dan met het eerste karakter achteraan.

```
<html>
<head>
<html>
<head>
  <title>lichtkrant demo</title>
  <script type="text/javascript">

function lichtkrant()
{
 var str = document.getElementById("krant").
   innerHTML;
 str = str.substr(1) + str.substr(0, 1);
 document.getElementById("krant").innerHTML = str;
 window.setTimeout(lichtkrant, 500);
}
  </script>
</head>
<body onload ="lichtkrant()">
 <span id="krant">deze tekst loopt over het scherm </
   span>
</body>
</html>
```

In bovenstaand script zal de functie `lichtkrant()` steeds na een halve seconde worden aangeroepen. Bij elke aanroep wordt de eerste letter van de tekst gehaald en er weer achter geplakt. Hierdoor lijkt de tekst steeds één positie naar links te verschuiven.

Met deze versie van het script is het echter niet mogelijk om de lichtkrant ooit te doen stoppen. Om dit te bereiken is het noodzakelijk om het zogenaamde 'id' van de timeout ter beschikking te hebben.
De functie `setTimout` retourneert namelijk een id. Dit id kan worden opgeslagen in een variabele.

**Voorbeeld**

```
var id;
id = setTimeout("alert('Hallo!')", 60000);
```

Dit id kan worden meegegeven aan de methode `clearTimeout` om de timer te laten stoppen.

**Voorbeeld**

```
  clearTimeout(id);
```

Hieronder is het voorbeeld van de lichtkrant uitgebreid met een knop waarmee de timeout kan worden stilgezet.

**Voorbeeld**

```
<html>
<head>
  <title>lichtkrant demo</title>

<script type="text/javascript">
 var id;
 function lichtkrant()
 {
 var str = document.getElementById("krant").
  innerHTML;
 str = str.substr(1) + str.substr(0, 1);
 document.getElementById("krant").innerHTML = str;
 id = window.setTimeout(lichtkrant, 500);
 }
  function stopkrant()
  {
    window.clearTimeout(id);
  }
</script>
</head>
<body onload ="lichtkrant()">
 <span id="krant">deze tekst loopt over het scherm </
  span>
 <br />
 <input type ="button" value="stop krant"
        onclick="stopkrant()" />
</body>
</html>
```

## 12.2 setInterval en clearInterval

De tweede techniek om een bepaalde functie te herhalen gebruikt iets andere functies, maar de functionaliteit is vergelijkbaar. Belangrijk verschil is wel dat de aanroep voor de herhaling bij deze techniek *buiten* de functie zelf plaatsvindt. De functies die u voor deze manier nodig hebt heten `setInterval` en

clearInterval. Hieronder ziet u het complete voorbeeld van de lichtkrant met behulp van deze twee functies.

```
<html>
<head>
  <title>lichtkrant demo</title>

<script type="text/javascript">
 var id = setInterval(lichtkrant,1000);

 function lichtkrant()
 {
 var str = document.getElementById("krant").
  innerHTML;
 str = str.substr(1) + str.substr(0, 1);
 document.getElementById("krant").innerHTML = str;
 }
 function stopkrant()
 {
   window.clearInterval(id);
 }
</script>
</head>
<body onload ="lichtkrant()">
 <span id="krant">deze tekst loopt over het scherm </
  span>
 <br />
 <input type ="button" value="stop krant"
        onclick="stopkrant()" />
</body>
</html>
```

Het verschil tussen setTimeout en setInterval is niet heel groot. Bij setTimeout moet u er rekening mee houden dat de volgende timeout pas wordt ingesteld als de functie klaar is. Wanneer dit relatief lang duurt klopt het tijdsinterval dus niet meer. Bij setInterval start de functie echt punctueel,

afhankelijk uiteraard van het interval, maar onafhankelijk van hoe lang de code duurt.

Een ander verschil is dat u `setTimeout` vanuit de herhalende functie zelf aanroept. Dit kan zeer handig zijn wanneer u bij opvolgende aanroepen andere parameters wilt meegeven. Deze mogelijkheid biedt `setInterval` niet, maar als u de parameters elke maal hetzelfde zijn zal `setInterval` waarschijnlijk iets overzichtelijker code opleveren.

# 13 FOUTEN OPSPOREN EN AFVANGEN

## 13.1 Debugging

Een groot aantal jaren geleden was het vinden van fouten in JavaScript een moeizame zaak. In moderne browsers vindt u echter een flink aantal tools om u hierbij te helpen. In vrijwel alle browsers kunt u via <F12> bij de tools voor ontwikkelaars komen. Hier kunt u onder andere uw code stap voor stap uitvoeren of op een bepaalde regel een onderbrekingspunt (breakpoint) zetten zodat de code precies op die regel stopt. De details verschillen per browser, maar in alle gevallen zijn de algemene min of meer gelijk. U doet er goed aan om in tenminste één grote browser vertrouwd te raken met de mogelijkheden zodat u een probleem snel kunt opsporen.

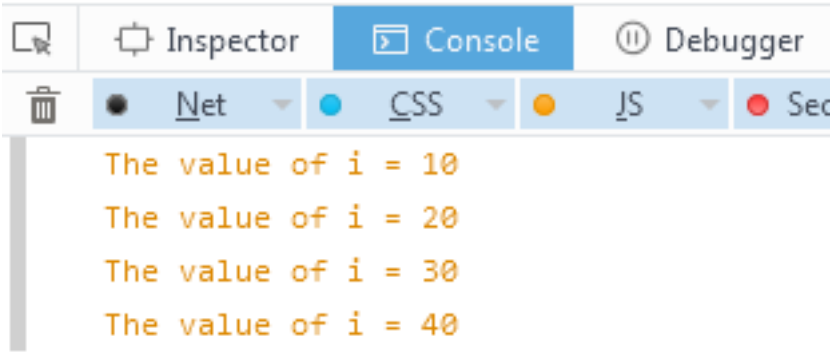
## 13.2 Console.log

Niet zelden wilt u bij het testen van uw code enkele tussenresultaten afbeelden. Zoals eerder vermeld is `document.write` meestal geen serieuze optie. U kunt `window.alert` gebruiken, maar dat werkt natuurlijk niet prettig als u veel informatie wilt laten zien, en bovendien is deie informatie verdwenen zodra u de dialoog wegklikt.

In deze gevallen kunt u het beste `console.log` gebruiken. De informatie die u afdrukt komt in de browser te voorschijn op het tabblad *Console*. Dat tabblad zit ingebouwd in elke browser, alleen zult u per browser even moeten kijken hoe u er precies komt. In onderstaand voorbeeld wordt simpelweg de waarde van de lusteller afgebeeld:

```
for (var i = 10; i < 50; i += 10) {
    console.log('The value of i = ' + i);
}
```

En hieronder ziet u het resultaat in de Console van FireFox.




## 13.3 Voorkomen van fouten

Hoewel u fouten op kunt vangen is voorkomen meestal beter dan genezen. Een van de meest voorkomende oorzaken van problemen in JavaScript is de onderlinge incompatibiliteit van browsers. Die zorgen er soms voor dat u een methode of property probeert te gebruiken die in de betreffende browser niet bestaat.

Er wordt hier (misschien ten overvloede) op gewezen dat het in JavaScript heel eenvoudig is om te controleren of een dergelijke methode of property wordt ondersteund, simpelweg door deze binnen een `if` te testen. Dus als u twijfelt of uw browser de methode `document.getElementById` ondersteunt kunt u dit gewoon aan de browser vragen, als volgt:

```
if( document.getElementById )
  // methode wordt ondersteund
```

Met behulp van deze techniek kunt u zeer veel fouten voorkomen.



computrain

## 13.4 Het gebruik van try ... catch

Code waarvan u denkt dat die wellicht fout kan gaan kunt u binnen een `try ... catch` blok plaatsen. De 'normale' code komt dan binnen het `try` gedeelte, het gedeelte waar u de problemen opvangt komt binnen het `catch` gedeelte. Schematisch ziet dit er zo uit:

**Voorbeeld**

```
try {
  // code die wellicht verkeerd gaat
}
catch ( errorobject ) {
  // code die probleem ondervangt
}
```

Zodra er in het `try` gedeelte een fout optreedt wordt de code daar afgebroken en wordt de code in het `catch` gedeelte uitgevoerd. Daarbij zorgt de JavaScript engine er voor dat er in de variabele 'errorobject' (de naam hiervan mag u zelf kiezen) een zinnige melding terechtkomt.

In oudere versies van JavaScript werden fouten opgevangen door te luisteren naar een event het zogenaamde `onError` event. Dit wordt tegenwoordig nauwelijks meer gebruikt. Groot voordeel van `try ... catch` boven het afvangen van het oudere `onError` event is dat de code hierna weer verder loopt. De code binnen het try blok wordt weliswaar afgebroken, maar code die zich onder het gehele `try ... catch` blok bevindt wordt gewoon uitgevoerd.

In onderstaande code wordt een niet-bestaande methode aangeroepen. Dat levert (uiteraard) een fout op, die door de `catch` wordt afgehandeld. Maar de code loopt na het afhandelen van het `try ... catch` blok gewoon verder. De melding erna zult u dus ook op het scherm zien.

**Voorbeeld**

```
try {
   document.onzin();
}
catch (errobject) {
  alert(errobject);
}
alert("tweede melding");
```

Dit fragment code levert dus twee meldingen op: de eerste melding geeft aan dat de betreffende methode niet bestaat, daarna volgt de tweede melding. Op deze manier kunt u dus verhinderen dat een riskant stukje code ervoor zorgt dat de rest van uw code niet meer wordt uitgevoerd.

## 13.5 Gebruik van throw

Nu is het afvangen van een fout wegens een niet-bestaande methode eigenlijk niet zo'n sterk voorbeeld. Tenslotte kunt u testen of die methode bestaat en op die manier voorkomen dat een fout optreedt.

Maar in het geval van een bibliotheek kan het zijn dat u een bestaande functie aanroept met verkeerde informatie. Denk bijvoorbeeld aan functie die als invoer een positief getal wil hebben. Wat moet de functie doen als de invoer 'min vijf' is? En wat zou een functie moeten doen als de gewenste invoer een getal tussen 1 en 10 moet zijn maar de invoer blijkt 17 te zijn? Een functie kan dit soort problemen niet zelf gaan oplossen. Het enige wat de functie kan doen is aangeven dat er iets fout gaat.

In dergelijke gevallen kunt u de functie het beste een fout laten genereren. Dit kunt u doen met `throw`. U geeft met `throw` zelf iets mee dat in de `catch` wordt opgevangen. Dat mag dus een foutmelding zijn, maar ook kunt u een compleet object meesturen. Het volgende voorbeeld geeft een idee van de mogelijkheden:

**Voorbeeld**

```
<html>
<head>
  <title>throw demo</title>
  <script type="text/javascript">
    function calculate()
    {
      try {
        var num = form1.getal.value;
        if( isNaN( num ))
          throw ("dat is geen getal");
        if ( num < 0 )
          throw ("Negatief getal!")
        form1.uitkomst.value = Math.sqrt(num);
        form1.getal.style.background = "white";
      }
      catch (errobject) {
        alert(errobject);
        form1.getal.style.background = "pink";
      }
    }
  </script>
</head>
<body>
  <form name="form1">
  Getal: <input type="text" name="getal"
          onblur="calculate()" />
  <br />
  Wortel: <input type="text" name="uitkomst" />
</form>
</body>
</html>
```

U ziet hier in het `body`-gedeelte een formulier. Als in de bovenste textbox een getal wordt ingetikt komt in de tweede textbox de waarde van de wortel te staan. Dat is tenminste wat er gebeurt als alles goed gaat. Als de invoer geen getal is of als het getal negatief is dan wordt de normale loop van de code met

behulp van een `throw` afgebroken. De code loopt daarna verder in het `catch` gedeelte. De waarde van de variabele 'errobject' bevat nu de meegegeven tekst. Deze wordt aan de gebruiker getoond en ook krijgt de achtergrond van de invoer een alarmerend lichtrode tint.

# 14 KORTE INTRODUCTIE JQUERY

## 14.1 Wat is jQuery?

jQuery is de laatste jaren de populairste Javascript-bibliotheek. Het is een gratis product, waarvan de code vrij beschikbaar is. De bedoeling van jQuery is om het schrijven van JavaScript te vereenvoudigen, om verschillen tussen browsers te ondervangen en om elementen van de DOM op simpele en eenduidige manier te benaderen.

## 14.2 Hoe komt u aan jQuery?

De eenvoudigste manier is om gewoon bovenaan een pagina de jQuery bibliotheek op te nemen. Deze wordt door diverse sites gratis ter beschikking gesteld. U kunt de bibliotheek ook downloaden van *jQuery.com* en op uw eigen site neerzetten. Een link naar een externe lokatie ziet er bijvoorbeeld zo uit: :

```
<script src="https://ajax.googleapis.com/ajax/
        libs/jquery/1.11.3/jquery.min.js">
</script>
```

*(Opmerking: de link is te lang om hier op één regel te passen. U dient bovenstaande link dus aan elkaar te schrijven).*

Als u daarentegen de bibliotheek downloadt kunt u gewoon naar het script refereren als een lokaal bestand. Alhoewel de bibliotheek krachtig en veelzijdig is valt het volume best mee. De meeste versies zijn ongeveer 100kB groot.

## 14.3 Versies van jQuery

Er zijn diverse versies van jQuery. De versies die met een 1 beginnen ondersteunen Internet Eplorer versies 6, 7 en 8. jQuery versies 2.x en hoger doen dit niet. Als u de ondersteuning van oudere versies van IE  nodig hebt zult u dus versies 1.x willen gebruiken. Hierbij moet worden aangetekend dat het gebruik van de oudere versies van IE zeer snel daalt. Per 2016 is het gebruik van IE8 gedaald tot veel minder dan 1%.

U kunt de jQuery bibliotheken gebruiken in normale versie of in 'minified' vorm. En minified versie is door een tool gestript van alle overbodige tekens. De meeste spaties zijn weg, variabelen zijn zo kort mogelijk gemaakt en commentaar is verwijderd. Dit maakt de code uiteraard vrijwel onleesbaar, maar de bibliotheek wordt er kleiner en sneller door. Als u jQuery uitsluitend wilt gebruiken en niet per se ook de code wilt zien kunt u het beste de minified versie gebruiken.

## 14.4 De $

Wat u ook doet in jQuery, *elke* referentie die u maakt naar een element begint met een $. Dit komt doordat de $ een speciale betekenis heeft. In feite is het een alias voor het woord jQuery. In plaats van de $ mag u dus ook het woord jQuery gebruiken. Maar in de praktijk doet niemand dat. Overal waar jQuery wordt gebruikt begint een referentie met een $.

## 14.5 Selectie van elementen

Achter de $ kunt u tussen haakjes aangeven welk element of welke elementen u wilt benaderen. De syntax die hiervoor wordt gebruikt is precies dezelfde syntax als die bij CSS wordt gehanteerd. Daarnaast wordt ook nog het woord 'this' ondersteund. Dus:

- $(#div1) voor een element met een id van 'div1'
- $(span) voor alle elementen van het type span
- $(p.large) voor alle p elementen met class 'large'
- $(this) het huidige element

## 14.6 De ready functie

Zoals u eerder zag moet veel JavaScript code pas worden uitgevoerd zodra het document geladen is. In jQuery is hiervoor een aparte methode, genaamd *ready*. Dit is een methode van het gehele document. In vrijwel alle gevallen ziet u daarom de volgende code:

```
$(document).ready( function () {
  // jQuery opdrachten
});
```

Deze code zegt dat het zodra het document geheel geladen (ready) is er een functie moet worden uitgevoerd. Deze functie wordt doorgaans neergezet als een zogenaamde *anonymous function*, ofwel een functie zonder naam. Een functie die op deze manier wordt gedefinieerd kan slechts op die plek worden gebruikt en niet vanuit andere code worden aangeroepen. In dit geval geeft dat niet, omdat de functie uitsluitend uitgevoerd wordt als het document geladen is en daarna niet meer.

## 14.7 Diverse jQuery functies

De kracht van jQuery zit hem nu vervolgens in het feit dat de bibliotheek een enorm aantal functies heeft. Indien u bijvoorbeeld een element van de *style* wilt wijzigen dan gebruikt u de functie *css*. Indien u de *innerHTML w*ilt wijzigen dan gebruikt u de functie *html*. Verder kunt u events afvangen op dezelfde wijze zoals u eerder deed met de functie *addEventListener*. Enkele voorbeelden mogen dit verduidelijken.

**Voorbeeld**

```
<script type="text/javascript">
  $(document).ready( function() {
        // opdracht 1
    $("#d1").css("color","red");
        // opdracht 2
    $("div").css("font-weight","bold");
        // opdracht 3
    $("#d2").click( function() {
      $(this).css("color","red");
    });
        // opdracht 4
    $("#button1").click( function() {
      $("#wait").html("busy....");
    });
  });
</script>
```

De eerste opdracht zorgt er voor dat het element met id “d1” een rode voorgrondkleur krijgt. Dit gebeurt direct na het laden van het formulier.

Op dezefde wijze ziet u hoe in opdracht twee het lettertype vet gemaakt wordt, maar in dit geval geldt dit voor de tag "div". Het betekent dus dat dit geldt voor *alle* div's op de pagina.

In de derde opdracht wordt het click event ingesteld van een element met id "d2". Dit gebeurt door er weer een andere anonieme functie aan te koppelen. In deze functie wordt de kleur van het element veranderd in rood, maar hier gebeurt dat dus alleen indien de gebruiker op het element klikt.

Op dezelfde wijze ziet u hoe in opdracht vier de tekst van een element met id "wait" gewijzigd wordt indien er op een element genaamd "button1" wordt geklikt.

Alhoewel het vaak wel even wennen is aan al die functies die binnen functies staan, en de opeenstapeling van accolades en haakjes die hiervan het gevolg is, ziet u toch ook dat dit alles in betrekkelijk weinig code kan worden neergezet. Een element met een bepaald id opvragen is heel eenvoudig en kan gebeuren zonder het gebruik van de u bekende aanroep naar *document.getElementById*. Om alle div-elementen te vinden hoeft u niet meer alle elementen één voor één langs in een lusje - jQuery zorgt er automatisch voor dat alle elementen worden meegenomen. U hoeft zich ook niet meer druk te maken over verschillend gedrag van browserversies, want die worden door jQuery feilloos afgehandeld. Er kan natuurlijk nog .veel meer worden gedaan met jQuery. Zo zijn er talloze functies voor diverse visuele effecten, aparte functies voor animaties en voor asynchrone aanroepen (AJAX). Dit alles valt echter ruim buiten de stof van deze cursus.

# 15 OEFENINGEN

## Oefening 1: JavaScript Fundamentals

1. Open vanuit de folder Exercises het bestand *exercise1.html* in Notepad (of in uw favoriete editor).
2. Er staat al een scriptsectie in het head-gedeelte van de pagina. Gebruik *document.write*, om een regel tekst toe te voegen die vermeldt dat deze tekst vanuit de head-sectie afkomstig is. Test het script en bekijk waar de regel tekst op de pagina verschijnt.
3. Voeg een tweede scriptsectie toe onder de regel commentaar. Voeg binnen dit script een regel commentaar toe met daarin uw naam en de datum waarop het script geschreven is.
4. Voeg in de tweede scriptsectie een paar regels toe die uw naam en een paar andere regels in de pagina zetten. Let erop dat elke regel tekst ook in de pagina op een verschillende regel verschijnt door steeds aan het einde een <*br*> tag toe te voegen. Test het script en merk op waar de teksten van dit gedeelte op de pagina verschijnen.
5. Voeg aan het script in de head-sectie een alert boodschap toe die u *Goedemorgen allemaal!* wenst. Test of dit in alle drie browsers (IE, FF, Chrome) werkt en let op de verschillen. Verander de boodschap zodanig dat elk woord op een aparte regel verschijnt.
6. Kopieer het html-bestand naar *exercise 1a.html*. Kopieer en plak het script uit de head-sectie naar een apart bestand genaamd *exercise1a.js*. Zorg er nu voor dat dit script kan worden gebruikt vanuit het html-bestand door het bestand op te roepen via '*src=*' en controleer dat dit inderdaad werkt.

**Facultatief gedeelte:**

7. Voeg een regel script toe aan het head-gedeelte (of aan het js-bestand) zodat de naam (appName) en versie (appVersion) van het navigator object worden afgedrukt. Test dit gedeelte in de drie browsers en let op de verschillen. Probeer ook de *language* en de *browserLanguage* van de navigator op te vragen en bekijk de uitvoer. U zult zien dat IE hier afwijkt van FF en Chrome. Dat is nu browser incompatibility!

8 Voeg tenslotte nog een regel toe waarin u de *window.screen.width* afdrukt. Met behulp van deze informatie kunt u de pagina eventueel aanpassen aan verschillende schermgroottes.

## Oefening 2: Elementaire variabelen

1 Open Exercise2.html
2 Definieer in het scriptgedeelte een variabele genaamd *stuksprijs* en initialiseer deze met behulp van een *window.prompt* waarmee u de gebruiker om een prijs vraagt. Definieer een tweede variabele met de naam *aantal* en initialiseer deze op dezelfde wijze.
3 Definieer vervolgens een variabele genaamd *prijs* en stel dit gelijk aan het aantal vermenigvuldigd met de stuksprijs. Gebruik *parseInt* en *parseFloat* om er zeker van te zijn dat het aantal inderdaad een geheel getal is en dat de stuksprijs een getal is. Druk het resultaat af in de webpagina. Test of dit werkt.
4 Definieer verder een variabele genaamd *BTW*. Geef deze variabele de waarde van 21% van de prijs (gebruik gewoon 0.21). Bereken vervolgens het eindbedrag (de prijs plus BTW). Druk zowel de BTW als het eindbedrag af op de webpagina. Test of dit werkt.
5 Geef tenslotte 5% korting op het eindbedrag. Gebruik de operator *= om dit te berekenen en druk het resultaat af.

**Facultatief gedeelte:**

6 Verander het vorige gedeelte in commentaar. Definieer drie variabelen: n1, n2 en n3. Initialiseer n1 met de waarde 5, n2 met 7 en n3 met "7". Druk nu n1+n2 en ook n1+n3 af. Welke uitvoer verwacht u? Controleer of u het goed voorspeld had. Als u in plaats van een optelling een vermenigvuldiging had gebruikt wat was er dan gebeurd?
7 Gebruik parseInt om n3 om te zetten naar het correcte formaat en controleer dat de twee optellingen nu een identiek resultaat opleveren.
8 Probeer een regel tekst op de pagina af te drukken die er uit ziet als: *5 + 7 = 12*, maar wel met gebruik van de variabelen waar dat mogelijk is. Merk op dat er extra haakjes nodig zijn rond de rekenkundige optelling om dit correct te krijgen.

## Oefening 3: Beslissingen en lussen

1. Open Exercise3.html
2. Definieer binnen het scriptgedeelte een variabele genaamd *cijfer*. Zet de waarde van deze variabele op nul. Geef deze variabele vervolgens een waarde door de gebruiker te vragen om een waarde tussen 1 en 10 (met behulp van *window.prompt*). Druk het resultaat af op de webpagina.
3. Als het cijfer een 6 of hoger is druk dan ook de tekst "Geslaagd!" af. Druk in overige gevallen de tekst "Helaas..." af. Test het script en controleer dat de juiste teksten verschijnen.
4. Test of het ingegeven cijfer echt wel een getal is via de *isNaN* functie. Als het cijfer ongeldig is druk dan een melding af dat het cijfer onjuist is. Toon anders de boodschappen als tevoren. Controleer dat ongeldige invoer nu wordt afgehandeld.
5. Maak een *do* … while lus rond de window.prompt en controleer voor *isNaN*. Het script moet net zo lang om een getal vragen tot een echt getal wordt gegeven.
6. Construeer nu een *while* lus rond de do … while lus. Deze lus moet doorgaan zolang het cijfer een waarde van onder de 1 of boven de 10 heeft. Controleer dat het script nu om invoer blijft vragen tot er een geldig cijfer tussen 1 en 10 is ingegeven.

## Oefening 4: Werken met for lussen

1 Open exercise4.html.
2 Maak een for lus die van 1 tot 5 loopt. In elke lusdoorgang moet de tekst 'Hallo' naar de pagina worden geschreven. Controleer dat de tekst 'Hallo' inderdaad vijf maal verschijnt.
3 Maak een tweede lus die van 1 tot 10 loopt. Druk binnen de lus regels tekst uit ongeveer als volgt:

    7 * 1 = 7
    7 * 2 = 14

    Maar gebruik de lusvariabele in plaats van de getallen 1 en 2 (enzovoort). Uiteindelijk moet de tafel van 7 in de pagina verschijnen. Controleer of dit werkt.
4 Maak een derde lus die een regel HTML afbeeldt die er uit ziet als:
    deze tekst wordt groter
    De lus moet van 8 tot 20 lopen en de lusvariabele moet in de tekst worden geplakt waar in bovenstaand voorbeeld het getal 12 is gebruikt. Als het goed is verschijnen er 13 regels tekst met een steeds groter lettertype. Let erop dat de gehele string binnen enkele aanhalingstekens moet komen want de html bevat zelf dubbele aanhalingstekens.
5 Maak tot slot nog een lus die van 200 afloopt naar 100 in stappen van 10. Druk een regel tekst af die bijna gelijk is aan de vorige opdracht,maar vervang het woord 'groter' door 'kleiner' en vervang verder de aanduiding 'pt' door '%'. Als het goed is levert dit 11 regels tekst op met een grootte die afneemt van 200% naar 100% in stappen van 10%.

**Facultatieve gedeeltes:**

6 Maak een lus die van 1 tot 100 loopt en alle getallen afdrukt op de pagina..
7 Verander de lus nu zodanig dat uitsluitend de **oneven** getallen worden afgedrukt.
8 Vraag vervolgens voor de lus om een maximum en vervang de waarde '100' door dat maximum
9 Definieer een variabele genaamd *totaal*. Druk de waarden van de getallen niet langer af maar tel ze in plaats daarvan op bij het totaal. Beeld na de lus de waarde van het totaal af. Controleer dat het totaal precies 900 is als er een maximum van 60 wordt ingegeven.

10 Maak een lusje van 1 tot 6. Schrijf binnen de lus een regel html die een paar woorden bevat binnen de tags <h1>, <h2> enzovoort tot en met <h6>. Controleer dat er 6 headers verschijnen met afnemende grootte.

## Oefening 5: Werken met functies

1 Open exercise5.html.
2 Voeg aan het scriptgedeelte in de header een functie toe genaamd *sayHello*. Deze functie moet gewoon de tekst 'Hallo' tonen in een *alert* dialoog. Roep de functie aan via een regel hetzij onder hetzij boven de functie (vergeet niet de verplichte haakjes toe te voegen). Test het script.
3 Verander de functie zodanig dat deze een parameter ontvangt genaamd *name*. Toon ook de waarde van deze variabele in de *alert*. Verander nu de aanroep naar *sayHello* zodat uw eigen naam wordt doorgegeven aan de functie. Het script moet nu niet alleen hallo zeggen maar ook uw naam tonen. Controleer dat dit werkt.
4 Wijzig de functie zo dat u deze ook zonder naam aan kunt roepen. Test hiertoe de variabele *name* eerst om te controleren of deze misschien gelijk is aan null. Als dat zo is, verander de waarde van *name* dan in 'onbekende'. Verander nu de aanroep weer tot zoals deze eerder was - dus zonder naam mee te geven. Het script moet nu hallo zeggen tegen een onbekende.
5 Maak nu eerst commentaar van de aanroepen naar alert om niet te veel dialogen op het scherm te zien.
6 Schrijf een functie genaamd *cube*. Deze functie zou een parameter moeten accepteren (bijvoorbeeld genaamd *number*). De functie moet dit getal tot de derde macht teruggeven (dus invoerwaarde 3 moet 27 opleveren, 4 levert 64 op enzovoort). Roep de functie aan met de waarde 10 en toon het resultaat in een alert. Controleer dat dit de waarde 1000 oplevert.
7 Maak eerst weer commentaar van de alert voordat u de volgende opdracht maakt.
8 Maak een functie genaamd *readMark*. Kopieer het gedeelte van een eerdere oefening waar een cijfer tussen 1 en 10 werd ingevoerd en plak dit binnen de functie. Vergeet niet om het ingelezen cijfer ook daadwerkelijk terug te sturen met een *return*. Roep de functie aan en toon het resultaat in een alert. Controleer dat er altijd een geldig cijfer wordt ingelezen en dat ongeldige invoer juist wordt verwerkt.

**Facultatief gedeelte:**

9 Verander de functie zodanig dat deze twee parameters ontvangt, genaamd *min* en *max*. Wijzig de code nu zo dat een cijfer tussen *min* en *max* moet worden ingevoerd. Vergeet niet om ook de tekst in de dialoog te wijzigen

zodat deze het juiste minimum en maximum aangeeft. Pas tenslotte de aanroep aan en geef de waarden 1 en 100 mee. Controleer dat het in te geven getal nu inderdaad tussen 1 en 100 moet liggen.

## Oefening 6: Omgaan met arrays

1 Open exercise6.html.
2 Definieer in het script in de body een new array (kies zelf een geschikte naam). Schrijf nu een lus van 0 tot en met 10. In elke lusdoorgang moet element nummer x van het array gelijk gesteld worden aan x. Toon na de lus de lengte van het array in de webpagina. Controleer dat de juiste lengte (11) verschijnt..
3 Definieer een variabele genaamd *totaal* en initialiseer deze op 0. Maak nu een tweede lus die van nul tot (maar niet tot en met!) de lengte van het array loopt. In elke lusdoorgang telt u de waarde van element x op bij het totaal. Toon dit totaal na de lus in de webpagina. Controleer dat hier 55 uitkomt.
4 Voeg een regel code toe na de eerste lus (maar voor de eerste write, waarin u de lengte toont). Zet in deze regel de waarde van element nummer 20 gelijk aan 20. Dit zorgt ervoor dat u een array met 'gaten' krijgt (een 'sparse array'). Wat is nu de nieuwe lengte? En wat is nu het nieuwe totaal?
5 Repareer de tweede lus door een extra if-statement toe te voegen. Alleen als het array-element een getal is (gebruik de isNaN functie en de not (!) operator ) moet het bij het totaal worden opgeteld. Controleer dat er nu de correcte waarde van 75 uitkomt.

**Facultatief gedeelte:**

6 Maak een functie genaamd *totalize* in de header. Deze functie accepteert één parameter. Deze parameter gaat een array van getallen ontvangen. Kopieer de code van de tweede lus naar de functie, inclusief de declaratie van de variabele *totaal*. Stuur na de lus de waarde van *totaal* terug als returnwaarde. Verander nu het script in de body en vervang de tweede lus door een aanroep van de functie. Controleer of het resultaat nog steeds juist is.

## Oefening 7: Gebruik van objecten.

1 Open exercise 7.html
2 Zet in het script in de head de *title* van het document op "Dit is een nieuwe tekst" (of iets anders dat leuk klinkt). Test of dit werkt.
3 Verander in de volgende regel de titel nogmaals. Voeg ditmaal twee uitroeptekens toe aan de bestaande titel. Gebruik deze keer het element genaamd "title" van het document array. Controleer dat ook op deze manier de titel gewijzigd kan worden.
4 Zet de bgColor van het document op "lightgreen". Probeer in IE of dit werkt (waarschijnlijk wel). Test vervolgens in Chrome en FF (hier werkt het waarschijnlijk niet). Het instellen van eigenschappen van het document kan meestal beter gebeuren nadat de pagina geheel geladen is.
5 Voeg in de head een functie toe met de naam *start*. Verplaats de regel die de achtergrondkleur instelt naar deze functie. Voeg nu een attribuut toe aan de body tag, als volgt:
<body onload="start()">
Controleer dat de achtergrondkleur nu in alle drie de browsers correct wordt ingesteld.
6 Voeg een *for... in* lus toe aan de functie *start*. De lus moet alle elementen van het document aflopen. Druk bij elke lusdoorgang de key en het geassocieerde element uit document array af. Elk element moet op een aparte regel komen ( gebruik een <br> element). Run dit script en bekijk de uitvoer met alle properties. Merk op dat de originele html verdwenen is en wordt vervangen door de uitvoer van dit gedeelte.
7 Verplaats de for…in lus naar het script in de body tag. Controleer dat de uitvoer nu binnen de bestaande html verschijnt. Dit laat zien hoe document.write prima werkt binnen een pagina maar nooit moet worden gebruikt als de pagina eenmaal geladen is.

**Facultatief gedeelte:**

8 Declareer In de head sectie, maar buiten de functie, een nieuwe variabele genaamd *cursus*. Initialiseer deze variabele met een *new Object*. Stel vervolgens de *lokatie* van dit object in op (bij voorbeeld) "Utrecht". En zet daarna de *cursuslengte* gelijk aan 3. Gebruik de array syntax om "onderwerp" in te stellen op "JavaScript". Merk op dat beide manieren om een eigenschap in te stellen hier gebruikt kunnen worden.

9 Verander de lus in het body script zodat deze nu niet meer de properties van het document object laat zien maar in plaats daarvan die van het *cursus* object. Controleer dat de zelfgedefinieerde eigenschappen een voor een verschijnen. Dit toont aan dat nieuwe objecten en nieuwe eigenschappen in JavaScript geheel vrij gedefinieerd kunnen worden. Merk tot slot op dat in de lus alleen de array syntax kan worden gebruikt. De syntax met een punt en de naam van de eigenschap werkt alleen als de naam van de eigenschap expliciet wordt genoemd.

## Oefening 8: Gebruik van DOM elementen

1 Open exercise8.html.
2 U ziet een pagina waar al een aantal div elementen staan. De eerste drie hebben al een id toegekend gekregen. Geef de vierde (lege) div een id van "div4". Bekijk de pagina in een browser.
3 Voeg een regel code toe aan de functie *start*. Gebruik *document.getElementById* om de eerste div op te sporen. Zet van dit element de *style.fontWeight* op "bold'. Controleer of dit werkt.
4 Voeg op dezelfde wijze een regel code toe die de tekst van de tweede div blauw maakt (gebruik hiervoor *style.color*).
5 Geef de derde div een linkermarge (*style.marginLeft*) van "10%".
6 Haal tenslotte de eigenschap *innerHTML* op van het vierde div element. Zet hierin een nieuwe tekst en zorg er bovendien voor dat deze tekst cursief wordt getoond.

**Facultatieve gedeeltes:**

7 Voeg een regel toe aan de functie *start* die alle div elementen van de pagina ophaalt via de functie *document.getElementsByTagName*. Dit levert een array op. Doorloop dit array (gebruik de length van het array voor de bovengrens van de lus) en verander van elk element de *style.backgroundColor* in "gold". Controleer dat alle div's nu een goudgele achtergrond krijgen.
8 Voeg nog een regel toe aan het script waarmee u de tweede div ophaalt. Zet hiervan de *style.display* op "none". De tweede div lijkt nu geheel te zijn verdwenen.
9 Haal de vijfde div op en sla deze op in een variabele. Maak nu een lusje van 1 tot en met 3. Maak binnen de lus een nieuw paragraaf-element, zet hiervan de tekst op iets als 'dit is paragraaf nummer 1' (of 2 of 3 uiteraard), en voeg de paragraaf toe aan de vijfde div. Controleer dat de pagina nu drie regels extra krijgt.

## Oefening 9: Gebruik van events met inline script

1 Open exercise 9.html.
2 U ziet dat er drie knoppen zijn gedefinieerd. Voeg aan de eerste knop een *onclick* attribuut toe waarmee u de waarde van *document.bgColor* instelt op 'lightgreen' (gewoon 'green' werkt ook maar dat is wel erg fel) . Hier zijn aanhalingstekens binnen aanhalingstekens nodig, dus let goed op dat alle soorten (enkele en dubbele) op de juiste plaats staan. Test of dit gedeelte werkt.
3 Zorg ervoor dat de twee andere knoppen ook werken. Gebruik als kleuren 'lightblue' en 'pink' als u wat zachtere kleuren prefereert. Test of deze knoppen nu ook werken.
4 Bekijk het eerste div element. Zorg ervoor dat de kleur inderdaad verandert (in blauw, bijvoorbeeld) als u op de tekst klikt. Gebruik *this.style.color* om de kleur van het element te benaderen. Het woord *this* refereert aan het element waarmee gewerkt wordt.
5 Implementeer bij het laatste element de *onMouseOver* en *onMouseOut* handlers en stel de kleur in op rood als de muis over de tekst beweegt en terug op zwart als de muis de tekst verlaat. Test de code.

**Facultatief gedeelte:**

6 Voeg een event handler toe aan de laatste regel van de pagina. Iedere keer als de gebruiker op de tekst klikt moet de *innerHTML* verlengd worden met een uitroepteken. Controleer dat de tekst hierdoor langer en langer wordt bij elke muisklik.

## Oefening 10: Events koppelen aan functies

1 Open exercise10.html

2 Voordat u met de events gaat werken voegt u eerst een paar gedeeltes toe aan het script in de head. Schrijf hier twee functies, namelijk *convertToCelsius* en *convertToFahrenheit*. De eerste functie moet een parameter krijgen die de temperatuur in graden Fahrenheit aangeeft en de temperatuur in graden Celsius terugsturen. De tweede functie moet het omgekeerde doen. De berekeningen zijn:

    Celsius = (Fahrenheit – 32) * 5 / 9

    Fahrenheit = Celsius * 9 / 5 + 32

   Gebruik in beide functies *parseFloat* om er zeker van te zijn dat u uit het argument een getal haalt. Zorg er voor dat het antwoord terugkomt als een geheel getal via het gebruik van *parseInt*.

3 Als deze twee functies af zijn kunt u het click event van de knoppen afvangen. U kunt de inhoud van de eerste textbox benaderen via *temperature.celsius.value*. Merk op hoe de naam van de textbox een eigenschap wordt van het formulier en hoe het formulier weer een eigenschap wordt van het document. Geef de waarde van de textbox door aan de juiste functie en ken het resultaat van de functie direct toe aan de andere textbox. Test of dit werkt. (ter controle: 20 graden Celsius is 68 graden Fahrenheit).

**Facultatief gedeelte:**

4 Ken dezelfde code toe aan de *onblur* events van de beide textboxen. Nu hoeft u niet langer op de knop te klikken om de waardes om te laten rekenen. Een druk op de <Tab> toets is al genoeg. Een klik elders op de pagina werkt ook, overigens, zolang de betreffende textbox de focus maar verliest.

## Oefening 11: Combineren van events met dynamische stijlen

1 Open exercise11.html
2 Op de pagina staan vier div's gedefinieerd en ook een aantal stijlen. In het eerste gedeelte worden de stijlen van de eerste twee div's aangepast.
3 Vang de onclick van de eerste div af en zet hier de *className* van het element op 'large'. Dit zou ervoor moeten zorgen dat de tekst wordt afgebeeld in een veel groter en vetter lettertype. Test dit uit.
4 Vang ook in de tweede div de *onclick* af en verander de class in 'festive' zodat de opmaak totaal anders wordt.
5 Maak voor het volgende gedeelte eerst een nieuwe functie, genaamd 'show'. Deze functie ontvangt een parameter, genaamd 'id'. Haal in de functie nu eerst het element op dat door dit id wordt aangegeven (gebruik *document.getElementById*). Sla dit element op in een variabele.
6 Vraag van het gevonden element de waarde op van *style.display*. Als deze waarde gelijk is aan een lege string, verander de waarde dan in 'none'. Als de waarde geen lege string was dan moet die juist veranderd worden in een lege string.
7 Handel nu de *onclick* events van de derde en vierde div elementen af. Hier zou u de functie *show* moeten aanroepen. Daarbij geeft u het id mee van het div element dat zich onder het h4 element bevindt binnen het div element waarop is geklikt. Geef dit id mee in de vorm van een string.
Test dit gedeelte. Als het allemaal werkt dan kunt u op de kopteksten van beide secties klikken waarna de inhoud wordt uitgeklapt en bij de volgende klik weer wordt ingeklapt.

**Facultatief gedeelte:**

8 Het is fraaier om aan de gebruiker duidelijk te maken dat er op de kopteksten geklikt kan worden. Definieer daarom een nieuwe stijl genaamd *h4.showable*. Zet binnen deze stijl de kleur op blauw en de cursor op een pointer. Geef nu in de html-opmaak de beide h4 elementen de class *showable*. Test of beide elementen zich nu gedragen als een link. De kleur van de koptekst zou nu blauw moeten zijn en de muiscursor zou moeten veranderen in een handje zodra de muis er overheen beweegt. Daarmee wordt duidelijk gemaakt dat de teksten klikbaar zijn.

## Oefening 12: Events koppelen met afbeeldingen

1 Bij de cursusbestanden vindt u ook vier plaatjes (picture1.jpg tot en met picture4.jpg). Die zullen bij deze oefening gebruikt worden.
Open exercise12.html
Als u deze pagina in de browser bekijkt ziet u vier mini-plaatjes. Het is de bedoeling dat u op de plaatjes kunt klikken waarna de grote versie verschijnt.

2 Maak eerst een functie genaamd *showimage* in de header. Deze functie moet een argument genaamd 'e' krijgen. Definieer binnen de functie een variabele genaamd *sourceimage*. Stel deze variabele in op *e.target*. Op deze manier krijgt u toegang tot het image element waar op is geklikt.

3 Zet nu verder in de functie een regel code waarin u de *src* van de *largepicture* instelt op de src van het image waar de gebruiker op heeft geklikt. Zet ook nog even de *style.display* van de grote image op een lege string, zodat deze zichtbaar wordt.

4 Zorg ervoor dat in de body tag een functie genaamd *start()* wordt aangeroepen bij het *onload* event.

5 Maak een nieuwe functie genaamd *start*. Zet daarin eerst de *style.display* van de grote afbeelding op 'none', zodat deze in eerste instantie onzichtbaar is. Maak vervolgens een lusje dat loopt over alle images van het document object behalve het laatste. In elke lusdoorgang roept u de *addEventListener* methode aan van het image en geeft u "click" mee als event en *showimage* (zonder haakjes!) als functie. Zo koppelt u deze functie aan het click event van alle kleine plaatjes.
Test deze code nu. Zodra u op een van de kleine plaatjes klikt verschijnt de grote versie.

**Facultatief gedeelte:**

6 Maak een nieuwe functie genaamd *next()*. Verhoog in deze functie de variabele *currentimage* (die staat al gedeclareerd) met 1. Als de variabele gelijk wordt aan *document.images.length -1* zet de waarde dan terug naar nul. Ken nu de *src* van image nummer *currentimage* toe aan de *src* van de grote image. Zorg er ook nog even voor dat deze grote afbeelding zichtbaar is.

7 Maak verder een functie *previous()* met bijna dezelfde code. Alleen verlaagt u hier de waarde van *currentimage* en als deze onder nul komt zet u deze op de indexwaarde van het voorlaatste image.

8 Koppel tenslotte de functie *next* aan het *click* event van de 'next' knop, en zo ook *previous* aan de 'previous' knop. Pas de stijldefinitie van img.small aan In het CSS-deel en voeg hier *display:none* toe, zodat alle kleine plaatjes onzichtbaar worden. Controleer dat u nu via de knoppen alle plaatjes een voor een kunt bekijken.

## Oefening 13: Strings, Numbers en Datums

1 Open exercise13.html
U ziet hier een tabel op de pagina met een aantal textboxen met daarvoor steeds een kleine vraag of opdracht. Als u op de knop onderaan klikt zou elke textbox gevuld moeten worden met de correcte antwoorden of gegevens.

2 Zoals u ziet wordt een functie genaamd *test()* gestart als u op de button klikt. Probeer binnen deze functie een voor een de opdrachten te maken (de teksten vertellen u wel wat per regel de bedoeling is). U hebt steeds de *value* nodig per textbox om de inhoud te lezen of in te stellen
Zodra u een bepaalde regel af hebt wilt u in sommige gevallen misschien de code omzetten in commentaar om te verhinderen dat er foutmeldingen komen als u met volgende regels bezig bent. Fouten zouden kunnen optreden als een textbox leeg blijft of het verkeerde soort gegevens bevat.
Methods en properties die van pas komen bij deze opdracht zijn onder andere:
Strings: toUpperCase, length , substr, substring, charAt, split
Numbers (Math): sqrt, floor
Dates: getTime, getFullYear, getDate, getMonth, toLocaleString, New Date(...)

**Facultatieve gedeeltes:**

3 Probeer om in de code controles in te bouwen. Het zou dan niet meer uit moeten maken of er in sommige textboxen misschien geen tekst staat of tekst in het verkeerde formaat.
Waar u onder andere op wilt controleren: de *length* van de *value* van een textbox en of de inhoud wel een getal is, via de *isNaN* function.

4 Voeg een extra rij toe aan de tabel met een textbox en als opdracht om het derde woord uit de zin in de textbox te halen. U kunt een zin splitsen in woorden met behulp van de *split* functie van een string, maar zorg er wel voor dat er in het resultaat minimaal drie woorden voorkomen voor u het derde woord uit het resultaat haalt.

5 Voeg nogmaals een rij toe aan de tabel met een textbox en de opdracht om de weekdag van vandaag te tonen. U krijgt de weekdag via de functie *getDay* van een Date, maar dit levert slechts een getal op tussen 0 en 6. Als u de weekdag in leesbare vorm wilt zien dan hebt u ook nog een array nodig.

Initialiseer dit array met de namen van de zeven weekdagen. Gebruik het resultaat van *getDay* nu als index voor het array en toon het bijbehorende element in de textbox.

## Oefening 14: Validatie van formulieren

1 Open exercise14.html
U ziet een formulier met vier velden. In het JavaScript gedeelte staan al een paar extra functies om bij de validatie te gebruiken. Er wordt aangenomen dat een naam alleen letters, spaties en streepjes mag bevatten. Een leeftijd moet een getal zijn tussen 0 en 99. In dit gedeelte kunt u de postcode negeren.

2 Ook is er al een functie '*validate*', die nog geen code bevat. Met deze functie zal de data gevalideerd worden. Daarom moet deze functie gekoppeld worden aan het *onsubmit* event van het formulier. Dit event moet bovendien de returnwaarde van de functie terugsturen om eventueel de verzending te annuleren. Daarom moet u het event als volgt koppelen: onsubmit="return validate()".

3 Declareer in de functie v*alidate()* een variabele genaamd 'dataok' en initialiseer deze met *true*. Declareer verder een variabele genaamd 'errorstring' en initialiseer deze met een lege string.

4 Haal nu eerst de inhoud op van de 'firstname' textbox. Als de inhoud leeg is(controleer de *length*) zet dan dataok op *false* en voeg een melding als 'voornaam niet ingevuld' toe aan de errorstring. Als de inhoud niet leeg is, roep dan *isValidName* aan Als deze functie 'false' terugstuurt zet dan weer dataok op *false* en voeg een gepaste melding toe aan de errorstring.

5 Kopieer en plak het blok code om behalve de voornaam ook de achternaam te controleren. Uiteraard wilt u deze keer de 'lastname' textbox controleren en ook zult u de foutmeldingen willen aanpassen.

6 Controleer onderin de functie de waarde van dataok. Als deze false is zet dan de inhoud van errorstring in de innerHTML van de div met een id van 'errorlist'. Retourneer tenslotte de waarde van dataok.

7 Probeer nu uit of de validatie inderdaad werkt. Lege namen of ongeldige karakters in een naam zouden foutmeldingen op moeten leveren. Als de namen correct zijn belandt u op de 'success' pagina.

8 Nu komt de leeftijd aan de beurt. De code lijkt op de controle van de namen, maar ditmaal gebruikt u de functie 'isValidAge'. Test uw pagina om te zien of alleen leeftijden tussen 0 en 99 worden geaccepteerd.

9

**Facultatief gedeelte:**

10 Op een formulier met veel velden is het wel zo gebruikersvriendelijk om de focus in te stellen op het bovenste element met een probleem. Definieer hiertoe een variabele genaamd 'elementinerror' en zet dit direct op *null*. Overal in de code waar u constateert dat de invoer verkeerd is, en dataok op false wordt gezet, controleert u nu ook of *elementinerror* nog steeds op *null* staat. Zo ja, stel het dan in op de textbox waar u de verkeerde invoer detecteert. Bijvoorbeeld, als blijkt dat bij de voornaam geen geldige waarde is ingevuld dan stelt u elementinerror in op *persondata.firstname*.

11 Roep tenslotte aan het eind van de functie (maar uiteraard alleen als dataok op false staat!) de methode *focus()* aan van *elementinerror*. Test uw code uit en controleer dat het bovenste veld met ongeldige invoer inderdaad de invoerfocus krijgt.

## Oefening 15: Validatie met regular expressions.

1 Open exercise15.html, of werk eventueel verder met exercise14 als u die afgekregen hebt. Het facultatieve gedeelte daarvan hoeft u niet af te hebben.

2 Maak een nieuwe functie, genaamd 'isValidPostCode', die een argument accepteert. Met deze functie gaat u bepalen of een gegeven invoer een geldige (Nederlandse) postcode bevat. Dit doel bereikt u met behulp van een regular expression. Zodoende wordt de code slechts twee regels. De regular expression voor een Nederlandse postcode is:
^[1-9][0-9]{3} ?[a-z]{2}$

3 Declareer nu in de eerste regel van de nieuwe functie een nieuwe regular expression, naar keuze met een nieuwe RegExp, of via de /expression/ methode. Als u de RegExp manier gebruikt, geef dan "i" mee als tweede argument om aan te geven dat de expression ongevoelig is voor hoofdletters/kleine letters. Als u de /exp/ manier neemt dan voegt u een 'i' toe na de tweede slash om hetzelfde te bereiken.

4 In regel twee van de functie retourneert u het resultaat van de *test* methode van de regular expression, waarbij u het argument dat u via de functie ontving meegeeft als de string om te testen. De *test* methode geeft *true* terug als er een match gevonden is, en anders *false*.

5 Maak binnen de *validate* functie een nieuw blokje code, analoog aan de andere blokjes code (copy/paste is het eenvoudigst), waarin u de inhoud van de *postalcode* textbox gaat testen. Zorg er wel voor dat u de juiste textbox controleert en pas ook de foutmeldingen aan.
Test het script – 1234AB zou geldig moeten zijn, maar 0246AA, 12345xx en 1234abc allemaal niet.

**Facultatief gedeelte:**

6 Kopieer/plak de markup van een rij van de tabel en voeg zo een extra rij toe waarin u om een bloedgroep vraagt. Geef de textbox een geschikte naam.

7 Maak een nieuwe functie genaamd 'isValidBloodType' net zoals de functie die de postcode controleert. Probeert u zelf de regular expression op te stellen: een bloedgroep is a,b, ab of o (gebruik het '|' karakter om alternatieven aan te geven in de expressie), gevolgd door ofwel een '+' of een '-'. Pas wel op dar u het plus teken 'escapet' met een backslash (dus als \+) om aan te geven dat u een letterlijke plus wilt hebben. En als u de

RegExp manier gebruikt, dan moet u zelfs de backslash nog eens 'escapen' (dus dan wordt het '\\+'). Als u de expressie niet voor elkaar krijgt kunt u ook nog proberen om deze te vinden op regexlib.com.

8 Maak tenslotte weer een blok code binnen de validatiefunctie om de bloedgroep te controleren. Test uw script. Zowel 'A+', 'AB-', 'o+' als 'b+' zouden in orde moeten zijn, maar 'aa+', 'ba-', 'b' of 'a++' zeker niet.

## Oefening 16: Gebruik van timers

1 Open exercise16.html
Het doel van de oefening is om de tijd te laten lopen op de webpagina. Hiervoor is een zogenaamde 'interval' nodig. Binnen het scriptblok is overigens al een variabele gedeclareerd, genaamd 'timerid'.

2 Maak een functie genaamd 'showTime'. Maak in deze functie eerst een nieuw *Date* object. Gebruik hiervan de methode *toLocaleTimeString* om de tijdsweergave te krijgen. Toon deze in de *innerHTML* van het element met ID 'timelabel'.

3 Maak een nieuwe functie genaamd 'startTime'. Roep binnen deze functie *window.setInterval* aan. Geef twee dingen door aan deze methode: ten eerste de naam van deze functie (zonder haakjes!) . Ten tweede een timeout van 1000 milliseconden. Deze methode geeft een ID terug. Sla deze op in de variabele 'timerid'.

4 Zorg er voor dat deze functie wordt gestart als de gebruiker op de 'start' knop klikt. Controleer dat de tijd inderdaad op de webpagina verschijnt en elke seconde wordt ververst.

5 Maak een tweede functie genaamd 'cancelTime'. Roep in deze functie *window.clearInterval* aan en geef het timerid mee. Maak ook de inhoud van het element met ID 'timelabel' leeg. Koppel deze functie aan het click event van de 'stop' knop. Ververs de pagina in de browser en controleer dat u het lopen van de tijd nu kunt starten en weer stoppen.

## Oefening 17: Fouten afvangen

1 Open exercise17.html
2 Activeer in de functie *start* de commentaarregel die een 'unknown' element probeert te vinden. Het script werkt nu niet. Probeer dit uit. Merk op dat de laatste regel van de functie *start* in dit geval niet wordt uitgevoerd.
3 Zet een *try* … *catch* constructie om de foutieve regel code. De regel met de fout moet binnen het *try* gedeelte komen, het *catch* deel heeft een argument (voor deze variabele mag u zelf een naam kiezen). Toon binnen het catch deel een *alert* met een bepaalde melding (iets als: 'er is een fout opgetreden') en daarachter de variabele die u via de catch hebt ontvangen. De laatste regel van de functie *start* moet buiten de try…catch blijven.
4 Test de code. Uw eigen foutmelding verschijnt in de browser. Merk op dat de laatste regel van de functie *start* nu wel wordt uitgevoerd en dat een nieuwe regel tekst op de webpagina verschijnt. Deze techniek is dus bijzonder geschikt om fouten op te vangen. Zonder een try…catch zouden die fouten verhinderen dat de rest van de code wordt uitgevoerd, maar nu de fout wordt 'afgevangen' loopt de rest van de code zoals bedoeld.

## Oefening 18: Introductie jQuery

1 Open exercise18.html
U ziet een pagina met enkele elementen. Bovenin is al een referentie gemaakt naar jQuery. Verder is de aanroep naar *document.ready* al neergezet. Bovendien is al een opdracht in jQuery toegevoegd. De kopregel wordt cursief als u er op klikt. Controleer in de browser dat dit inderdaad het geval is.
Voeg nu op dezelfde wijze een aantal jQuery opdrachten toe, binnen de aanroep naar *document.ready*. Geef steeds de goede selector mee (hetzij een id, hetzij een html-tagnaam), selecteer het juiste event en roep vervolgens de juiste jQuery-functie aan.

2 Zorg er voor dat de eerste div verdwijnt als u er op klikt. U kunt *$(this).hide()* gebruiken.

3 Zorg er voor dat het font van de tweede div vet wordt als u er op klikt.

4 Zorg er verder voor dat deze tweede div blauw wordt als u er met de muis boven hangt en weer zwart wordt als u de muis verplaatst. Gebruik hiervoor de events *mouseenter* en *mouseleave*.

5 Zorg er voor dat de tekst van de derde div vervangen wordt door "New text" als u er op klikt. Hiervoor kunt u de *$(this).html()* functie gebruiken.

6 Zorg er tenslotte voor dat elke paragraaf rood wordt wanneer u er op klikt.

**Facultatief gedeelte:**

7 Zorg er voor dat de beide knoppen de gekleurde blokken laten verdwijnen en weer verschijnen. U kunt als selector "input:button" gebruiken maar u moet daar nog wel even het goede id achter plakken. In de functie roept u van div4 en div5 de methodes *fadeOut* en *fadeIn* aan. U kunt meegeven hoelang de 'fade' moet duren, ofwel in milliseconden ofwel met de woorden "slow" en "fast". Probeer dit kort uit.

# 16 EXERCISES

## Exercise 1: JavaScript Fundamentals

1 From the Exercises folder open *exercise1.html* in Notepad (or your favorite editor).
2 There is already a script section in the head of the page. Using *document.write*, add a line to the script that writes a line of text indicating this text was written from the head section. Test the result and notice where the text appears on the page.
3 Add a second script section below the comment line. In this script add a comment line mentioning your name and the date it was written.
4 In the second script section add a few lines writing your name and a few other lines to the page. Make sure each item is on a different line by writing *<br>* tags as well. Test the result and notice where the results of this part appears on the page.
5 Add an alert message to the script in the header wishing you *Good Morning*. Test whether this works in all three browsers (IE, FF, Chrome) and notice the differences. Now change the message so that each word appears on a different line.
6 Copy the html-file to Exercise 1a.html. Cut and paste the script in the header section to a separate file, called *exercise1a.js*. Now include this file from the html file (use '*src=*') and check that this works.

**Optional part:**

7 Add a line to the script in the header (or to the js-file) that prints the current name (appName) and version (appVersion) of the navigator object. Test this part in all three browsers and notice the differences. Try also to request the *language* and *browserLanguage* from the navigator and check the results. Notice how IE differs from FF and Chrome here. That's browser incompatibility!
8 Finally add a line to test the screen width by printing *window.screen.width*. With this information you could modify the page to adapt itself to different screen sizes.

## Exercise 2: Basic variables

1. Open Exercise2.html
2. In the script part define a variable called *unitprice* and initialize it with *window.prompt* asking the user for a price. Define a second variable called *amount* and initialize this the same way.
3. Next define a variable called *price* and set it equal to the amount multiplied by the unitprice. Use *parseInt* and *parseFloat* to make sure the amount is an integer and the price is a number. Print the result in the webpage. Test your script.
4. Define a further variable called *VAT*. Set this variable equal to 21% of the price (simply use 0.21). Next calculate the totalprice (price + VAT). Print both the VAT and the totalprice on the page. Test your script.
5. Finally give a 5% discharge on the totalprice. Use the *= operator to achieve this and print the result.

**Optional part:**

6. Comment out the earlier part. Define three variables: n1, n2 and n3. Set n1 equal to 5, n2 equal to 7 and n3 equal to "7". Next print n1+n2 and also n1+n3. What results do you expect? Check whether you were right. If you use multiplication instead of addition what will happen?
7. Use parseInt to convert n3 to the correct format and check that the two additions yield equal results.
8. Try to get a line on your page that looks like: *5 + 7 = 12,* using the variables wherever possible. Notice how you will need extra parentheses around the arithmetic addition to get the correct result.

## Exercise 3: Decisions and loops

1. Open Exercise3.html
2. Inside the script part define a variable called *mark*. Initialize this variable to zero. Next give this variable a value by asking the user for a mark between 1 and 10 (using *window.prompt*). Display the result of the mark inside the web page
3. If the mark is 6 or higher then additionally display a text like "Success". In other cases display a text like "Too bad…". Try your script and check that both texts display correctly when applicable.
4. After the mark has been entered check whether it is really a number by using the *isNaN* function. If the mark is incorrect then display a message that the format was incorrect. In all other cases display the text as before. Check whether incorrect input is now handled correctly.
5. Create a *do ... while* loop around the window.prompt and the check for *isNaN*. The script should ask for a number until a real number has been provided.
6. Create a *while* loop around the do … while loop. This loop should continue as long as the mark is below 1 or above 10. Check that the script will now keep running until a valid mark between 1 and 10 has been supplied.

## Exercise 4: Working with for loops

1 Open exercise4.html.

2 Start a loop which runs from 1 to 5. Every time the text 'Hello' should be written to the page. Check that the page displays the word 'Hello' five times.

3 Start another for loop which runs from 1 to 10. Within the loop print lines of text to produce output like this:

    7 * 1 = 7

    7 * 2 = 14

    But use the loop variable instead of the numbers 1 and 2 (and so on) in order to display the table of 7. Check the output.

4 Start a third for loop that displays a line of html that looks like this:
this text gets larger
The loop should run from 8 to 20, and the loop variable should be inserted where the number 12 is used in the example line. You should see 13 lines of text with increasing font size. Make sure you use single quotes for the entire string, because the html has double quotes inside.

5 Finally start another loop that runs from 200 down to 100 in steps of 10. You should display a line similar to the one from the previous loop, but replace the word 'larger' by 'smaller' and also replace the font indicator 'pt' by '%'. You should see 11 lines of text, with a font size decreasing from 200% to 100% in steps of 10%

**Optional parts:**

6 Create a loop that runs from 1 to 100 and displays all the numbers on the page.

7 Next modify the loop in such a way that it only displays the **odd** numbers

8 Next, ask for a maximum before starting the loop and replace the '100' with that number

9 Define a variable called *total.* Instead of displaying all the numbers add each of them to the total. After the loop display the total. Check that the total is exactly 900 when the maximum is 60.

10 Create a loop that runs from 1 to 6. Inside write a line of html that contains a few words inside the tags <h1>, <h2> etc up to <h6>. You should see 6 lines of header texts with decreasing size.

## Exercise 5: Creating functions

1 Open exercise5.html.
2 In the head section add a function called *sayHello* to the script part. The function should simply display 'Hello' in an alert box. Call this function with a line either below or above the function (don't forget the mandatory parentheses). Test your script.
3 Modify the function to take a parameter called *name*. Add the text of the name variable to the alert box. Now modify the call to sayhello to pass your own name to the function. The script should now say hello to you and specify your name as well. Test whether this works.
4 Modify the function to allow for an unspecified name. Before the alert test the name to see whether it may be equal to null. If so, change the value of name into 'stranger'. Now modify the call to what it was before – without a name specified. The script should now say hello to a stranger.
5 To prevent too many alert boxes first comment out the alert lines before you continue.
6 Create a function called *cube*. This function should accept a single parameter (for instance called *number*). It should return this number raised to the power 3 (so passing 3 should yield 27, 4 should yield 64 etc.). Call the function passing 10 as the argument and show the result in an alert box. Check whether the result is 1000.
7 To prevent too many alert boxes, again, first comment out the alert line before you continue.
8 Create a function *readMark*. Copy the part from the previous exercise where a mark was read in between 1 and 10 and paste this inside the function. Make sure you return the mark. Call the function and show the result in an alert. Check that a mark is correctly read and that illegal marks are handled.

**Optional part:**

9 Modify the function to accept two parameters called *min* and *max*. Now change the code so that a mark between *min* and *max* is read. Make sure you also modify the text in the dialog to indicate the boundary values. Finally modify the call to the function to specify 1 and 100 and check that a number should now be specified between 1 and 100 indeed.

## Exercise 6: Working with arrays

1. Open exercise6.html.
2. In the script in the body part define a new array (choose a fitting name yourself). Next create a for loop from 0 to 10. In each iteration set element x of the array equal to x. After the loop show the length of the array in the web page. Test your script and see if the correct length appears.
3. Define a variable called *total* and initialize it to zero. Next create a second loop, which runs from zero up to (but not including) the length of the array. In each iteration add element x to the total. After the loop show the total in the web page. Check that the total is equal to 55.
4. Add a line of code after the first loop, but before the first write (which shows the length). In this line set element number 20 equal to 20. This will turn your array into a 'sparse array' – an array with gaps in it. Run the code – what is the new length? And what shows up for the total?
5. Fix the second loop by inserting an if statement. Only if the array element is a number (use the isNaN function and the not (!) operator ) should it be added to the total. Test your script and check that the total is now equal to 75.

**Optional part:**

6. Create a function in the web header part called *totalize* that will accept a single parameter. This parameter will be an array of numbers. Inside the function copy the code from the second loop including the *total* variable. Make sure you return the value of *total* after the loop. Now modify the main script and replace the second loop by a call to the function. Check whether the result is still correct.

## Exercise 7: Using and creating objects.

1 Open exercise 7.html
2 In the head part of the script set the *title* property of the document equal to "This is a new text" (or something else you fancy). Test whether your script works.
3 Add a line that changes the title by adding two exclamation marks. This time use the "title" element of the document array. Notice how this syntax works as well.
4 Set the *bgColor* equal to "lightgreen". Test this line In IE (it probably works). Test it also in Chrome and FF (the line probably won't work). Setting document properties should mostly be done after the page is complete.
5 Add a function called *start* to the head part of the page. Move the line that sets the background color to this function. Add an attribute to the body tag, like this:

    <body onload="start()">

   Check that the background color is now set correctly in all browsers.
6 Add a *for...in* loop to the start function. The loop should run over all elements of the document object. Inside the loop print both the key and the associated value from the document array, each of them on a single line (use a <br> element). Run this script. Notice how the original html content is replaced.
7 Move the for...in loop to the script inside the body tag. Notice how the output is now embedded inside the original html. This shows why document.write should never be used after the page is complete.

**Optional part:**

8 In the head section, outside the function, declare a new variable called *course*. Initialize this variable to a *new Object*. Next set the *location* property of this object to (e.g.) "Utrecht". Similarly set the *length* to 3. Use the array syntax to set "topic" to "JavaScript". Notice how both approaches will work here.
9 In the body script modify the loop to run over the properties of the course object instead of those of the document object. Notice how your user-defined properties appear one by one. This shows how you can create your own objects with dedicated properties. Also notice how in the loop you

can only use the array syntax - the property syntax will only work if the property name is explicitly specified.

## Exercise 8: Using DOM elements

1 Open exercise8.html.
2 You will see a page with a few div elements already specified. The first three already have an id. Give the fourth (empty) div an id of "div4". View the page in a browser.
3 Add a line of text to the start function. Use *document.getElementById* to obtain the first div. Set the *style.fontWeight* of this element to "bold'. Check whether this works.
4 Similarly add a line which will turn the text inside the second div element to blue (use *style.color*).
5 Give the third div a left margin (*style.marginLeft*) of "10%".
6 Finally retrieve the *innerHTML* property of the fourth div element. Set this value to some new text and also make this text display in an Italic font.

**Optional parts:**

7 Add a line to the start function which will get all div elements by using the function *document.getElementsByTagName.* Loop over this array with a for loop (use the length property of the array in the condition) and for each element change its *style.backgroundColor* to "gold".
8 Add another line to the script retrieving the second div and set its *style.display* to "none". Notice how this div now appears to have totally disappeared.
9 Get the fifth div and save this in a variable. Now run a loop from 1 to 3. Inside the loop create a new paragraph element, set its text to something like 'this is paragraph number 1' (or 2 or 3 obviously) and add the new element to the fifth div. Check that the page now contains three extra lines.

## Exercise 9: Using events with inline scripting

1 Open exercise 9.html.
2 Notice how three buttons are defined. At the first button add an *onclick* atribute to set the value of the *document.bgColor* equal to 'lightgreen' ('green' works as well but may be a bit too bright) . You will need quotes inside quotes, so make sure you have all the quotes in the right place. Test whether this works.
3 Similarly make the two other buttons work as well. Use the colors 'lightblue' and 'pink' if you want softer colors rather than bright ones. Test these two buttons as well.
4 Look at the first div element. Make sure the color indeed changes when you click on it (to blue, for instance). You can use *this.style.*color to change the color of this element. The keyword *this* refers to the element we are working with.
5 For the last element implement the onMouseOver and *onMouseOut* handlers to set the color of the div to red and to reset the color to black, respectively. Test whether this works.

**Optional part:**

6 Add an event handler for the last line of the page. Each time the user clicks the text an exclamation mark should be appended to the *innerHTML*. Check that the text gets ever longer every time you click it.

## Exercise 10: Using events and functions

1 Open exercise10.html

2 Before handling events add some JavaScript to the head section. Here you should write two functions called convertToCelsius and *convertToFahrenheit*. The first function should accept a parameter specifying the temperature in Fahrenheit and return the value in degrees Celsius. The second function should do the opposite. The formulas for conversion are:

    Celsius = (Fahrenheit – 32) * 5 / 9

    Fahrenheit = Celsius * 9 / 5 + 32

   In each function use *parseFloat* to make sure you extract a number from the argument. You can make sure to return an integer result from the function by using *parseInt* on the return value.

3 Once these two functions are complete, handle the click event of the buttons. You can get to the values of the upper text box by using *temperature.celsius.value*. Notice how the name of the text box becomes a property of the form (and the form name is automatically a property of the document). You should pass the value of one text box into the correct function and assign the result of the function to the other text box. Check whether this works. (to check: 20 degrees Celsius is equivalent to 68 degrees Fahrenheit).

**Optional part:**

4 Assign the same code to the *onblur* events of the text boxes. Now you no longer need to click the buttons to convert the values – a simple <Tab> will do. A click anywhere works as well as long as the text box loses the focus.

## Exercise 11: Using events with dynamic styles

1 Open exercise11.html
As you will see there are four divs defined and a few styles as well. First we will deal with the styles of the first two divs.

2 In the first div, handle the *onclick* and set the *className* of the element to 'large'. This should change the text to a much larger and heavier font. Test whether this works.

3 In the second div handle *onclick* and set the class name to 'festive' to make the content appear completely different.

4 For the next part first create a new function called 'show'. This function should accept a parameter called 'id'. In the function first get the element indicated by the id argument, using the function *document.getElementById.* Store this value is an variable.

5 Next in the function check the value of this element's *style.display*. If the value is equal to an empty string you should change it to 'none'. If it is not an empty string you should change it into an empty string.

6 Finally handle the *onclick* events of the third and fourth div elements. You should call the show function and pass the id of the div that is inside the main div element below the h4 elements. Notice that these names should be passed as strings.
Test the results if it all works correctly you will be able to click on the header text of both sections to make the contents appear and disappear again.

**Optional part:**

7 To make it clear that the headers are clickable define a new style called *h4.showable*. In this style set the color to blue and the cursor to a pointer. Now assign the class *showable* to both h4 header elements. Test whether they now both look like a link. Their color should be blue and the cursor should change into a hand symbol when the mouse hovers over them, thus making them appear clickable.

## Exercise 12: Handling Events with images

1 Among the course files you will also find four pictures (picture1.jpg to picture4.jpg). These are needed for this exercise.
Open exercise12.html
You will see four mini-pictures in the browser. The idea is that we will be able to click on them to see the larger picture.

2 First create a function called *showimage* in the head section. This function should accept a single argument called 'e'. Inside the function define a variable called *sourceimage*. Next set this variable equal to *e.target*. This way we will obtain the image element that has been clicked.

3 Complete the function by setting the *src* property of the *largepicture* equal to the src property of the image that was clicked. Also set *style.display* of the large picture to an empty string, to make it visible.

4 In the body tag, make sure a function called *start()* is called when the *onload* event occurs.

5 Create a new function called *start*. First, set the *style.display* of the large picture to 'none', to hide it. Next create a for loop over all images of the document object except the last one. In each iteration call the *addEventListener* method of each image and pass "click" for the event name and *showimage* (without parentheses!) for the function. This way we link the code to each of the small images.
Test the code. Whenever you click a small image the large picture should appear below.

**Optional part:**

6 Create a function called *next()*; in this function increment the variable *currentimage* (already defined in the script section). If it is equal to *document.images.length -1* then reset it to zero. Next assign the src of image number *currentimage* to the large picture – also make sure this large picture is visible.

7 Similarly make a function *previous()* that has almost the same code except that it decrements *currentimage* and restarts at the last picture if that variable goes below zero.

8 Finally link *next* to the click event of the next button, and likewise *previous* to the previous button. Adapt the style definition of img.small to include

*display:none* to hide all the small images. Test to see if the buttons will now enable you to view all the pictures one by one.

## Exercise 13: Strings, Numbers and Dates

1 Open exercise13.html
You will see a table with a number of text boxes and a small question or task before it. When you click the button all text boxes should be filled with the correct data.
Complete the function *test()* that will run when the button is clicked. Try to complete the tasks one by one (the labels before the text boxes will tell you what to do). You will need the *value* of each text box to read its contents or to store new text into it.
Whenever one task is complete you may in some cases want to comment the code out to prevent errors when working on further tasks (errors may occur when text boxes have no text, or the text is in an incorrect format).
Methods and properties you may find useful when working on these tasks include:
Strings: toUpperCase, length , substr, substring, charAt, split
Numbers (Math): sqrt, floor
Dates: getTime, getFullYear, getDate, getMonth, toLocaleString, New Date constructors.

**Optional parts:**

2 Try to build in checks on the correct input so that all code will run without problems. Now incorrect or missing input from any text box will not generate errors.
3 Things to check for include: the length of the content of a text box and the correct format by using the *isNaN* function.
4 Add another table row with a text box and the task to retrieve the third word of the sentence in the text box. You can use the *split* function of a string to split the text into words, but make sure you check that the sentence contained at least three words before trying to select the third word.
5 Add a final table row with a text box and the task to retrieve the current day of the week. You can use the *getDay* function of a Date to get the correct day, but this will only yield a number from 0 to 6. If you want the weekday to be displayed as a string you will need to define an array. Initialize this array with the names of the seven days of the week. Next use the result of *getDay* as an index into the array and display the corresponding element in the text box.

## Exercise 14: Form validation

1 Open exercise14.html
You will see a form with four fields. In the JavaScript section there are already a few extra functions to help the validation. It is assumed that a name can only contain letters, spaces or hyphens, and an age must be a number between 0 and 99. We will ignore the postal code in this exercise.

2 There is also an empty function called *validate*. This function will be used to validate the data, so it should handle the *onsubmit* event of the form. The onsubmit must return the functions return value as well to (possibly) cancel the submission, so make it look like this: onsubmit="return validate()".

3 In the *validate* function declare a variable *dataok* and initialize this to *true*. Also declare a variable *errorstring* and initialize this variable to an empty string.

4 First retrieve the content of the firstname text box. If the text is empty (check the *length*) set dataok to *false* and add a text like 'missing first name' to the errorstring. If it is not empty call *isValidName* and if this returns false again set dataok to *false* and add an appropriate message to the errorstring.

5 Cut and paste the previous block of code to check the last name the same way, but obviously use the lastname text box this time and also customize the error messages for the last name.

6 At the end of the function check the value of dataok. If it is false assign the errorstring to the innerHTML of the div with ID 'errorlist'. Finally return the value of dataok.
Check to see if the validation works at this point. Empty names or illegal characters in the names should produce an error, valid input should be redirected to the 'success' page.

7 Next, check the age. The code is similar to the name checks, but use the function isValidAge this time. Test to see whether now only ages between 0 and 99 are allowed.

**Optional part:**

8 On a large form it is user-friendly to set the focus to the first element in error. To achieve this define a variable called *elementinerror* and initialize it to *null*. Everywhere in the code where an error is detected and dataok is set to false, also check *elementinerror* to see if it is still equal to *null*. If it is, set it to the element with the error. For instance, when checking the first

name and detecting a problem, set *elementinerror* equal to *persondata*. *firstname*.

9 Finally, at the end of the function, when dataok is false, call the *focus()* method on the *elementinerror*. Test the code and check whether the focus is given to the first element that has a problem.

## Exercise 15: Using regular expressions

1. Open exercise15.html, or continue working with exercise14 if you managed to finish it. The optional parts of that exercise are not necessary for this one.
2. Create a new function called 'isValidPostCode', that accepts a single argument. This function will indicate whether the argument is a valid (Dutch) postal code. To achieve this we will use a regular expression. By using this the code will only be two lines. The regular expression for a postal code in the Netherlands is:
   ^[1-9][0-9]{3} ?[a-z]{2}$
3. In the first line of the new function create a regular expression, either with a new RegExp, or with the /expression/ syntax. If you use the RegExp pass "i" as the second argument to make the expression case insensitive. If you use /exp/ syntax add an 'i' after the second slash to achieve case insensitivity.
4. In line two of the function return the result of the *test* method of the regular expression and pass the function argument as the string to test. The *test* method will return true if the string matches, otherwise it returns false.
5. In the *validate* function create a block of code similar to the other ones (copy/paste is helpful) to test the contents of the *postalcode* text box. Make sure you use the correct text box in the code and also modify the error messages.
   Test your code – 1234AB should be a valid code, but 0246AA, 12345xx and 1234abc should all fail.

**Optional part:**

6. Copy the markup for a table row and add an extra row to the form asking for a blood type. Give the text box an appropriate name. Create a function called 'isValidBloodType' similar to the one used for the postal code. Try to create the regular expression yourself: a blood type is either a,b, ab or o (use the '|' sign to indicate alternatives in the expression) followed by either a '+' or a '-'. Make sure you 'escape' the plus sign to indicate a real plus sign (use \+) and if you use the RegExp syntax take care to escape the backslash as well (so you should end up with '\\+'). If you are unable to construct the correct expression you may also try to find it on regexlib.com.

7 Finally create another block of code in the validate function to check the blood type text box. Test your code. Types such as 'A+', 'AB-', 'o+' or 'b+' should be valid, but 'aa+', 'ba-', 'b' or 'a++' should not.

## Exercise 16: Using timers

1 Open exercise16.html
The purpose of the exercise is to make a clock run on the web page. This will be done using an interval function. Notice that a variable called 'timerid' is already defined in the script block.

2 Create a function called 'showTime'. In this function first create a new *Date* object. Next use its *toLocaleTimeString* method to obtain the current time. Store this string in the *innerHTML* of the element with an ID of 'timelabel'.

3 Create a function called 'startTime'. In this function call *window.setInterval*. Pass two arguments to this method: firstly the name of this function (without parentheses!). Secondly a timeout period of 1000 milliseconds. The method returns an ID. Store this value in the variable called 'timerid'.

4 Ensure that this function is called when the user clicks the 'start' button. Check to see that the time is displayed on the web page and is refreshed every second.

5 Create a second function called 'cancelTime'. In this function call *window.clearInterval* and pass the timerid. Link this new function to the click event of the 'stop' button. Test the web page and check that the time display can now both be started and stopped.

## Exercise 17: Handling errors

1 Open exercise17.html
2 In the start function uncomment the line that tries to find an 'unknown' element. The JavaScript now won't work. Test your code. Notice that the last line of the function will not be executed.
3 Put a *try ... catch* construction around the incorrect line. The line with the error goes inside the *try* part, the *catch* part has an argument (you are free to choose your own name for this variable). Inside the catch part show an *alert* with a specific text (something like: 'a serious error occurred') followed by the variable you received in the catch phrase. The last line of the start function should be outside the try…catch.

   Test your code. You will now see an error alert in the browser. Also notice that the last line of the start function now does get executed and a new line of text appears on the page. This technique is therefore useful to trap errors. Without a try…catch those errors would prevent other code to be executed, but when the error is 'caught' further code will be executed as it normally would.

## Exercise 18: Introduction to jQuery

1 Open exercise18.html
You will see a page with several elements. At the top you will already find a reference to jQuery. The call to *document.ready* has been put in as well. And a single jQuery command has been added. The first line of the page will be shown in italic when you click it. Check that this does indeed happen.
Now add a number of jQuery commands, similar to the one already there, to the *document.ready* call. In each case you should supply the correct selector (either an id or an HTML tag name), specify the event you want and use the appropriate jQuery function.

2 Make sure that the first div disappears when you click it. You can use *$(this).hide()* to achieve this.

3 Make sure the font of the second div will become bold when you click it.

4 Also ensure that the second div will become blue when you hold the mouse over it and black again when you move the mouse away again. For this you can use the events *mouseenter* and *mouseleave*.

5 Make sure that the text of the third div is replaced by "New text" when you click it. For this you can use the function *$(this).html().*

6 Finally make sure that every paragraph turns red when you click it.

**Optional part:**

7 Make sure that both buttons do as they promise (fade the colored blocks). For the selector you should use "input:button" but you will also have to add the correct id to that. In the functions you can use *fadeOut* and *fadeIn* on both "div4" and "div5". You can indicate how long the fade effect will take by specifying "slow", "fast" or a number in milliseconds. Try to make all of these methods work.