



# **JAVASCRIPT PROGRAMMING FUNDAMENTALS**

Projecten

**Deze cursus is eigendom van de VDAB**

## Inhoudsopgave

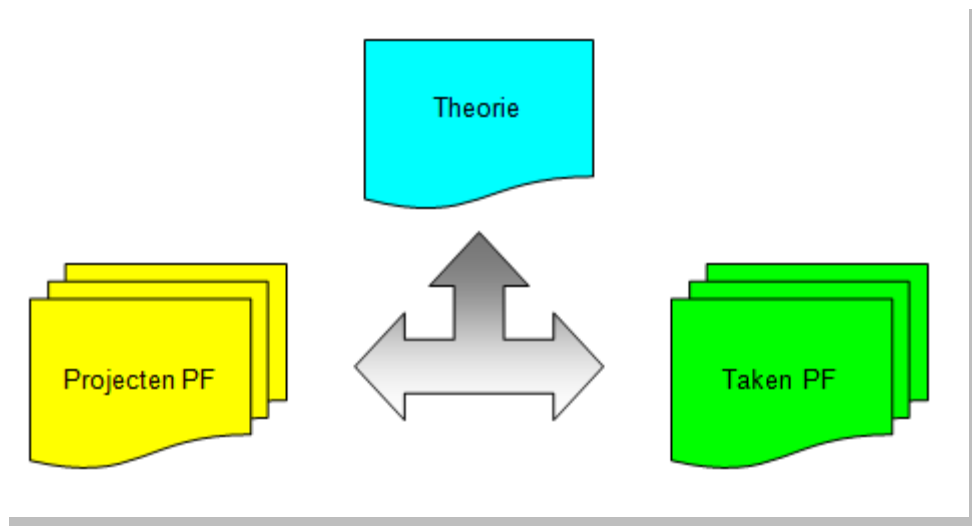
<b>1</b>	<b>INLEIDING.....</b>	<b>3</b>
1.1	Software .....	3
1.1.1	<i>Editors.....</i>	<i>3</i>
1.1.2	<i>De browsers.....</i>	<i>4</i>
1.1.3	<i>Extra software.....</i>	<i>4</i>
1.2	Conventies .....	4
1.3	Basisbestanden.....	5
<b>2</b>	<b>DE PROJECTEN.....</b>	<b>7</b>
2.1.1	<i>Het Document Object Model, een kennismaking.....</i>	<i>8</i>
2.1.2	<i>Examenresultaten.....</i>	<i>20</i>
2.1.3	<i>Getallenreeksen.....</i>	<i>30</i>
2.1.4	<i>Image gallery op drie manieren .....</i>	<i>39</i>
2.1.5	<i>Arrays en objecten.....</i>	<i>55</i>
2.1.6	<i>Een kalender.....</i>	<i>75</i>
2.1.7	<i>CookieBank .....</i>	<i>87</i>
2.1.8	<i>ISBN-10 validatie .....</i>	<i>106</i>
2.1.9	<i>Formulier: Birdy Airways.....</i>	<i>116</i>
2.1.10	<i>Quiz van de week .....</i>	<i>133</i>
<b>3</b>	<b>COLOFON .....</b>	<b>153</b>

## 1 INLEIDING

Voor deze module "*Javascript Programming Fundamentals*" krijg je drie delen:

Dit is het **projectenboek** dat hoort bij de **Programming Fundamentals** van de handleiding Javascript. Dit is het praktische deel van waaruit je vertrekt.

Er zijn nog twee andere delen: de **Theorie** en de **Taken Programming Fundamentals**.



We raden je aan als volgt te werken:

1. Voer een **project** uit in het "**Projecten PF**" boek
2. Lees de **bijhorende theorie** in het "**Theorie**" deel
3. Maak de **bijhorende taken** zoals ze aangegeven worden in het "**Projecten PF**" boek.  
De taken staan in het "**Taken PF**" boekje.

Het **Theorie** deel bevat ook topics voor het "*Projecten Advanced*" gedeelte, een module die niet iedereen moet doornemen, dus lees enkel wat aangegeven wordt.

**Bij elk project staat duidelijk vermeld welke theoretische topics je dient na te lezen.**

Als bijvoorbeeld in een project een **switch** statement gebruikt wordt, en je hebt dat nog nooit gebruikt, dan ben je verondersteld de syntax van deze structuur na te kijken in de theorie. In dit "Projecten" gedeelte wordt geen theoretische uitleg gegeven.

### 1.1 Software

#### 1.1.1 Editors

De oefeningen in de cursus kan je maken met eender welke tekstverwerker of je kan een editor gebruiken. Deze handleiding stelt zich op dat vlak neutraal op: dit is geen handleiding "*Javascript & DOM met programma X*".

De keuze van editor is vrij: je werkt best verder in de editor die je gebruikte voor HTML&CSS.

Er zijn er heel wat:

- Adobe **Dreamweaver**
- **Visual Studio**
- **Eclipse**
- **Aptana**
- **Zend studio**
- **PHPedit**
- **Nvu** webauthor (freeware) [www.nvu.com](http://www.nvu.com)
- **FirstPage** HTML Editor (freeware) [www.evrsoft.com](http://www.evrsoft.com)

### 1.1.2 De browsers

Alle moderne browsers voeren Javascripts uit, maar ondersteunen niet altijd alle webstandaarden of hebben eigen objecten, methods en properties.

Het is essentieel dat je je script altijd test in meerdere browsers.

Installeer daarom nu:

- **FireFox (FF)**
- **Internet Explorer (IE)**
- **Chrome (Chr)**
- **Opera (O)**

eventueel ook **Safari**

### 1.1.3 Extra software

Om makkelijk fouten in je Javascript op te sporen, te *debuggen*, kan je de hulp van sommige browserplugins goed gebruiken.

Installeer daarom nu:

- **Firebug** voor FF
- **IE Developers tools**
- **Chrome developer tools**



#### Opmerking:

Firebug en andere debuggers vertragen het lezen van een webpagina. Voor websites die zwaar gebruik maken van Javascript, zoals GMail, Google Maps en videosites e.a. kan dit erg storend werken.


Als je Firebug niet nodig hebt, zet hem af.

## 1.2 Conventies



De volgende opmaak wordt hier gehanteerd:

- **Javascript objecten, properties en methods** worden in een rood monotype font geschreven, bv. `document.getElementById()`.
- **HTML elementen** en hun **attributen** worden in een blauw monotype font geschreven, bv. `div`.
  - lees `div#content` als het `div` element met de `id` "content"
  - lees `div.keuzevakjes` als een `div` element met de `class` "keuzevakjes"
- Grotere stukken code worden omkaderd:

```
var getal = Number('3.5') // retournt 3.5
var probleemgetal = Number('3.5 witte muizen') // retournt NaN
```

- Bestandsnamen (*cookiebank.html*), eigen variabelen (*eltem*), Engelstalige begrippen (*closure*) worden in schuinschrift gezet.
- Het teken  wordt gebruikt om aan te duiden dat de volgende regel aan de huidige gevoegd dient te worden. De beperkte bladbreedte noodzaakt ons soms lange statements in een aantal lijnen te splitsen.

Bijvoorbeeld:

```
if (nieuwSaldo<=0){
    strBericht = "Uw saldo is onvoldoende om dit 
                bedrag af te halen. ";
    strBericht += "U kunt maximaal " + eval(saldo-1) + 
                " Euro afhalen.";
    waarschuwing(strBericht);
    bedragVeld.value=saldo-1;
    bedragVeld.focus();
}
```

Hierin moet je de eerste twee lijnen van de `if` schrijven als

```
strBericht = "Uw saldo is onvoldoende om dit bedrag af te halen. ";
strBericht += "U kunt maximaal " + eval(saldo-1) + " Euro afhalen.";
```

- Een **tip** dient om je aandacht te trekken op een nuttige levenswijsheid, veelvoorkomend probleem, een handige oplossing:



" Wijsheid verwerf je op 3 manieren: ten eerste door na te denken, de meest edele manier, ten tweede door na te bootsen, dat is het gemakkelijkst en tenslotte door ervaring, die de bitterste methode is. (*Confusius*)"

### 1.3 Basisbestanden

Alle oefenbestanden vind je in de bijhorende *zip* file. Alle beeldmateriaal is voor het gemak gecentraliseerd in één map.

Zorg zelf voor een mappenstructuur op je website, want bestandsnamen uit verschillende projecten kunnen dezelfde zijn.

Het basisbestand *js\_form Ontvanger.htm* wordt in nogal wat formulieren gebruikt: het is handig om te controleren welke gegevens een formulier doorstuurt (en of hij ze hoegenaamd doorstuurt) zonder daarbij **.Net**, **Java** of **PHP** te moeten gebruiken.

## **2 DE PROJECTEN**

Voer deze projecten in volgorde uit en maak de aangeduide taken.

Het is belangrijk dat je dieper ingaat op de behandelde thema's door ook de relevante theorie hoofdstukken te lezen!

## 2.1.1 Het Document Object Model, een kennismaking

### Doel

Een kennismaking met het **Document Object Model** (DOM) in een browser.  
We leren via de verschillende '*developer tools*' van de browsers de DOM tree tonen en bepaalde eigenschappen inspecteren.

We maken onze code helder en duidelijk leesbaar door er voldoende spatiering, tabs en commentaar in te voorzien.

We proberen enkele zeer eenvoudige DOM ingrepen

### Theorie

Lees de volgende theorie topics na:

- Het Document Object Model
- naamconventies voor variabelen

### Duurtijd:

30 min

### Basisbestand

*documentObjectModel.html*.

### Het DOM

Als je het basisbestand opent ziet het er zo uit in een browser:



Wat is er gebeurd om deze pagina op je scherm te krijgen?

- Je browser richt een **verzoek**, een HTTP *request*, aan een **webserver** om een HTML pagina te krijgen
- de webserver stuurt de HTML pagina door: een tekstbestand met HTML codes



- de browser interpreteert, ***parsed***, de HTML code die wordt vertaald naar **nodes**
- de browser bouwt daarmee een ***Document Object Model*** in zijn intern **geheugen**, een soort **boomstructuur** van **element-** en andere **nodes**: de ***document tree***
- deze DOM wordt opnieuw geïnterpreteerd om er een beeld mee op te bouwen



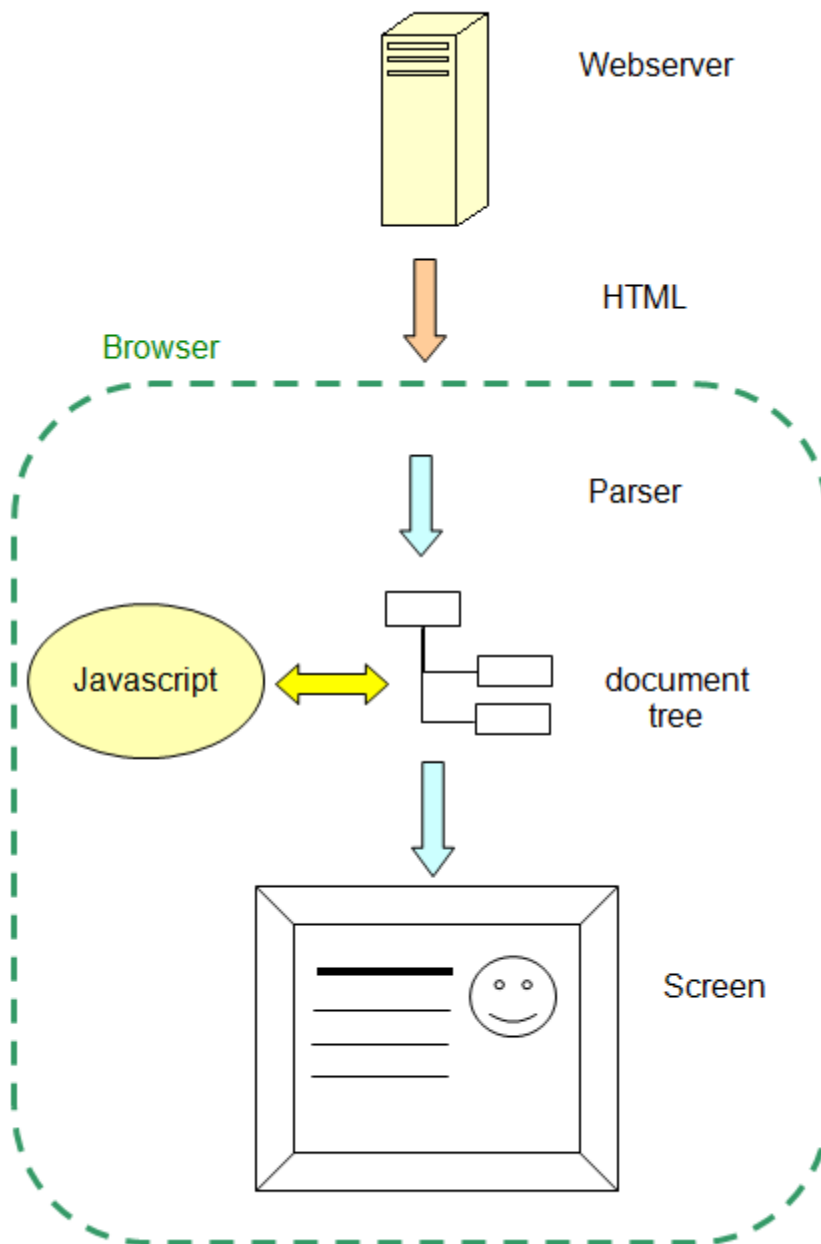
het **beeld** van de HTML pagina op het scherm is **direct gekoppeld** aan de **document tree** in het werkgeheugen van de browser, niet aan de HTML code van de pagina!

En dan is er nog **JavaScript**.

Elke browser heeft een versie van Javascript ingebouwd. Met deze scriptingtaal kunnen we niet enkel allerlei bewerkingen, berekeningen doen maar ook de *document tree* **rechtstreeks manipuleren**, zodat dit een onmiddellijk effect heeft op het beeld.

JavaScript stelt ons dus in staat de oorspronkelijke HTML pagina helemaal te wijzigen **nadat** deze ingelezen werd.

Een overzicht van de *flow*:



### De *document tree* zichtbaar maken

In onze editor zien we de HTML van de pagina, in de browser zien we het beeld die deze ervan maakt, maar waar kunnen we de ***document tree*** bekijken? Daarvoor gebruiken we best een plugin. Die laat ons ook toe allerlei eigenschappen te zien die anders verborgen blijven, heel interessant om te debuggen - fouten te achterhalen.

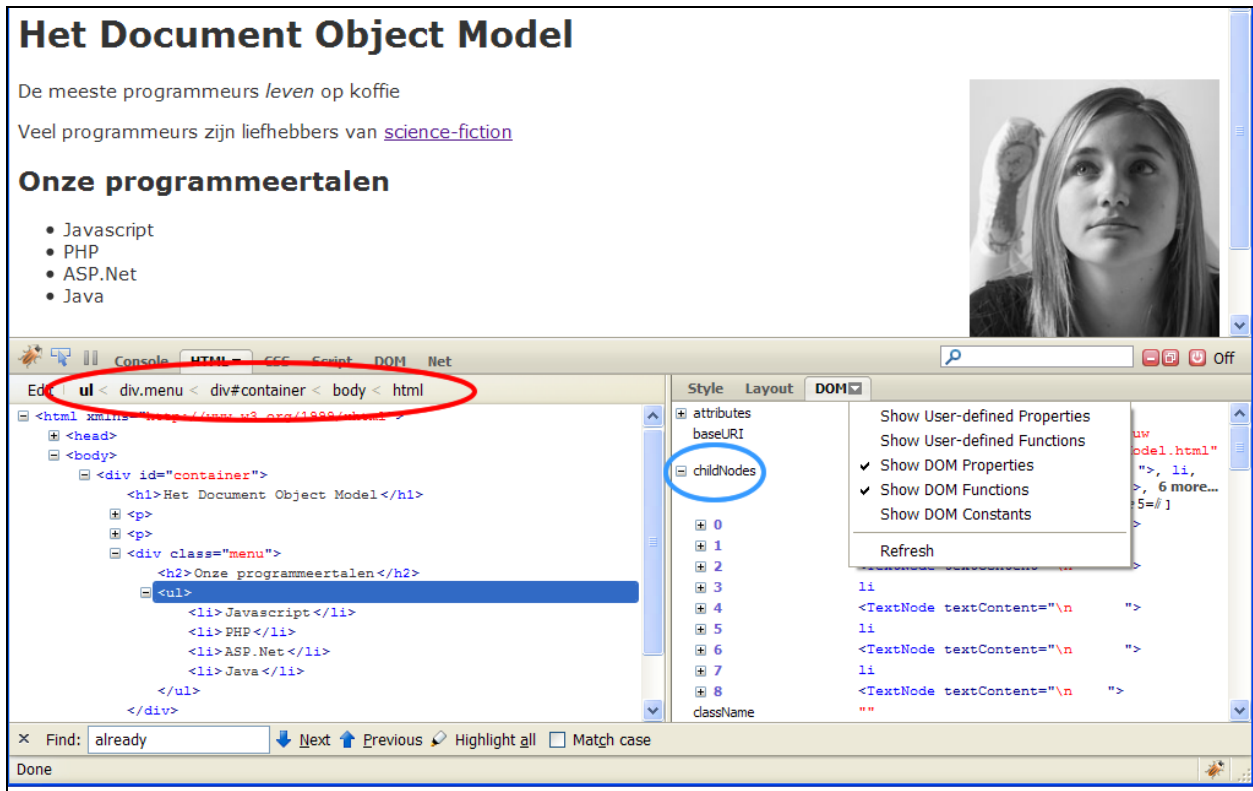
We openen ons basisbestand *documentObjectModel.html* pagina in 3 verschillende browsers: FF, IE en Chrome

## Firebug in Firefox

Eenmaal geladen in FireFox starten we **Firebug**.

Klik in het FireBug paneel bovenaan op het menu **HTML**. Het paneel heeft nu twee helften. Klik in de rechterhelft op het menu **DOM** en kies uit zijn menu SHOW DOM PROPERTIES en SHOW DOM METHODS.

je ziet nu dit:



Klik aan de linkerkant op het + van het **ul** element. Je ziet nu de **li** elementen uitgevouwen.

Bemerk:

- De hierarchie van **parent** elementen in de bovenbalk (rode ovaal).
  - Beweeg de muis erover: ze worden aangeduid in de pagina
  - Klik op een **parent** element: het wordt geselecteerd
- Klik opnieuw op het **ul** element. Kijk naar de rechterhelft:
  - je ziet een DOM property **childNodes** (blauwe ovaal) met daaronder 8 genummerde **childNodes**. Acht? er zijn maar 4 **li** elementen? Ook de "nieuwe lijn" karakters (\n) worden beschouwd als een *node*
- Onderzoek nog wat verder de mogelijkheden van het DOM paneel. Bemerk o.a. de verschillende DOM properties die aangegeven worden voor elke element/node die je selecteert, zoals bijvoorbeeld de **id** of de **className**
- FireBug gaat nog een stapje verder: het laat je toe de *document tree* direct te manipuleren, we proberen twee dingen:

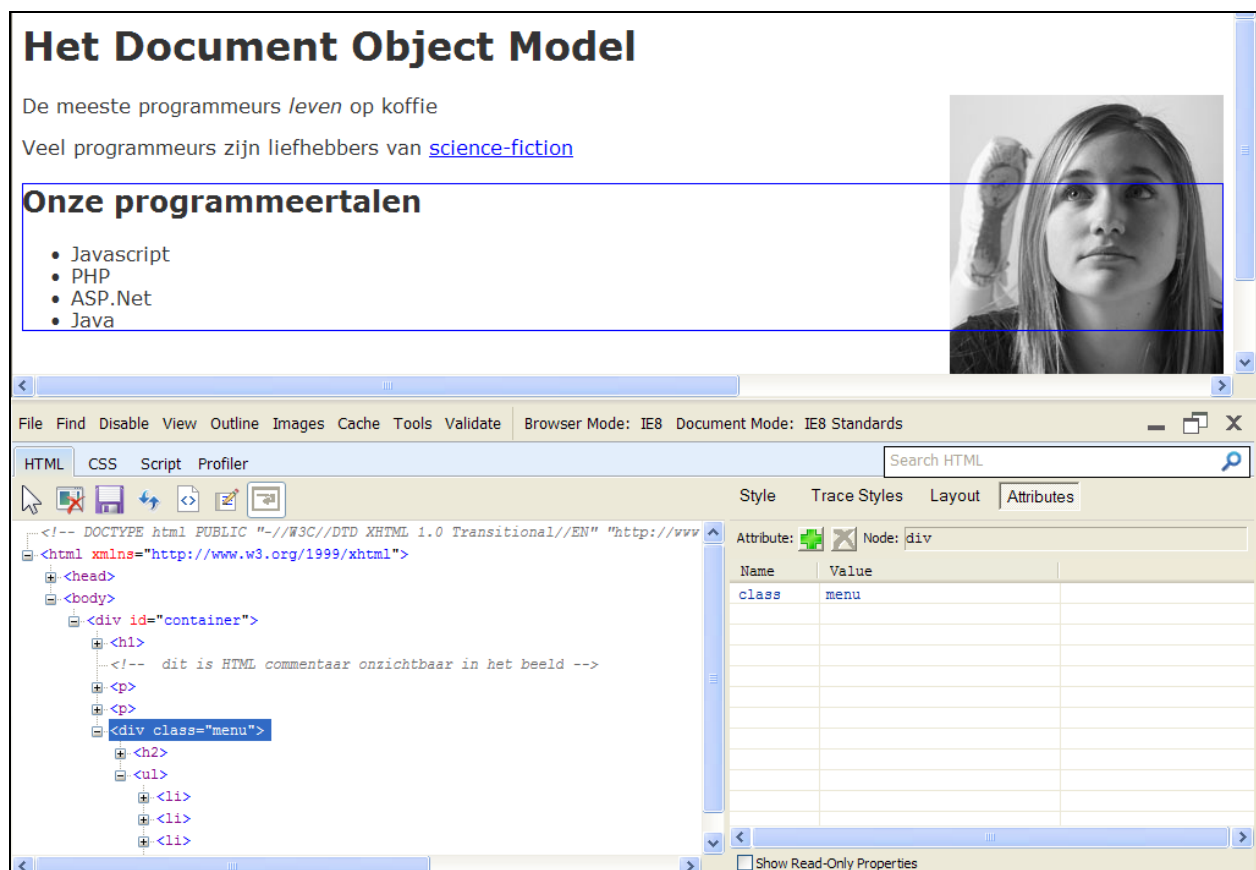
- selecteer één van de **p** elementen in FireBug, doe een rechtsklik en kies DELETE ELEMENT uit het shortcutmenu: het element verdwijnt...
- selecteer het **img** element in het HTML paneel in FireBug, in het rechterpaneel kies het STYLE menu. je ziet de CSS styles voor het element. Je bemerkt **float:right**. Wijzig 'right' in 'left'. Het effect is onmiddellijk zichtbaar.

De wijzigingen die je in Firebug kunt doen, zijn echter maar van tijdelijke aard: ze dienen enkel om te *debuggen*, te kijken wat het effect is van een wijziging. De HTML code kan door een webdeveloper tool niet permanent gewijzigd worden.

## Developer tools in IE

In **IE** start je de **IE Developer tools** door F12 te drukken. Je kan dit programma ook als apart venster instellen met de functie "Unpin" (icoon rechtsboven).

Ook hier vinden we twee panelen als we links het HTML menu kiezen. Het uitzicht is ongeveer gelijk, maar de DOM eigenschappen kan je hier niet bekijken. Attributen van de elementen wel:



Bemerk:

- ook hier kan je de CSS styles direct wijzigen en het effect daarvan bekijken.

## Chrome en zijn developer tools

Ook Chrome heeft **developer tools** en een **Javascript console**.

Bekijk nu je pagina in Chrome, klik rechtsboven op het *instellingen* icoon en kies **tools**, daarna **developer tools** of **Javascript Console**.

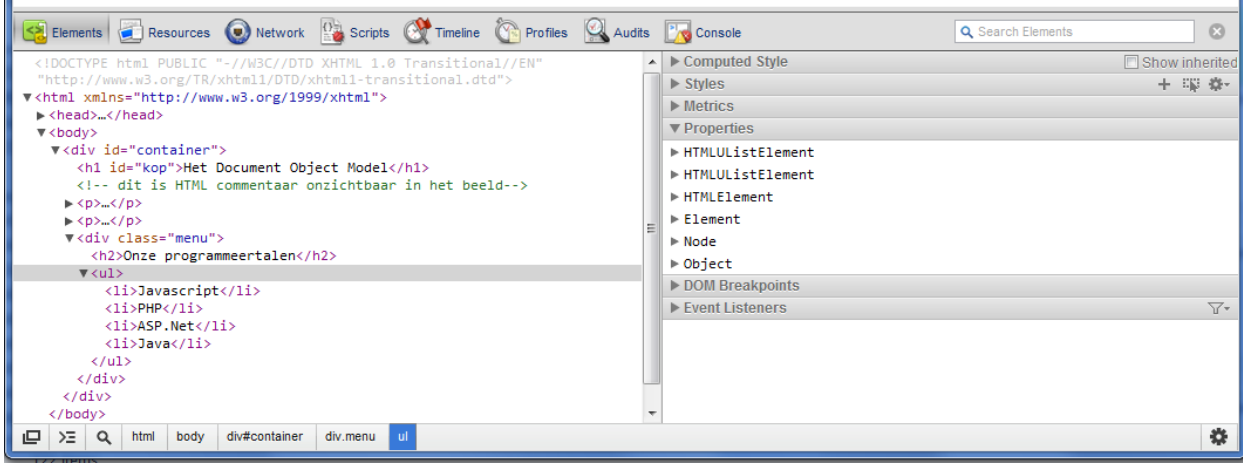
## Het Document Object Model

De meeste programmeurs *leven* op koffie

Veel programmeurs zijn liefhebbers van [science-fiction](#)

### Onze programmeertalen

- Javascript
- PHP
- ASP.Net
- Java



Klik ook hier op het **ul** element in de *document tree*.

Merk hier ook de lijn van *parents* op onderaan de pagina en aan de rechterzijde de verschillende panelen die informatie geven over het element: *Computed style*, *Styles*, *Metrics*, *Properties*, *DOM breakpoints* en *Event Listeners*. Chrome is de enige browser die informatie geeft over Event listeners.

Enkele korte opdrachtjes:

- hoeveel **childNodes** heeft het **h1** element?
- wat is de **parentNode** van de **div#menu**?

### de script tag

Nu maken we een eerste javascript waarin we enkele wijzigingen aan het ingeladen document maken.

Daarvoor hebben we een `script` element nodig. Typ de volgende code in de `head` van het document onder het `style` element:

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

Bemerkingen:

- de `script` tag kan zowel in de `body` of in de `head` van het document geplaatst worden.  
Dat mag **eender waar**: meestal plaatst men een script in de `head`, maar dat is geen *must*. Sommige auteurs plaatsen een script zelf consequent als **laatste element** in de `body` tag, dan hebben ze nooit problemen met referenties (leggen we straks uit)
- het is beter `script` elementen onder `style` elementen te zetten zodat stylesheets eerst ingeladen worden.
- nog beter is de JS in een **apart bestand** te plaatsen en dat te koppelen via de `script` tag. In verdere projecten passen we dit toe.  
Toch zal een *lokaal* script soms noodzakelijk zijn.
- afhankelijk van de HTML standaard heeft de `script` tag een attribuut nodig:
  - in XHTML is het attribuut `type` verplicht:  
het duidt het MIME type aan, in dit geval is de waarde steeds `"text/javascript"`. Achterwege laten is echter geen probleem: het is tenslotte de server die het MIME type bepaalt, niet de webpagina
  - in HTML5 is geen enkel attribuut nodig.  
Het MIME type wordt steeds verondersteld `"text/javascript"` te zijn.
  - voor een extern script zullen we straks het `src` attribuut gebruiken.
- De Javascript **statements** staan tussen de begin en eindtag

Om deze reden gebruiken we in dit project een `type` attribuut.

### Enkele eenvoudige ingrepen

We willen enkele “*onmiddellijke*” wijzigingen aan de pagina maken zodat de client deze ervaart als aanwezig vanaf de start...

*kan "onmiddellijk" wel?*

- De browsers leest de HTML en Javascript code in de volgorde van de paginatekst, dus een javascript1 die boven een javascript2 staat, wordt eerst uitgevoerd.

- Javascript statements worden **direct uitgevoerd**, dus een berichtenvenster (**alert**) laten verschijnen kan gebeuren vóór er ook maar iets van de pagina te zien is.
- Voor wijzigingen aan de elementen kan dat echter niet, want we moeten wachten tot de browser alle HTML gelezen heeft en die in een **document tree** heeft omgezet.  
Het ogenblik waarop de browser alles geparsed heeft is een **Event**, een **gebeurtenis**, in dit geval het **load** event van het **window** object.
- Wij kunnen wachten op dit event met **window.onload**. Dus alle code die in de event handler van dit event staat, wordt niet onmiddellijk uitgevoerd maar wacht op het afvuren van het event **load**.

In eerste instantie willen we de tekst van de titel dynamisch wijzigen.

We typen de volgende code:

```
<script type="text/javascript">
  window.onload = function(){

  }
</script>
```

Bespreking:

- **window.onload** is het event waaraan we een **event handler** koppelen: een **function** statement
- Het **function** statement is hier **anoniem**, het heeft geen naam, het zal gewoon uitgevoerd worden als het event *vuurt*

Nu vullen we de **function** verder op:

```
<script type="text/javascript">
  window.onload = function(){
    /* wijzig de titel */
    var eTitel      = document.getElementById('kop'); //referentie naar node
    eTitel.innerHTML = "De document tree";
  }
</script>
```

Bespreking:

- We plaatsen eerst wat **commentaar** om onze acties te verduidelijken. In Javascript kan je commentaar zetten:
  - tussen  
    /\*  
    commentaar  
    hier





- We verwijderen eerst de aanwezige *textNode* met de method `removeChild` die we de eerste (en enige) *childNodes* meegegeven.
- We maken een nieuwe *textNode* aan die we opvullen met tekst en die we invoegen in het `li` element.

Merk ook op dat we alle variabelen laten voorafgaan door een **prefix** waarmee we willen aangeven welk *datatype* de variabele zal bevatten.

Zo duidt *eBolletjes* aan dat het hier een element of een collectie elementen betreft, geen tekst of geen getal. Het gebruik van prefixes is een conventie – een afspraak – die niet verplicht is maar veel verduidelijkt.

Je vraagt je misschien af waar het probleem kan liggen als je een variabele *aantal* noemt, iedereen begrijpt toch dat het hier over een getal gaat? Inderdaad, maar de problemen beginnen bij situaties zoals deze.

*Een invulveld vraagt de leeftijd van een persoon:*

```
var leeftijd = document.getElementById('leeftijd');
leeftijd     = leeftijd.value;
leeftijd.innerHTML = "";
```

Dit kan fout gaan omdat de variabele *leeftijd* hier *hergebruikt* wordt voor verschillende dingen: elementen en getallen. Dit is veel duidelijker:

```
var eLeeftijd = document.getElementById('leeftijd');
var nLeeftijd = eLeeftijd.value;
eLeeftijd.innerHTML = "";
```

Lees meer over prefixes in het hoofdstuk **Syntax – variabelen** in de theorie.

*Waarom gebruiken we hier geen `innerHTML`?*

Hier maken we gebruik van DOM node en element methods puur om ze te demonstreren. `innerHTML` had evengoed gekund, en zou het verwijderen van de *textNode* overbodig gemaakt hebben. Vooral voor het invoegen of wijzigen van grotere stukken html, is `innerHTML` te prefereren omdat het ook veel sneller werkt.

Tenslotte willen we een nieuw *list item* “Perl” toevoegen achteraan. Voeg de volgende code toe, nog altijd binnen de haakjes van de **function**:

```
...
/* een nieuw item invoegen achteraan de lijst */
var eLijst = document.getElementsByTagName('ul')[0];
var eItem  = document.createElement('li');
var sTekst = document.createTextNode('Perl');
eItem.appendChild(sTekst);
eLijst.appendChild(eItem);
...
```

Bespreking:

- we refereren de lijst met `document.getElementsByTagName('ul')`: ook hier maakt `getElementsByTagName` een collection van **alle** `ul` elementen in de pagina, dus gebruiken we de array index `[0]` om aan de eerste `ul` aan te duiden
- nu maken we een nieuw `li` element aan met `document.createElement('li')`
- we maken ook een `TextNode`
- die we daarna toevoegen aan het `li` element
- het `li` element wordt achteraan toegevoegd

## Attributen

We moeten niet enkel weten hoe we de inhoud van een element moeten instellen, maar ook hoe we een attribuut kunnen zetten of wijzigen.

Het document bevat een figuur, een `img` element. Wat er getoond wordt is afhankelijk van het pad in het `src` attribuut. Het `alt` attribuut bevat een plaatsvervangende tekst als er een probleem zou zijn met het beeldbestand. De figuur heeft geen `title` attribuut, nochtans belangrijk om een *tooltip* te tonen als je met de muis over de figuur glijdt.

```
...
/* attributen zetten en wijzigen */
var eImg          = document.getElementsByTagName('img')[0];
eImg.src          = "../images/elise.jpg";
var sTooltip      = "Elise, system engineer";
eImg.setAttribute("alt",sTooltip);
eImg.setAttribute("title",sTooltip);
}

</script>
```

Bespreking:

- eerst maken we een collection `img` elementen met `getElementsByTagName`
- daar nemen we het eerste (en enige) element van, door de array index `[0]` te gebruiken
- in de meeste gevallen kan je een attribuut direct instellen door het te behandelen als een property (eigenschap) van het DOM element, dus hier `eImg.src`, dat we een nieuw pad geven naar een ander beeldbestand en waarmee je de figuur echt verandert.
- we maken een tekstje aan in `sTooltip`
- deze maal gebruiken we de DOM method `setAttribute` waarmee je eender welk attribuut kunt aanmaken en wijzigen. De syntax is:

`elm.setAttribute(attribuutnaam, waarde)`

hier wijzigen we de tekst van het `alt` attribuut en maken een nieuw `title` attribuut aan

Wat heeft de voorkeur? Directe property gebruiken of `setAttribute`? Er is niet altijd een directe property beschikbaar en tegenwoordig werkt `setAttribute` in alle browsers en voor alle attributen, dus maak je eigen keuze, maar zorg dat je steeds uittest in alle browsers!.

In dit project leerde je werken met slechts enkele van de belangrijke DOM methods en attributes. Er zijn er uiteraard veel meer en je zult ermee kennismaken in de volgende projecten.

Wil je echter een overzicht van alle methods en attributes, kijk dan zeker de hoofdstukken "*de belangrijkste DOM Attributes*" en "*de belangrijkste DOM Methods*" na in het theorie boek.

**Taken:**

Maak nu de volgende taken:

- DOM Intro

## 2.1.2 Examenresultaten

### Doel

Een verdere kennismaking met Javascript statement en DOM properties en methods.  
Koppelen van een script via een event handler aan een knop, unobtrusive javascript.  
Kennismaking met de Javascript console in FireBug.

### Theorie

Lees de volgende theorie topics na:

- syntax
- data types, variabelen, scope
- structuren: `if`
- DOM
- onclick event handler

### Duurtijd:

30 min

### Basisbestand

*selectie\_examens.html.*

### Examen

Bekijk even de broncode van het basisbestand:

- Er zijn 3 `input` velden en één `button` element
- Allemaal hebben ze een `id` en elk invulveld heeft ook een `name` attribuut.

### De script tag

Plaats deze keer de `script` tag als voorlaatste element – dus net vóór de `</body>` tag in de HTML:

```
...  
<div id="output"></div>  
<script>  
  
</script>  
</body>  
</html>
```

### Bemerkingen:

- We plaatsen deze keer het script achteraan om te bewijzen dat DOM referenties leggen op die plek geen `window.onload` nodig heeft omdat het script het laatste is wat de browser leest, en dus alle elementen reeds ingeladen zijn.

## Een eigen functie maken

Als we op de knop 'Go' klikken moeten de ingevulde punten geëvalueerd worden en moet er een bericht verschijnen dat zegt of de student geslaagd is of niet.

We zijn momenteel optimistisch en veronderstellen dat de gebruiker braaf **getallen** zal ingeven, **validatie** van de gegevens volgt in verdere projecten.

We hebben dus twee zaken te doen:

- een script maken dat de punten evalueert
- dit script *unobtrusive* koppelen aan de “Go” knop

We maken een begin aan onze functie in het **script** element:

```
<script>

function evalueer(){
  /* evalueert of een student geslaagd is of niet */
  alert('test');
}

</script>
```

Bespreking:

- Deze keer schrijven we een **benoemde functie**, die bestaat uit:
  - het sleutelwoord **function** wordt gevolgd door
  - de **unieke naam** van de functie, gevolgd door
  - een paar ronde haakjes ( ) met eventuele argumenten. Daarna volgt
  - het *block* statement tussen de accolades { }.
- We hebben een voorlopig statement in de functie gezet: **alert()** zal een berichtenvenster doen verschijnen

De functie is klaar, maar zal niet vanzelf starten, we moeten de functie als *eventhandler* installen voor het klikken op de “Go” knop.



### Terminologie:

- **event**: de gebeurtenis, bv. **click**, **submit**, **load**, ...
- **event handler**: de functie die reageert op het event, moet je zelf maken en koppelen

Om de knop te koppelen moeten we een referentie ernaar leggen:

```
<script>

var eKnop      = document.querySelector('#goKnop');
eKnop.onclick = evalueer;
```

```
function valueer(){
  /* valueert of een student geslaagd is of niet */
  alert('test');
}
</script>
```

Bespreking:

- De method `querySelector` met de selector `#goKnop` refereert één element. Deze method is identiek aan `getElementById('goKnop')`. Merk wel het spoorwegteken op. We demonstreren deze nieuwe method hier als alternatief.

De methods `querySelector` en `querySelectorAll` zijn verschillend omdat ze gebruik maken van **CSS selectors** zoals `#menu` of `.menu` of `div.zijbalk > ul`. De enorme verscheidenheid van de CSS selectors geeft ons veel meer mogelijkheden dan we kunnen hebben met `getElementById` of `getElementsByTagName`.

- Het `click event` van de knop wordt opgevangen door de *event handler* `onclick` en toegewezen aan de functie `valueer`.

Let op hoe we een **referentie** naar de functie leggen: **geen ronde haakjes!**

Let ook op: JS is **hoofdlettergevoelig**: `Valueer` zal niet werken

Deze manier om een event handler te koppelen aan een element is de klassieke manier, straks zien we een nieuwere en betere methode.

De syntax:

```
elm.onevent = functie
```

enkele voorbeelden:

```
eKnop.onclick = valueer
```

```
eFormulier.onsubmit = valideer
```

Deze klassieke eventhandlers hebben het voordeel dat ze kort en duidelijk zijn, ze hebben o.a. het nadeel dat je maar naar één functie kunt verwijzen.

Probeer nu de knop uit: je krijg het test bericht te zien.

## De Javascript console

Heb je een *developer tool* actief in je browser? Zoniet, doe dat dan nu, want we zullen de `alert()` vervangen door iets beters.

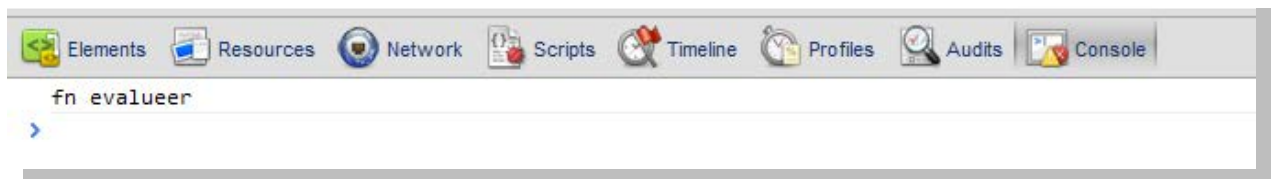
Zorg ervoor dat het **Console** panel van je *developer tool* open staat en actief is: alle developer tools hebben een console paneel. Het console panel toont normaal Javascript foutberichten, maar we kunnen er ook onze eigen berichten in tonen. Zo kunnen we het gebruiken om te **debuggen**: fouten opsporen.

Het voordeel van een console bericht is dat deze het programma niet verstoren en onderbreken zoals een alert doet.

Vervang nu de `alert()` door een `console.log` statement:

```
    }  
    function evaluateer(){  
    /* evalueert of een student geslaagd is of niet */  
        console.log('fn evaluateer');  
    }  
}
```

En probeer de knop opnieuw uit. Je ziet nu



Zo weten we dat de event handler werkt.

Nu vullen we de functie `evaluateer` verder aan met verschillende statements:

```
function evaluateer(){  
    /* evalueert of een student geslaagd is of niet */  
  
    //alert('test')  
    console.log('fn evaluateer')  
  
    // de invulvelden zijn objecten  
    var eWiskunde      = document.getElementById('wiskunde');  
    var eBoekhouden    = document.getElementById('boekhouden');  
    var eInformatica    = document.getElementById('informatica');  
  
    //de punten  
    var nWisk           = eWiskunde.value;  
    var nBoek           = eBoekhouden.value;  
    var nInfr           = eInformatica.value;  
  
}
```

Bespreking:

- Met het sleutelwoord `var` gevolgd door de unieke naam `eWiskunde` maken we een **lokale variabele** aan.
  - Een **variabele** is een stukje geheugen waarin een waarde kan opgeslagen worden.
  - De variabele is **lokaal** (*local scope*), m.a.w. hij is enkel gekend **binnen de functie**.
  - deze variabele bevat een DOM element

- De eerste drie variabelen zijn dus eigenlijk DOM elementen: de input velden
- Een tweede reeks variabelen bevatten de **value** van die velden.  
Dit zijn de waarden die ingevuld worden (of niet) door de gebruiker. We gebruiken de prefix *n* om aan te geven dat het hier getallen zijn
- Wat er precies zal ingevuld worden zijn we niet zeker, maar we *hopen* voorlopig op een correcte input



Tip: gebruik commentaar om een statement */\* tijdelijk te neutraliseren \*/*, wis die niet uit

Om een en ander uit te testen voegen we opnieuw een console-statement in:

```
...
// de invulvelden zijn objecten
var eWiskunde      = document.getElementById('wiskunde');
var eBoekhouden    = document.getElementById('boekhouden');
var eInformatica    = document.getElementById('informatica');

//de punten
var nWisk           = eWiskunde.value;
var nBoek           = eBoekhouden.value;
var nInfr           = eInformatica.value;

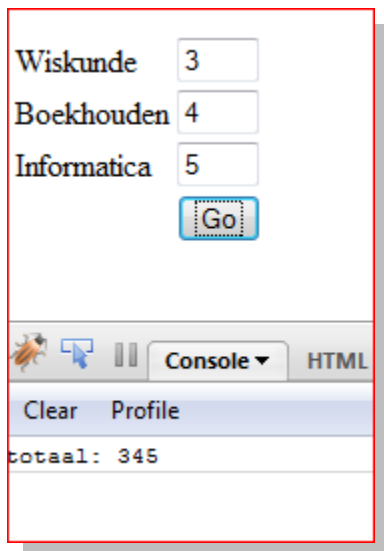
var nTotl           = nWisk + nBoek + nInfr;
var nGmid           = nTotl/3;

console.log("totaal: " + nTotl);
...
```

Probeer nu je functie uit door iets in te vullen en op de knop Go te klikken.

Het resultaat kan verrassend zijn:





Waarom worden de getallen **na** elkaar gezet en niet opgeteld?

- JS variabelen zijn **niet getypeerd**, dat wil zeggen dat er geen **data type** aan toegewezen wordt tijdens de declaratie.  
Je zegt gewoon `var nWiskunde`, maar daarmee bevat het nog geen getal (**number**)...
- De waarde van een invulveld wordt **altijd als tekst (string)** gelezen
- Het `+` teken in JS wordt ook gebruikt als **concatenation operator**:  
dat is een operator die twee stukken tekst aan elkaar naait,  
zoals in `"hello" + " oostende"`

Dus ziet JS er geen probleem in om deze twee stukjes tekst gewoon te *concateneren* en in de `var nTotl` te plaatsen.

Moest je hetzelfde doen met de vermenigvuldiging of de deling, dan zou dat probleem zich niet voordoen: JS kan een vermenigvuldiging niet anders uitvoeren dan met twee getallen en probeert een **impliciete conversie** van een **String** naar een **Number**: je krijgt een correct resultaat.

*Hoe lossen wij ons probleem op?*

We zijn verplicht ervoor te zorgen dat onze variabelen van het **Number** data type zijn. We passen de conversiefunctie `parseInt()` toe op de inhoud.  
`parseInt()` converteert de input naar een *Integer Number*: een geheel getal

```
...
//de punten
var nWisk       = parseInt(eWiskunde.value);
var nBoek       = parseInt(eBoekhouden.value);
var nInfr       = parseInt(eInformatica.value);

var nTotl       = nWisk + nBoek + nInfr;
console.log('totaal: ' + nTotl);
...
```

Probeer opnieuw: nu telt hij de getallen correct op.

**Validatie** van de input is in feite een absolute noodzaak, maar dat stellen we even uit tot later.

We gaan verder met de opdracht.

*Voorwaarde 1: de student is geslaagd als hij minstens 6/10 haalt voor wiskunde.*

Hiervoor maken we gebruik van een **selectie** structuur: **if**.

```
...
    var nTotl          = nWisk + nBoek + nInfr;
    console.log('totaal: ' + nTotl);

    // de evaluatie
    var sBericht = "";
    if( nWisk >= 6 ) {
        sBericht = "U bent geslaagd";
    }
    else {
        sBericht = "U bent NIET geslaagd";
    }
    console.log(sBericht);
...
```

Bespreking:

- we initialiseren de variabele *sBericht*
- een **if** structuur bevat
  - een **test** : de ronde haakjes ( ) waarin een
  - **voorwaarde** staat: *wiskunde* >= 6
- als deze voorwaarde **true** evalueert, wordt
- de **block statement** - alles in de { } – uitgevoerd
- indien de test **false** evalueert, gebeurt er **niets** en gaat het programma door met het volgende statement
- de **else** bevat een block statement dat uitgevoerd wordt als de test **false** evalueert. Dit deel is niet verplicht

Probeer dit uit..

**Voorwaarde 2:**

*de student is geslaagd als hij minstens 6/10 haalt voor wiskunde en minstens 12/20 voor boekhouden en informatica samen.*

Dit zijn twee voorwaarden waaraan voldaan moet worden. We wijzigen de code

```
if(nWisk>=6 && nBoek + nInfr>=12){
    sBericht = "U bent geslaagd";
}
else {
    sBericht = "U bent NIET geslaagd";
}
console.log(sBericht);
```

- bemerk dat er nu sprake is van twee tests die gecombineerd worden met een **logische EN**. In Javascript gebruiken we daarvoor de **&&** operator

### Voorwaarde 3:

*de student is geslaagd als hij minstens 6/10 haalt voor wiskunde en minstens 12/20 voor boekhouden en informatica samen. Als aan die voorwaarde niet voldaan is, maar hij scoort een 10 voor informatica, is hij toch geslaagd.*

Wijzig de code:

```
if((nWisk>=6 && nBoek + nInfr>=12) || nInfr==10 ){
    sBericht = "U bent geslaagd";
}
else {
    sBericht = "U bent NIET geslaagd";
}
console.log(sBericht);
```

- voorwaarde 2 heeft nu twee mogelijkheden: we combineren die met een **logische OF**. In Javascript is dat de **||** operator
- Hier is een overkoepelend paar haakjes nodig

### Voorwaarde 4:

*Bovenop de vorige voorwaarde definiëren we: als hij een gemiddelde heeft hoger dan 7 heeft hij 'onderscheiding' anders 'voldoende'.*

Daarvoor berekenen we beter vooraf het gemiddelde.

```
var nTotl      = nWisk + nBoek + nInfr;
var nGmid      = nTotl/3;
//console.log('totaal: ' + nTotl + '; gemiddelde:' + nGmid);

// de evaluatie
var sBericht = "";
if((nWisk>=6 && nBoek + nInfr>=12) || nInfr==10 ){
    sBericht = "U bent geslaagd";
    if(nGmid >= 7){
        sBericht += " met onderscheiding";
    }
    else{
        sBericht += " met voldoende";
    }
}
```

```
    }  
  }  
  else {  
    sBericht = "U bent NIET geslaagd";  
  }  
  console.log(sBericht);  
}
```

Bespreking:

- de **+=** operator **concateneert** (*duur* woord voor aan elkaar naaien) de string variabelen.

`bericht += " met voldoening"` is hetzelfde als  
`bericht = bericht + " met voldoening";`

- hier werd een **if** structuur **genest** in het eerste deel van een andere **if**. Let er goed op dat alle block statements op de juiste manier afgesloten werden.

De functie *evalueer* doet nu zijn werk. Het is echter niet de bedoeling dat de gebruiker het eindresultaat moet lezen in de Javascript Console, die is enkel bedoeld om fouten op te sporen. Daarom moeten we het bericht nu op een *normale* manier doen verschijnen.

Voeg in de HTML onder de tabel een **div** element toe:

```
...  
</tbody>  
</table>  
<div id="output"></div>  
<script>  
...  
</script>  
</body>  
</html>
```

Hierin zullen we het bericht plaatsen.

Commentarieer alle `console.log` statements en vervolledig het script:

```
...  
//console.log(bericht);  
  
//output naar div  
var eOutput      = document.querySelector('#output');  
eOutput.innerHTML = sBericht;
```

Bespreking:

- opnieuw maken we via de **id** van het **div** element een referentievariabele
- en plaatsen de tekst er in met de method **innerHTML**. Het feit dat die de inhoud telkens overschrijft, speelt hier in ons voordeel omdat we dan telkens nieuwe getallen kunnen evalueren.

Wiskunde

6

Boekhouden

6

Informatica

7

Go

U bent geslaagd met voldoening

**Taken:**

Maak nu de volgende taken:

- Deling
- Kindergeld

## 2.1.3 Getallenreeksen

### Doel

Kennismaking met de verschillende **iteratie** structuren. Eenvoudige validatie van getallen.

### Theorie

Lees de volgende theorie topics na:

- syntax
- data types, variabelen, scope
- NaN, isNaN
- arrays
- structuren: for, while

### Duurtijd:

30 min

### Basisbestand

*getallenreeksen.html*.

Bekijk de broncode van het basisbestand. We hebben je wat werk bespaard en al heel wat ingevuld.

```
<script>

window.onload = function(){

    //=====variabelen=====
    var eOutput = document.getElementById('output');
    var eKnop    = document.getElementById('deKnop');
    var eGetal   = document.getElementById('getal');

    //=====event handlers=====

    eKnop.onclick = lussenMaar;

} //einde window.onload

//=====functies=====

function lussenMaar(){
    /* Testfunctie voor iteraties */
    console.log(eGetal.value);
}

</script>
```

Bemerk het volgende:

- dit is een HTML5 bestand, maar dat maakt weinig uit voor de Javascript die we hier zullen schrijven.
- de `script` tag staat opnieuw in de `head` tag. Een `window.onload` is dus opnieuw nodig.
- Eerst maken we de elementvariabelen `eOutput`, `eKnop` en `eGetal`.
- een `click` event handler voor `eKnop` verwijst naar een functie `lussenMaar()`. Bemerk dat deze functie **niet** in de `window.onload` staat
- de functie `lussenMaar()` bevat een `console.log` statement voor de `value` van `eGetal`

Zorg ervoor dat je de Javascript console kan zien in de *developer tool* van je browser.

### Scope probleem

Als we nu onmiddellijk op de knop 'Go' klikken merken we dat er een probleem is: de Javascript console toont een *Reference* fout: `eGetal` is niet gekend.

**Oorzaak:** `eGetal` is een variabele die gedeclareerd werd in de `window.onload` en dus beperkt is tot de *scope* van deze functie. De functie `lussenMaar` bevindt zich niet **binnen** de `window.onload` en kent deze variabele dus niet.

**Oplossing(en):** er zijn een aantal mogelijke oplossingen voor dit *scope* probleem, de een al beter dan de ander:

1. **niet zo goed:** gebruik **globale variabelen**:

declareer de variabele `eGetal` **vóór** en **buiten** de `window.onload` en ken hem zijn waarde toe **binnen** de `onload`.

Door zijn declaratie wordt `eGetal` **globaal** en zal gekend zijn door alle functies. Dus zo:

```
<script>

var eGetal; //global var
window.onload = function(){

    //=====variabelen=====
    eGetal = document.getElementById('getal');
    ...
}
```

*dit lijkt gemakkelijk maar is de minst goede oplossing:*

- *bedenk dat je dit zult moeten doen voor alle variabelen die de je nodig hebt in zelfstandige functies, dus minstens ook voor `eOutput`.*
- *globale variabelen veroorzaken conflicten met andere scripts*

Helemaal zonder globale variabelen kunnen we momenteel niet, daarvoor kennen we nog onvoldoende Javascript.

2. **meestal OK:** zet de functie `lussenMaar` **binnen** de `window.onload` functie. Doordat de functie dan binnen de scope van de `window.onload` valt, zal hij

de variabele kennen.

*deze oplossing is aanvaardbaar voor een klein script, maar:*

- *als je veel eventhandler functies hebt is het niet langer werkbaar*
- *de functie is niet langer zelfstandig en niet bereikbaar voor andere functies tenzij die ook in de `window.onload` zitten*

3. **goed**: geef **argumenten** aan de functie door:

als we het getal dat ingevuld wordt, doorgeven aan de functie, kan die deze verwerken. Het vergt echter een verandering aan de `onclick` event handler. Omdat we nu verwijzen naar de functie `lussenMaar` en geen haakjes gebruiken, kunnen we geen argument doorgeven.

wijzig het script als volgt:

```
<script>

window.onload = function(){

    //=====variabelen=====
    var eOutput = document.getElementById('output');
    var eKnop    = document.getElementById('deKnop');
    var eGetal  = document.getElementById('getal');

    //=====event handlers=====

    eKnop.onclick = function(){
        var nGetal = eGetal.value;
        eOutput.innerHTML = lussenMaar(nGetal);
    };

} //einde window.onload

//=====functies=====

function lussenMaar(n){
    /*      Testfunctie voor iteraties : FOR
           @n      getal
    */
    console.log(n);
}

</script>
```

Bespreking:

- om een argument door te geven aan een event handler functie moet je deze vervatten in een **anonieme functie**



- door **geen gebruik te maken van globale variabelen** en wel **argumenten** te gebruiken, behoudt de functie *lussenMaar* zijn onafhankelijkheid en kan hij door andere scripts ook gebruikt worden
- de waarde van het invulveld wordt ingelezen in de var *nGetal* op het moment van de klik. Deze waarde wordt doorgegeven aan de functie *lussenMaar* als **argument**
- de functie *lussenMaar* geeft een **return** waarde: die wordt in de **innerHTML** van het output element geplaatst

Nu moeten we functie *lussenMaar* invullen.

### For lus

Met een **for** lus **itereren** we een bepaald aantal maal door een *block* statement. Het aantal iteraties is vooraf gekend. We testen even uit:

```
window.onload = function(){

    //=====variabelen=====
    var eOutput = document.getElementById('output');
    var eKnop    = document.getElementById('deKnop');
    var eGetal   = document.getElementById('getal');

    //=====event handlers=====
    eKnop.onclick = function(){
        var nGetal = eGetal.value;
        eOutput.innerHTML = lussenMaar(nGetal)
    };
} //einde window.onload

//=====functies=====
function lussenMaar(n){
    /*    Testfunctie voor iteraties
        @n    getal, verplicht, max aantal iteraties
    */
    console.log(n);
    var sTekst = "";
    for(var i=0; i< n ; i++){
        sTekst += i + "&nbsp;";
    }
    return sTekst;
}
```

Vul een getal in in het inputveld en klik 'Go'.

Bespreking:

- merk op dat in de commentaar van de functie we uitleg geven over de argumenten van de functie: we zeggen wat hun verondersteld datatype is , of ze optioneel zijn of niet en wat ze doen

- de **for** statement lust zoveel maal tot de voorwaarde niet meer vervuld is. De statements binnen het block van de **for** lus worden dus evenveel maal uitgevoerd.  
Hier *concateneren* we telkens de variabele **i** + een **spatie**, zo bevat de **returnwaarde** een reeks getallen telkens gescheiden met een spatie
- de haakjes van de **for** lus bevat 3 delen:
  - **initialisatie** van een ingebouwde tellervariabele, **i**. je kan die var noemen zoals je wil: **'teller'** of **'j'**, **'x'** of **'y'**. De initiële waarde is 0
  - die ingebouwde teller **i** hebben we met een **var** gedeclareerd in de **for** structuur. Dat is echter geen garantie dat **i** enkel binnen de **for** lus gekend is, Javascript kent enkel *function scope*, geen structuur scope
  - in het middengedeelte van de lus staat de **voorwaarde**: zolang **i** kleiner dan **n** blijft
  - in het laatste deel verhogen of verlagen we **i**. De syntax **i++** betekent een verhoging met **1**, dus zoveel als **i = i + 1**
- de functie eindigt met een **return** statement gevolgd door de waarde die de functie teruggeeft aan de oproeper.  
Elke code na de **return** wordt niet meer uitgevoerd

Als je bijvoorbeeld het getal 5 ingevuld hebt, krijg je de getallen 0 1 2 3 4.  
Vijf getallen dus.

## Validatie van getallen

Deze maal willen we zeker zijn dat er een getal ingevuld wordt. Als dat niet het geval is, tonen we een foutbericht.

Voor we de waarde aan de functie doorgeven doen we een controle, in de **onclick** dus:

```
eKnop.onclick = function(){
    var nGetal = eGetal.value;
    if(nGetal==""||isNaN(nGetal)){
        alert(' Deze functie werkt enkel met getallen');
    }
    else{
        eOutput.innerHTML = lussenMaar(parseInt(nGetal))
    }
};
```

Bespreking:

- de waarde van een invulveld is ALTIJD een **String**, wat je ook invult
- Javascript zet waarden die **op getallen lijken**, zoals "3", **impliciet** (automatisch) om naar een getal als hij dat nodig acht, bv. bij een vermenigvuldiging. Maar is de bewerking niet eenduidig (bijvoorbeeld bij een +) en dan gebeuren zaken die je niet verwacht : concatenatie.

- we controleren die waarde van het invulveld eerst op een lege string OF met de functie `isNaN()`.
  - `isNaN()` (*is Not A Number*) retournt `true` als een waarde onmogelijk in een getal kan worden omgezet en `false` als dat wel mogelijk is.  
`isNaN()` heeft een schoonheidsfoutje: de functie reageert niet op een lege string, daarom de eerste test
- als de waarde niet `NaN` is, dan doen we verder: we geven de waarde door aan de functie maar zetten hem zelf om d.m.v. `parseInt()`. Deze functie zet de waarde om naar een echt number, een integer = geheel getal.

Probeer dit even uit.

Een `for` lus kan ook naar beneden tellen. Probeer even:

```
for(var i=n; i>0 ; i--){  
  ...  
}
```

Deze lus start met het inputgetal en eindigt met 1.

`for` lussen kunnen gecombineerd worden. Keer even terug naar de stijgende vorm van de lus en wijzig verder:

```
function lussenMaar(n){  
  /*    Testfunctie voor iteraties : FOR  
        @n    getal, verplicht, max aantal iteraties  
  */  
  //console.log(n);  
  var sTekst = "";  
  for(var i=1; i<= n ; i++){  
    for(var j=1;j<=n;j++){  
      sTekst += i*j + "&nbsp;";  
    }  
    sTekst += "<br>";  
  }  
  
  return sTekst;  
}
```

Dit levert de tafel van vermenigvuldiging op in evenveel rijen als kolommen.

Bespreking:

- voor elke 'buitenlus' met teller `i` voeren we een 'binnenlus', met teller `j` uit en schrijven we ook een `br` element
- de 'binnenlus' vermenigvuldigt beide tellers, die bereikbaar zijn voor elkaar, en plakt er een spatie achter zodat er wat tussenruimte tussen de getallen komt

Elke combinatie van `for` lussen is mogelijk.

## while lus

Met een **while** structuur is het aantal lussen onbepaald. Dat gebruik je dus best als je het aantal niet weet, maar als aan een voorwaarde voldaan moet worden.

De voorwaarde wordt eerst geëvalueerd op **true/false** voor de uitvoering van de lus.

Als die voorwaarde **true** blijft kan de lus oneindig voortduren.

Kopieer de functie *lussenMaar* en commentarieer het origineel. Pas in de kopie de code aan:

```
...  
var i = 1;  
while (i<=n){  
    sTekst += i + "&nbsp;";  
    i++;  
}  
...
```

Bespreking:

- een **while** lus heeft geen ingebouwde teller. Daarom maken we onze eigen teller vooraf aan, ook **i** genaamd, en initialiseren die op **0**
- een **while** lus is eenvoudiger: er is slechts de voorwaarde waaraan voldaan moet zijn, zolang duurt de lus
- we moeten zelf de teller verhogen: **i++**

Ook de dubbele lus kunnen we hiermee uitvoeren:

```
function lussenMaar(n){  
    /*    Testfunctie voor iteraties : DUBBELE WHILE  
        @n    getal, verplicht, max aantal iteraties  
    */  
    console.log(n);  
    var sTekst = "";  
    var i = 1;  
    while(i<=n){  
        var j = 1;  
        while(j<=n){  
            sTekst += i*j + "&nbsp;";  
            j++;  
        }  
        sTekst += "<br>";  
        i++;  
    }  
    return sTekst;  
}
```

Bespreking:

- let er op dat de teller **j** bij de start van elke 'buitenlus' (**i**) opnieuw geïnitieerd moet worden

Een **while** lus is echter vooral interessant **als het aantal iteraties onbekend is**.

Dit is bijvoorbeeld handig met een **Array** (een tabelvariabele).

Verander de event handler van de **onclick** naar:

```
eKnop.onclick = function(){
    eOutput.innerHTML = arrayLus();
};
```

en maak de functie *arrayLus*:

```
function arrayLus(){
    /*    Testfunctie voor iteraties : WHILE met array
        @n    getal, verplicht, max aantal iteraties
    */
    var aGetallen = [12,2,4,8,5];

    var sTekst = "";
    var i = 0;
    while(i<aGetallen.length){
        sTekst += aGetallen[i] + "&nbsp;";
        i++;
    }
    console.log(sTekst);
    return sTekst;
}
```

Bespreking:

- de tabelvariabele *aGetallen* is een letterlijk **Array** , geschreven met vierkante haakjes `[ ]` en gescheiden door komma's. Later meer over arrays.
- de teller *i* wordt geïnitieerd op 0
- zolang de teller kleiner blijft dan het aantal items in getallen – deze wordt gegeven door de **length** property – blijft de lus doorgaan
- we tonen elk item in *getallen* met zijn index:  
`getallen[0]` is het eerste item en `getallen[4]` is het laatste item.  
Om die reden moet de teller met **0** beginnen en stoppen voor de **5** bereikt is want `getallen[5]` is in feite een zesde item. Zoveel items zijn er echter niet, dus zouden we een fout krijgen.

Omdat arrays dikwijls gebruikt worden als een soort 'databank' en het aantal items onzeker is of wijzigt, is de **while** lus ideaal om er doorheen te lussen.



Pas op voor een **oneindige lus**!

Als je in vorig voorbeeld de statement **i++** vergeet, blijft de voorwaarde **true** en blijft de **while** **oneindig** voortlussen.

De enige manier om eruit te raken is de browser afsluiten... en als je dan je bestand niet

opgeslagen hebt ben je de klos...

**Taken:**

Maak nu de volgende taken:

- Faculteit

## 2.1.4 Image gallery op drie manieren

### Doel

In dit project maken we in verschillende stappen drie verschillende versies van een eenvoudige "moderne kunst" *image gallery*. Onderweg leren we hoe met DOM scripting elementen op te bouwen en af te breken. We maken ook kennis met events en leren een array gebruiken.

### Theorie

Lees zeker de volgende theorie topics na:

- DOM methods en properties
- lusstructuren
- events, `onload`, `onmouseover`, `onclick`, `onchange`
- anonieme functie
- arrays
- feature sensing, object detection

### Duurtijd

2 uur

### Basisbestanden

*image\_gallery.html*, *image\_gallery.css*, *gallery\_images.txt* + art images

Bekijk de broncode van het basisbestand *image\_gallery.html*: het bevat een aantal hyperlinks en er is ook een stylesheet aan gekoppeld.

Controleer eerst of de beelden bij de oefenbestanden bereikbaar zijn volgens het pad van de hyperlinks. Is dat niet zo, maak dan wijzigingen.

```
...  
<li><a href="art/gerard_richter1.jpg" title="Gerard Richter">Gerard Richter</a></li>  
...
```

Als je nu op een hyperlink klikt, wordt het beeld geopend in een aparte pagina: je bent verplicht om telkens terug te keren naar de vorige pagina, dat is niet erg gebruiksvriendelijk.

### eerste versie: gewoon klikken

In een eerste versie willen we het beeld in dezelfde pagina bekijken. We hebben reeds plaats voorzien aan de rechterzijde, in de kolom `#kader`. We plaatsten eerst een tijdelijk beeld in deze kolom, een zogenaamde 'plaatshouder'.

Voeg de volgende `img` tag toe aan `div#kader`:

```
...  
<div id="kader">  
  
</div>  
...
```

Nu hebben we een vaste plek waar de beelden getoond zullen worden: `img#plaatshouder`. Dit beeld (600 X 600px) neemt zowel in de breedte als in de hoogte maximaal plaatst in: de foto's die we zullen tonen zijn nooit breder of hoger.

We zetten onze eerste scriptversie in een extern javascript bestand: maak `image_gallery_versie1.js` en koppel dat via een `script` tag in de webpagina:

```
...  
<link href="image_gallery.css" rel="stylesheet" media="all" />  
<script src="image_gallery_versie1.js"></script>  
</head>  
...
```

Wat moet er gebeuren?

- het basisbeeld moet vervangen worden door de aangeklikte foto
- de hyperlink mag zijn standaardactie – het openen van het beeld in het volledige venster – niet meer uitvoeren

We beginnen met een `window.onload` event handler in onze javascript file:

```
// Image_gallery_versie1.js  
// een Javascript_PF project  
  
window.onload = function () {  
    var eImg          = document.getElementById('plaatshouder');  
    //nieuwe eventhandler voor alle hyperlinks in de menubalk  
    var eSidebar      = document.querySelector(aside);  
    var eLinks        = eSidebar.getElementsByTagName('a');  
    console.log('sidebarLinks %s', eLinks.length);  
  
    for(var i=0;i<eLinks.length;i++){  
        eLinks[i].addEventListener('click',function (e){  
            e.preventDefault();  
            toonFoto(this, eImg);  
        })  
    }  
}
```

Bespreking:

- het `window.onload` event wacht tot de *document tree* geladen is



- we maken een referentievariabele naar de 'plaatshouder' `img` met de klassieke `getElementById`
- we maken een referentievariabele naar de `aside` met `querySelector`. Het resultaat geeft geen wezenlijk verschil met de vorige method: het eerste `aside` element dat gevonden wordt, is hier ook het enige aanwezig
- vanuit de variabele `eSidebar`, maken we een *collection* van hyperlinks met `getElementsByTagName('a')`.

We merken nog eens op dat dat in tegenstelling tot `document.getElementById()`, `getElementsByTagName()` kan toegepast worden vanuit/op eender welke DOM element, dus ook `eSidebar.getElementsByTagName()`

- dit resulteert in een *collection* DOM elementen, waar we via `console.log` even de `length` property van opvragen, om te zien of hun aantal klopt. Commentarieer dit statement straks
- Daarna lussen we doorheen alle items van de collectie en stellen een nieuwe event handler in voor het `click` event: een anonieme functie die twee zaken doet:
  - Om de standaard actie van een *event* uit te schakelen gebruiken we de method `e.preventDefault()`, waarbij `e` het event zelf is. Je kan het dus ook voor andere events gebruiken, niet enkel op een `click`.
  - het `this` object doorstuurt naar een functie `toonFoto`. Het `this` object in een eventhandler stelt het object voor waarop de eventhandler plaatsheeft, hier is dat de hyperlink. We geven dus een referentie naar de hyperlink door aan de functie

We maken nu eerst even een dummy versie van `toonFoto` om te kunnen testen. Plaats deze functie buiten de `window.onload`:

```
function toonFoto(eLink,eImg){
  /* wisselt de bron van het src attribuut van de img#beeld
  @ eLink, een hyperlink element
  @ eImg, plaatshouder img
  */

  console.log(eLink.href);
}
```

Test uit: als je op een hyperlink klikt zie je zijn href attribuut in de JS console, maar de link toont geen foto meer. Probeer alle hyperlinks.

## Een meer efficiënte selectie

We brengen een verbetering aan in de manier waarop we de hyperlinks collectie maken. Vervang de volgende twee lijnen code door deze nieuwe:

```
window.onload = function () {

    //nieuwe eventhandler voor alle hyperlinks in de menubalk
    //var eSidebar = document.getElementById(aside);
    //var eLinks   = eSidebar.getElementsByTagName('a');
    eLinks        = document.querySelectorAll(aside a');
    console.log('sidebarLinks %s', eLinks.length);

    for(var i=0;i<eLinks.length;i++){
        eLinks[i].addEventListener('click',function (e){
            e.preventDefault();
            toonFoto(this);
        })
    }
}
```

Bespreking:

- de method `querySelectorAll` is veel korter (en sneller) in het maken van contextuele selecties: de selector `aside a` selecteert alle `a` elementen in alle `aside` elementen in de pagina.  
Omdat er geen ander `aside` element in de pagina is kunnen we dit doen, anders zouden we gericht moeten werken

## Een andere foto tonen

De functie `toonFoto()` vullen we nu verder in:

```
function toonFoto(eLink, eImg){
    /* wisselt de bron van het src attribuut van de img#beeld
    @ eLink, een hyperlink element
    @ eImg, plaatshouder img
    */

    //console.log(eLink.href);
    eImg.src = eLink.href;
}
```

Bespreking:

- de functie heeft twee argumenten:
  - `eLink` waarin het `this` object terechtkomt: een `a` element dus
  - `eImg`, een `img` element: de vaste plaatshouder van images op de pagina
- de werking is zeer simpel: het `src` attribuut van `eImg` krijgt de waarde van het `href` attribuut van de hyperlink. Dit is voldoende om bij een klik een andere foto te tonen.

Probeer dit even uit.

## Extra informatie tonen

We willen nu ook extra informatie tonen bij elke foto. Die halen we uit het **title** attribuut van elke hyperlink. Deze tekst plaatsen we in een dynamisch aangemaakt **p** element dat we eerst onder de foto willen plaatsen:

```
function toonFoto(eLink, eImg){
  /* wisselt de bron van het src attribuut van de img#beeld
  @ eLink, een hyperlink element
  @ eImg, plaatshouder img
  */

  //console.log(eLink.href);
  eImg.src          = eLink.href;
  var sInfo         = eLink.getAttribute('title');
  var eInfo         = document.createElement('p');
  eInfo.id          = "info";
  eInfo.innerHTML   = sInfo;
  eImg.parentNode.appendChild(eInfo);
}
```

Bespreking:

- in *sInfo* halen we de informatie op uit het **title** attribuut van de hyperlink. De standaard DOM method om een attribuut te lezen is **getAttribute()**:

```
eLink.getAttribute('title')
```

vandaag lukt dat in quasi alle browsers. Niet zo lang geleden was dat niet altijd het geval. Daarom zullen vele auteurs nog steeds de directe manier proberen, als een *property* van het object (die ook niet voor **alle** attributen werkt):

```
eLink.title
```

het resultaat is hetzelfde.

- we maken een **p** element aan met **document.createElement('p')** en plaatsen die in de *var eInfo*
- we geven dit element een **id** attribuut om het later gemakkelijk te kunnen herkennen
- we plaatsen de tekst in het *paragraph* element met **innerHTML**.
- klein probleempje: we kunnen het niet toevoegen aan het **img** element *eImg* zelf: een **img** element is namelijk een EMPTY element en kan dus enkel *attribuutNodes* hebben... Het is dus de bedoeling de info als **sibling** (=zusje/broertje) toe te voegen **na** de **img**.

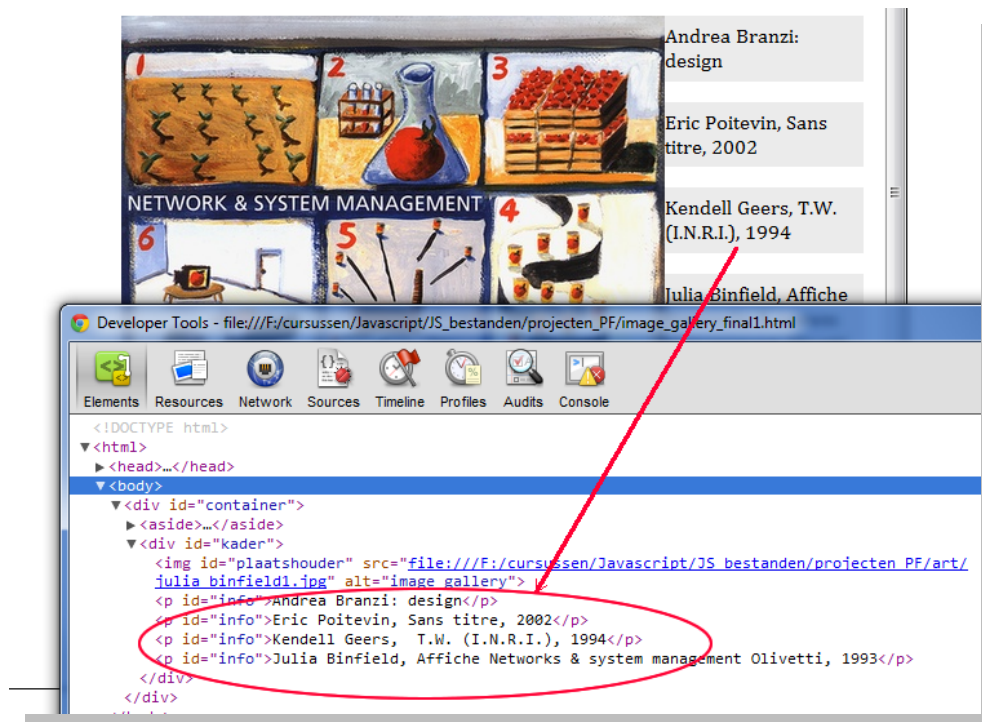
Daarom moeten we eerst de **parentNode** van de **img** refereren en *eInfo* daaraan toevoegen met **appendChild()**. Deze method voegt een element altijd als **laatste** child toe.

Uitproberen. Het werkt maar...

Twee problemen:

1. *de layout is niet altijd goed: de tekst komt soms rechts van het beeld te staan*  
oorzaak: smalle fotos geven rechts ruimte voor tekst; oplossing: CSS
2. *na een aantal kliks blijf je de info van andere beelden meeslepen*  
oorzaak: bij elke klik wordt een nieuw `p` element aangemaakt

Bekijk met je *webdeveloper tool* de DOM tree:



We passen de functie *toonFoto* aan:

```
function toonFoto(eLink, eImg){
  /* wisselt de bron van het src attribuut van de img#beeld
  @ eLink, een hyperlink element
  @ eImg, plaatshouder img
  */

  //console.log(eLink.href);
  eImg.src      = eLink.href;
  var sInfo     = eLink.getAttribute('title');
  var eInfo     = document.getElementById('info');
  if(eInfo){
    //eInfo bestaat reeds
    eInfo.innerHTML = sInfo;
  }
  else {
    var eInfo     = document.createElement('p');
    eInfo.id      = "info";
  }
}
```

```
eInfo.innerHTML      = sInfo;
eImg.parentNode.appendChild(eInfo);
}
}
```

Bespreking:

- eerst maken we een referentie naar een element met `id "info"`
- als deze variabele niet `undefined` is (dit element bestaat dus), wijzigen we gewoon de `innerHTML` met nieuwe tekst, zoniet maken we een nieuw element aan

Probeer uit. Probleem 2 opgelost.

Het layout probleem kan opgelost worden enkel met CSS (`clear` property) , maar we pakken het anders aan: ipv de tekst onder de image te plaatsen, zetten we die liever erboven.

We wijzigen de functie:

```
...
else {
    var eInfo          = document.createElement('p');
    eInfo.id           = "info";
    eInfo.innerHTML     = sInfo;
    //eImg.parentNode.appendChild(eInfo);
    eImg.parentNode.insertBefore(eInfo,eImg.parentNode.firstChild);
}
}
```

Bespreking:

- de `insertBefore()` method voegt een node als *child* toe op een plek ten opzichte van een andere *childNodes*:

```
eImg.parentNode.insertBefore(eInfo, eImg.parentNode.firstChild);
```

- voegt de node *eInfo* toe
- als *child* van `eImg.parentNode`
- vóór de node `eImg.parentNode.firstChild`, dus vóór de eerste *childNodes* van deze *parent*

dus staat *eInfo* nu als eerste *childNodes* en is de `img` een plaatsje opgeschoven

Gebruik `insertBefore()` dus als je niet op de "laatste plaats in de rij" wil invoegen.

Test dit even uit en gebruik je *developer tool* om je te vergewissen van de structuur:

```
<!DOCTYPE html>
<html>
  <head>...</head>
  <body>
    <div id="container">
      <aside>...</aside>
      <div id="kader">
        <p id="info">Kendell Geers, T.W. (I.N.R.I.), 1994</p>
        
      </div>
    </div>
  </body>
</html>
```

Je bemerkt het `p` element als *child* van `div#kader` en als *sibling* van de `img`.

### Versienummer

Als toetje voegen we het versienummer van onze image gallery toe aan de paginatitel. Voeg toe aan het script vóór en in de `window.onload`:

```
var versie = " versie 1.0";

window.onload = function () {

  //versie info
  var eKop      = document.querySelector('h1');
  eKop.innerHTML = eKop.innerHTML + versie;

  var eImg      = document.getElementById('plaatshouder');
  ...
}
```

### Tweede versie: rollover

De image gallery zou nog veel eleganter zijn moesten de beelden wijzigen als de muis over de links beweegt: een "roll-over" effect!

Sla het javascript op als `image_gallery_versie2.js` en wijzig de koppeling in de HTML file naar dit bestand.

Wijzig dan ook het versienummer in het script.

De `click` event handler hebben we nu niet meer nodig, alhoewel die gerust mag blijven staan: het kan geen kwaad.

We voegen wel een nieuwe event handler toe :

```
var versie = " versie 2.0";

window.onload = function () {
```

```
//versie info
var eKop          = document.querySelector('h1');
eKop.innerHTML    = eKop.innerHTML + versie;

var eImg          = document.getElementById('plaatshouder');
//var eSidebar    = document.querySelector('aside');
//var eLinks      = eSidebar.getElementsByTagName('a'); //collection
var eLinks        = document.querySelectorAll('aside a'); //collection
//console.log('eLinks %s',eLinks.length);

for(var i=0;i<eLinks.length;i++){
    eLinks[i].addEventListener('click',function (e){
        e.preventDefault();
        toonFoto(this,eImg);
    })
    eLinks[i].addEventListener('mouseover',function (e){
        toonFoto(this,eImg);
    })
}

}
```

Bespreking:

- met `addEventListener` voegen we een event handler toe voor het `mouseover` event.  
Deze methode krijgt vanaf nu de voorkeur op de 'oude' manier - `onclick` . Wat meer uitleg over deze methode volgt later.
- de `e.preventDefault()` is hier onnodig want een `mouseover` heeft geen default action die ons stokken in de wielen kan steken
- al de rest blijft: dit demonstreert dat functions gemaakt als "black boxes" – dus enkel afhankelijk van hun argumenten – hun voordeel hebben.

Je kan nu reeds uittesten: tweede versie klaar!

### derde versie: met keuzelijst

In deze versie willen we in plaats van hyperlinks in de sidebar een keuzelijst (`select` element). En niet enkel dat, we willen het hele systeem (keuzelijst en image gallery) dynamisch opbouwen vanuit gegevens die ons aangeleverd worden in een `Array`.

Deze versie vraagt naast een aangepast script ook wat verbouwingswerken aan de HTML, sla daarom

- *image\_gallery\_versie2.html* op als *image\_gallery\_versie3.html*
- en ook *image\_gallery\_versie2.js* als *image\_gallery\_versie3.js*.

Je begrijpt dat een kunstgalerij met slechts 5 kunstwerken een beetje magertjes is... Zo zal het zelden gebeuren dat de hyperlinks vast in het document staan, er zullen er meer zijn en ze zullen regelmatig veranderen. De gegevens voor de galerij worden meestal *server-side* gegenereerd vanuit een database en dan doorgegeven aan de *client-side* als een **Array** of **JSON** object.

Open het bestand *gallery\_images.txt*: je bemerkt een **Array**:

```
var aModernArt = [  
  
  ["gerard_richter1.jpg","Gerhard Richter, Juni n° 527, 1983","Gerard Richter"],  
  ["andrea_branzi.jpg","Andrea Branzi: design","Andrea Branzi"],  
  ["eric_poitevin.jpg","Eric Poitevin, Sans titre, 2002","Eric Poitevin"],  
  ["kendell_geers.jpg","Kendell Geers, T.W. (I.N.R.I.), 1994","Kendell Geers"],  
  ["julia_binfield1.jpg","Julia Binfield, Affiche Networks & system management Olivetti,  
    1993","Julia Binfield"]  
  
]
```

Bespreking:

- een **Array** is een tabelvariabele: het bevat meerdere waarden gescheiden door **komma's**. dit array is **letterlijk** aangemaakt: de waarden zijn simpelweg in **vierkant haakjes** `[ ]` geschreven.
- Dit **Array** is twee-dimensioneel: het *aModernArt* bevat 5 elementen die zelf ook nog eens een **Array** zijn: let op de komma's na elk binnenarray:

```
["andrea_branzi.jpg","Andrea Branzi: design","Andrea Branzi"],
```

is *aModernArt[1]*

- het binnenarray bevat respectievelijk het beeldbestand, de info en de waarde die getoond moet worden in hyperlink of keuzelijst. Zo is

```
aModernArt[1][0] de bestandsnaam: "andrea_branzi.jpg" en  
aModernArt[1][1] de info: "Andrea Branzi: design" en  
aModernArt[1][3] de artiest: "Andrea Branzi "
```

In eerste instantie moeten we het invoegen in onze toepassing: zullen we *kopiëren*? Niet echt wenselijk want het bevat enkel gegevens. Vermits het geldige JS statements bevat is de beste manier het bestand op te slaan als een *.js* bestand en te koppelen: Sla op *gallery\_images.txt* als *gallery\_images.js* en voeg dan een koppeling toe in het HTML bestand:

```
<script src="gallery_images.js"></script>  
<script src="image_gallery_versie3.js"></script>  
</head>
```

*Is de volgorde van deze scripts belangrijk?*

Scripts worden in volgorde gelezen, en als in het tweede script iets uitgevoerd wordt waarvoor een item uit het eerste script nodig is, dan is de volgorde van belang.



We testen nu even uit of de koppeling in orde is:

Plaats de statement als eerste lijn in de `window.onload` van het script.

```
// Image_gallery_versie3.js
// een Javascript_PF project
// dropdown uit array versie

var versie = " versie 3.0";

window.onload = function () {
    console.log(aModernArt[0][0]);    //is aModernArt aanwezig?
    ...
}
```

Bij het laden krijg je nu in de Javascript console de bestandsnaam van het eerste kunstwerk te zien. Indien dit correct is, commentarieer de `console.log()`.

Maar het is duidelijk dat alles wat volgt **volledig afhankelijk** is van de **aanwezigheid** van het **Array** `aModernArt`. Dit is té belangrijk om te negeren:

als het array `aModernArt` niet geladen is:

- ☞ kan er vanalles vreselijk fout gaan en op het scherm van de gebruiker terechtkomen: WILLEN WE NIET...☹
- ☞ is dit de schuld van ?
  - de ontwikkelaar, die bijvoorbeeld een fout maakte in de `src` van de `script` tag, in dat geval
    - krijgt de ontwikkelaar een foutbericht, niet de gebruiker, de ontwikkelaar verhelpt de fout
  - de gebruiker: het meest voorkomende probleem is het inactief staan van Javascript, in dat geval
    - krijgt de gebruiker een bericht waarin hij op de hoogte gebracht wordt van het feit dat er iets niet juist is en dat hij dat kan verhelpen

Om die reden moeten we aan *object detection* doen en *graceful degradation*.

## object detection

Eerst controleren we de aanwezigheid van het array. Voeg een `if` structuur toe aan de `window.onload`:

```
window.onload = function () {

    //array geladen?
    if(typeof aModernArt == "undefined"){
        throw new Error("array aModernArt niet gevonden");
    }
    else{
        console.log(aModernArt[0][0]);
    }
}
```

```
//versie info
var eKop          = document.querySelector('h1');
eKop.innerHTML    = eKop.innerHTML + versie;
...
}

}
```

Bespreking:

- de **if** structuur test het type van de variabele *aModernArt*: als dit "undefined" is, dan bestaat de variabele niet, in dat geval
  - werpen we een Javascript **Error**. Dit zal als een fout te zien zijn in de javascript console. Verder gebeurt er niets meer, de ontwikkelaar zal wel de console in het oog houden
- het vervolg van de code komt dus in de **else**

Probeer uit: maak een foute referentie naar *gallery\_images.js* en controleer de Javascript console.

Voor het geval dat Javascript inactief is voorzien we een special tekst die vast in de pagina staat, maar door Javascript verborgen wordt. Je snapt het: geen Javascript, geen verbergen.

Voeg de volgende **div** toe aan de HTML van de pagina:

```
...
<div id="container">
  <div id="noscript">
    <h3>Probleem!</h3>
    Deze website gebruikt Javascript: dit is momenteel afgezet in uw browser. <br>
    Activeer Javascript via het menu van uw browser en ververs de pagina.
  </div>
  <aside>
    <h1>Modern Art Image Gallery</h1>
    <ul>
    ...
```

en nu voegen we volgende code toe aan de **window.onload**:

```
window.onload = function () {
  //noscript verbergen
  var eNoScript = document.getElementById('noscript');
  eNoScript.style.display = "none";

  //array geladen?
  if(typeof aModernArt == "undefined"){
  ...
```

Test dit ook uit: zet Javascript af en weer aan in je browser.

Nu dit opgelost is kunnen we verder doen.

## dynamisch een keuzelijst maken

Bij het laden van de pagina moet de keuzelijst dynamisch aangemaakt worden.

Verwijder eerst alle vaste hyperlinks uit de sidebar:

```
...
<aside>
  <h1>Modern Art Image Gallery</h1>
  <!-- keuzelijst hier -->
</aside>
...
```

Dat gebeurt opnieuw bij het `window.onload` event. Verwijder alle code betreffende de hyperlinks, maar laat de functie `toonFoto` staan.

```
...
if(typeof aModernArt == "undefined"){
    throw new Error("array aModernArt niet gevonden");
}
else{
    //console.log(aModernArt[0][0]);
    //versie info
    var eKop          = document.querySelector('h1');
    eKop.innerHTML    = eKop.innerHTML + versie;
    //plaatshouder
    var eImg          = document.getElementById('plaatshouder');
    //dynamische keuzelijst
    var eKeuzelijst    = maakKeuzelijst(aModernArt);
    var eSidebar       = document.querySelector('aside');
    eSidebar.appendChild(eKeuzelijst);
}
} //einde window.onload
```

Bespreking:

- de variabele `eKeuzelijst` zal het `select` element bevatten dat zal aangeleverd worden door de functie `maakKeuzelijst`.  
We geven die als argument het array mee
- `eSidebar` is een referentie naar het `aside` element
- waaraan we de `select` toevoegen als laatste child

Een keuzelijst is een `select` element met een aantal `option` elementen. Als een gebruiker iets kiest uit de lijst wordt de relevante foto + info getoond.

Maak nu de functie `maakKeuzelijst()`:

```
function maakKeuzelijst(a){
    /*
```

```
return SELECT element
@a array van images
*/
var nArt          = a.length;
var eSelect       = document.createElement('select');
eSelect.id        = "keuzelijst";
//standaard option element
var eOption       = document.createElement('option');
eOption.innerHTML = "Maak een keuze";
eOption.setAttribute("value", "");
eSelect.appendChild(eOption);
//andere option elementen met artiesten
for(var i=0;i<nArt;i++){
    var eOption    = document.createElement('option');
    eOption.innerHTML = a[i][2];
    eOption.value   = i;
    eSelect.appendChild(eOption);
}

return eSelect;
}
```

Bespreking:

- eerst tellen we het aantal element in het array *a*
- in *eSelect* maken we een DOM **select** element
- we geven het een **id** om het later te kunnen bereiken
- we maken een standaard **option** element aan met de tekst "*Maak een keuze*" en voegen dat onmiddellijk toe aan het **select** element en de **value** "" (lege string)
- we lussen doorheen het array *a* (*aModernArt*):
  - we maken telkens een nieuw **option** element aan
  - we schrijven het derde item van het subarray als **innerHTML** in dit option
  - en geven het een **value** met de waarde van de lusvariabele *i*
  - voegen het **option** element aan de *eSelect* toe
- de functie returned de **select**

Even uittesten: je krijgt nu een 'dropdown' met 5 keuzes, allemaal '*testkeuze*'.

Ze reageren nog wel niet. Daarvoor hebben we een eventhandler nodig. Die voegen we toe aan de **window.onload**:

```
...
eSidebar.appendChild(eKeuzelijst);
eKeuzelijst.addEventListener("change",function(e){
    var waarde = this.value;
    console.log(waarde);
    if(waarde!="" && waarde!=null){
```

```
        toonFoto(waarde,eImg)
    }

    });

}

} //einde window.onload
```

Bespreking:

- met `addEventListener` maken we een event handler voor het `change` event
- de waarde die geselecteerd wordt in de lijst kunnen we weten met `this.value`, `this` zijnde de `select`.  
Het `console` statement toont ons dat
- als die waarde geen lege string is of ontbreekt, roepen we de functie `toonFoto` aan
- merk op dat `toonFoto` gewijzigd is: het eerste argument is nu de *waarde* die we lazen

De functie `toonFoto` moet dus ook enkele veranderingen ondergaan:

```
function toonFoto(nIndex, eImg){
/* wisselt de bron van het src attribuut van de img#beeld
@ nIndex, een hyperlink element
@ eImg, plaatshouder img
@ aModernArt array, global
*/

    aArt    = aModernArt[nIndex]; //subarray
    sPad    = aArt[0];           //source
    sInfo   = aArt[1];           //info
    sNaam   = aArt[2];           //naam

    eImg.src    = "art/" + sPad;
    var eInfo   = document.getElementById('info');

    if(eInfo){
        //wijzig info
        eInfo.innerHTML    = sInfo;
    }
    else {
        //maak nieuwe p#info aan
        var eInfo          = document.createElement('p');
        eInfo.id            = "info";
        eInfo.innerHTML     = sInfo;
        eImg.parentNode.insertBefore(eInfo,eImg.parentNode.firstChild);
    }
}
```

Bespreking:

- het eerste argument van de functie, *nIndex*, is nu een getal

- waarmee we onmiddellijk uit de globale variabele *aModernArt*, het zoveelste element afzonderen in *aArt*
- waaruit we respectievelijk de source, de info en de naam van het kunstwerk lezen
- opnieuw stellen we het *src* attrib van *eImg* in, deze keer moeten we echter het pad vervolledigen (vroeger zat dat vervat in de hyperlinks)
- de rest van de functie blijft gelijk

Test uit.

**Taken:**

maak nu

- Feature sensing
- Toon-verberg figuren

## 2.1.5 Arrays en objecten

### Doel

De bedoeling van dit project is je te leren werken met de verschillende soorten "tabelvariabelen": **Array** en **Object**.

Als je een verzameling gegevens wil opslaan of doorgeven, wanneer gebruik je best een **Array** en wanneer een **Object**? Hoe behandel je ze?

Terzelfdertijd leren we ook omgaan met event handlers: hoe laat je een knop iets uitvoeren? wat zijn de mogelijke problemen?

### Theorie

Lees zeker de volgende theorie topics na:

- data types: eenvoudige types, objecten, arrays
- variabelen, scope
- gezonde programmeerprincipes in Javascript
- DOM methods
- event handlers

### Duurtijd

3 uur

### Basisbestand

*personeel.html*

### Een personeelsbestandje

We maken gebruik van ingeladen gegevens om een personeelsbestand weer te geven. Daarnaast gebruiken we een formulier om nieuwe gegevens aan het bestand toe te voegen.

Open het basisbestand en bekijk de aanwezige Javascript code en ook de HTML van het formulier.

## JS PF project: Personeel

Dit project leert je omgaan met array en object variabelen die gebruikt worden om (tijdelijk) gegevens in op te slaan. Deze pagina bevat een kleine databank van personen waar we aan kunnen toevoegen en die we kunnen tonen.

### Opdracht:

*Om een nieuw personeelslid toe te voegen, vul de nodige gegevens in*

\* verplicht veld

naam \*

leeftijd \*

functie \*

sexe \* man ☐ vrouw ☐

gehuwd? ☐

kind1

kind2

kind3

3 personeelsleden in de databank

In de `script` tag van het startbestand vinden we twee globale variabelen en een `window.onload` handler.

Deze laatste bevat een aantal DOM referentievariabelen:

```
window.onload = function(){

    //=====DOM REFERENTIES=====

    //knoppen
    var eToevoegen      = document.getElementById('toevoegen');
    var eMaakLijst      = document.getElementById('maakLijst');
    //invulvelden, keuzelijsten, etc...
    var eNaam           = document.getElementById('naam');
    var eFunctie        = document.getElementById('functie');
    var eSexe           = document.frmPersoneelslid.sexe;
    var eLeeftijd       = document.getElementById('leeftijd');
    var eGehuwd         = document.getElementById('gehuwd');
    var eKind1          = document.getElementById('kind1');
    var eKind2          = document.getElementById('kind2');
    var eKind3          = document.getElementById('kind3');
```



```
//andere
var eOutput      = document.getElementById('output');
var eTeller      = document.getElementById('teller');
...
```

Merk op dat de control `eSexe` op de oude DOM0 manier gerefereerd wordt: via de name van het formulier. Dit is perfect geldig voor formulier *controls*.

## De gegevens:

Erboven vinden een tabelvariabele *aFuncties*:

```
var aFuncties = ["instructeur","bediende","manager","arbeider"];
```

een eenvoudig **Array**.

en de tabelvariabele *aoPersoneel*:

```
var aoPersoneel = [
  {
    id:4678,
    naam:"Roger Mary",
    functie:"instructeur",
    leeftijd: 65,
    sexe:"m",
    gehuwd:true,
    kinderen:[
      {naam:"Liesbeth",
        leeftijd: 26,
        sexe:"v"}
    ],
    vrienden:24
  },
  {
    naam:"Evelyn Van Welsenaers",
    leeftijd: 44,
    sexe:"v",
    gehuwd:true,
    kinderen:[
      {
        naam:"Patrick",
        leeftijd: 12,
        sexe:"m"
      },
      {
        naam:"Jonas",
        leeftijd: 14,
        sexe:"m"}
    ],
    functie:"bediende",
  }
];
```

```
        id:1025,
        vrienden:11

    },
    {
        leeftijd: 27,
        sexe:"v",
        gehuwd:false,
        id:9007,
        functie:"arbeider",
        naam:"Heidi Vercouteren",
        vrienden:6
    }
]
```

*aoPersoneel* is dus een **Array** van **Objecten**:

```
var aoPersoneel = [
    { ... personeelslid ... },
    { ... personeelslid ... },
    { ... personeelslid ... },
    ...
]
```

Elk personeelslid wordt voorgesteld als een **letterlijk object** in accolades **{ }**. Zo'n object heeft **attributen** (properties) die telkens in een **name:value** paar beschreven zijn.

De waarde van een eigenschap kan eender welk datatype zijn, bv *leeftijd* is een **Number**, *gehuwd* is een **Boolean**, *kinderen* is een **Array**, etc...

Een *personeelslid* beschrijven als een **Object** heeft voordelen ten opzichte van een **Array**:

- de **volgorde** van de attributen is onbelangrijk
- de **aan-/afwezigheid** van een attribuut speelt geen rol (vb. Heidi heeft geen *kinderen*)
- nieuwe attributen kunnen *on-the-fly* **toegevoegd** worden zonder invloed te hebben op de andere objecten
- we kunnen de waarde van een attribuut **gemakkelijk opvragen** via zijn **name**

Bemerk ook wat de prefixes van die variabelen ons vertellen:

- **aFuncties**: een eenvoudige **array**
- **aoPersoneel**: een **array** van **objecten**

De HTML bevat een formulier waarmee we extra personeelsleden kunnen toevoegen aan *aoPersoneel*.

**Keuzelijst opvullen met gegevens:**

De keuzelijst *functie* is momenteel leeg. Er is enkel een leeg **select** element te zien. We zullen die opbouwen met de data uit *aFuncties*.

```
...
//=====KEUZELIJST OPVULLEN =====

sOpties = "<option value=''>--- kies een functie ---</option>";
for(var i=0;i<aFuncties.length;i++){
    sOpties += "<option>" + aFuncties[i] + "</option>";
}
eFunctie.innerHTML = sOpties;
```

Bespreking:

- de var *sOpties* bevat reeds een eerste optie met de value "" (lege string). Een dergelijk eerste item is altijd aan te raden omdat gebruikers anders zonder kijken de eerste functie selecteren. Het noopt je wel tot valideren
- we lussen doorheen het array van functies en voegen telkens een nieuwe **option** toe aan de string
- deze wordt nadien als **innerHTML** geplaatst in de select *eFuncties*

Opmerking:

- dit is één van de gevallen waar we te maken krijgen met **browserproblemen**: in IE is het best mogelijk dat **innerHTML** niet werkt om de **select** op te vullen met opties: test het uit!
- De reden? geen reden... blijkbaar hebben vooral **select** elementen er last van in IE
- wat kunnen we eraan doen? geen **innerHTML** gebruiken: we kunnen in dit geval beter DOM methods gebruiken

Vervang bovenstaande code door:

```
//=====KEUZELIJST OPVULLEN =====

var eDF = document.createDocumentFragment();
var eOption1 = document.createElement('option');
var sValue1 = document.createTextNode("--- kies een functie ---");
eOption1.appendChild(sValue1);
eOption1.value = "";
eDF.appendChild(eOption1);
for(var i=0;i<aFuncties.length;i++){
    var eOption = document.createElement('option');
    var sValue = document.createTextNode(aFuncties[i]);
    eOption.appendChild(sValue);
    eDF.appendChild(eOption);
}
eFunctie.appendChild(eDF);
```

Bespreking:

- hier gebruiken we een `documentFragment`:

Een `documentFragment` is een DOM node die niet één HTML element voorstelt, maar gebruikt kan worden als tijdelijke verzameling van meerdere elementen, ideaal om 'stukken' HTML mee op te bouwen.

- we doen dit om te vermijden dat we teveel invoegingen doen in de dom tree, beter eerst alles verzamelen en slechts één keer invoegen
- merk ook op dat we expliciet de `value` van de eerste option op "" (lege string) zetten. Doe je dat niet dan heeft het element geen `value` attribuut en zal de validatie straks fout lopen

### De opdracht:

is tweevoudig: de lijst van personeelsleden tonen en nieuwe toevoegen.

Vermits we reeds enkele personeelsleden hebben, maken we eerst een functie `fnToonPersoneel` en binden die als event handler aan de knop "toon personeel".

Dit doen we op een nieuwe en betere manier:

```
...  
//=====EVENT HANDLERS=====  
  
ePersoneelsLijst.addEventListener('click',function(){  
    eOutput.innerHTML = fnToonPersoneel(aoPersoneel);  
});  
...
```

Voeg deze code toe aan de `window.onload`.

bespreking:

- de DOM method `addEventListener` is de aangeraden manier om een functie als handler te binden aan een event. Het biedt een aantal voordelen t.o.v. de oude manier – `onclick`:
  - dezelfde methode kan eender welke event binden
  - je kan een event ook gemakkelijk 'ontbinden' met `removeEventListener`
  - je kan meerdere handlers koppelen aan hetzelfde event
- De syntax is

`element.addEventListener(eventtype,functie[,capture])`

- het *event type* is een String met het soort event: `'click'`, `'submit'`, `'mousedown'`,...
- de *functie* is de event handler: je kan hier een anonieme functie gebruiken of doorverwijzen naar een aparte functie (zonder haakjes dus)

- *capture* is een **optionele Boolean** met standaardwaarde **false** die aangeeft of *capture* moet gebruikt worden. Aangezien dit argument optioneel is gebruiken we het niet, MAAR ... oudere browsers kunnen een fout werpen door zijn afwezigheid: plaats dan een **false** als laatste argument .

Dan zou ons voorbeeld er zo uitzien:

```
ePersoneelsLijst.addEventListener('click',function(){
    eOutput.innerHTML = fnToonPersoneel(aoPersoneel);
},false);
```

- hier kunnen we niet rechtstreeks schrijven  
`ePersoneelsLijst.addEventListener('click',fnToonPersoneel)`  
omdat we het argument *aoPersoneel* willen doorgeven aan de functie en de returnwaarde in het output element plaatsen.  
We zijn verplicht hem eerst in een anonieme functie te verpakken

`addEventListener('click',...)` is beter dan `onclick` omdat:

- het de standaard DOM manier is
- het op alle DOM elementen werkt, niet enkel op HTML
- je meerdere eventhandlers voor het hetzelfde event kunt binden
- je één van deze met `removeEventListener` kunt 'afhalen'
- je *capture* kunt gebruiken

### Een geneste *bulleted list*:

Nu maken we de functie *fnToonPersoneel* buiten de `window.onload`; Deze functie zal de eigenschappen van de personeelsleden overlopen en ze weergeven. Hier zullen we dat doen als een bolletjeslijst, maar het spreekt voor zich dat je eender welke HTML structuur kunt bouwen.

```
function fnToonPersoneel(aoData){
    /*
    return HTML String met UL lijst van properties en hun waarden van alle objecten in
    @aoDataarray van objecten
    */
    var sLijst = "";
    if(aoData.length>0){
        //overloop array van objecten
        for(var i=0;i<aoData.length;i++){
            sLijst += "<ul>";
            var oPersoon = aoData[i];
            //overloop alle eigenschappen van object
            for(var key in oPersoon){
```

```
        var propWaarde = oPersoon[key];
        sLijst += "<li>" + key + ":" + propWaarde+ "</li>";
    }
    sLijst += "</ul>";
}
return sLijst;
}
```

Bespreking:

- De functie *fnToonPersoneel()* verwacht een array van objecten (*aoData*)
- de functie bouwt een **string** variabele *sLijst* op die uiteindelijk via een **return** statement als resultaat teruggegeven wordt aan de *caller*
- de bedoeling is dat tekst een HTML string bevat met een aantal **ul** elementen die zelf **li** elementen bevatten. We moeten oppassen dat we geen ongeldige HTML teruggeven (zoals **<ul></ul>**) als het array leeg is, daarom het eerste **if** statement (*aoData.length>0*)
- met een **for** lus doorlopen we alle elementen in het array
  - voor elk item openen/sluiten we een **ul** element
  - elk item is een object dus stellen we voor ons eigen gemak de variable **oPersoon = aoData [i]**
- nu doorlopen we alle eigenschappen van dit persoonsobject met een **for in** lus
  - in **for(var key in oPersoon)** is **key** een variabele die de *sleutel* voorstelt, m.a.w. een eigenschap van het object
    - **key** is de naam van de property
    - **oPersoon[key]** is dan de waarde voor die property, die we voor het gemak gelijkstellen aan de var *propWaarde*
  - voor elk object bouwen we zo verder aan de HTML string en maken een *list item* **li**, met de naam van de property en de waarde ervan als tekst
- bij het einde van elk *oPersoon*'s object sluiten we de **</ul>** af

Als we dit uittesten met het bestaande *aoPersoneel*, krijgen we dit te zien:

toon personeel

- id:4678
- naam:Roger Mary
- functie:instructeur
- leeftijd:65
- sexe:m
- gehuwd:true
- kinderen:[object Object]
- vrienden:24
- naam:Evelyn Van Welsenaers
- leeftijd:44
- sexe:v
- gehuwd:true
- kinderen:[object Object],[object Object]
- functie:bediende
- id:1025
- vrienden:11

We merken op:

- de eigenschappen worden getoond in de volgorde zoals ze in het object staan
- de kinderen worden slechts getoond als **[object Object]**

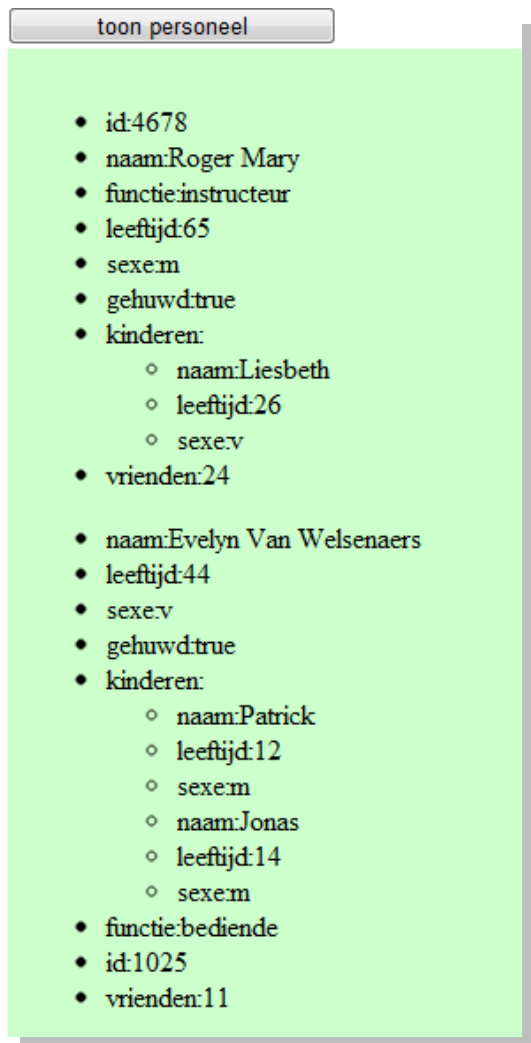
Om dit laatste probleem op te lossen bedenken we dat een kind in feite ook een object is en dat we dit evengoed kunnen behandelen als een personeelslid. Daarom de volgende aanpassing aan het script:

```
...
for(var i=0;i<aoData.length;i++){
    //overloopt alle eigenschappen
    sLijst += "<ul>";
    var oPersoon = aoData[i];
    for(var key in oPersoon){
        var propWaarde = oPersoon[key];
        if (Array.isArray(propWaarde)){
            sLijst += "<li>" + key + ":" + fnToonPersoneel(propWaarde) + 
                "</li>";
        }
        else{
            sLijst += "<li>" + key + ":" + propWaarde + "</li>";
        }
    }
    sLijst += "</ul>";
}
...
```

Bespreking:

- de eigenschap *kinderen* bevat – als hij aanwezig is – ook een Array met Objecten. We kunnen ons voorstellen dat we nog dergelijke eigenschappen zouden kunnen hebben, bijvoorbeeld *projecten*
- daarom gebruiken we de static method `Array.isArray()` om te testen of de waarde van een eigenschap een echt array is:
  - is dat het geval dan roepen we *fnToonPersoneel* **recursief** aan zodat binnenin deze eigenschap een nieuwe **ul** lijst geproduceerd wordt
  - is dat niet het geval dan doen we zoals daarstraks en maken een gewoon **li** element

Nu ziet de output er beter uit:



### Nieuwe personeelsleden toevoegen:

We willen nu via het formulier in staat zijn nieuwe personeelsleden aan de data toe te voegen.

Onderaan het formulier is er de knop “*gegevens toevoegen*”. We zorgen eerst voor een eventhandler ervoor:



```
...

//=====EVENT HANDLERS=====
ePersoneelsLijst.addEventListener('click',function(){
    eOutput.innerHTML = fnToonPersoneel(aoPersoneel);
});
eToevoegen.addEventListener('click', function(){
    ...
});
...
```

Nu vullen we de anonieme functie op, eerst halen we alle gegevens uit de formulervelden.

De *event handler* :

```
eToevoegen.addEventListener('click',function(){
    //formulierwaarden aflezen

    var sNaam          = eNaam.value;           //String
    var nLeeftijd       = eLeeftijd.value;       //String
    var sKind1          = eKind1.value;          //String
    var sKind2          = eKind2.value;          //String
    var sKind3          = eKind3.value;          //String
    var bGehuwd         = eGehuwd.checked;       //Boolean
    var sFunctie        = eFunctie.value;        //String

    var sSexe           = undefined ;            //String of undefined
    //bepaal value sexe
    for(var i=0;i<eSexe.length;i++){
        if(eSexe[i].checked==true){sSexe=eSexe[i].value};
    }

    console.log(sNaam + nLeeftijd + sFunctie + sSexe + bGehuwd + sKind1+ sKind2+ sKind3);

});
...
```

Bespreking:

- eerst lezen we de eenvoudige tekstvelden met hun **value**: *naam, leeftijd, kind1-3*
- de checkbox *gehuwd* controleren we via zijn **checked** property die een **boolean** is
- een keuzelijst heeft ook een **value**, die zoals je weet twee mogelijke waarden heeft:
  - de inhoud van het **value** attribuut, als dat aanwezig is
  - de tekst in het **option** element als er geen **value** attribuut is

in ons geval is dat voor de functies de tekst, voor de eerste optie een lege string

- voor het radiobutton veld eSexe is de zaak niet zo rechtlijnig.
  - we stellen expliciet de var sSexe in op **undefined**
  - de **eSexe.value** opvragen retourneert een **undefined**, omdat het hier over een groep, een array, van **input** elementen gaat.
  - daarom moeten we doorheen alle keuzerondjes lussen die deel uitmaken van de groep eSexe (die dezelfde **name** hebben) en elkeen's **checked** property controleren.
  - dan lezen we de waarde van het **value** attribuut (m|v) van het aangevinkte rondje en die zetten we in de variabele sSexe
  - is geen van de twee rondjes aangevinkt, dan blijft de waarde van sSexe op **undefined** staan en kunnen we valideren.
- Een **console.log** statement laat ons toe de ingevulde waarden eens te controleren

In een volgende stap valideren we de waarden **op eenvoudige wijze** (geen specifieke foutboodschappen) en voegen die dan toe aan aoPersoneel via de functie *fnPersoneelslidToevoegen*.

We vervolgen de eventhandler functie:

```
...
console.log(sNaam + nLeeftijd + sFunctie + sSexe + bGehuwd + sKind1+ sKind2 + sKind3);

//eenvoudige validatie
if(sNaam == "" || isNaN(nLeeftijd) || typeof(sSexe)=="undefined" || sFunctie=="") {
    //niet goed
    console.log('validatie NOK');
    alert('één van de verplichte velden is niet ingevuld')
}
else {
    //ok
    console.log('validatie OK');
    fnPersoneelslidToevoegen(sNaam,nLeeftijd,bGehuwd,sFunctie,sSexe,[sKind1,sKind2,sKind3])
    ;
}

});
...
```

Bespreking:

- de validatie is *alles of niets*: één foutboodschap. **Console** statements tonen ons welke beslissing genomen wordt
- de *naam* wordt gecontroleerd op een lege string
- de *leeftijd* wordt gecontroleerd op een cijfer: **isNaN** geeft true als het geen getal is

- de `eSexe` mag niet **undefined** zijn. De operator **typeof** retournt het data type van de variabele
- de *functie* van de persoon mag geen lege string zijn
- Als alles Ok is gaan we verder en sturen alle gegevens als argumenten door naar een functie *fnPersoneelslidToevoegen*

We maken nu deze functie (ook buiten de **window.onload**):

```
function fnPersoneelslidToevoegen(naam,leeftijd,gehuwd,functie,sexe,aKindnamen){
  /*
  maakt een personeelslid object en voegt toe aan aoPersoneel
  @naam      String,
  @leeftijd   Number,
  @gehuwd     Boolean,
  @functie    String,
  @sexe       String,
  @aKindnamen Array, optioneel, array van kindnamen
  */
  var persoon          = new Object();
  persoon.naam         = naam;
  persoon.leeftijd     = leeftijd;
  persoon.functie      = functie;
  persoon.gehuwd       = gehuwd;
  persoon.sexe         = sexe;
  persoon.vrienden     = 0;
  persoon.id           = parseInt( (Math.random() * 10000) + 1);
  var aKindNamen       = aKindnamen || []; //optioneel argument opvangen
  var aantalKinderen   = aKindnamen.length;
  if(aantalKinderen>0){
    persoon.kinderen    = []; //maak een property
    for(var i=0;i<aantalKinderen;i++){
      if(aKindnamen[i]!=""){
        var kind = new Object();
        kind.naam = aKindnamen[i];
        persoon.kinderen.push(kind);
      }
    }
  }
  aoPersoneel.push(persoon);
  //console.log(aoPersoneel)
}
```

Bespreking:

- we maken een **object**variable *persoon*. We doen dat met een **constructor** (**new Object**). Een constructor is een functie.  
Dit komt op hetzelfde neer als **var persoon = { };**
- elk argument wordt als waarde ingesteld voor een gelijknamige **property** van *persoon*

- het aantal *vrienden* wordt op 0 gezet
- het *id* getal wordt hier een willekeurig getal tussen 10000 en 1. Dat doen we aan de hand van de **Math.random** method, het max en het min getal. Omdat het resultaat een geheel getal zou zijn, passen we er nog eens **parseInt** op toe. Het spreekt voor zich dat dit in een reële situatie op een andere manier zal gebeuren...
- Het argument *aKindnamen* is **optioneel**: dat betekent dat het volledig achterwege gelaten kan worden in de functieaanroep. Omdat we echter de lengte van het array *aKindnamen* berekenen voegen we deze lijn toe:

```
var aKindNamen = aKindnamen || [];
```

dit betekent zoveel als "*als er geen **argument** aKindnamen is, zet dan de **variabele** aKindnamen gelijk aan een leeg array*"

- daarna lussen we doorheen het array *aKindnamen*. Indien leeg gebeurt er niets meer. Indien er minstens één kind is
  - maken we voor *persoon* een nieuwe eigenschap *kinderen* aan met een leeg array als waarde
  - voor elke niet-lege naam die we vinden, maken we een nieuw *kind* **object** aan
  - de naam wordt dan toegevoegd als waarde van de eigenschap *naam* van dat *kind* object
  - het *kind* object zelf wordt toegevoegd (**push**) aan het array *kinderen*

Het is dus belangrijk ervoor te zorgen dat eventuele kinderen dus allemaal terecht komen als **objecten** in een **array**, dezelfde structuur als het originele *aoPersoneel*.

- tenslotte wordt de objectvariable *persoon* aan het array *personen* toegevoegd met de array method **push**

*Je vraagt je misschien af of het aantal kinderen in het formulier niet beter variabel gemaakt moet worden? Inderdaad, dat zou ideaal zijn, maar dat brengt ons te ver en heeft weinig meer met het doel van dit project te maken.*

## Teller bijhouden

Momenteel staat onderaan het formulier een teller die het aantal personeeleden toont: in het **span** element met de **id** *teller* staat oorspronkelijk een 3. We willen dit dynamisch regelen. We maken een functie *fnUpdateTeller*:

We voegen volgende code toe:

```
function fnUpdateTeller(n){  
  /*  
    update de teller in de span#teller  
    @n    increment/verhoging
```

```
*/  
var eTeller      = document.getElementById('teller');  
var nTeller      = parseInt(eTeller.innerHTML);  
nTeller         = nTeller + n;  
eTeller.innerHTML = nTeller;  
  
}
```

en een laatste lijntje aan *fnPersoneelslidToevoegen*:

```
...  
aoPersoneel.push(persoon);  
fnUpdateTeller(1);  
}
```

Bespreking:

- de waarde van de teller wordt gelezen via `innerHTML`, die hier geen problemen oplevert
- deze `string` variabele wordt omgezet naar een `number` met `parseInt()`
- `nTeller` wordt verhoogd met het argument `n`
- en we plaatsen het getal terug in `eTeller`

### formulier reset

Na het “opslaan” van één personeelslid, moeten we een volgende opslaan. Daarom maken we de inputvelden leeg en plaatsen de cursor opnieuw in het eerste vak. Voeg deze code toe aan de event handler voor de knop toevoegen:

```
...  
else {  
    //ok  
    fnPersoneelslidToevoegen(sNaam,nLeeftijd,bGehuwd,sFunctie,sSexe,↵  
                                [sKind1,sKind2,sKind3]);  
  
    //formulier reset voor volgende toevoeging  
    document.frmPersoneelslid.reset()  
    eNaam.focus();  
}  
});
```

Bespreking:

- met `document.frmPersoneelslid` refereren we het formulier. Met de method `reset()` voeren we een initialisatie ervan uit: alle waarden van de velden worden terug op hun `defaultValue` (oorspronkelijke waarde) gezet.
- met `eNaam.focus()` plaatsen we de cursor opnieuw in dat veld

### Verborgten data

De bolletjeslijst die we hier gemaakt hebben is een zeer eenvoudige structuur. Zonder grote moeite kunnen we van onze gegevens een veel aangenamer voorstelling maken. daarbij kunnen gebruik maken van de gegevens van elk persoon om opmaak te voorzien of om meer interactiviteit aan de gebruiker te bieden.

Dat doen we in eerste instantie door de lijst te verlaten voor een *blok* voorstelling. Met dit voorbeeld zul je ook zien dat tabellen echt niet nodig zijn. Het startbestand is reeds voorzien van een aantal toepasselijke style rules.

Kopieer de functie *fnToonPersoneel* naar *fnToonPersoneel2* . Nu vervangen we de *ul* en *li* elementen door *div* en *span*:

```
function fnToonPersoneel2(aoData){
/*
return HTML String met DIVs en SPANs van properties en hun waarden van alle objecten in
@aoData array van objecten
*/

var sLijst = "";
if(aoData.length>0){
//overloopt het array
for(var i=0;i<aoData.length;i++){
//overloopt alle eigenschappen
var oPersoon = aoData[i];
sLijst += "<div class='persoon'>";

for(var key in oPersoon){
var propWaarde = oPersoon[key];
if (Array.isArray(propWaarde)){
sLijst += "<span class='prop'>" + key + "
"</span><span class='val'>" +
fnToonPersoneel2(propWaarde)+ "</span>";
}
else{
sLijst += "<span class='prop'>" + key + "
":</span><span class='val'>" +
propWaarde+ "</span>";
}
}
sLijst += "</div>";
}
}
return sLijst;
}
```

Wijzig nu ook de event handler van de knop "toon personeel" met de nieuwe functie:

```
ePersoneelslijst.addEventListener('click',function(){
eOutput.innerHTML = fnToonPersoneel2(aoPersoneel);
});
```

Als je dit nu uittest krijg je een heel ander beeld:

toon personeel

id:	4678												
naam:	Roger Mary												
functie:	instructeur												
leeftijd:	65												
sexe:	m												
gehuwd:	true												
kinderen	<table> <tr> <td>naam:</td> <td>Liesbeth</td> </tr> <tr> <td>leeftijd:</td> <td>26</td> </tr> <tr> <td>sexe:</td> <td>v</td> </tr> </table>	naam:	Liesbeth	leeftijd:	26	sexe:	v						
naam:	Liesbeth												
leeftijd:	26												
sexe:	v												
vrienden:	24												
naam:	Evelyn Van Welsenaers												
leeftijd:	44												
sexe:	v												
gehuwd:	true												
kinderen	<table> <tr> <td>naam:</td> <td>Patrick</td> </tr> <tr> <td>leeftijd:</td> <td>12</td> </tr> <tr> <td>sexe:</td> <td>m</td> </tr> </table> <table> <tr> <td>naam:</td> <td>Jonas</td> </tr> <tr> <td>leeftijd:</td> <td>14</td> </tr> <tr> <td>sexe:</td> <td>m</td> </tr> </table>	naam:	Patrick	leeftijd:	12	sexe:	m	naam:	Jonas	leeftijd:	14	sexe:	m
naam:	Patrick												
leeftijd:	12												
sexe:	m												
naam:	Jonas												
leeftijd:	14												
sexe:	m												
functie:	bediende												
id:	1025												
vrienden:	11												

## Data-\* attribuut

Nu is het onze bedoeling twee properties niet in het overzicht te tonen maar die op een andere manier te gebruiken: *id* en *vrienden*.

De *id* property zullen we gebruiken voor een *id* en een *title* attribuut van het *div* element en de property *vrienden* voor een HTML5 *data-\** attribuut.

We passen opnieuw aan:

```
...
for(var i=0;i<aoData.length;i++){
    //overloopt alle eigenschappen
    var oPersoon = aoData[i];
    sLijst += "<div class='persoon'" +
        " id='pers_" + oPersoon.id + "'" +
        " title='personeelsnummer: " + oPersoon.id + "'" +
        " data-vrienden='" + oPersoon.vrienden + "'" +
        ">";
    for(var key in oPersoon){
        var propWaarde = oPersoon[key];
        if(key != "id" && key != "vrienden" ){
            if (Array.isArray(propWaarde)){
                sLijst += "<span class='prop'" + key + " " +
                    "</span><span class='val'" +
                    "fnToonPersoneel2(propWaarde)+ "</span>";
            }
        }
    }
}
```

```
        }
        else{
            slijst += "<span class='prop'>" + key + "
                    "</span><span class='val'>" + 
                    propWaarde+ "</span>";
        }
    }
}

if(typeof oPersoon.vrienden !== "undefined") {
    slijst += "<button class='like' title='voeg een vriendje toe'>Like</button>";
}
slijst += "</div>";
}
...
```

Bespreking:

- het **div** element krijgt een aantal attributen erbij:
  - **id**: een samenstelling van '*pers\_*' en de *id* property, bv. '*pers\_4678*'
  - **title**: de tekst 'personeelsnummer: 4678'
  - **data-vrienden**: het getal van de property *vrienden*

**data-vrienden** is een voorbeeld van een **data-\*** attribuut. **Dataset** attributen kan je naar willekeur aanmaken in HTML5, zo zou je net zo goed een **data-kleur**, **data-breedte**, **data-beschrijving** kunnen maken. De bedoeling is dat je deze attributen gebruikt als **lokale databank** van gegevens die niet zichtbaar moeten zijn. Via Javascript zijn deze waarden gemakkelijk te bereiken.

Bemerk ook dat je een property van een object rechtsreeks kunt aanspreken met de **dot notatie**: **persoon.vrienden** of met de **haakjes notatie**: **persoon['vrienden']**.

- in de **for in** lus maken we nu via een **if** een uitzondering voor die twee properties zodat ze niet meer getoond worden
- achteraan de **div** voegen we ook een **button** element in met de **class** 'like'. Deze CSS class zal zorgen voor een achtergrondbeeld voor de knop. De CSS rules hiervoor zijn al aanwezig in het bestand. De knop wordt enkel ingevoegd als de *oPersoon* een eigenschap *vrienden* heeft die niet **undefined** is, dus kinderen krijgen er geen

Als je nu de personeelslijst opnieuw toont, zie je Like knoppen voor alle personen:



naam:	Evelyn Van Welsenaers												
leeftijd:	44												
sexe:	v												
gehuwd:	true												
kinderen	<table border="1"> <tr> <td>naam:</td> <td>Patrick</td> </tr> <tr> <td>leeftijd:</td> <td>12</td> </tr> <tr> <td>sexe:</td> <td>m</td> </tr> </table> <table border="1"> <tr> <td>naam:</td> <td>Jonas</td> </tr> <tr> <td>leeftijd:</td> <td>14</td> </tr> <tr> <td>sexe:</td> <td>m</td> </tr> </table>	naam:	Patrick	leeftijd:	12	sexe:	m	naam:	Jonas	leeftijd:	14	sexe:	m
naam:	Patrick												
leeftijd:	12												
sexe:	m												
naam:	Jonas												
leeftijd:	14												
sexe:	m												
functie:	bediende												
<input type="button" value="Like"/>													

## Event registratie van dynamische knoppen

Leuke knoppen maar er gebeurt niets als je erop klikt...Normaal, want we moeten nog een event registratie doen voor elke dynamisch aangemaakte knop.



Het is met *plain Javascript* onmogelijk een event handler op voorhand te koppelen! (dat kan wel in jQuery).

Voor elke nieuwe knop die je aanmaakt, moet je de event handler koppelen **na** de aanmaak.

Misschien denk je dat we de event handler kunnen meegeven via de HTML String en de **innerHTML**. Als je Javascript en HTML gescheiden wil houden, kan dat niet. Het is één van de nadelen van het gebruik van innerHTML, hadden we de div's en knoppen aangemaakt met DOM methods, dan was het gemakkelijker geweest.

We zijn dus verplicht **achteraf** een event handler te koppelen voor alle knoppen.

Voeg een verwijzing naar een functie *fnRegLikeKnoppen* toe aan de ePersoneelsLijst event handler:

```
ePersoneelslijst.addEventListener('click',function(){
    eOutput.innerHTML = fnToonPersoneel2(aoPersoneel);
    fnRegLikeKnoppen();
});
```

Nu maken we deze functie:

```
function fnRegLikeKnoppen(){
    /*
     * event registratie voor de like knoppen
     * kan in JS enkel NA het aanmaken van de knop
     *
     */
}
```

```
var eLikes = document.querySelectorAll('.like');
for(var i=0;i<eLikes.length;i++){
    eLikes[i].addEventListener('click',function(){
        var ePersoon    = this.parentNode;
        var nVriendjes = parseInt(ePersoon.dataset['vrienden'] )+1;
        ePersoon.dataset['vrienden'] = nVriendjes;
        alert("Deze persoon heeft er een vriendje bij: " + nVriendjes);
    })
}
```

Bespreking:

- eerst maken we een collection van de knoppen met `querySelectorAll`. In tegenstelling tot `querySelector` maakt deze method een collection elementen aan, niet slechts één element. We hebben de class selector `.like` gebruikt
- met een `for` lus lussen we doorheen de collection om de event handler te binden.
  - we voegen aan elke knop een `click` event handler toe die het `this` keyword gebruikt: het event heeft plaats op de knop, dus is `this` de knop zelf.
  - dus is `this.parentNode` de `div` waarin de `button` zit,
  - met `dataset['vrienden']` lezen we het `data-vrienden` attribuut, we zetten het om naar een `integer` en hebben in `nVriendjes` het aantal vrienden
  - we tellen er 1 bij op en plaatsen het opnieuw in het attribuut
  - we tonen een bericht dat we er eentje toegevoegd hebben

Het project is klaar.

Taken:

- TagCloud

## 2.1.6 Een kalender

### Doel

In dit project leren we voornamelijk werken met datums, het **Date** object en zijn vele methods. Daarnaast maken we ook kennis met enkele veelgebruikte **String** methods.

We bouwen een jaarkalender waarin we de huidige datum en je eerstkomende verjaardag aanduiden. We maken ook een eigen Javascript library.

### Theorie

Lees zeker de volgende theorie topics na:

- het **Object** object
- het **Date** object
- data types, variabelen, scope
- arrays
- String methods: **substr()**, **toUpperCase()**, **split()**
- lusstructuren
- DOM intro
- **innerHTML**

### Duurtijd

2 ½ uur

### Basisbestanden

*kalender.html; kalender.css*

### Een eigen library

Open het basisbestand : het is voor één keer een XHTML document, dat maakt ook geen enkel verschil uit voor dit project.

Bekijk de code: het bevat twee containers: een **div#output** en een **div#kalender**. Er is ook een extern stylesheet *kalender.css* aan gekoppeld met enkele *style rules* voor de opmaak van de toekomstige kalender.

De functies die we deze maal zullen maken, kunnen nuttig blijken voor andere pagina's, daarom zullen we beginnen met de opbouw van een eigen **library** (bibliotheek). Een dergelijke library bevat dus geen pagina-specifieke scripts, enkel algemeen bruikbare.

Maak een leeg tekstdocument aan en sla het op als *nuttig\_lib.js*.

Om onze 'output' elementen meerdere keren te kunnen opvullen zorgen we eerst voor een functie die de inhoud van een element leegt.

Plaats de volgende functie in je nieuwe library:

```
// JavaScript library

/***** DOM functies *****/

function leegNode(objNode){
/* verwijdert alle inhoud/children van een Node
@ objNode: node, verplicht, de node die geleegd wordt
*/
while(objNode.hasChildNodes()){
    objNode.removeChild(objNode.firstChild)
}
}
```

We delen onze *library* in in categorieën: dit is een DOM functie, straks volgen andere.



Vermeld altijd in commentaar bij een functie:

- wat de functie juist **doet** en eventueel welke waarde hij **returnt**
- voor elk **argument** (soms voorafgegaan door een @ teken):
  - de naam van het argument,
  - het verwachte datatype
  - een korte omschrijving
  - eventueel of het optioneel is

Hoe werkt de functie *leegNode()*?

- de functie heeft een argument *objNode*: dat is een variabele die gebruikt wordt binnen de functie. De naam *objNode* geeft een aanduiding dat we een DOM *node* verwachten – geen getal of tekst of nog iets anders  
Als je de functie dus gebruikt zal je een referentie naar een *node* – een element bijvoorbeeld - moeten meegeven aan de functie, bijvoorbeeld

```
leegNode(document.getElementById('inhoudstafel'))
```

- De **while** lus lust zolang tot de voorwaarde **false** wordt: in dit geval tot er geen *childNodes* meer overblijven. De method **hasChildNodes()** returnt een **true** zolang een *node* nog *childNodes* heeft.
- in elke **while** lus wordt de eerste *childNodes* verwijderd. Dat zou evengoed de laatste *childNodes* kunnen zijn, tenslotte blijven er geen meer over
- merk op dat de method **removeChild()** enkel vanuit de *parent* kan uitgevoerd worden: een node kan zichzelf niet vernietigen, enkel zijn parent kan dat

Maak nu een **koppeling** in *kalender.html* naar de library met een **script** tag:

```
...
<script type="text/javascript" src="nuttig_lib.js"></script>
</head>
...
```



Let op! schrijf een **script** tag naar een extern script nooit als

```
<script type="text/javascript" src="nuttig_lib.js" />
```

Strikt genomen is dit een XHTML EMPTY tag, dus in theorie is dit correct, in de praktijk werkt het niet ...



Als je **meerdere libraries** koppelt kan je eventueel wat verduidelijken door tussen de begin- en eindtag *html commentaar* te zetten:

```
<script type="text/javascript" src="nuttig_lib.js">
<!-- algemene js library -->
</script>
```



Elke geldige URL is toegelaten: het is bijvoorbeeld perfect toegelaten te koppelen naar een script op het web. Je kunt daar een bestaande library gebruiken of je eigen script op het web plaatsen:

```
<script type="text/javascript" ↵
src="http://www.google.com/jsapi"></script>
```

Sla beide bestanden op.

Nu zullen we dit even uitproberen: in *kalender.html* zetten we nog een **script** tag met wat code:

```
<script type="text/javascript">
window.onload = function (){
    // DOM elementen
    var divOutput      = document.getElementById('output');
    var divKalender    = document.getElementById('kalender');

    //leeg inhoud
    leegNode(divOutput); //functie uit nuttig_lib.js
}
</script>
```

Bespreking:

- De **window.onload** handler maakt twee object variabelen, **divOutput** en **divKalender** aan
- de **leegNode()** functie werkt op de inhoud van **divOutput**: de tekst verdwijnt onmiddellijk

Nu we weten dat onze library functie werkt mag je de tekst in `divOutput` verwijderen en ook de code wissen of commentarieren.

## De huidige datum en tijd

Nu beginnen we aan de kalender: we willen de huidige datum en tijd in de pagina plaatsen. Ook hier hebben we code die herbruikbaar is: het gaat in onze library *nuttig\_lib.js*.

Maak een categorie *Datum, tijd functies* aan en plaats een globale variabele:

```
/****** Datum, tijd functies *****/  
  
//globale datum objecten  
var vandaag = new Date();
```

- De variabele *vandaag* zit niet in een functie en heeft daarom een *global scope*
- Hij bevat de huidige datum als **Date** object.  
Een **Date** object kan aangemaakt worden **mét** en **zonder argumenten**. Zonder argumenten neemt het de systeemdatum (dus de huidige datum en tijd) over.

## Het Date object

Een **object** in het algemeen heeft **properties** (eigenschappen) en **methods** (methodes). Het **Date** object heeft voornamelijk methods die tijden *geven* of *instellen*.

De variabele *vandaag* is dus een **Date** object: het is een datum en tijd terzelfdertijd. Maar om enkel het jaar, of de maand, de dag of de tijd te weten te komen moeten je die eruit "extraheren" met een method. Er zijn er veel, dat kan je nagaan in de theorie van **Date**.

Twee methods die *hapklare brokjes* opleveren in de vorm van een *lokaal* opgemaakte tekst (*lokaal = in de taal van je besturingssysteem*) zijn `.toLocaleDateString()` en `.toLocaleTimeString()`.

We maken een functie aan in *nuttig\_lib.js* die een **string** retournt met de huidige datum en tijd.

```
function getVandaagStr(){  
  //returnt een lokale datumtijdstring  
  
  var strNu = "Momenteel: " + vandaag.toLocaleDateString() + ", ";  
  strNu += vandaag.toLocaleTimeString();  
  return strNu;  
}  
//-----
```

De functie *getVandaagStr()* **returnt** een **string**. Deze tekst gebruiken we nu in ons *output* element:

```
window.onload = function (){
  // DOM elementen
  var divOutput      = document.getElementById('output');
  var divKalender    = document.getElementById('kalender');

  //plaatst huidige datum-tijd in output element
  divOutput.innerHTML = getVandaagStr();
}
```

Nu we de techniek een beetje onder de knie hebben, beginnen we aan een echte kalender.

## Een jaarkalender

De bedoeling van dit project is dynamisch een volledige jaarkalender op het scherm te plaatsen bij het inladen van de pagina.

We maken nu een functie *maakJaarKalender()*. Deze delegeert op zijn beurt het aanmaken van de maanden naar een nieuwe functie *maakMaandTabel()*.

Deze laatste produceert een tabelletje voor elke maand. De opmaak wordt verzorgd door de CSS in *kalender.css*. Te wijzigen naar believen.

## Enkele handige arrays

We voegen enkele bruikbare **arrays** toe aan onze library *nuttig\_lib.js*:

```
//-----datum arrays-----

//dagen volgens getDay() volgorde
var arrWeekdagen= new Array('zondag', 'maandag', 'dinsdag', 'woensdag', 'donderdag',
  'vrijdag', 'zaterdag');

//vervang feb dagen voor een schrikkeljaar
var arrMaanden= new Array(['januari',31], ['februari',28], ['maart',31], ['april',30],
  ['mei',31], ['juni',30], ['juli',31], ['augustus',31], ['september',30],
  ['oktober',31], ['november',30], ['december',31]);
```

Vermits je de theorie over **arrays** al gelezen hebt, weet je dat in JS er enkel geïndexeerde arrays bestaan, ook geen echt multi-dimensionele arrays, maar dat je die kunt simuleren.

Het *arrWeekdagen* is een 1-dimensioneel array die de volgorde van de datum functie *getDay()* volgt: dus zondag= 0, maandag = 1, etc...

Het *arrMaanden* is een 2-dimensioneel array: het arrays bevat elementen die zelf een array zijn. Zo bevat *arrMaanden[0]* het array **['januari',31]**.

Om in een multi-dimensioneel array één item te localiseren gebruiken we meerdere indexen: zo is het aantal dagen voor februari te bereiken via `arrMaanden[1][1]`. De eerste index leidt ons naar het tweede element, waarvan dan opnieuw het tweede element genomen wordt.

Beide arrays gaan je goed van pas komen als je met datums werkt. Vermits ze niet in een functie staan hebben ze een **global scope**.

Ook enkele globale variabelen die we overal kunnen gebruiken:

```
...
//globale datum objecten te gebruiken in je pagina
var vandaag          = new Date();
var huidigeDag       = vandaag.getDate(); //dag vd maand
var huidigeWeekDag   = vandaag.getDay(); //weekdag
var huidigeMaand     = vandaag.getMonth();
var huidigJaar       = vandaag.getFullYear();
...
```

Bemerk dat we uit ons **Date** object *vandaag* met methods andere gegevens puren.

Nog een noodzakelijke functie is testen of het behandelde jaar een **schrikkeljaar** is om het max aantal dagen voor februari te vervangen. Daarom deze handige *isSchrikkeljaar()* functie die:

```
//-----
function isSchrikkeljaar(jaar){
/* test voor schrikkeljaar
jaar: number, verplicht
return: boolean
*/
eindwaarde=false;

if (!isNaN(jaar)){
  if (jaar%4===0){
    eindwaarde=true;
    if(jaar%100===0){
      eindwaarde=false;
      if(jaar%400===0){
        eindwaarde=true;
      }
    }
  }
}
return eindwaarde;
}
```

Zoek zelf maar eens uit wanneer een jaar een schrikkeljaar is?



## Een maand

We beginnen met het genereren van een tabelletje voor één maand. De functie *maakMaandTabel()* voegen we toe **buiten** de `window.onload` event handler:

```
function maakMaandTabel(kalenderJaar, maandIndex){
    /*
    Dependency: nuttig_lib
    Return: string, voor innerHTML: een tabelletje met een maandoverzicht
    @kalenderJaar: integer, 4 digit jaar
    @maandIndex: integer, van 0-11

    */

    //controle argumenten:
    if (isNaN(kalenderJaar) || (kalenderJaar.toString().length!=4)){
        return "fout jaargetal";
    }
    if (isNaN(maandIndex) || (maandIndex<0) || (maandIndex>11)){
        return "fout maandgetal";
    }

    //weekdag van de eerste dag van de maand
    var start_datum = new Date(kalenderJaar, maandIndex, 1);
    var start_weekdag = start_datum.getDay();

    //bepaal einddag vr die maand, mogelijke uitzondering februari van schrikkeljaar
    var eindDag = arrMaanden[maandIndex][1];
    if((maandIndex==1) && (isSchrikkeljaar(kalenderJaar))){eindDag=29}

    //opbouw returnwaarde string
    strMaandTabel = "<table class='kalender'>\n";
    // titelrij
    strMaandTabel += "<tr><th colspan='7'>" + arrMaanden[maandIndex][0]+ " ";
    strMaandTabel += kalenderJaar + "</th></tr>\n";

    //dagtitels
    strMaandTabel += "<tr>";
    for(var i=0;i<7;i++){
        strMaandTabel += "<td>" + arrWeekdagen[i].substr(0,2).toUpperCase() + "</td>";
    }
    strMaandTabel += "</tr>\n";

    var dag      = 1;
    var teller   = 0;

    while(dag<=eindDag){
        // weekrij
        strMaandTabel += "<tr>";
        for (var i=0;i<7;i++){
```

```
//teken cellen, met of zonder dag ingevuld
var strId          = ''; // id samengesteld uit maandIndex en dagnummer
var strDagNummer   = ''; //het dagnummer
//schrijf de dagen
if ((teller>=start_weekdag)&&(dag<=eindDag)) {
    strDagNummer = dag;
    strId = " id='" + kalenderJaar + "_" + maandIndex + "_" + dag + "'";
    dag++;
}
//schrijf de cel
strMaandTabel += "<td " + strId + ">" + strDagNummer + "</td>";
teller++;
}
strMaandTabel += "</tr>\n";
}
strMaandTabel += "</table>\n";
return strMaandTabel;
}
```

Wat uitleg:

- de functie heeft twee argumenten: een jaartal van 4 digits en een maandgetal dat start met 0 en maximaal 11 is
- de functie heeft een *dependency* op *nuttig\_lib.js*, met andere woorden, die library moet gekoppeld zijn, omdat er functies en arrays uit gebruikt worden
- de functie **returnt** een HTML string, *strMaandTabel*, die een **table** element bevat. Deze string kunnen we dan via **innerHTML** in een container plaatsen
- eerst controleren we die argumenten: indien die fout zijn returnen we onmiddellijk met een foutbericht
- daarna bepalen we welke weekday de eerste dag van deze maand is: dat doen we door een **Date** te construeren met de argumenten en er daarna de method **getDay** op los te laten: die returnt een getal tussen 0 (zondag) en 6 (zaterdag)
- daarna bepalen we de laatste dag van de maand, meestal 28, 30 of 31. Dat halen we uit het maandarray. Voor een schrikkeljaar is februari een uitzondering
- dan beginnen we de opbouw van de tabel
- Titelrij: in **arrWeekdagen[i].substr(0,2).toUpperCase()** wordt het array *arrWeekdagen* doorlopen en van elke item worden de eerste twee letters in hoofdletters omgezet
- Weekrijen:
  - De **var dag** houdt het dagnummer bij
  - Omdat niet in alle cellen een dag komt houdt de **var teller** het aantal cellen (**td**) bij
  - Zolang het maximum aantal maanddagen niet bereikt is wordt een volledige rij van 7 cellen toegevoegd

- Omdat de inhoud en de attributen van deze cellen afhankelijk zijn van nogal wat voorwaarden, volgen we een bepaalde **strategie**:
  - De vars `strId` en `strDagNummer` worden eerst **leeg** gedeclareerd en eventueel ingevuld door de voorwaarde verderop.
  - We schrijven de volledige string voor de `<td>` tag aan het einde van de `for` lus
  - alle nodige variabelen worden daarbij ingebouwd, ongeacht hun inhoud.
- Binnen die 7 cellen wordt pas gestart met het plaatsen van het daggetal als `start_weekdag = teller`.
- `dag` wordt met 1 verhoogd
- Een `id`, samengesteld uit `kalenderJaar_maandIndex_DagNummer`, wordt aan de cel toegekend als er ook een `dagNummer` in staat. Lege cellen behoeven geen `id`.  
De `id`'s zullen nodig zijn als we achteraf een bepaalde cel willen bereiken of aanklikken.

Om dit uit te proberen moeten we de functie aanroepen: voeg toe aan de `window.onload`:

```
window.onload = function(){
    divOutput = document.getElementById('output');
    divKalender = document.getElementById('kalender');

    divKalender.innerHTML = maakMaandTabel(2010,1);
}
```

We zien de maand februari van 2010 verschijnen.

Probeer andere maanden/jaren en controleer vooral de weekdag van de eerste dag: klopt die?

Zo kunnen we de huidige maand tonen. Wijzig bovenstaande codelijn in:

```
...
divKalender.innerHTML = maakMaandTabel(huidigJaar,huidigeMaand);
...
```

De variabelen `huidigJaar` en `huidigeMaand` komen uit `nuttig_lib`.

Het resultaat ziet er als volgt uit:

## JS basis project: JS Kalender

Momenteel: Monday, February 01, 2010, 11:48:05

februari 2010						
ZO	MA	DI	WO	DO	VR	ZA
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28						

### De huidige datum aanduiden

In een kalender worden bepaalde dagen aangeduid met een kleurtje.

We kunnen twee strategieën volgen:

- ofwel duiden we de dagen aan *tijdens* de opbouw van de maandtabel
- ofwel duiden we dagen nadien aan

De laatste optie lijkt de beste omdat ze flexibeler is en los staat van de andere functie.

Om een datum aan te duiden gebruiken we CSS: het is enkel een kwestie van de juiste cel een **class** te geven. De nodige stylerules voor die class hebben we al voorzien in ons stylesheet.

Omdat we ook andere dagen (verjaardag, etc...) zullen aanduiden, maken we er een generische functie van:

```
//-----  
function dagAanduiden(oDatum,CSS_Class){  
/*  
nodig:   CSS class in stylesheet  
         id in element  
  
@ oDatum: Datum object van aan te duiden dag  
@ CSS_Class: CSS class dient aanwezig te zijn  
*/  
  
//welk jaar, maand en dag?  
var dDag      = oDatum.getDate();  
var dMaand    = oDatum.getMonth();  
var dJaar     = oDatum.getFullYear();  
  
//construeer id voor cel  
var strId = dJaar+"_"+dMaand+"_"+dDag;  
var dCel = document.getElementById(strId);  
if (dCel){  
    dCel.className = CSS_Class;  
}
```

```
}
```

Wat doen we:

- Uit het argument *oDatum* wordt het jaar, de maand en de dag afgeleid en de vermoedelijke *id* van de cel samengesteld
- Dan wordt een DOM referentie gelegd naar de cel met die *id*
- Als deze geslaagd is, en het object *dCel* dus bestaat, stellen we zijn *className* in op het doorgegeven argument *CSS\_Class*

Deze functie zullen we nu gebruiken om de huidige datum aan te duiden. We zetten de code in de *window.onload*:

```
window.onload = function(){
//uitvoering na onload
  divOutput = document.getElementById('output');
  divKalender = document.getElementById('kalender');
  divKalender.innerHTML = maakMaandTabel(huidigJaar,huidigeMaand);
  dagAanduiden(vandaag,'vandaag'); // huidige datum aanduiden
}
```

Als je dit uitprobeert wordt de huidige datum oranje.

## Een jaarkalender

Het is nu een eenvoudige stap om een jaarkalender te maken. De functie *maakJaarKalender()* voert *maakMaandTabel()* 12 maal uit:

```
function maakJaarKalender(kalenderJaar){
  /*
  Dependency: maakMaandTabel()
  Return: string, voor innerHTML: 12 maandtabellen

  @ kalenderJaar: integer, 4 digit jaar
  */

  strJaarKalender="";
  for(var i=0;i<12;i++){
    strJaarKalender += "<div class='maandContainer'>";
    strJaarKalender += maakMaandTabel(kalenderJaar,i);
    strJaarKalender += "</div>";
  }
  return strJaarKalender;
}
```

Bespreking:

- de functie retournt ook een HTML string die je in een container kan plaatsen
- met een *for* loop lussen we 12 maal om de maandtabelletjes te genereren

- die worden in een **div** geplaatst, enkel maar voor de layout, want sommige maanden veroorzaken een hogere tabel

Nu moeten we nog de call in het **window.onload** event aanpassen:

```
...  
divKalender.innerHTML = maakJaarKalender(huidigJaar);  
...
```

En we kunnen ook nog onze verjaardag aanduiden:

```
...  
//dagen aanduiden  
var verjaardag = new Date(huidigJaar, 11, 14);  
dagAanduiden(vandaag, 'vandaag');  
dagAanduiden(verjaardag, 'verjaardag');  
...
```

Het resultaat ziet er zo uit (in *Chrome*):

## JS basis project: JS Kalender

Momenteel: Monday, February 01, 2010, 11:42:34

januari 2010							februari 2010							maart 2010							april 2010							mei 2010							juni 2010							
ZO	MA	DI	WO	DO	VR	ZA	ZO	MA	DI	WO	DO	VR	ZA	ZO	MA	DI	WO	DO	VR	ZA	ZO	MA	DI	WO	DO	VR	ZA	ZO	MA	DI	WO	DO	VR	ZA	ZO	MA	DI	WO	DO	VR	ZA	
					1	2																																				
3	4	5	6	7	8	9	7	8	9	10	11	12	13	7	8	9	10	11	12	13	4	5	6	7	8	9	10	2	3	4	5	6	7	8	6	7	8	9	10	11	12	
10	11	12	13	14	15	16	14	15	16	17	18	19	20	14	15	16	17	18	19	20	11	12	13	14	15	16	17	9	10	11	12	13	14	15	13	14	15	16	17	18	19	
17	18	19	20	21	22	23	21	22	23	24	25	26	27	21	22	23	24	25	26	27	18	19	20	21	22	23	24	16	17	18	19	20	21	22	20	21	22	23	24	25	26	
24	25	26	27	28	29	30	28							28	29	30	31				25	26	27	28	29	30		23	24	25	26	27	28	29	27	28	29	30				
31																												30	31													
juli 2010							augustus 2010							september 2010							oktober 2010							november 2010							december 2010							
ZO	MA	DI	WO	DO	VR	ZA	ZO	MA	DI	WO	DO	VR	ZA	ZO	MA	DI	WO	DO	VR	ZA	ZO	MA	DI	WO	DO	VR	ZA	ZO	MA	DI	WO	DO	VR	ZA	ZO	MA	DI	WO	DO	VR	ZA	
					1	2	1	2	3	4	5	6	7					1	2	3	4							1	2	3	4	5	6					1	2	3	4	
4	5	6	7	8	9	10	8	9	10	11	12	13	14	5	6	7	8	9	10	11	3	4	5	6	7	8	9	7	8	9	10	11	12	13	5	6	7	8	9	10	11	
11	12	13	14	15	16	17	15	16	17	18	19	20	21	12	13	14	15	16	17	18	10	11	12	13	14	15	16	14	15	16	17	18	19	20	12	13	14	15	16	17	18	
18	19	20	21	22	23	24	22	23	24	25	26	27	28	19	20	21	22	23	24	25	17	18	19	20	21	22	23	21	22	23	24	25	26	27	19	20	21	22	23	24	25	
25	26	27	28	29	30	31	29	30	31					26	27	28	29	30			24	25	26	27	28	29	30	28	29	30				26	27	28	29	30	31			
																					31																					

Taken:

maak nu:

## 2.1.7 CookieBank

### Doel

Leren werken met cookies: we maken een pagina die het saldo van de gebruiker bijhoudt en stortingen en geldafhalingen toelaat.

### Theorie

Lees de volgende theorie topics na:

- Storage

### Duurtijd

2 uur

### Basisbestand

*cookiebank.html*.

### CookieBank

De *CookieBank* is een online bank die de naam en het banksaldo van een bezoeker onthoudt. Dat doen we door enkele *cookies* te schrijven op de PC van de bezoeker.

In de theorie over **Storage** heb je gelezen dat er twee methodes zijn om *state* gegevens op te slaan: **cookies** en HTML5 **Web Storage**. Beide slaan gegevens op met *naam=waarde* paren. In eerste instantie leren we cookies gebruiken, daarna veranderen we het systeem naar Web Storage.

In een eerste fase willen we testen of dit een nieuwe klant is of een terugkerende, misschien heeft hij al een rekening geopend?

Bij een eerste bezoek vragen we zijn naam en slaan die op in een cookie/storage item. Bij elk volgend bezoek kijken, lezen we het cookie/storage item en maken er gebruik van.

In dit project maken we vooral gebruik van DOM methods, zelden van **innerHTML**.

Het startbestand is HTML5.

### Cookiefuncties

Plaats nu eerst deze drie functies in je library *nuttig\_lib.js*:

```

/***** cookies *****/

function setCookie(naam,waarde,dagen){
/*plaatst een cookie

naam: cookienaam;
waarde: de inhoud van het cookie
dagen: optioneel, het aantal dagen dat het cookie geldig blijft vanaf nu
        indien afwezig wordt het een session cookie
*/
    var verval = "";
    if(dagen){
        //vandaag global bovenaan deze lib;
        var vervalDatum = new Date(vandaag.getTime()+dagen*24*60*60*1000);
        verval = vervalDatum.toUTCString();
    }
    document.cookie = naam + "=" + waarde + ";expires=" + verval;
}
//-----
function getCookie(naam){
/*leest een cookie

naam: cookienaam
*/
    var zoek = naam + "=";
    if (document.cookie.length>0){
        var begin = document.cookie.indexOf(zoek);
        if (begin!=-1){
            begin += zoek.length;
            var einde = document.cookie.indexOf(";", begin);
            if (einde==-1){
                einde = document.cookie.length;
            }
            return document.cookie.substring(begin, einde);
        }
    }
}
//-----
function clearCookie(naam){
/*
verwijdert een cookie

naam: cookienaam
*/
    setCookie(naam,"",-1);
}

```



## Welkom

Net als in vorige scripts plaatsen we alle statements die onmiddellijk uitgevoerd moeten worden in een `window.onload` event. Open het basisbestand en plaats een `script` tag.

We leggen eerst referenties naar enkele elementen en stellen standaardwaarden in:

```
...
<script>
window.onload = function(){

    //DOM elementen
    var eOutput      = document.getElementById('output');
    var eKnopKrediet  = document.getElementById('krediet');
    var eKnopDebiet   = document.getElementById('debiet');

    //standaardwaarden
    var sMsg          = '';                // bericht aan gebruiker
    var sNaam          = 'nieuwe klant';    // standaard invulling naam
    var nSaldo         = 0;                // standaard saldo

    } //einde window.onload
</script>
```

Als dit de eerste maal is dat de bezoeker onze website bekijkt, dan is er geen cookie, ofwel is dit niet zijn eerste bezoek en dan bestaat er een cookie op zijn PC.

In elk geval moeten we testen voor de aanwezigheid van het cookie en een bericht op het scherm plaatsen. We maken daarbij gebruik van de cookie functies in onze *nuttig\_lib.js*:

vervolledig de `window.onload`:

```
//standaardwaarden
...

//test cookie
if(getCookie('klantnaam')){
    //gekende klant
    var sNaam      = getCookie('klantnaam');
    var nSaldo     = getCookie('saldo');

    //bericht
    sMsg += "Welkom " + sNaam + ",";
    sMsg += "uw saldo bedraagt " + nSaldo + " Euro";
}
else{
    //eerste bezoek
    sMsg += "Welkom beste bezoeker. ";
    sMsg += "Als u bij ons een nieuwe rekening opent, ontvangt u een startsaldo van 100 Euro!";
}
```

```
}  
  
} //einde window.onload
```

Deze code is uitvoerbaar maar nog niet volledig:

- de library functie `getCookie()` returnt ofwel de waarde van een cookie *'klantnaam'* ofwel `null` indien die niet gevonden wordt
- afhankelijk van dit resultaat schrijven we twee verschillende teksten naar de var `sMsg`:
  - ofwel de klantnaam met zijn saldo (via een ander cookie)
  - ofwel een welkomsttekst voor een nieuwe bezoeker

Dit bericht moet nu nog getoond worden in de `div#output`, dat doen we door een `documentFragment` samen te stellen. Een `documentFragment` is een DOM node die niet één HTML element voorstelt, maar gebruikt kan worden als tijdelijke verzameling van meerdere elementen, ideaal om 'stukken' HTML mee op te bouwen.

Vervolg de code onder de `if` structuur:

```
...  
else {  
    ...  
}  
  
// generische DOM elementen  
var dfBericht = document.createDocumentFragment();  
var eNl       = document.createElement('br');  
  
//vervolledig documentFragment en voeg in  
var tNode = document.createTextNode(sMsg);  
dfBericht.appendChild(tNode);  
dfBericht.appendChild(eNl.cloneNode(false));  
dfBericht.appendChild(eNl.cloneNode(false));  
eOutput.appendChild(dfBericht);  
  
} //einde window.onload
```

- we maken twee generische DOM elementen aan, de `documentFragment` en een `br` element. Beide bestaan dus enkel in de JS geheugenruimte
- daarna maken we een `TextNode` met het samengestelde bericht aan
- en voegen die achteraan het `documentFragment` toe
- daar komt nog eens een kloon van het `br` element bij, tweemaal, om wat ruimte onder de tekst te krijgen.  
het is noodzakelijk het element te klonen, je kan het geen tweemaal invoegen. Voor de method `cloneNode()` is het argument `deep` op `false` gezet, want het heeft toch geen children

- tenslotte wordt het **documentFragment** ingevoegd in **eOutput**

Probeer dit even uit, je moet de welkomsttekst voor een nieuw klant krijgen.

Naast de tekst willen we ook nog een knop waarmee de nieuwe klant een nieuwe rekening kan openen of waarmee hij een bestaande rekening kan afsluiten.

We voegen een en ander toe aan het script:

```
if(getCookie('klantnaam')){
    //gekende klant
    var sNaam      = getCookie('klantnaam');
    var nSaldo     = getCookie('saldo');

    //outputbericht
    sMsg = "Welkom " + sNaam + ",";
    sMsg += "uw saldo bedraagt " + nSaldo + " Euro";
    //knop
    var eKnop      = maakKnop('Sluit rekening');
    eKnop.addEventListener('click',rekeningSluiten); //eventhandler
}
else{
    //eerste bezoek
    sMsg = "Welkom beste bezoeker. ";
    sMsg += "Als u bij ons een nieuwe rekening opent, ontvangt u een startsaldo van 100 Euro!";
    //knop
    var eKnop      = maakKnop('Open rekening');
    eKnop.addEventListener('click',rekeningOpenen);
}

// generische DOM elementen
var dfBericht = document.createDocumentFragment();
var eNl       = document.createElement('br');

//vervolledig documentFragment en voeg in
var tNode = document.createTextNode(sMsg);
dfBericht.appendChild(tNode);
dfBericht.appendChild(eNl.cloneNode(false));
dfBericht.appendChild(eNl.cloneNode(false));
dfBericht.appendChild(eKnop);

eOutput.appendChild(dfBericht);
...
```

- Om een **button** element aan te maken, die in het ene geval de rekening sluit, in het ander opent, gebruiken we een functie *maakKnop*
- de var *eKnop* krijgt in beide gevallen onmiddellijk een event handler toegewezen
- en wordt aan het **documentFragment** toegevoegd

Nu schrijven we de drie functies, onder het `window.onload` event:

```
/******FUNCTIES******/

function maakKnop(tekst){
  /*
   *returnt een DOM button element
   */
  var eKnop    = document.createElement('button');
  var sTekst   = document.createTextNode(tekst);
  eKnop.appendChild(sTekst);
  eKnop.setAttribute('type','button');
  return eKnop;
}
//-----

function rekeningOpenen(){
  console.log('rekening openen');
}
//-----

function rekeningSluiten(){
  console.log('rekening sluiten');
}
```

Bespreking:

- de functie *maakKnop()* retournt een `button` element, dit kan je dus een *factory function* noemen: hij produceert op aanvraag
- de twee andere functies worden straks verder gespecificeerd maar ze moeten bestaan om ze te kunnen koppelen als event handlers

Probeer dit uit, de knoppen geven enkel `console` berichten.

## CookieBank

Welkom bij CookieBank, de bank met smaak!

Welkom beste bezoeker. Als u bij ons een nieuwe rekening opent, ontvangt u een startsaldo van 100 Euro!

Open rekening

\* verplicht

bedrag \*

+ -

De volgende stap is de functie *rekeningOpenen()*:

```
function rekeningOpenen(){
  //console.log('rekening openen');
  var sNaam = window.prompt("Uw naam, graag?", "");
  if (sNaam!="" && sNaam!=null){
    setCookie('klantnaam', sNaam, 100);
    setCookie('saldo', 100, 100);
  }
}
```

Bespreking:

- als de gebruiker de functie opstart verschijnt een dialoogvenster (`window.prompt`) dat ofwel de naam retournt ofwel een `null` waarde (in het geval van *cancel*)
- het tweede argument van `window.prompt` is optioneel, maar gebruiken we toch om de vervelende *'undefined'* default waarde in IE te vermijden
- als de naam gegeven is plaatsen we deze in het cookie *klantnaam* met de library functie `setCookie`. De levensduur van het cookie wordt op 100 dagen ingesteld
- op dezelfde manier wordt het cookie *saldo* gezet op een waarde van 100 (Euro)

Test dit even uit.

Je zal opmerken dat als je de knop *open Rekening* gebruikt, er niets te zien is. Enkel als je de pagina **ververst (F5)** zie je je naam en je saldo verschijnen. Dat zou onmiddellijk moeten gebeuren.

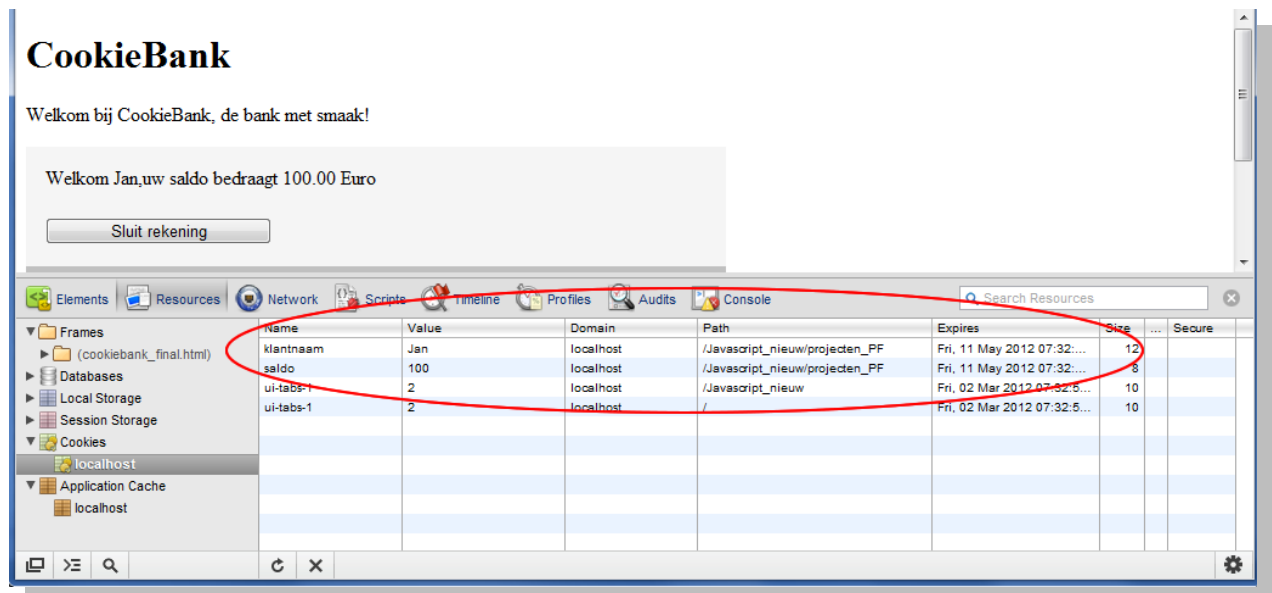
De cookies worden enkel gelezen bij het inladen van de pagina, daarom zullen we bij elke wijziging zelf de pagina moeten verversen:

```
function rekeningOpenen(){
  //console.log('rekening openen')
  var sNaam = window.prompt("Uw naam, graag?", "");
  if (sNaam!="" && sNaam!=null){
    setCookie('klantnaam', sNaam, 100);
    setCookie('saldo', 100, 100);
    window.history.go(0);
  }
}
```

- Om met JS een pagina te doen verversen gebruiken we het `window.history` object met de method `go(index)`. `window.history` bevat een array van url's die je bezocht hebt. Het heeft de methods `back()` en `forward()` (enkel indien `back()` gebruikt is) die overeen komen met de *back* en *forward* functie van de browser. De method `go()` neemt een integer als argument en laat ons toe een aantal pagina's *backward* of *forward* te gaan in de lijst. Gebruik we het argument `0` dan *updaten* we de huidige pagina.

## Cookies nakijken

Via onze browser *webtools* (Firebug etc..) kan je direct nagaan of een website/domein cookies geplaatst heeft. We geven hier het voorbeeld van **Chrome developer tools**, maar zoek even in je favoriete tool en je zult de instellingen voor *Cookies* ook vinden.



Je ziet bij *Resources* links de Cookies voor de *localhost* (waar ik momenteel bezig ben), dan zie je alle cookies met hun name, value, Domain, Path en Expiry date. Meestal kan je via zo'n tool ook een cookiewaarde wissen.

Merk ook op dat er andere cookies te zien zijn voor *localhost* (ui-tabs, een cookie van jquery UI), dat komt omdat ik op andere tabbladen in mijn browser enkele andere bestanden open heb staan waar die gebruikt worden: je ziet de cookies hier **per domein**.

Bekijk je hiermee bijvoorbeeld een Google pagina, dan sta je versteld van het aantal cookies!

Nu hebben we ook een manier nodig om de cookies te vernietigen en de begintoestand te herstellen: *Rekening sluiten*.

Voeg toe aan je script:

```
function rekeningSluiten(){
    //vernietigt de rekening, dus de cookies
    clearCookie('klantnaam');
    clearCookie('saldo');
    window.history.go(0);
}
```

- De functie *rekeningSluiten()* maakt gebruik van de library functie *clearCookie* om zowel de naam als het saldo te wissen. De pagina wordt onmiddellijk ge-updatet.

## Geld storten en afhalen

Via de knoppen '+' en '-' kan de gebruiker een bedrag storten of afhalen. Die knoppen bestaan reeds en er is al een referentie naar gelegd, maar ze werken niet: er is nog geen event handler voor gedefinieerd. Dat doen we nu eerst in de `window.onload`:

```
...
//event handler voor + - knoppen
eKnopKrediet.addEventListener('click', function(){ berekenen('+')});
eKnopDebiet.addEventListener('click', function(){ berekenen('-')});

} //einde window.onload
```

Bespreking:

- de functie *berekenen()* maken we hierna.
- Die verwacht een argument : '+' of '-'. Omdat het onmogelijk is dat argument mee te geven in de verwijzing naar de event handler, zijn we verplicht de echte event handler te verpakken in een anonieme functie.  
Je kan dus niet schrijven

```
eKnopKrediet.addEventListener('click', berekenen('+'));
```

daarmee zou de functie *berekenen* onmiddellijk uitgevoerd worden en niet ingesteld worden als event handler

De functie *berekenen()* kan zowel zowel geldstortingen als afhalingen aan:

```
function berekenen(bewerking){
    /*
    storting of geldafhaling
    @bewerking = een '+' of een '-' teken
    */

    var nNieuwSaldo = 0;
    var eBedrag     = document.getElementById('bedrag');
    var sBedrag     = eBedrag.value;
    var sSaldo      = getCookie('saldo');
    var sBericht    = "";

    if(sSaldo != null && sSaldo != ""){

        if(sBedrag != "" && !isNaN(sBedrag)){

            nSaldo      = parseFloat(sSaldo);
            nBedrag      = parseFloat(sBedrag);
            switch (bewerking) {
                case '+':
                    nNieuwSaldo = nSaldo + nBedrag;
                    break;
                case '-':
                    nNieuwSaldo = nSaldo - nBedrag;
                    break;
            }

        }

    }

}
```

```
        setCookie('saldo', nNieuwSaldo, 100);
        window.history.go(0);
        eBedrag.value = "";

    }
    else{
        alert('U moet een correct bedrag ingeven');
    }
}
else{
    //geen saldo = geen rekening
    var bOpenen = window.confirm('U heeft nog geen rekening geopend, nu even doen?');
    if(bOpenen===true){rekeningOpenen()}
}
}
```

#### Bespreking:

- de var *nNieuwSaldo* wordt geïnitieerd op 0
- dan controleren we de input van het veld *eBedrag*: als dat leeg is of geen getal bevat (via de functie *isNaN*), geven we met een *alert* een waarschuwing.
- dan lezen we het *saldo* cookie: als dat onbestaande is of een lege string, dan vragen we met een *window.confirm* (OK/Cancel knop) of we alsnog een rekening moeten openen.
  - de return waarde van een *window.confirm* is een *Boolean*
  - dan sturen we door naar *rekeningOpenen()*
- als alles OK is (bedrag ingevuld en rekening bestaat),
  - zetten we de waarde van *sBedrag* om in een *Floating point Number* met *parseFloat*: de var *nBedrag*
  - en ook van de waarde uit het cookie: *nSaldo*



Dit is **belangrijk**: de meeste problemen met berekeningen komen voort uit het verkeerd typeren van getallen: het is aan jou om ze te converteren!

- waarden van cookies worden steeds als **string** gereturt!
- waarden van velden worden steeds als **string** gereturt!

- afhankelijk van de gebruikte knop wordt de variabele *nNieuwSaldo* berekend en opnieuw in het cookie *saldo* geplaatst.
- de pagina wordt geüpdatet om de resultaten onmiddellijk zichtbaar te maken
- het statement *eBedrag.value = ""* zorgt ervoor dat het veld bedrag opnieuw leeg komt voor een volgende opdracht

Probeer dit nu uit.

Je bemerkt mogelijk een probleem met decimale getallen: een bedrag zoals 10,20 wordt afgerond naar 10.



Dat komt omdat **JS enkel de punt erkent als decimaal teken**, wat de landinstelling van je besturingssysteem ook is.

De meest efficiënte manier om dit te verwerken is een eventuele **komma** te vervangen door een **decimale punt** vóór de verwerking.

```
...
var sBericht  = "";
var re       = /,/;
sBedrag      = sBedrag.replace(re, '.');

var nNieuwSaldo = 0;
...
```

We gebruiken daarvoor een **regular expression** en de **string** functie **replace**: een komma wordt vervangen door een punt. Omdat we pas enkele lijnen lager er een echt getal van maken zal dit correcte berekeningen geven.

Je kan nu ook opmerken dat we beter twee decimalen tonen in het saldo van de klant. Daarvoor gebruiken we de **Number** method **toFixed(i)** die als argument het aantal decimalen neemt dat je wil tonen.

Wijzig de code als volgt:

```
...
if(getCookie('klantnaam')){
    //gekende klant
    var sNaam      = getCookie('klantnaam');
    var nSaldo     = parseFloat(getCookie('saldo')).toFixed(2);
    ...
}
...
```

### In het rood

We kunnen niet toelaten dat er meer geld afgehaald wordt dan er op de rekening staat: er moet minstens 1 € op blijven staan.

Daarvoor maken we de volgende aanpassingen in *berekenen()*:

```
...
switch (bewerking) {
    case '+':
        nNieuwSaldo = nSaldo + nBedrag;
        break;
    case '-':
        nNieuwSaldo = nSaldo - nBedrag;
        break;
}
```

```
if (nNieuwSaldo<=0){
    var nMax      = nSaldo-1;
    sBericht      += "Uw saldo is onvoldoende om dit bedrag af te halen. ";
    sBericht      += "U kunt maximaal " + nMax + " Euro afhalen.";
    eBedrag.value = nMax;
    eBedrag.focus();
    toonWaarschuwing(sBericht);
}
else{
    setCookie('saldo',nNieuwSaldo,100);
    window.history.go(0);
    eBedrag.value = "";
}
}
else{
    alert('U moet een correct bedrag ingeven');
}
...
```

Bespreking:

- zoals je ziet controleren we eerst het nieuwe saldo vooraleer het cookie te plaatsen
- we maken een waarschuwingstekst aan en laten die door de (nog aan te maken functie *toonWaarschuwing()*) plaatsen.



**alert()** popups vervelen zeer snel en onderbreken het programma nodeeloos:

- voor debuggen, gebruik **console.log**
- voor waarschuwingen, gebruik een speciaal HTML element voor berichten

- we helpen de gebruiker door het maximaal toegelaten bedrag in het veld te plaatsten

In de pagina zit een **div** met de **class** waarschuwing. Deze CSS class heeft het **display** rule op **none** staan: het element is onzichtbaar.

De functie *toonWaarschuwing()* plaatst een tekst in de **div** en toont hem opnieuw.

```
function toonWaarschuwing(msg){
    /*
    toont een waarschuwingstekst in divWarning
    msg = detekst
    */

    var eWarning = document.querySelector('.waarschuwing');
    eWarning.innerHTML = msg;
    eWarning.style.display = "block";
}
```

```
}
```

- we gebruiken **querySelector** om hiermee het eerste element met die class te refereren
- eerst wordt het bericht er in geplaatst
- dan maakt **display = "block"** de box zichtbaar: de *style rule* in het stylesheet heeft minder gewicht dan deze **inline** style

Opmerking:

- de techniek van berichten doen verschijnen/verdwijnen in de pagina wordt heel veel toegepast. Je kan je eigen varianten erop aanpassen aan de plaats en de omstandigheden.  
Zo kan je bijvoorbeeld meerdere berichten olijsten door ze te verpakken in **li** elementen die je dan in een **ul** plaatst.  
of je kan er een speciaal **label** element voor voorzien rechts van het inputveld.

Onze *cookieBank* is grotendeels af. als je zelf nog verbeteringen kunt aanbrengen, probeer maar.

De volledige code ziet er als volgt uit:

```
<script>

window.onload = function(){

    //DOM elementen
    var eOutput      = document.getElementById('output');
    var eKnopKrediet  = document.getElementById('krediet');
    var eKnopDebiet  = document.getElementById('debiet');

    //standaardwaarden
    var sMsg          = '';                // bericht aan gebruiker
    var sNaam          = 'nieuwe klant';    // standaard invulling naam
    var nSaldo         = 0;                // standaard saldo

    //test cookie
    if(getCookie('klantnaam')){
        //gekende klant

        var sNaam      = getCookie('klantnaam');
        //var nSaldo    = getCookie('saldo');
        var nSaldo = parseFloat(getCookie('saldo')).toFixed(2);
        //bericht
        sMsg += "Welkom " + sNaam + ", ";
        sMsg += "uw saldo bedraagt " + nSaldo + " Euro";

        //knop
```

```
        var eKnop      = maakKnop('Sluit rekening');
        eKnop.addEventListener('click',rekeningSluiten); //eventhandler
    }
    else{
        //eerste bezoek
        sMsg  = "Welkom beste bezoeker. ";
        sMsg += "Als u bij ons een nieuwe rekening opent, ontvangt u een startsaldo van
100 Euro!";
        //knop
        var eKnop      = maakKnop('Open rekening');
        eKnop.addEventListener('click',rekeningOpenen);
    }

    //inhoud output zone
    var dfBericht = document.createDocumentFragment();
    var eNl       = document.createElement('br');
    var tNode     = document.createTextNode(sMsg);
    dfBericht.appendChild(tNode);
    dfBericht.appendChild(eNl.cloneNode(false));
    dfBericht.appendChild(eNl.cloneNode(false));
    dfBericht.appendChild(eKnop);
    eOutput.appendChild(dfBericht);

    //event handlers voor + - knoppen
    eKnopKrediet.addEventListener('click', function(){ berekenen('+')});
    eKnopDebiet.addEventListener('click', function(){ berekenen('-')});

} //einde window.onload

/*****FUNCTIONIES*****/

function maakKnop(tekst){
    /*
    retournt een DOM button element
    */
    var eKnop      = document.createElement('button');
    var sTekst     = document.createTextNode(tekst);
    eKnop.appendChild(sTekst);
    eKnop.setAttribute('type','button');
    return eKnop
}

//-----

function rekeningOpenen(){
    //console.log('rekening openen')
    var sNaam = window.prompt("Uw naam, graag?", "");
    if (sNaam != ""){
        setCookie('klantnaam',sNaam,100); //nuttig_lib functie
        setCookie('saldo',100,100);
        window.history.go(0);
    }
}
```

```
}
//-----

function rekeningSluiten(){
    //vernietigt de rekening, dus de cookies
    //console.log('rekening sluiten')
    clearCookie('klantnaam');
    clearCookie('saldo');
    window.history.go(0);
}

//-----

function berekenen(bewerking){
    /*
    storting of geldafhaling
    @bewerking = een '+' of een '-' teken
    */
    var eBedrag    = document.getElementById('bedrag');
    var sBedrag    = eBedrag.value;
    var sSaldo     = getCookie('saldo');
    var sBericht   = "";
    var re         = /,/;
    sBedrag        = sBedrag.replace(re, '.');

    var nNieuwSaldo = 0;

    if(sSaldo!=null && sSaldo!=""){

        if(sBedrag!="" && !isNaN(sBedrag)){

            nSaldo        = parseFloat(sSaldo);
            nBedrag       = parseFloat(sBedrag);
            switch (bewerking) {
                case '+':
                    nNieuwSaldo = nSaldo + nBedrag;
                    break;
                case '-':
                    nNieuwSaldo = nSaldo - nBedrag;
                    break;
            }
            if (nNieuwSaldo<=0){
                var nMax          = nSaldo-1;
                sBericht          += "Uw saldo is onvoldoende om dit bedrag af te
halen. ";
                sBericht          += "U kunt maximaal " + nMax + " Euro afhalen.";
                eBedrag.value = nMax;
                eBedrag.focus();
                toonWaarschuwing(sBericht);
            }
        }
        else{
```

```

        setCookie('saldo',nNieuwSaldo,100);
        window.history.go(0);
        eBedrag.value = "";
    }
}
else{
    alert('U moet een correct bedrag ingeven');
}
}
else{
    //geen saldo = geen rekening
    var bOpenen = window.confirm('U heeft nog geen rekening geopend, nu even doen?');
    if(bOpenen===true){rekeningOpenen()}
}
}
//-----

function toonWaarschuwing(msg){
    /*
    toont een waarschuwingstekst in divWarning
    @msg = de tekst
    */
    //console.log(msg)
    var eWarning = document.querySelector('.waarschuwing');
    eWarning.innerHTML = msg;
    eWarning.style.display = "block";
}

</script>

```

## Web Storage

Deze cookiebank maakten we met traditionele cookies. Nu maken we een versie die gebruik maakt van HTML5 **Web Storage**. Het wordt enkel gemakkelijker.

Lees zeker nog eens de theorie over Web Storage na.

Sla je bestand op onder de gewijzigde naam *html5cookiebank.html*.

Infeite is de transitie naar het gebruik van Web Storage zeer eenvoudig: het enige wat we moeten doen is alle cookie functies vervangen door storage methods.

We beginnen er aan:

1. Verwijder de **script** tag naar *nuttig\_lib.js*
2. pas de code in de **window.onload** aan:

We moeten wel voorzichtig zijn: heeft de gebruiker een browser die Web Storage aankan? daarom bouwen we een *feature detection* als extra test in:

```

...
//standaardwaarden
var sMsg = ''; // bericht aan gebruiker

```

```
var sNaam          = 'nieuwe klant';    // standaard invulling naam
var nSaldo         = 0;                 // standaard saldo

if(localStorage){
    //kan deze browser dit script aan?
    console.log('localStorage OK');

    if(localStorage.klantnaam){
        //gekende klant
        var sNaam      = localStorage.klantnaam;
        var nSaldo     = parseFloat(localStorage.saldo).toFixed(2);
        //bericht
        sMsg += "Welkom " + sNaam + ",";
        sMsg += "uw saldo bedraagt " + nSaldo + " Euro";

        //knop
        var eKnop      = maakKnop('Sluit rekening');
        eKnop.addEventListener('click',rekeningSluiten); //eventhandler
    }

    else{
        //nieuwe klant, eerste bezoek
        sMsg = "Welkom beste bezoeker. ";
        sMsg += "Als u bij ons een nieuwe rekening opent, ontvangt u een
startsaldo van 100 Euro!";
        //knop
        var eKnop      = maakKnop('Open rekening');
        eKnop.addEventListener('click',rekeningOpenen);
    }
}
else {
    //gebruik cookies
}
...
```

Bemerk hoe we eenvoudig de storage items opvragen via hun property:

**localStorage.klantnaam**

3. pas de code in de functie *rekeningOpenen()* aan:

```
function rekeningOpenen(){
    var sNaam = window.prompt("Uw naam, graag?","");
    if (sNaam != ""){
        localStorage.setItem('klantnaam',sNaam);
        localStorage.setItem('saldo',100);
        window.history.go(0);
    }
}
```

4. pas de code in de functie *rekeningSluiten()* aan:

```
function rekeningSluiten(){
```

```
//vernietigt de rekening, dus de cookies
localStorage.clear();
window.history.go(0);

}
```

5. pas de code in de functie *berekenen()* aan:

```
function berekenen(bewerking){
    /*
    storting of geldafhaling
    @bewerking = een '+' of een '-' teken
    */
    var eBedrag    = document.getElementById('bedrag');
    var sBedrag    = eBedrag.value;
    var sSaldo     = localStorage.getItem('saldo');
    var sBericht   = "";
    var re         = /,/;
    sBedrag        = sBedrag.replace(re, '.');

    var nNieuwSaldo = 0;

    if(sSaldo!=null && sSaldo!=""){

        if(sBedrag!="" && !isNaN(sBedrag)){

            nSaldo        = parseFloat(sSaldo);
            nBedrag        = parseFloat(sBedrag);
            switch (bewerking) {
                case '+':
                    nNieuwSaldo = nSaldo + nBedrag;
                    break;
                case '-':
                    nNieuwSaldo = nSaldo - nBedrag;
                    break;
            }
            if (nNieuwSaldo<=0){
                var nMax = nSaldo-1;
                sBericht += "Uw saldo is onvoldoende om dit bedrag af te halen. ";
                sBericht += "U kunt maximaal " + nMax + " Euro afhalen.";
                eBedrag.value = nMax;
                eBedrag.focus();
                toonWaarschuwing(sBericht);
            }
            else{
                localStorage.setItem('saldo',nNieuwSaldo);
                window.history.go(0);
                eBedrag.value = "";
            }
        }
        else{

```



```
        alert('U moet een correct bedrag ingeven');
    }
}
else{
    //geen saldo = geen rekening
    var bOpenen = window.confirm('U heeft nog geen rekening geopend, nu even doen?');
    if(bOpenen===true){rekeningOpenen()}
}
}
```

Dit is voldoende. Test uit.

Het is duidelijk dat je beide methodes kunt gebruiken: als de browser het aankan gebruik storage, zoniet val terug op cookies. Kan je nu zelf deze combinatie maken?

### Taken

Maak nu:

- Taalkeuze

### 2.1.8 ISBN-10 validatie

#### Doel

Hoe valideer je codes zoals een ISBN nummer? Dergelijke codes – net als bankrekeningnummers – maken gebruik van een controlegetal dat moet kloppen volgens een welbepaald algoritme (*checksum*).

Bij deze eerste benadering maken we gebruik van een ruime selectie **String** methods.

#### Theorie

Lees de volgende theorie topics na:

- de **%** operator
- **NaN**, **isNaN()**
- **function**, structuren
- de **String** methods **substr**, **charAt**, **replace**
- **parseInt**, **parseFloat**

#### Duurtijd

<sup>1/2</sup> uur

#### Basisbestand

*isbn.html*.

#### ISBN-10 en ISBN-13

Het *International Standard Book Number (ISBN)* , of het *Internationaal Standaard Boeknummer* is een unieke code om de uitgave van een boek of document te identificeren.

Neem een willekeurig boek uit de kast en kijk op de achterzijde, dan bemerk je twee codes, bijvoorbeeld:

- ISBN-10: 1-59059-908-X
- ISBN-13: 978-1-59059-908-2

Eerst even wat uitleg: we citeren de Nederlandstalige Wikipedia:

*"Het **ISBN-10** nummer bestaat uit vier delen, vaak, maar niet noodzakelijkerwijs gescheiden door een scheidingsstreepje. De vier delen geven het volgende aan:*

- *de taal of het land van herkomst (Nederlandstalig = 90),*
- *de uitgever,*
- *het documentnummer en*
- *een controlecijfer.*

*Het controlecijfer is modulo 11, waarbij de '10' wordt vervangen door de letter 'X'.*

Het *controlecijfer* van een ISBN-10 code moet gelijk zijn aan de rest van de deling, som (van de eerste 9 getallen telkens vermenigvuldigd met de index van hun positie) / 11, de **modulo** 11 dus:

$$x_{10} = (1x_1 + 2x_2 + 3x_3 + 4x_4 + 5x_5 + 6x_6 + 7x_7 + 8x_8 + 9x_9) \bmod 11$$

*Tegenwoordig worden vrijwel alle boeken voorzien van een EAN-[streepjescode](#), waarin het ISBN terugkomt, wat de automatische verwerking van boeken sterk vergemakkelijkt.*

*Het **ISBN-13** nummer werd in 2007 ingevoerd omdat in sommige landen de 10 codes opraakten.*

*Het nieuwe ISBN bestaat uit 5 elementen (3 met variabele lengte, het eerste en het laatste met vaste lengte), die gescheiden worden door streepjes of spaties. Het is als volgt opgebouwd: ISBN 978-90-77287-02-6 of ISBN 978 90 77287 02 6.*

*Het controlegetal wordt berekend op basis van de voorgaande cijfers uit het ISBN. In bovenstaand voorbeeld is dat 6. Het controlegetal is altijd een cijfer*

*De berekening van het controlecijfer gaat volgens onderstaande formule.*

*In woorden: opgeteld worden het eerste cijfer (9 dus), het derde, het vijfde enzovoort, plus driemaal het tweede cijfer ( $3 \cdot 7 = 21$ ), driemaal het vierde enzovoort. Van de som wordt het laatste cijfer genomen (wiskundiger gezegd: er wordt modulair gerekend) en dat wordt van 10 afgetrokken.*

*Dat is het controlegetal en dat wordt het dertiende cijfer.*

$$x_{13} = (10 - (x_1 + x_3 + x_5 + x_7 + x_9 + x_{11} + 3 \cdot (x_2 + x_4 + x_6 + x_8 + x_{10} + x_{12}))) \bmod 10 \bmod 10$$

## Strategie

Wij willen een validatie maken voor zowel de ISBN-10 als een ISBN-13 code.

Het is duidelijk dat we beter een strategie bedenken om een ingegeven waarde te analyseren:

- bevat de waarde spaties of koppeltekens? verwijder ze
- is de lengte 10 of 13 karakters?
- voor ISBN-10: vervang een eventuele controlecijfer X door een 10
- is het code gedeelte wel numerisch?
- bereken het controlegetal volgens het juiste algoritme
- return een **boolean** volgens een goed of slecht resultaat

Je zal op het internet zeker een aantal oplossingen hiervoor vinden, toch blijf je beter even bij de les zodat je straks de nodige *skills* hebt om de code van de voordeur van de bank van Zwitserland te valideren...

## Het startbestand

Het startbestand bevat een eenvoudig formulier en een `script` tag met een `window.onload`. Een aantal globale variabelen worden ingevuld: een output zone, het formulier en het isbnveld (de name kw is nodig voor *isbndb.com*).

De referenties naar het formulier en het veld gebeuren op de DOM0 manier, dus niet met `getElementById()`, alhoewel dat evengoed had gekund.

Het bevat ook een array met enkele voorbeeldnummers die d.m.v. enkele statements opgelijst worden in de pagina. Je kan ze copy/pasten om ze te controleren. Gebruik ook de ISBN's die je vindt op de boeken in je bibliotheek als testcases. Het formulier stuurt een waarde door naar *isbndb.com*, een gratis service die een *paging* doet het gerelateerde boek te vinden. Dat lukt niet altijd, vooral Nederlandstalige boeken geven problemen.

## Het formulier

We moeten ervoor zorgen dat enkel geldige ISBNwaarden doorgestuurd worden door het formulier. Dat betekent dat we het `submit` Event moeten onderscheppen met een eigen event handler.

Ook dit proberen we *unobtrusive* te doen, dus zonder javascript in de HTML te plaatsen. We doen dit in de `window.onload`:

```
window.onload = function(){

    divOutput      = document.getElementById('output');
    frm             = document.frmISBN;
    isbnVeld        = frm.kw; //name kw noodzakelijk voor isbndb.com

    //lijst testwaarden
    var strNummers = "";
    for(var i=0;i<arrISBN.length;i++){
        strNummers += arrISBN[i] + "<br />";
    }
    divOutput.innerHTML = strNummers;

    //event handler voor form.submit onderscheppen
    frm.onsubmit = function(){
        console.log(this.name + ' submit hier');
        return false;
    }

}
```

Bespreking:

- we stellen het `onsubmit` event van het formulier gelijk aan onze anonieme functie

- de functie heeft op het einde een **return false**: daarmee stoppen we het **submit** event: de gegevens worden niet doorgestuurd
- de **console.log** is uiteraard tijdelijk om te controleren of het werkt



als tijdens de **submit** een foutje in je script sluipt, zal de **submit** toch doorgaan. Gebeurt dat, dan weet je dat je een fout moet opsporen!

De **Javascript console** toont nu bij een **submit** een bericht zoals "frmISBN submit hier".

We wijzigen de event handler:

```
...
frm.onsubmit = function(){
    var isbn = isbnVeld.value;
    var geldig = isValidISBN(isbn);
    console.log("waarde is een geldig isbnnummer: " + geldig);
    return false;
}
...
```

Bespreking:

- we lezen de waarde van het invulveld in de variabele *isbn*
- **voorlopig** tonen we het resultaat van de functie *isValidISBN()* in de console

### isValidISBN()

De functie *isValidISBN()* retournt een **boolean** die vertelt of een doorgegeven waarde een geldig ISBN is. Plaats de functie buiten de **window.onload**.

De basisstructuur moet zijn:

```
function isValidISBN(isbn){
    /*
    if(geldig){
        return true;
    }
    else
    {
        return false;
    }
    */
}
```

Om een waarde geldig te verklaren moet het argument *isbn* (de doorgegeven waarde) verwerkt en getest worden.

Eerst moeten alle spaties en koppeltekens verwijderd worden:

```
function isValidISBN(isbn){
  isbn = isbn.replace(/\s/gi,""); //verwijder alle spaties
  isbn = isbn.replace(/\-/gi,""); //verwijder alle koppeltekens
}
```

- daarvoor gebruiken we **String** method **replace()**.
- De variabele *isbn* wordt telkens opnieuw herbruikt.
- Het eerste argument is de te vervangen tekst, het tweede de nieuwe tekst.
- voor het eerste argument gebruiken we een **regular expression**:

**`/\s/gi`**

- **`/\s/`** is een patroon dat een **wit-ruimte karakter** zoekt, dus een spatie, een tab, een nieuwe lijn.
- De flags **gi** erna, zorgen ervoor dat het patroon global is, en dus alle spaties,tabs zoekt en hoofdletterongevoelig, hier van geen belang
- het tweede argument is een lege string: `""`. Die vervangt dus alle spaties
- ook voor de tweede vervanging gebruiken we een **regular expression**:

**`/\-/gi`**

- hier zoeken we een koppelteken: de regex is dus samen gesteld uit `/ /` met er in `\-`, een ge-escaped koppelteken.
- we vervangen **alle** koppeltekens door de flags **gi**

Nu kunnen we nagaan of de waarde de juiste lengte heeft ( 10 of 13). We beginnen met een ISBN-10:

```
function isValidISBN(isbn){
  isbn = isbn.replace(" ","","g");
  isbn = isbn.replace("-","","g");
  var l = isbn.length;
  if(l==10){
    return true;
  }
  else
  {
    return false;
  }
}
```

- We zetten de basisstructuur op met een **if else** en returnen in beide gevallen een **boolean**
- de **String** method **length** telt het aantal karakters van een stringwaarde

Nu kunnen we de *isbn* variabele opsplitsen in code en controlecijfer:

```
function isValidISBN(isbn){
  isbn = isbn.replace(" ", "", "g");
  isbn = isbn.replace("-", "", "g");
  var l = isbn.length;
  if(l==10){
    var basis9 = isbn.substr(0,9);
    var control = isbn.substr(9);

    //return true
  }
  else
  {
    return false;
  }
}
```

- voor *basis9* retournt de **String** method **substr()** het tekstdeel beginnend vanaf het eerste karakter (index **0**) met een lengte van 9 karakters
- voor *control* hebben we enkel het laatste karakter nodig: **substr(9)** bevat nu slechts één argument, de indexpositie: **9** is het tiende karakter. Een tweede argument - aantal karakters - wordt niet gegeven, dus krijgen we alles wat rest: slechts één karakter
- een **true** waarde returnen is wat voorbarig: we commentariëren het even

Van *basis9* kunnen we zeker eisen dat het numeriek is. Met numeriek bedoelen we dat het een **number** zal opleveren **als** het geconverteerd wordt, het heeft '*numeriek potentieel*', momenteel is het nog geen **number** , maar een **string**.

We hoeven het nog niet om te zetten naar een getal, maar er mogen geen andere karakters meer in voorkomen:

```
function isValidISBN(isbn){
  isbn = isbn.replace(" ", "", "g");
  isbn = isbn.replace("-", "", "g");
  var l = isbn.length;
  if(l==10){
    var basis9 = isbn.substr(0,9);
    var control = isbn.substr(9);
    if(!isNaN(basis9)){
      //return true;
    }
    else {
      return false;
    }
  }
  else
  {
    return false;
  }
}
```

- de functie `isNaN` evalueert eender welke waarde en retournt `true` als iets NIET numeriek is, dus retournt `!isNaN` een `true` als iets WEL numeriek is
- we zouden ook van onze lib functie `isGetal()` kunnen gebruik maken

Het controlecijfer moet ook gevalideerd:

```
...
if(!isNaN(basis9)){
    control = control.replace("X","10","gi");
    control = parseInt(control);
    if(!isNaN(control)){
        //return true;
    }
    else{return false;}
}
else
{
    return false;
}
...
```

- In het geval van ISBN-10 is het mogelijk dat er een 'X' als controlecijfer staat: dat wordt *eventueel* vervangen door een '10'
- we proberen met de conversiefunctie `parseInt()` het controlecijfer om te zetten naar een integer getal. Deze functie retournt altijd iets: ofwel een integer ofwel de waarde `NaN`
- de waarde `NaN` kan enkel getest worden met de functie `isNaN()`, nooit met een `==`

Nu we zeker zijn dat zowel `basis9` als `control` 'numeriek' zijn, kunnen we verder gaan met de validatie: het eigenlijke controle algoritme:

```
function isValidISBN(isbn){
    isbn = isbn.replace(" ","","g");
    isbn = isbn.replace("-","","g");
    var l = isbn.length;
    if(l==10){
        var basis9 = isbn.substr(0,9);
        var control = isbn.substr(9);
        if(!isNaN(basis9)){
            control = control.replace("X","10","gi");
            control = parseInt(control);
            if(!isNaN(control)){
                var sum = 0;
                for(var i=0;i<basis9.length;i++){
                    sum += parseInt(basis9.charAt(i)) * (i+1);
                }
                var modulo = sum % 11;
                return (control==modulo);
            }
        }
    }
}
```



```
        }
        else{return false;}
    }
    else {
        return false;
    }
}
else
{
    return false;
}
}
```

- met een **for** lus doorlopen we alle karakters van de **string** *basis9*. De method **charAt(i)** leest één karakter op een bepaalde positie in de **string**.
- elk karakter wordt omgezet in een integer, vermenigvuldigd met de lusindex+1 en dit product wordt opgeteld bij de variabele *sum*
- de rest van de deling van *sum* door 11 is de *modulo* 11: JS gebruikt daarvoor de **%** operator
- de **boolean** waarde van de vergelijking *modulo* versus controlcijfer wordt gereturned

Test dit nu uit met een aantal ISBN-10 waarden.

Nu passen we de Event Handler voor het form aan zodat deze de gereturnde waarde doorgeeft aan de **submit** van het formulier. Op die manier bepaalt een geldig ISBN direct of er ge-submit wordt:

```
window.onload = function(){
    divOutput      = document.getElementById('output');
    frm             = document.frmISBN;
    isbnVeld        = frm.kw;

    //lijst testwaarden
    var strNummers = "";
    for(var i=0;i<arrISBN.length;i++){
        strNummers += arrISBN[i] + "<br />";
    }
    divOutput.innerHTML = strNummers;

    //event handler voor form.submit onderscheppen
    frm.onsubmit = function (){
        var isbn      = isbnVeld.value;
        var geldig    = isValidISBN(isbn)
        //console.log("waarde is een geldig isbnnummer: " + geldig);
        return geldig;
    }
}
```

Om de gebruiker er ook van op de hoogte te brengen dat er iets mis is - de reden waarom hij niet doorgestuurd wordt naar isbndb.com, tonen we een foutbericht.

We doen dat door een `label` element in te voegen naast het input element:

```
<p>Vul een ISBN-10 of ISBN-13 nummer in: <input type="text" name="kw" id="kw" /><input  
  type="submit" value="Valideer"/>  
<label class="error" id="fout">ongeldig ISBN nummer</label></p>
```

Daarbij hoort een *selector* in het interne stylesheet:

```
...  
label.error{  
  display:none;  
  color:red;  
}  
...
```

Het `label` is dus oorspronkelijk niet zichtbaar. Nu tonen/verbergen we de fout na de validatie:

```
window.onload = function(){  
  
  divOutput      = document.getElementById('output');  
  labelFout      = document.getElementById('fout');  
  frm            = document.frmISBN;  
  isbnVeld       = frm.kw; //name kw noodzakelijk voor isbndb.com  
  
  //lijst testwaarden  
  var strNummers = "";  
  for(var i=0;i<arrISBN.length;i++){  
    strNummers += arrISBN[i] + "<br />";  
  }  
  divOutput.innerHTML = strNummers;  
  
  isbnVeld.onfocus = function(){  
    labelFout.style.display = "none";  
  }  
  
  //event handler voor form.submit onderscheppen  
  frm.onsubmit = function (){  
    var isbn      = isbnVeld.value;  
    var geldig   = isValidISBN(isbn)  
    //console.log("waarde is een geldig isbnnummer: " + geldig);  
    if(geldig===false){  
      labelFout.style.display = "inline";  
    }  
    return geldig;  
  }  
}
```

```
}
```

**Bespreking:**

- we maken een referentie naar het `label` element
- in de `frm.onsubmit` testen we de waarde van *geldig* en indien `false` tonen we het `label`
- van zodra de cursor weer in het `input` veld staat - `isbnVeld.onfocus` – verbergen we de foutboodschap opnieuw

Het gebruik van **vaste** foutboodschappen in een formulier die je toont/verbergt, is een makkelijke manier van werken.

Klaar!

**Taak:**

- **ISBN-13**  
Dezelfde functie `isValidISBN()` moet nu ook voorzien voor ISBN-13 codes.

## 2.1.9 Formulier: Birdy Airways

### Doel

Een formulier *client-side* valideren en dynamisch maken

### Theorie

Lees de volgende theorie topics na:

- Formulieren
- DOM intro

### Duurtijd

4 uur

### Basisbestanden

*birdy.html, js\_form Ontvanger.html*

### Het formulier

Bekijk de broncode van het basisbestand: het bevat een formulier dat zijn gegevens doorstuurt naar *js\_form Ontvanger.htm*: zo kunnen we controleren **of** er **iets** doorgestuurd wordt. Zorg ervoor dat beide bestanden in dezelfde map zitten.

### Universele ontvanger van gegevens

Dit JS script schrijft alle *naam=waarde* paren van eender welk form op de pagina

Gebruik een GET als method

```
optVluchtType = retour
vertrekL = aAHO
vertrekdatum = 12/12/2008
aankomstL = aBRI
retourdatum = 13/12/2008
volwassenen = 1
kinderen = 0
peuters = 0
passagier1 = Jan+Vandorpe
```

Het **form** element controleert de breedte van het formulier dat voor de rest enkel bestaat uit **fieldset**, **label** elementen en controls. Er is voldoende ruimte voorzien aan de rechterkant voor foutberichten.

Er zijn twee strategieën om een formulier te valideren: ofwel valideren we op het ogenblik van de **submit**, de meest gebruikte strategie, ofwel valideren we elke control tijdens/net na de input. Ook deze mogelijkheid wordt toegepast maar je moet er voorzichtig mee zijn. Combineren van de twee is eveneens mogelijk.

Wij kiezen voor de klassieke manier: tijdens de **submit**.

Dit formulier heeft ook wat velden die afhankelijk zijn van de keuze van een ander veld: zo is een retourvlucht onnodig als je maar een enkel ticket koopt. De passagiersnamen zijn afhankelijk van het aantal passagiers uiteraard. Om die reden moeten we de controls ook dynamisch kunnen manipuleren en daar zal de validatie zich aan moeten aanpassen. Maar eerst doen we alsof er niets aan de hand is en gaan van start met de validatie.

We beginnen met een `script` tag en een `window.onload` event, enkele referenties, een eventhandler voor het formulier met een testberichtje:

```
<script type="text/javascript">
window.onload = function (){

    //DOM referenties
    var eFrmVlucht      = document.frmVlucht;
    var eRetour         = document.frmVlucht.optRetour;
    var eEnkel          = document.frmVlucht.optEnkel;
    var eVolw           = document.frmVlucht.volwassenen;
    var eKind           = document.frmVlucht.kinderen;
    var ePeut           = document.frmVlucht.peuters;

    //formulier submit
    eFrmVlucht.addEventListener('submit', function (e){
        e.preventDefault();
        console.log('wie is this?' + this.name);

    });

}
</script>
```

- de DOM referenties naar het formulier en de controls doen we deze keer volgens de DOM0 standaard die gebruik maakt van de `name` attributen (niet `id` !):

```
document.nameForm en
document.nameForm.nameControl
```

Je mag uiteraard evengoed een modernere methode toepassen. Voor de meeste controls zijn de methods **equivalent** met dien verstande dat sommige een `id` gebruiken, andere een `name`:

```
document.frmVlucht.kinderen
document.getElementById('kinderen')
document.querySelector('#kinderen')
```

Voor sommige elementen, zoals een groep radio buttons zijn deze methodes **niet** equivalent.

Bijvoorbeeld (controleer de HTML), de groep "enkel/retour": deze bestaat uit twee `input type="radio"` elementen met een verschillend `id` en dezelfde `name`. Dus zijn

```
document.frmVlucht.optVluchtType en
```

`document.getElementById('optRetour')` niet gelijk, maar `document.frmVlucht.optRetour` wel aan hierboven

- bemerk het argument `e` in de anonieme functie: dat stelt het `submit` event zelf voor. Onmiddellijk daarna roepen we de method `e.preventDefault()` op om de **standaardactie** van het event te verhinderen, m.a.w. de `submit` gaat niet door
- het is interessant om te zien dat `this` in dit geval het `form` element is

### de submit eventhandler

Eenmaal je gecontroleerd hebt dat de event handler werkt, vervolledigen we die en laten de validatie van het formulier doen door een aparte functie `valideer()`. Het resultaat van deze functie zal bepalen of de `submit` doorgaat of niet:

```
eFrmVlucht.addEventListener('submit', function (e){
    e.preventDefault();
    var bValid = valideer(this);
    console.log('formulier ' + this.name + ' valideert ' + bValid);
    if(bValid===true) this.submit();
})
```

We maken onmiddellijk een rudimentaire functie `valideer()` buiten de `window.onload`:

```
function valideer(frm){
    var bValid= true;
    return bValid;
}
```

Bespreking:

- deze functie is de "*hoofdcontroleur*": hij bepaalt of de `submit` doorgaat of niet door de **Boolean** die hij doorgeeft aan de `submit` event handler
- de functie heeft een argument `frm`. Dat is het `this` object doorgegeven door de event handler. Hier is dat het form element
- de functie retournt de var `valid`, die voorlopig op `true` is gezet

### Velden valideren via hun **Class**

Voor elk veld bepalen we

- a) of het gecontroleerd moet worden
- b) volgens welke validatieregel(s) er moet gecontroleerd worden

We doen dat door aan elk te controleren veld, in de HTML, een `class` toe te kennen. Bijvoorbeeld, als een tekstveld *verplicht* is geven we de `class required` aan het `input` element. Moet een veld een *geldig emailadres* zijn, voegen we de `class email` toe. Voor elke verdere validatieregel geven we een `class` erbij.

We doen dit nu: geef de `class required` aan de elementen `username` en `wachtwoord`. Is er al een `class`, dan zet je die er gewoon bij gescheiden door een spatie:

```
...  
<input type="text" name="username" id="username" class="text required" title="uw  
  gebruikersnaam (emailadres)" />  
...
```

(Je vindt een overzicht van alle classes van de formelementen achteraan dit project)

Omdat er meerdere validatieregels zullen zijn, verzamelen we die in een **objectvariabele** `oFouten`. Maak deze globale variabele aan vóór de `window.onload`.

`oFouten` bevat voor elke validatieregel een property dat zelf een object is met twee properties: `msg` en `test`. De `msg` is het foutbericht dat we doen verschijnen, de `test` is een method, een functie die de test uitvoert.

De structuur is dus zo:

```
var oFouten = {  
  validatieregell1: {  
    msg: "...bericht...",  
    test: function () {  
      //testfunctie retournt een boolean  
    }  
  },  
  validatieregell2: {  
    msg: "...bericht...",  
    test: function () {  
      //testfunctie retournt een boolean  
    }  
  },  
  ...  
}
```

We starten met een eerste validatieregel: *required*.

```
...  
var oFouten = {
```

```
required:{
    /* enkel voor input type="text|password" */
    msg: "verplicht veld",
    test: function (elem){
        return elem.value != "";
    }
}
...

```

Bespreking:

- het object *oFouten* heeft een property *required* dat zelf twee properties *msg* en *test* heeft
- *test* is een **method**, dat wil zeggen een **property** die een **functie** bevat.
  - die moet een **true/false** returnen
  - we controleren of het element niet leeg is

*Zal deze test wel werken voor een keuzelijst of een checkbox?*

Vooraleer we kunnen proberen moeten we de functie *valideer()* verder uitwerken:

```
function valideer(frm){

    var bValid = true; //optimistisch geen fouten

    //lus doorheen alle form elementen van het formulier
    for(var i=0;i<frm.elements.length;i++){
        //verwijder vorige foutboodschappen
        hideErrors(frm.elements[i]);
        //valideer veld
        var bVeld = valideerVeld(frm.elements[i]);
        console.log("het element %s met name %s valideert %s", ↵
                    frm.elements[i].nodeName,frm.elements[i].name, bVeld);
        if(bVeld ===false){ bValid = false; }
    }

    return bValid;
}

```

Bespreking:

- de var *bValid* wordt optimistisch op true gezet. Deze variabele wordt aan het einde gereturnet. maar een ketting is zo sterk als zijn zwakste schakel, dus als 1 test faalt, faalt de volledige controle, daarom
- overlopen we alle controls in het formulier door met een **for** loop te lussen doorheen de collectie **frm.elements**. Dit is een ingebakken DOM0 verzameling die **alle controls** (tekstvelden, checkboxes, keuzelijsten,...) in een **form** element bevat.
  - voor elke control verwijderen we eerst alle vorige foutboodschappen met *hideErrors*, een functie die we nog moeten maken



- voor elke control voeren we een functie *valideerVeld* uit. Die functie moet ook een **boolean** returnen. We maken deze functie onmiddellijk hierna.
- als er één control **false** valideert, valideert het volledige formulier **false**



Het **console.log** statement bevat %s parameters. Dat zijn **printf** statements die simpelweg door een string vervangen worden. Die strings moeten, in de juiste volgorde en gescheiden door komma's, erop volgen.  
Zowel FireBug als de Javascript consoles van IE en Chrome ondersteunen deze syntax

In dit voorbeeld zal de output van de **console.log** de opeenvolging tonen van form controls die doorlopen wordt. Je krijgt dus iets te zien in de aard van *"Het element INPUT met de name username valideert false"*

Deze log is heel nuttig en toont je voor elke element of hij wel/niet valideert. Je bemerkt ook dat de **fieldset** elementen ook overlopen worden. Aangezien ze echter geen fouten **class** bevatten, kan dat geen kwaad.

Nu maken we de functie *valideerVeld()* die één enkele control valideert:

```
function valideerVeld(elem){
    //valideert één veld volgens zijn class

    var aFoutBoodschappen = [];

    for (var fout in oFouten){
        var re = new RegExp("(^|\\s)" + fout + "(\\s|$)"); //regex
        // fouten class aanwezig?
        if(re.test(elem.className)){
            var bTest = oFouten[fout].test(elem);
            console.log("het element %s met name %s wordt gevalideerd voor %s: %s", ↵
                        elem.nodeName,elem.name, fout, bTest);
            if(bTest === false){
                aFoutBoodschappen.push(oFouten[fout].msg);
            }
        }
    }

    if(aFoutBoodschappen.length>0){
        showErrors(elem, aFoutBoodschappen);}
    return !(aFoutBoodschappen.length>0);
}
```

Bespreking:

- de functie heeft als argument een DOM element *elem*
- we maken een leeg **array** *aFoutBoodschappen* aan

- we overlopen met een **for in** loop alle eigenschappen van het object *oFouten*. Zoals je weet is een eigenschap een validatieregel, bijvoorbeeld *required*.
  - we maken een **regular expression** aan die de naam van de validatieregel bevat. Deze *regular expression* zoekt de naam binnen een **string**: hij kan eender waar staan, vooraan, midden, achteraan
  - we gebruiken de Javascript method **test** om de *regular expression* te zoeken in de **className** van het element
  - als de validatieregel voorkomt als **class** van het element, dan voeren we de method *test* uit van die eigenschap, dus bijvoorbeeld, als de **string** "required" voorkomt in de **class** van het element, dan wordt de method **oFouten['required'].test()** uitgevoerd op het element
  - als deze method een **false** returnt – de waarde van het veld is dus niet goed - dan gebruiken we de array method **push** om het bericht van de fout toe te voegen aan het *aFoutBoodschappen* array.

Dus in ons voorbeeld, wordt in dat geval **fouten['required'].msg** toegevoegd aan het array

- als nadien blijkt dat het array *aFoutBoodschappen* items bevat – bedenk dat een element meerdere fouten kan genereren - gebruiken we de functie *showErrors* om alle fouten te tonen voor dat element. Deze functie moeten we nog maken.
- uiteindelijk returnen we een **false/true** , die teruggaat naar *valideer()*

Maak nu nog even twee functie skeletten voor *showErrors* en *hideErrors*, zodat we kunnen uittesten:

```
function showErrors(elem, errors){
    //toont alle fouten voor één element
}
function hideErrors(elem){
    //verbergt alle foutboodschappen
}
```

Probeer nu even uit en bekijk de output in de javascript console (FireBug, IE console, Chrome console).

Het is tijd om de functie *showErrors* en *hideErrors* te vervolledigen.

```
function showErrors(elem, aErrors){
    /*
    toont alle fouten voor één element
    @elem          element, te valideren veld
    @aErrors       array, fouten voor dit element
    */
```

```
var eBroertje = elem.nextSibling;
if(!eBroertje || !(eBroertje.nodeName == "UL" && eBroertje.className == "fouten" )){
    eBroertje = document.createElement('ul');
    eBroertje.className = "fouten";
    elem.parentNode.insertBefore(eBroertje, elem.nextSibling);
}
//plaats alle foutberichten erin
for(var i=0;i<aErrors.length;i++){
    var eLi = document.createElement('li');
    eLi.innerHTML = aErrors[i];
    eBroertje.appendChild(eLi);
}
}
```

Bespreking:

- de functie heeft de argumenten *elem*, de control waarvoor fouten bestaan, en *errors*, een array van foutberichten
- eerst zoeken we de *nextSibling* van het element
- indien die niet bestaat of hij is geen *ul* element met de *class* "fouten", dan
  - maken we een *ul* element en geven die de *class* "fouten"
  - voegen die in als eerste *sibling* van de control ( er onmiddellijk na dus)
- we overlopen het *aErrors* array en plaatsen elk bericht als *li* element in de *ul*
- het uitzicht van deze *list* bepalen we via het stylesheet

En *hideErrors*:

```
function hideErrors(elem){
    /*
    verwijdert alle fouten voor één element
    @elem          element, te valideren veld
    */
    var eBroertje = elem.nextSibling;
    if(eBroertje && eBroertje.nodeName == "UL" && eBroertje.className == "fouten" ){
        elem.parentNode.removeChild(eBroertje);
    }
}
```

Bespreking:

- de functie heeft het argument *elem*, de control waarvoor fouten bestaan
- ook hier zoeken we de *nextSibling* van het element
- en als die bestaat en hij is een *ul* met de *class* "fouten", dan verwijderen we die

## Een getal

Tijd om een andere validatieregel toe te voegen.

Voeg voor de invulvelden *volwassenen*, *kinderen*, *peuters* een `class` "aantal" toe.

Het veld *volwassenen* is dus "required" EN "aantal", terwijl *kinderen* en *peuters* enkel "aantal" hebben.

We breiden het *oFouten* object uit met een nieuwe eigenschap:

```
var oFouten = {  
  
  required:{  
    ...  
  },  
  aantal:{  
    msg: "getal verwacht",  
    test: function(elem){  
      //aantal test enkel de inhoud als getal als er een inhoud is  
      if(elem.value != ""){  
        return !isNaN(elem.value) && elem.value>0;  
      } else {return true;}  
    }  
  }  
}
```

Opmerkingen:

- LET OP de komma na de eigenschap *required* !
- de property *aantal* heeft ook een *msg* en een *test*
- *aantal* is op zichzelf geen verplicht veld, daar zorgt *required* voor. Om die reden testen we de inhoud enkel als die niet leeg is.
- `isNaN` retournt `true` als een waarde niet kan geconverteerd worden naar een getal. `!isNaN` geeft dus `true` als iets een getal is.
- terzelfdertijd zorgen we er ook voor dat die waarde groter dan 0 is

Probeer dit even uit.

## Een datum valideren

de datumvelden *vertrekdatum* en *retourdatum* moeten een datum van het formaat `d/m/jjjj` bevatten. Eerst mag je aan beide elementen de `class` "datum" toevoegen.

We voegen ook een eigenschap aan het *oFouten* object toe:

```
var oFouten = {  
  
  ...  
  },  
  datum:{  
    msg: "datum ongeldig (d/m/yyyy)",  
  }  
}
```

```
test: function(elem){
    // dd/mm/yyyy
    var re_datum = /^[0-9]|[0,1,2][0-9]|3[0,1])\/([\d]|1[0,1,2])\/\d{4}$/;
    if(elem.value != ""){
        return re_datum.test(elem.value);
    } else {return true;}
}
}
```

Bespreking:

- de *test* voor deze validatieregel maakt gebruik van een **regular expression**. Om de werking te begrijpen, zie het theoretisch gedeelte
- deze regex is een patroon **d/m/yyyy**
- ook deze regel is op zich niet verplicht dus testen we enkel als er iets in het vak getypt wordt
- de return waarde volgt als een boolean uit de functie **re\_datum.test(elem.value)**

### Een keuzelijst valideren

Uit de keuzelijst *vertrekl* moet een luchthaven gekozen worden. De keuzelijst heeft een eerste optie "*kies vertrek luchthaven*" waarvan de **value** leeg is.



Gebruik bij een eerste optie met een vraag ("*maak uw keuze*") bij voorkeur een lege waarde: die is gemakkelijk te valideren, zowel aan client- als aan serverside.

Zet nu eerst de **class required** in het element en probeer. Tot onze verbazing werkt de validatie onmiddellijk!

Niet moeilijk te begrijpen want de test *required* controleert op een lege string.

### Enkel of Retour?

Bij een enkele reis moet er geen *retourDatum* zijn en die moet dan ook niet gevalideerd worden. We kunnen de control *retourDatum*

- **desactiveren (disable):**  
⇒ *geen naam=waarde* paar wordt doorgestuurd naar de server
- **verbergen:**  
⇒ een *leeg naam=waarde* paar wordt doorgestuurd naar de server, bv **voornaam=**
- **verwijderen:**  
⇒ *geen naam=waarde* paar wordt doorgestuurd naar de server

In elk geval moet de situatie omkeerbaar zijn als de gebruiker van mening verandert. Standaard staat de optie *Retour* aangevinkt en is dit vak dus *required*.

De eenvoudigste oplossing is de eerste, voornamelijk omdat dan geen *naam=waarde* paar wordt doorgestuurd. We kunnen ook combineren met de tweede mogelijkheid om duidelijker te zijn.

Eerst moeten we event handlers instellen voor de radiobuttons *Retour* en *Enkel*, dat doen we in de `window.onload`:

```
...
    eRetour.addEventListener('click', function(){           //radio retour
        vluchtType(this.value);
    });
    eEnkel.addEventListener('click', function(){           //radio enkel
        vluchtType(this.value);
    })
...
```

Bespreking:

- we verwijzen naar een functie *vluchtType* in een anonieme functie. waarom niet onmiddellijk verwijzen naar *vluchtType*? Omdat we enkel zo in staat zijn een argument door te geven
- het argument is de *value* van de radiobutton (*this*) waarop geklikt wordt

Nu maken we de functie *vluchtType*:

```
function vluchtType(sType){
    /*
    des-/activeert de retourdatum
    @sType String, 'enkel'|'retour'
    */

    var eRetourDatum      = document.getElementById('retourdatum');
    var eLabelRetourdatum = document.getElementById('labelRetourdatum');

    if(sType == "enkel"){
        //geen retourdatum
        var sClass      = eRetourDatum.className.replace("required","");
        eRetourDatum.className      = sClass;
        eRetourDatum.disabled       = true;
        eRetourDatum.style.display  = "none";//verberg label
        eLabelRetourdatum.style.display  = "none";//verberg label
    }
    else{
        eRetourDatum.disabled       = false;
        var sClass                  = eRetourDatum.className + " required";
        eRetourDatum.className      = sClass;
        eRetourDatum.style.display  = "inline";
        eLabelRetourdatum.style.display  = "inline";
    }
}
```

```
}  
}
```

Bespreking:

- het argument *sType* bevat dus de waarde van het aangeklikte keuzerondje, "enkel" of "retour"
- We leggen twee referenties
  - naar de datum control
  - en naar zijn label
- als een enkele reis gekozen wordt
  - verwijderen we de **class** "required" uit zijn attribuut. We doen dat door middel van de **String** functie **replace**: het woord "required" wordt vervangen door een lege string. Als "required" niet aanwezig is, dan gebeurt er niets.

We kunnen de **className** niet zomaar overschrijven, want er zijn andere classes aanwezig en die moeten blijven staan

- de retourdatum control wordt **disabled** gezet: nu kan je er de focus niet meer op zetten en dus niets meer invullen
  - hij wordt verborgen door zijn **display** eigenschap op **none** te zetten
  - het label wordt ook verborgen
- voor een retourreis
  - zetten we eerst **disabled** af, want anders kan je er geen wijzigingen in aanbrengen
  - voegen we de **class** "required" toe
  - en tonen zowel control als label opnieuw

Test nu even uit.

## Verschillende luchthavens

We willen dat de aankomstluchthaven verschilt van de vertrekluchthaven.

Een specifieke (*custom*) test is hier de kortste weg. We maken een property *aankomstL* in ons *oFouten* object:

```
...  
,  
aankomstL:{  
  msg: "aankomstluchthaven moet verschillen van vertrekluchthaven",  
  test: function (elem){  
    //CUSTOM TEST: aankomstLuchthaven moet verschillen van vertrekluchthaven  
    if(elem.value != ""){  
      var aL = elem.value; //aankomstluchthaven  
      var vL = document.frmVlucht.vertrekL.value; //vertrekLuchthaven
```

```
        return !(aL==vL);
    }else{ return true;}}
...
```

Bespreking:

- ook hier testen we enkel als er een waarde in het veld zit
- de test vergelijkt de waarden van de twee keuzelijsten en retournt de tegengestelde boolean waarde van de test: dus

```
return !(aL==vL);
```

is hetzelfde als

```
if(aL==vL){
    return false;
}
else{
    return true;
}
```

## TimeVortex?

Is het mogelijk dat je terugkeert vóór je vertrokken bent? *No way...* enkel *Dr. Who* weet hoe dat moet, wij controleren best of de retourdatum na de vertrekdatum valt.

We verwachten minstens een verschil van één dag tussen vertrek- en retourdatum.

Voeg een `class retourDatum` toe aan het `#retourdatum` element.

Voeg nu een fout `retourDatum` toe aan het `oFouten` object:

```
...
retourDatum:{
    msg:"retourdatum > vertrekdatum",
    test: function (elem){
        /* CUSTOM TEST: retourdatum minstens 1 dag na vertrekdatum
        * 1 dag later = 86400000 ms
        * beide formaten dd/mm/yyyy
        */
        if(elem.value != ""){
            var aD = elem.value; //aankomstdatum
            var vD = document.frmVlucht.vertrekdatum.value; //vertrekdatum
            var dag = 86400000; //dag in ms
            //retourdatum
            var arrD1 = aD.split('/');
            var D1 = new Date(parseInt(arrD1[2]),parseInt(arrD1[1])-1,↵
                parseInt(arrD1[0]));
            //vertrekdatum
            var arrD2 = vD.split('/');
```



```
var D2 = new Date(parseInt(arrD2[2]),parseInt(arrD2[1])-1,↵
    parseInt(arrD2[0]));

var verschil = D1-D2;
return (verschil>=dag);
} else {return true;}
}
...
```

#### Bespreking:

- dit is een *custom test*: **specifiek** bedoeld om die twee datumvelden te vergelijken. Het is mogelijk om een method te maken die toepasselijk is voor eender welke controls, maar dat zou ons in deze basiscursus te ver brengen
- ook deze test gebeurt slechts als er een inhoud is in de retourdatum, het is niet zijn taak die inhoud te verplichten
- we lezen de twee datums in, uiteraard zijn dat gewone **String** variabelen
- De test moet de twee datums vergelijken. Deze keer volstaat een 'patroon' – een regex – niet. We zijn verplicht er echte **Date** objecten van te maken
  - **Date** objecten bestaan uit milleseconden, daarom moet ons minimaal verschil – een *dag* - ook uitgedrukt worden in milliseconden
  - we maken een array *arrD1* met de **split()** functie die de waarde van het datumveld (in het formaat *dd/mm/yyyy*) in 3 items splitst gebaseerd op het / karakter.
  - *arrD1[0]* bevat dus het dagdeel, *arrD1[1]* het maanddeel en *arrD1[2]* het jaargedeelte
  - daarmee construeren we een **Date** object, er zorg voor dragend dat we echte getallen (**parseInt**) gebruiken voor deze vorm van de constructor:

```
new Date(jaarGetal, maandGetal, dagGetal)
```

- het *maandGetal* moet met 1 verminderd worden omdat januari de maand **0** is
- als je de geldigheid van deze datum wil controleren gebruik **console.log(D1.toLocaleDateString())**, die moet exact dezelfde datum teruggeven
- We doen hetzelfde voor de tweede datum
- het verschil tussen deze twee **Date** objecten is uitgedrukt in milliseconden
- als dat verschil minder is dan één dag, dan valideert deze test **false** en wordt de waarschuwing getoond

#### De passagiersnamen

Er moet minstens één passagiersnaam ingevuld worden, dit veld bestaat reeds. Maar indien er meerdere passagiers zijn moeten ook hun namen ingevuld en gevalideerd worden.

We hebben daarbij een aantal mogelijkheden:

- de velden bestaan maar zijn onzichtbaar gemaakt
- de velden bestaan reeds maar zijn niet aanklikbaar (*disabled*)
- de velden worden *on-the-fly* aangemaakt

Elk heeft voor- en nadelen, maar omdat we onzeker zijn betreffende het aantal, kiezen we voor de laatste mogelijkheid.

We hebben ook een "ankerplaats" voorzien waarin we de extra passagiernamen zullen plaatsen: onderaan het formulier zit een `span` element met de `id` "extras".

Plaats deze event handlers voor het `blur` event in de `window.onload`:

```
...
eVolw.addEventListener('blur',passagierNamen);
eKind.addEventListener('blur',passagierNamen);
ePeut.addEventListener('blur',passagierNamen);
...
```

Het `blur` event heeft plaats als de `focus` van een veld verdwijnt: je klikt ergens anders, je gebruikt de Tab toets om de cursor te verplaatsen.

De functie `passagierNamen()` die het aantal extra invulvelden zal controleren:

```
function passagierNamen(){
/*
 * controleert dynamisch het aantal passagiernamen
 * passagier 1 blijft steeds bestaan
 * veronderstelt de aanwezigheid van een SPAN#EXTRAS
 */

//aantallen velden
var eVolw          = document.frmVlucht.volwassenen;
var eKind          = document.frmVlucht.kinderen;
var ePeut          = document.frmVlucht.peuters;
var eExtras        = document.getElementById('extras');

if(!valideerVeld(eVolw)||!valideerVeld(eKind)||!valideerVeld(ePeut)){
    return;
}

//getalwaarden
var nVolw          = parseInt(eVolw.value); //ook required
var nKind          = (eKind.value=="")?0:parseInt(eKind.value);
var nPeut          = (ePeut.value=="")?0:parseInt(ePeut.value);
var nPassagiers    = nVolw + nKind + nPeut;
var sInhoud        = "";

for(var i=0;i<nPassagiers-1;i++){
    sInhoud += maakPassagier(i+2);
}
eExtras.innerHTML = sInhoud;
```

```
}

```

Bespreking:

- we referen de aantallen velden en ook de *target span*
- we maken gebruik van onze *valideerVeld* functie om foutieve inhoud op te sporen en af te breken indien nodig
- de getalwaarde van elk veld wordt ingelezen met *parseInt*, eventueel een leeg veld vervangen door een 0
- er wordt een totaal gemaakt
- daarna vullen we de span op met evenveel label en input elementen als het totaal. Dat gebeurt door een *innerHTML* string te laten genereren door nog een functie: *maakPassagier()*.

```
function maakPassagier(i){
/*
 * maakt een label en een passagiernaamveld
 * @i integer
 * @return string HTML
 */
  var str = "";
  str += "<label for=\"passagier\"+i+\">Passagier "+i+":</label>\n";
  str += "<input type=\"text\" name=\"passagier\"+i+\"\" id=\"passagier\"+i+\"\" ";
  str += "title=\"voornaam en famlilienaam van de passagier\" "
        class=\"extra_p text required\" />\n";
  str += "<br />\n";
  return str;
}
```

Wat gebeurt hier:

- we bouwen een stringvariabele op die een *label* en een *input* element aanmaakt
- het *i* getal wordt verwerkt in de *id* van de *input*

Het dynamisch aanpassen van het aantal passagiersnamen is nu volledig.

De validatie van de nieuwe namen gebeurt automatisch door de aanwezigheid van de class "required".

Overzicht van de validatie classes die de formelementen moeten hebben. De classes in *italic* zijn classes die niets met validatie te maken hebben maar gebruikt worden voor de opmaak van het veld:

Element id	Class(es)
------------	-----------

---

username	<i>text</i> required
wachtwoord	<i>text</i> required
onthoulogin	
optVluchtType	required
vertrekL	required
vertrekdatum	<i>text</i> required datum
aankomstL	required aankomstL
retourdatum	<i>text</i> required datum retourDatum
volwassenen	<i>number</i> required aantal
kinderen	<i>number</i> aantal
peuters	<i>number</i> aantal
passagier1	<i>text</i> required

## 2.1.10 Quiz van de week

### Doel

Een clientside wekelijkse quiz maken waarvan de gegevens geleverd worden in JSON formaat

### Theorie

Lees **eerst** de volgende theorie topics na:

- JSON

### Basisbestanden

*quiz.html, quiz.json*

### Duurtijd

4 uur

### De opdracht:

De gebruiker test zijn parate kennis aan de hand van een aantal multiple choice vragen.

De quiz wijzigt elke week, qua inhoud en aantal vragen. Sommige vragen hebben ook een figuur, andere niet. Als de gebruiker een keuze maakt, wordt er onmiddellijk aangegeven of het antwoord goed/slecht was en wordt een begeleidende tekst getoond. Een score wordt bijgehouden en getoond na de laatste vraag.

In een eerste fase bouwen we de vragen onder elkaar op, daarna maken we ze meer dynamisch waarbij elke vraag gevolgd wordt door een andere, tenslotte de eindscore.

De gegevens voor de quiz worden aangeleverd in JSON formaat, Je krijgt hier een voorbeeld JSON string in het bestand *quiz.json*. Je mag echter veronderstellen dat in productieomstandigheden de JSON string dynamisch geproduceerd wordt door een serverside script (PHP, .Net, Java).

Deze quiz is ook niet beveiligd: iemand die code HTML en Javascript kan lezen, kan het correcte antwoord achterhalen, het is gewoon voor de "fun".

Het basisbestand bevat reeds heel wat CSS.

## JS PF project: Quiz van de week

De gebruiker test zijn parate kennis aan de hand van een aantal multiple choice vragen. De quiz wijzigt Als de gebruiker een keuze maakt, wordt er onmiddellijk aangegeven of het antwoord goed/slecht was.

De gegevens voor de quiz worden aangeleverd in JSON formaat. Je krijgt hier een voorbeeld JSON : beveiligd: iemand die code HTML en Javascript kan lezen, kan het correcte antwoord achterhalen.

### Quiz van week 9



### Stappenplan:

Er zijn verschillende stappen te ondernemen in dit project:

- de JSON String interpreteren
- een HTML structuur bouwen die vragen, media en antwoorden bevat
- een feedback tekst tonen voor elk antwoord
- een functie die de score bijhoudt en toont
- en een interface maken die telkens één vraag met een keer toont met hyperlinks "Vorige", "Volgende" etc..

### JSON lezen

We veronderstellen dat je eerst de theorie gelezen hebt en dus weet wat JSON is.

Open het bestand *quiz.json*. Het bevat een JSON string variabele *jsonQuiz* met alle gegevens zoals die ook door een serverside script geleverd zouden worden.

De structuur van die gegevens is als volgt:

```
var jsonQuiz = '{  
  
  "datum": "2012-02-28",  
  "vragen": [  
    {
```

```
        "vraag"      : "Welke film ...",
        "antwoorden" : ["Rundskop", ...],
        "correct"    : 2,
        "tekst"       : "A separation ...",
        "media"       : ["oscar.jpg"],
        "url"         : "http://..."
    },
    ...
]
}'
```

Bemerk de *apostrophen* die de volledige JSON omvatten.

### Opmerkingen:

1. de structuur hierboven is wat verduidelijkt met *nieuwe lijnen*, maar het is belangrijk dat je dat NIET doet. Invoegen van "*Enters*" e.d. zullen fouten opleveren. De variabele moet één lange string blijven zonder speciale karakters.
2. JSON in een bestand leveren is een beetje een *artificiële situatie*: meestal wordt een JSON string ge-retourned door een serverside script (PHP, Java, .Net) dat antwoordt op een Ajax call. Die stuurt dit in één keer door naar de Ajax eventhandler. De string in *quiz.json* is echter wél een geldig voorbeeld van wat je zult ontvangen.

Om met deze gegevens iets te kunnen doen moeten ze geïnterpreteerd worden, *ge-parsed*.

We voegen twee **script** tags toe aan *quiz.html*:

```
<script src="quiz.json" type="text/javascript"></script>
<script>

var oQuiz      = {};

window.onload = function(){

    oQuiz      = JSON.parse(jsonQuiz);
    console.log(oQuiz.vragen[0].vraag);

    var eQuiz   = document.getElementById('quiz');
    eQuiz.appendChild(maakDfQuiz());

} //einde window.onload

function maakDfQuiz(json){
    // return documentFragment voor quiz
    // @json      JSON string

    var dfQuiz   = document.createDocumentFragment();

    return dfQuiz;
}
```

```
</script>
```

Bespreking:

- de eerste `script` leest het bestand `quiz.json`. Omdat daarin de variabele `jsonQuiz` staat, wordt deze geladen
- de tweede `script` tag bevat
  - een global variabele `oQuiz` die momenteel een leeg object bevat
  - een `window.onload`
- in de `window.onload` converteren we eerst de JSON variabele met `JSON.parse()` naar het Javascript object `oQuiz`
- we testen `oQuiz` door met een `console.log` statement de `vraag` property van de eerste vraag te tonen
- we leggen een referentie naar het element `#quiz`
- we *appenden* de returnwaarde van de functie `maakDfQuiz()`: dit zal een `documentFragment` zijn, een DOM structuur die geen equivalent HTML element heeft, het is eerder een "brochette" van HTML elementen. Het dient specifiek om grotere stukken HTML gemakkelijk als één geheel in de DOM te kunnen plaatsten.

Bekijk dus de *javascript console* met een webtool: daar moet je de vraag zien verschijnen: als je dat ziet is het object `oQuiz` aanwezig.

### Native JSON support?

We gaan er hier van uit dat je browser *native support* (ingebouwde functionaliteit) heeft voor het `JSON` object en dat de methods `parse()` en `stringify()` aanwezig zijn. Maar is dat ook zo bij de browser van de *client*?

Het zou wel eens kunnen dat hij nog met een oudere browser zit zonder ingebouwde JSON ondersteuning. Dan moeten we aan *feature detection* doen en *graceful degrade*.

Dus redeneren we misschien in deze termen:

```
if(window.JSON){
  //native support
  var oQuiz = JSON.parse(jsonQuiz);
}
else {
  //geen native support-> laad een json library
  ...
}
```

Neen, het is eenvoudiger. De library `json2.js` (startbestanden) neemt dit voor zijn rekening: het eerste wat de library doet is zelf testen voor de aanwezigheid van een



JSON object, als dat *native* aanwezig is laat hij alles met rust, zoniet maakt hij het object met zijn methods aan.

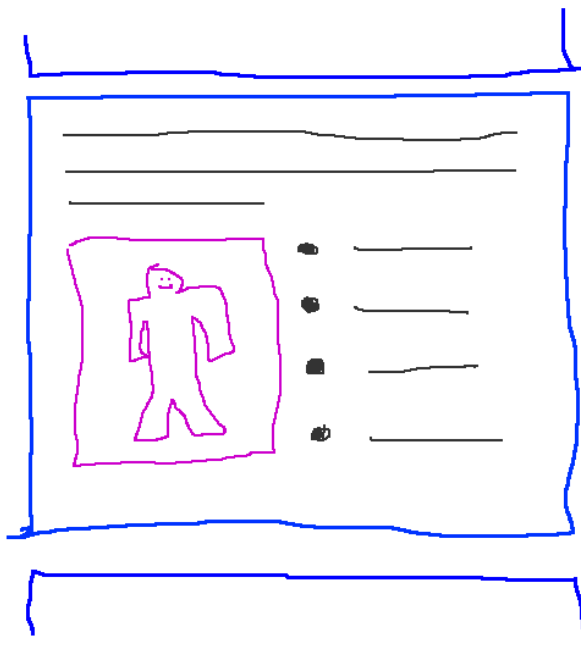
Daarom sluiten we deze library in vóór de andere scripts:

```
<script src="json2.js"></script>
<script src="quiz.json"></script>
<script>
  var oQuiz    = {};
  window.onload = function(){
  ...
```

Nu is er **altijd** JSON ondersteuning.

### De HTML structuur van de quiz bouwen

De basisstructuur die we voor ogen hebben is een opeenvolging van "Vraagcontainers" (**div**), die elke een "vraag" bevatten (**p**), één of geen "images"(**img**) en een "bolletjeslijst" (**ul**) met mogelijke "antwoorden".



De basis HTML structuur zien we dus zo:

```
div - |
      | - p
      | - img
      | - ul - |
                | - li
                | - li
                | - li
```

De layout voor deze structuur laten we volledig over aan de CSS (die reeds aanwezig is). We bouwen nu de functie *maakDfQuiz()* verder uit.

## Een titel

```
function maakDfQuiz(){
  //return documentFragment voor quiz

  //quiz structuur: DIV met IMG's en UL

  var dfQuiz    = document.createDocumentFragment();
  var aVragen   = oQuiz.vragen;
  var nVragen   = aVragen.length;

  //Titel
  var dDatum    = new Date(oQuiz.datum);
  var sTitel    = "Quiz van week " + dDatum.getWeek(); //augmentation method
  var eTitel    = document.createElement('h2');
  eTitel.appendChild(document.createTextNode(sTitel));
  dfQuiz.appendChild(eTitel);

  return dfQuiz
}
```

### Bespreking:

- we beginnen eerst met het **documentFragment** *dfQuiz* aan te maken. Dit is gewoon een lege container.
- voor het gemak maken we een variabele *aVragen* die het array met alle quizvragen zal bevatten
- we tellen het aantal vragen in *nVragen*
- eerst maken we een titel aan waarin het nummer van de week vervat zal zijn:
  - daarvoor maken we een nieuw **Date** object *dDatum* aan de hand van de string *oQuiz.datum*. Zo zie je dat het gebrek aan een *Date* data type in JSON, snel opgelost wordt.
  - alhoewel **Date** een heleboel methods heeft, bestaat er geen ingebouwde method *getWeek* die je het weeknummer geeft: we zullen nu onmiddellijk een *augmentation* van **Date** schrijven om deze method te krijgen
- we maken een **h2** element aan waarin de gecombineerde tekst geplaatst wordt en dit element wordt toegevoegd aan het **documentFragment**
- aan het einde van de functie *return* we het **documentFragment**, zodat het in de **window.onload** kan gebruikt worden.

Voeg volgende code toe aan het **script** element:

```
<script>
//=====GLOBALS ETC=====
//Date augmentation voor method getWeek
Date.prototype.getWeek = function() {
```

```
var onejan = new Date(this.getFullYear(),0,1);
return Math.ceil((((this - onejan) / 86400000) + onejan.getDay()+1)/7);
}
var oQuiz = {};

//=====window onload=====

window.onload = function(){
  ...
}
```

Dit is een *augmentation* van het **Date** object: vanaf nu is een method *getWeek* beschikbaar op eender welk **Date** object. Hoe augmentation werkt, wordt uitgelegd in de advanced cursus, we gaan hier niet verder in op.

## De vraagcontainers

Voor de volgende code is het goed als de je structuur van ons object *oQuiz* voor ogen houdt. We gaan verder na de titel:

```
...
dfQuiz.appendChild(eTitel);
// de Vragen
for (var i=0;i<nVragen;i++){
  //DIV als container vr elke vraag
  var oVraag = aVragen[i];
  var eVraagContainer = document.createElement('div');
  eVraagContainer.setAttribute("class","vraag");
  eVraagContainer.setAttribute("data-index",i);
  eVraagContainer.id = "vraag_" + i;
  var eVraag = document.createElement('p');
  var sVraag = document.createTextNode(oVraag.vraag);
  eVraag.appendChild(sVraag);
  eVraagContainer.appendChild(eVraag);

  //eventuele images

  //Antwoorden

  //Feedback

  dfQuiz.appendChild(eVraagContainer);
} //einde lus Vragen

return dfQuiz;
}
```

Bespreking:

- we lussen met een **for** doorheen het array *aVragen* waarvan er *nVragen* zijn
  - voor het gemak bewaren we het object dat in *aVragen[i]* zit in de var *oVraag*
  - voor elke vraag maken we een **div** element
  - die een CSS **class** "vraag" meekrijgt
  - die ook een dataset attriboot "**data-index**" meekrijgt die de index van de vraag bevat
  - en ook nog een **id** als "vraag\_n" om later de container makkelijk te kunnen identificeren
  - in deze **div** voegen we nog een **p** element toe die de *vraag* property van het object *oVraag* als tekst krijgt
  - we voorzien reeds zones voor de volgende taken
  - aan het eind van alle vragen worden de containers telkens toegevoegd aan het **documentFragment**

Als je deze code uitvoert moet je reeds een titel en alle vragen te zien krijgen: test uit.

### Eventuele images

De property *media* van een *vraag* object is normaal een bestandsnaam, maar de *media* property kan ook volledig afwezig zijn.

Als er een image is, voegen we die ook toe aan de vraagcontainer:

```
...
eVraagContainer.appendChild(eVraag);

//eventuele images
if(oVraag.media){
    var eImage = document.createElement('img');
    eImage.src = "../images/" + oVraag.media;
    eVraagContainer.appendChild(eImage);
}
//Antwoorden

//Feedback

dfQuiz.appendChild(eVraagContainer);
} //einde lus Vragen
return dfQuiz;
}
```

Bespreking:

- eerst kijken we of er een *media* property is. Een lengte testen van een niet bestaande property zou een fout geven
- daarna maken we een **img** element aan waarin we de bestandsnaam samenvoegen met het pad naar de bestanden (pas aan aan je eigen noden)

- we voegen het `img` element toe aan de `div.vraag`

## De Multiple Choice antwoorden

We maken een *unordered list* voor de antwoorden:

```
...
//Antwoorden
var eAntwoordenLijst = document.createElement('ul');
var aAntwoorden      = oVraag.antwoorden;
var nAntwoorden      = aAntwoorden.length;
//lus doorheen alle antwoorden
for (var k=0;k<nAntwoorden;k++){
    var eAntwoord = document.createElement('li');
    eAntwoord.appendChild(document.createTextNode(aAntwoorden[k]));
    eAntwoordenLijst.appendChild(eAntwoord);
}
eVraagContainer.appendChild(eAntwoordenLijst);

//Feedback
...
```

Bespreking:

- we maken een `ul` element
- de var `aAntwoorden` bevat het array met alle mogelijke antwoorden
- we lussen er doorheen en maken voor elk een `li` element dat aan de `ul` toegevoegd wordt
- de `ul` wordt toegevoegd aan de `div`

Nu kunnen we uittesten.

## Antwoorden als hyperlinks

Nu hebben we een basisstructuur maar geen enkele interactiviteit. Om de quiz te kunnen evalueren moeten we weten wat er geantwoord wordt. We moeten:

- de antwoorden als hyperlink voorzien
- een event handler voor die links registreren die
  - de vraag evalueert
  - onmiddellijk feedback geeft of het goed/slecht was met
  - de begeleidende tekst
- de goede/slechte antwoorden tellen en een eindscore tonen

Eerst moeten we hyperlinks voorzien in de `li` elementen. We wijzigen het gedeelte *//antwoorden* in de functie *maakDfQuiz()*:

```
...
for (var k=0;k<nAntwoorden;k++){
    var eAntwoord = document.createElement('li');
    var eLink      = document.createElement('a');
    eLink.setAttribute("href","#");
    eLink.setAttribute("data-index",k);
    eLink.appendChild(document.createTextNode(aAntwoorden[k]));
    eAntwoord.appendChild(eLink);
    eAntwoordenLijst.appendChild(eAntwoord);
}
...
```

Bespreking:

- nu staat de tekst in een hyperlink
- bemerk dat je een **href** attribuut **moet** geven aan het **a** element, anders gedraagt het zich niet als hyperlink
- we geven ook een dataset attribuut "**data-index**" mee aan elke hyperlink waarin de index van het antwoord staat

## Event handler voor de hyperlinks

Om een antwoord te evalueren hebben we een eventhandler nodig en een manier om te weten of dit het juiste antwoord was. Daarnaast zullen we nog gegevens nodig hebben over de vraag zelf.

We wijzigen de hyperlinks opnieuw en slaan in elke link op of dit de juiste is of niet (security kan ons niet schelen):

```
...
//lus doorheen alle antwoorden
var aAntwoorden = oVraag.antwoorden;
var nAntwoorden = aAntwoorden.length;
var nCorrect    = oVraag.correct;

for (var k=0;k<nAntwoorden;k++){
    var eAntwoord = document.createElement('li');
    var eLink      = document.createElement('a');
    eLink.setAttribute("href","#");
    eLink.setAttribute("data-index",k);
    var bCorrect   = (k==nCorrect);
    eLink.setAttribute("data-correct",bCorrect);
    eLink.addEventListener("click",function (e){evalVraag(e, this)});
    eLink.appendChild(document.createTextNode(aAntwoorden[k]));
    eAntwoord.appendChild(eLink);
    eAntwoordenLijst.appendChild(eAntwoord);
}
eVraagContainer.appendChild(eAntwoordenLijst);
...
```

Bespreking:

- in *oVraag.correct* staat een index van het juiste antwoord, dat gaat in de var *nCorrect*
- de var *bCorrect* bevat een **boolean** als test of de index van dit antwoord (1<sup>ste</sup>, 2<sup>de</sup>, ... hyperlink) overeenkomt met *nCorrect*
- deze **boolean** waarde slaan we op in het attribuut **data-correct** van de hyperlink

In elke hyperlink kan je dus nu zien of dit antwoord juist is of niet. Zoals gezegd, verwachten we van ons publiek niet dat ze deze code (kunnen) nakijken.

- we registreren een event handler voor de **click** van de hyperlink. Omdat we argumenten willen meegeven aan onze eventhandler *evalVraag*, moeten we die verpakken in een anonieme functie.
  - de anonieme functie heeft een argument *e*: dat is het event zelf. we geven die door aan *evalVraag* als eerste argument
  - In deze anonieme functie stelt **this** de hyperlink zelf voor. We moeten die ook meegeven als argument, samen met het *oVraag* object

## het antwoord valideren

Nu schrijven we een eerste versie van de event handler *evalVraag()*:

```
function evalVraag(e,link){  
  //evalueert een quizvraag  
  e.preventDefault();  
  console.log(link.dataset["correct"]);  
  
}
```

Bespreking:

- de functie heeft 3 argumenten:
  - *e*, het **click event** zelf
  - *link*, het hyperlink element (vanuit **this**)
  - *oVraag*, het vraag object waar dit één van de antwoorden van is
- **e.preventDefault()** verhindert de standaard actie van de hyperlink ('ergens naar toe gaan')
- het **console.log** statement test even of we het **data-correct** attribuut van de hyperlink kunnen lezen

Probeer dit uit: bekijk de console en bekijk de HTML: kan je de **data-correct** attributen van de links zien?

Nu we weten of dit antwoord het juiste was, kunnen we verder handelen: we tonen een tekst die bestaat uit het woord "CORRECT/FOUT" + de begeleidende tekst die in *oVraag.tekst* zit.

```
function evalVraag(e,link){
```

```
//evalueert een quizvraag
e.preventDefault();

//zoek de parent Vraagcontainer van de lienk
var eVraag = function(node){
    while(node.parentNode){
        if(node.parentNode.className=='vraag'){
            return node.parentNode;
        }
        node=node.parentNode
    }
}(lienk)

var nVraag      = parseInt(eVraag.dataset["index"]);
var nAntwoord   = parseInt(lienk.dataset["index"]);
var sCorrect    = lienk.dataset["correct"];
var bJuist      = (sCorrect=="true");
var sJuist      = bJuist?"correct":"fout";

}
```

#### Bespreking:

- eerst moeten we een referentie leggen naar de "Vraag" container waarin dit aangeklikte antwoord zit: strikt genomen is dat de tweede parent (**lienk.parentNode.parentNode**).  
De kans zit er echter in dat we in de toekomst de structuur van onze vraag wijzigen en daarom gebruiken we hier een generische (maar wat complexere) manier van werken:
  - de var *eVraag* zal de *parent* bevatten
  - deze *parent* wordt gereturned door de anonieme functie met argument *node*
  - de functie wordt onmiddellijk uitgevoerd dank zij de dubbele accolades achteraan, waarin het element *lienk* wordt doorgegeven (aan *node*)
  - via een **while** lus
    - met voorwaarde (**node.parentNode**)  
betekent: heeft deze node nog een *parentNode*?
    - als deze *parentNode* ook nog een CSS **class** heeft met waarde 'vraag'? dan is dit diegene die we zoeken: **return** die
    - zoniet wordt de *parentNode* de huidige node en gaan we een trapje hoger in de node-stamboom tot we vinden wat we zoeken
- de var *nVraag* leest het getal dat in het attribuut **data-index** van de vraagcontainer staat
- de var *nAntwoord* leest het getal dat in het attribuut **data-index** van de antwoord-link staat



- de var *sCorrect* leest de tekst dat in het attribuut **data-correct** van de antwoord-link staat
- de var *bJuist* is een **boolean** die de tekst van *sCorrect* evalueert
- de var *sJuist* maakt een relevante tekst aan die verder gebruikt zal worden

In een volgende stap zullen we feedback geven aan de gebruiker over zijn antwoord. In het object *oQuiz* is er een tekst voorzien die wat uitleg geeft over het juiste antwoord. Die moeten we eerst nog toevoegen aan de vraagcontainer.

Pas de functie *maakDfQuiz* aan:

```
...
}

    eVraagContainer.appendChild(eAntwoordenLijst);

    //Feedback
    var eFeedback = document.createElement('p');
    eFeedback.setAttribute("class", "feedback");
    eFeedback.appendChild(document.createTextNode(oVraag.tekst));
    eVraagContainer.appendChild(eFeedback);

    dfQuiz.appendChild(eVraagContainer);
} //einde lus Vragen
...
```

Bespreking:

- er wordt een extra element **p.feedback** toegevoegd aan elke **div.vraag**
- de tekstuele inhoud ervan komt uit **oQuiz.tekst**

In de CSS staat echter een stylerule **p.feedback {display:none}**. De container wordt dus niet getoond.

Nu kunnen we ervan gebruik maken in *evalVraag()*:

```
...
var sJuist          = bJuist?"correct":"fout";

var eFeedback      = eVraag.querySelector("p.feedback");
var eCorrect       = eFeedback.querySelector("span.correct");
var sResultaat     = "antwoord " + ++nAntwoord + " was " + sJuist;

if(eCorrect){
    eCorrect.innerHTML = sResultaat; //er was al een span
}
else{
    eFeedback.innerHTML = "<span class='correct'>" + sResultaat + "
                           </span><br>" + eFeedback.innerHTML;
```

```
    }  
    eFeedback.style.display = "block";  
}
```

Bespreking:

- vanuit *eVraag* gaan we op zoek naar *p.feedback* die deze bevat. Bemerkt dat je *querySelector* *contextueel* kunt gebruiken: niet vanuit het *document* maar vanuit *eVraag* zelf
- we zoeken ook een *span.correct* binnen *p.feedback*. Misschien bestaat deze niet
- de string *sResultaat* bevat de tekst of het antwoord goed/fout is
- als een *span.correct* gevonden werd zetten we die tekst er in, zoniet
- maken we een *span.correct* met de tekst en plaatsen die bij de bestaande tekst in *eFeedback*
- tenslotte tonen we de feedbackcontainer door zijn *display* op "block" te zetten. De positionering in de CSS zorgt ervoor dat deze tekst onderaan elke vraag staat

## Score bijhouden

Aan het eind moet de gebruiker zijn score zien:

*aantal juiste antwoorden / totaal aantal vragen (aantal beantwoorde vragen)*

Het zal ook mogelijk zijn terug te keren en een vraag anders te beantwoorden.

Om de scores bij te houden zullen we het *oQuiz* object uitbreiden met een array *scores* waarin we – in volgorde van de vragen – een *boolean* waarde zetten voor elk antwoord.

We voegen dit array toe onmiddellijk **na** het opvullen van het object, dus helemaal bovenaan ons script:

```
window.onload = function(){  
  
    oQuiz      = JSON.parse(jsonQuiz);  
    oQuiz.score = [];  
  
    var eQuiz  = document.getElementById('quiz');  
    eQuiz.appendChild(maakDfQuiz());  
  
} //einde window.onload
```

Doe dit niet vóór het opvullen met de *JSON.parse*, want anders wordt het overschreven.

Tijdens de evaluatie van elke vraag – in *evalVraag()* - houden we nu de juiste/foute score bij voor elke vraag:

```
...
var sJuist          = bJuist?"correct":"fout";
oQuiz.score[nVraag] = bJuist;
console.log(oQuiz.score);
var eFeedback       = eVraag.querySelector("p.feedback");
```

In het array `oQuiz.score` gebruiken we nu de index van de vraag om een `true/false` te noteren.

Met het console statement kan je het Array controleren na het beantwoorden van een vraag.

Uiteindelijk willen we het resultaat zien. We zullen dit tonen in een extra `div` die als laatste container in de quiz staat. We moeten die eerst nog aanmaken in `maakDfQuiz()`. Voeg deze code toe onderaan deze functie:

```
...
} //einde lus Vragen

//extra container eindscore
var eScoreContainer = document.createElement('div');
eScoreContainer.setAttribute("class","vraag");
eScoreContainer.id     = "score";
dfQuiz.appendChild(eScoreContainer);

return dfQuiz;
}
```

Bespreking:

- we maken een `div` element
- met dezelfde `class` als de andere vragen (komt straks van pas)
- met een `id` "score" om hem makkelijk te kunnen identificeren
- we voegen die toe als laatste element van het `document.Fragment`

Nu vind je onderaan een lege "vraag".

Om de score te kunnen tonen hebben we een aparte functie nodig die we zullen oproepen na elke evaluatie:

```
function eindScore(){
  //update de tekst met eindscore in eindscorecontainer
  //@ oQuiz global

  var nVragen      = oQuiz.vragen.length;
  var nBeantwoord   = 0;
  var nJuist        = 0;
  var eScore        = document.querySelector("#score");
```

```
for (var i=0;i<oQuiz.score.length;i++){
    if(typeof(oQuiz.score[i])!= "undefined") {++nBeantwoord}
    if(oQuiz.score[i]==true) {++nJuist}
}

var sScore = "<p class='score'>Uw score is " + nJuist+ "/" + nVragen + " <br>(" + nBeantwoord + " beantwoord)</p>";
eScore.innerHTML = sScore;
}
```

Bespreking:

- de functie leest in *nVragen* het aantal vragen uit het *oQuiz* object
- het element *eScore* is de extra div die we net aanmaakten
- het aflezen van het aantal juiste antwoorden uit *oQuiz.score* is niet zo eenvoudig als je zou denken: veronderstel dat de user de vragen met index 0,1 en 3 beantwoord en dus de 3<sup>de</sup> vraag overslaat. Dan bevat het array de volgende inhoud:

```
[true, false, undefined, true]
```

het aantal beantwoorde vragen is dus 3, niet 4. We moeten dus de **undefined** waarden eruit filteren, daarom

```
if(typeof(oQuiz.score[i])!= "undefined") {++nBeantwoord}
```

- de uiteindelijke tekst wordt samengesteld en via **innerHTML** in de score container geplaatst. De nodige style rules zijn reeds aangemaakt.

Nu nog enkel de functie oproepen vanuit *evalVraag()*:

```
...
eFeedback.style.display = "block"; //toon feedback

eindScore();
}
```

## User Interface

Tot nu toe staan alle vragen onder elkaar op de pagina. We willen dit wijzigen zodat de vragen als een stapel kaarten op elkaar liggen – eerste vraag boven – en er via een hyperlink naar de volgende vraag geklikt kan worden. Aan het eind zien we de eindscore.

We hebben sowieso navigatie nodig, hier opnieuw in de vorm van hyperlinks. Die moeten aangemaakt worden in de quiz container, dus in *maakDfQuiz()*. Omdat dit een navigatiebalkje wordt dat voor (bijna) alle vragen identiek is, is de code in *maakDfQuiz()* kort: voeg toe tussen "antwoorden" en "feedback".

```
...
for (var k=0;k<nAntwoorden;k++){
    ...
}
eVraagContainer.appendChild(eAntwoordenLijst);

//NAVBAK MET VORIGE VOLGENDE
eVraagContainer.appendChild(maakNav(i));
//FEEDBACK BERICHT
var eFeedback = document.createElement('p');
...
```

- De returnwaarde van een functie *maakNav(i)* wordt hier aan de vraagcontainer toegevoegd
- het argument *i* is de index van de vraag

Nu maken we de functie *maakNav()*:

```
function maakNav(index){
    //returnt navbalkje met vorige volgende hyperlinks
    //@index      de index van deze vraag

    var nMaxIndex      = oQuiz.vragen.length;
    var nVorigeIndex    = index-1;
    var nVolgendeIndex  = index+1;

    var eNav            = document.createElement('div');
    eNav.setAttribute("class","nav");

    //eerste vraag geen Vorige hyperlink
    if(index>0){
        var eVorige      = document.createElement('a');
        eVorige.setAttribute("href","#");
        eVorige.setAttribute("title","Vorige vraag:" + nVorigeIndex);
        eVorige.innerHTML = "<<";
        eVorige.addEventListener("click",function (e){toonVraag(nVorigeIndex)});
        eNav.appendChild(eVorige);
    }
    if(index<=nMaxIndex){
        var eVolgende     = document.createElement('a');
        eVolgende.setAttribute("href","#");
        eVolgende.setAttribute("title","Volgende vraag: " + nVolgendeIndex);
        eVolgende.innerHTML = ">>";
        eVolgende.addEventListener("click",function (e){toonVraag(nVolgendeIndex)});
        eNav.appendChild(eVolgende);
    }
}
```

```
        return eNav;
    }
```

Bespreking:

- deze functie retournt een `div.nav` met twee hyperlinks: "Vorige" "Volgende"
- we bepalen het aantal vragen (`nMaxIndex`), de index van de vorige vraag (`nVorigeIndex`), de index van de volgende vraag (`nVolgendeIndex`)
- we maken een `div` met de `class` "nav"
- de eerste vraag hoeft geen "Vorige" (we willen geen lus maken)
- voor alle andere vragen maken we de twee hyperlinks aan en voegen die toe aan `div.nav`
- bemerk de eventhandler `toonVraag()` die de respectieve index meekrijgt als argument

Nu hebben alle vragen een extra navbalkje meegekregen, maar ze staan nog steeds onder elkaar. Om een "kaartendeck" te krijgen moeten we één iets wijzigen in de CSS. Ga naar het interne stylesheet en zoek de rule `.vraag`. Wijzig zijn `position` in "absolute":

```
<style type="text/css">
#quiz {
    position:relative;
}
.vraag {
    position:absolute;
    top:30px;
    left:0;
...

```

Alle "kaartvragen" liggen nu op elkaar, maar de laatste – de eindscore – ligt bovenaan...

We zorgen ervoor dat de eerste vraag boven ligt door hem een `z-index` te geven. Voeg dit ene lijntje toe aan `maakDfQuiz()`:

```
...
//DIV VRAAG CONTAINER
    var oVraag          = aVragen[i];
    var eVraagContainer = document.createElement('div');
    eVraagContainer.setAttribute("class","vraag");
    eVraagContainer.setAttribute("data-index",i);
    eVraagContainer.id    = "vraag_" + i;

    if(i==0){eVraagContainer.style.zIndex=10;}

    var eVraag          = document.createElement('p');
...

```

Door de eerste vraag een **zIndex** van 10 te geven, komt deze "bovendrijven". Via zijn hyperlink kunnen we dan verder navigeren naar de volgende vraag.

We moeten nog de eventhandler *toonVraag()* maken:

```
function toonVraag(index){
    /*
    toont een vraag dr de z-index hoog te zetten
    @index index in de collection 'vraag' elementen (inclusief de eindscore container)
    */

    var eVragen = document.querySelectorAll(".vraag");
    for(var i=0;i<eVragen.length;i++){
        if(i==index){
            eVragen[i].style.zIndex=10;
        }
        else{
            eVragen[i].style.zIndex=0;
        }
    }
}
```

Bespreking:

- de functie krijgt een indexgetal doorgegeven als argument: deze "vraag" zal hij tonen door hem een hoge **z-index** te geven
- eerst maken we een *collection* van elementen met de **class** "vraag" via **querySelectorAll**
- we lussen doorheen deze collection en zetten alle **zIndex**- en op 0 met uitzondering van de gevraagde index: die krijgt 10

Taken:

Maak nu:

- De JS PF eind oefening



### 3 COLOFON

**Sectorverantwoordelijke:**

**Cursusverantwoordelijke:**

Jean Smits

**Didactiek en lay-out:**

Jan Vandorpe

**Medewerkers:**

Jan Vandorpe  
Adinda Mattens

**Versie:**

juli 2012

**Nummer dotatielijst:**