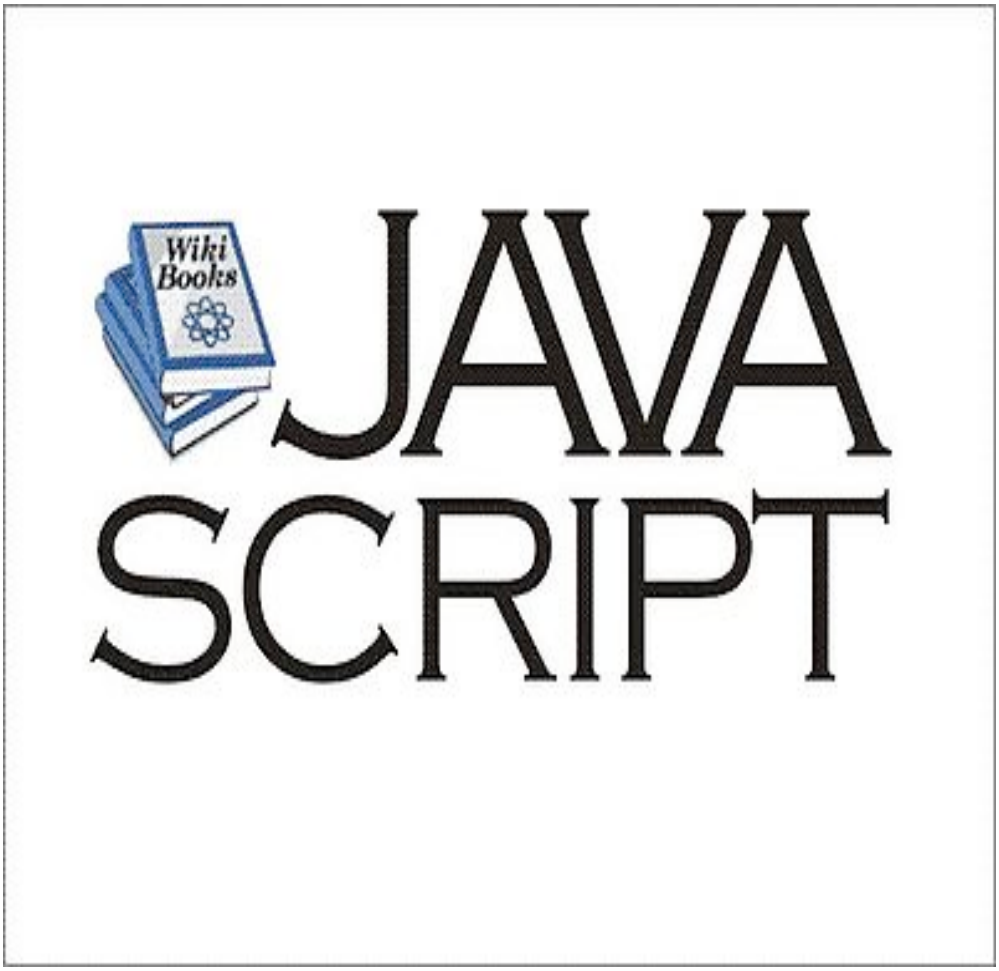
Cursus Javascript



ICT Academie Nova College

Inhoudsopgave

Inleiding

Het is verstandig om:

1. De voorbeeldcodes uit te proberen, zodat je ziet wat er gebeurt.
2. Eerst te lezen, dan nog een keer lezen, en als je het dan nog niet begrijpt vragen.
3. Geen spelletjes te spelen / hyves / msn en gewoon aan je werk te gaan: je hebt je tijd nodig.

Wat is Javascript?

Javascript is een scripttaal en wordt gebruikt om op webpagina's interactie met de gebruiker te voorzien, effecten te gebruiken en formuliergegeven s te controleren. Javascript-instructies zijn in HTML-documenten ingebed, daarom moet je een basiskennis van HTML hebben voordat je vlot aan de slag kan met Javascript. In deze cursus wil ik je een idee geven van de mogelijkheden die Javascript biedt om attractieve, interactieve pagina's te bouwen.
Javascript is echter bijzonder uitgebreid, vandaar dat je in deze cursus ook veel voorbeeld-codes zal vinden.
Javascript kent heel wat ingebouwde objecten en functies, we zullen er heel wat leren kennen in de loop van deze cursus.

Javascript – Java

Javascript is niet hetzelfde als Java.
Hieronder vind je een overzicht van de verschillen:

1. Java is een programmeertaal waarvan je de programmacode moet compileren, terwijl Javascript een scripttaal is waarvan de programmacode pas door de browser op de computer van de eindgebruiker wordt geïnterpreteerd.
2. De broncode in Javascript is normaal gesproken volledig in HTML-documenten ingebed terwijl een Java-applet alleen maar een tag om de applet uit te voeren in de HTML-code wordt opgenomen.
3. In Java moet je variabelen declareren (strong typing), net als in hogere programmeertalen als C/C++. Bovendien moet je het type variabele zelf toewijzen (tekenreeks, getal). In Javascript moet je variabelen echter niet declareren. Dit wordt loose typing genoemd. Dit wil zeggen dat een variabele automatisch door Javascript wordt gedefiniëerd wanneer je dat niet zelf doet. Het type variabele kan je tijdens de uitvoering van het programma veranderen.
4. Javascript gebruikt dynamic-binding terwijl Java static-binding toepast. Static-binding betekent dat de Java-compiler reeds tijdens het compileren, dat wil zeggen tijdens de omzetting in machinetaal, de verwijzingen naar objecten controleert. Bij Javascript zijn deze verwijzingen dynamisch en kunnen ze zelfs nog worden veranderd tijdens de uitvoering van het programma.

Hoofdstuk 1
Javascript : Basisbegrippen

1. Plaatsing van Javascript-code
2. Syntax-regels
3. Variabelen
4. Basisobjecten, eigenschappen en methoden

1.1 Plaatsing van Javascript-code

De script-tag

De script-tag geeft aan dat er gebruik zal worden gemaakt van een scripttaal.
Attributen:

- *type* : Het type scripttaal:
  - voor Javascript: *text/javascript*
  - voor VBscript: *text/vbscript*
- *language* : de gebruikte taal, afgekeurd ten voordele van type.
- *defer* : attribuut dat aangeeft dat vanuit dit codeblok geen inhoud zal worden opgegeven (geen document.write-statements)
- *src* : verwijzing naar een bestand met script-code

Ingebedde Javascript-code

Javascript-code kan worden ingebed in (X)HTML-pagina's met de **script**-tag

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

hier komt de Javascript-code

// -->
</script>
```

De HTML commentaar-tags zorgen voor een correcte afhandeling door browsers die geen scripting ondersteunen.
Deze (oudere) browsers negeren de script-tag en zouden de code gewoon op het scherm tonen. Dit vermijden we door de eigenlijke Javascript-code tussen HTML commentaar-tags te plaatsen.

Javascript herkend de <!-- - tag en negeert alle code op deze regel.
De HTML sluittag --> wordt voorafgegaan door //, dit is een manier om Javascript-commentaar te voorzien, daar anders in browsers die Javascript wel ondersteunen de tag --> tot een Javascript-fout zou leiden.

Javascript-includes

Je kan vanuit een webpagina ook verwijzen naar een bestand waarin Javascript-code zit opgeborgen:

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

Noscript
Met het **noscript**-element kan je de gebruiker ervan op de hoogte stellen dat de Javascript-code niet door de browser kan worden uitgevoerd.
Browsers die scripts ondersteunen kennen deze tag en weten dat ze de inhoud moeten negeren. Browsers die geen scripts ondersteunen kennen deze tag niet, negeren dus de **noscript-tag** en tonen de inhoud ervan op het scherm.

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

hier komt de Javascript-code

// -->
</script>
<noscript>
Javascript wordt niet uitgevoerd door de gebruikte browser
</noscript>
```

1.2 Syntax-regels

- Afzonderlijke woorden worden in Javascript door een of meer spaties of door een tabstop van elkaar gescheiden. Operatoren als = of + en soortgelijke tekens hoeft u niet noodzakelijk te scheiden met een spatie of tabstop.
- Elk statement (opdracht) eindigt met een puntkomma .
- Javascript is hoofdlettergevoelig !
- **commentaar** bij de Javascript-code kan als volgt worden ingevoerd :

commentaar op 1 regel : **//**

```
var i = 1;      // ken de waarde 1 toe aan variabele i .
```

commentaar op meerdere regels : **/* ... */**

```
/* Dit is commentaar
en wordt niet geïnterpreteerd */
```

- Voor kommagetallen : gebruik een punt i.p.v. een komma !

1.3 Variabelen

Variabelen gebruik je om waarden in het geheugen van de computer vast te houden. Ze zijn de fundamentele gegevenseenheden van elke programmeertaal.
Variabelen kunnen getallen of tekenreeksen (strings) bevatten. Variabelen kunnen op een willekeurige plaats in het programma worden uitgelezen of veranderd.
In tegenstelling tot de meeste programmeertalen zijn variabelen in Javascript **untyped** : de aard (teken, integer,...) van de variabele wordt automatisch toegewezen (net zoals in Visual Basic).
Een variabele declareren en initialiseren
Een variabele wordt in Javascript met het woord **var** gedeclareerd :

```
var i;
var j;
```

Je kan ook verschillende variabelen tegelijk declareren :

```
var i,j;
```

Nadat je een variabele hebt gedeclareerd kan je er een **waarde** aan **toewijzen**.
De toekenningsoperator is in Javascript het gelijkheidsteken (**=**)

In Javascript kan je de **declaratie** (aanmaak) en **initialisatie** (eerste opvulling) van een variabele ook in 1 stap doen:

```
i=7;
j="Hallo iedereen";
var i = 7;
```

Het bereik (scope) van een variabele
Het bereik van een variabele kan lokaal of globaal zijn. Globale variabelen kunnen op willekeurige plaatsen in het programma worden aangeroepen. Lokale variabelen worden in een functie of codeblok gedeclareerd. Hun bereik beperkt zich dus tot deze functie of codeblok. Opgepast, wanneer je variabelen gebruikt ben je niet verplicht het sleutelwoord **var** te gebruiken, maar dan hebben ze steeds een **globale** impact.

1.4 Basisobjecten, eigenshappen en methoden

Met Javascript kan je objecten aanmaken en benaderen. Daar je met Javascript een webpagina moet kunnen benaderen, zijn reeds een aantal objecten voor de ontwikkelaar te benaderen:

- ***navigator*** : informatie over de gebruikte browser
- ***window*** : verwijst naar het huidige browservenster
- ***document*** : verwijst naar de huidige webpagina - het documentvenster

Er zijn nog heel wat andere objecten te benaderen, meer hierover verderop in de cursus.

Voorbeeld op volgende pagina >>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE>Cursus Javascript</TITLE>

<script type="text/javascript">
<!--
// vullen van variabelen
cursus = "cursus Javascript";
school = '';
mededeling = "Welkom op deze webpagina";
dagen = 3;
kleur = 'red';
window.status = mededeling; //bericht in de statusbalk
// -->
</script>

</HEAD>
<BODY>
<h3>Welkom !</h3>

<script type="text/javascript">
<!--
// tekst op het document
document.write('Welkom in de ' +cursus +' bij het <b>' +school +'.</B>');
document.write('<p>We zijn reeds ' +dagen +' dagen <i>(=' +(dagen*24) +'
uren)</i> met Javascript bezig.');
document.write('<p style="color:' +kleur +'">Hopelijk vind je het leuk</font>');
// -->
</script>

</BODY>
</HTML>
```

In deze toepassing zie je twee script-blokken: 1 in de hoofding en 1 in de body van de webpagina. Als algemene regel zet je de code die geen document.write-instructies bevat best in de hoofding van de pagina. De code die hier staat wordt uitgevoerd vooraleer er iets op het scherm verschijnt.
Wij vullen er een aantal variabelen, en zorgen ervoor dat de eigenschap *status* van het object *window* gevuld wordt met een tekenreeks.
Plaats je een script-blok binnen het **body**-gedeelte van de pagina, dan wordt het script uitgevoerd op de plaats waar je het blok hebt ingebed, de plaatsing van het script-blok bepaalt hier dus de plaats van de regels die met *document.write* op het scherm worden gezet. ***document*** is het object dat verwijst naar het huidige documentvenster van de webpagina. Wil je vanuit Javascript inhoud op het document plaatsen dan kan dit door de methode *write* van het **object *document*** toe te passen.

In een document.write-statement kan je tekenreeksen samenvoegen (concatineren) met het + teken. Merk op dat je in de tekenreeksen HTML-tags kan gebruiken. Deze worden door de browser geïnterpreteerd.

- Je kan in Javascript ook berekeningen doen met numerieke variabelen.
- Je kan het type van een variabele opvragen met de functie-operator *typeof* :

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
 <script type="text/javascript">
<!--
cursus = "cursus Javascript";
lessen = 10;
prijs = 45.5;
// -->
</script>
</HEAD>
<BODY>
<h3>Welkom !</h3>
<script type="text/javascript">
<!--
document.write('<p>Het volgen van de opleiding \" +cursus +'\' omvat '
                              +lessen +' lessen en kost € ' +prijs +'</p>');
document.write('<b>cursus</b> is van het type: ' +typeof(cursus) +'<BR>');
document.write('<b>lessen</b> is van het type: ' +typeof(lessen) +'<BR>');
document.write('<b>prijs</b> is van het type: ' +typeof(prijs) +'<BR>');
document.write('<b>document</b> is van het type: ' +typeof(document) +'<BR>');
// -->
</script>
</BODY>
</HTML>
```

Dit komt er alsvolgt uit te zien:

Welkom!

Het volgen van de opleiding 'cursus Javascript' omvat 10 lessen en kost € 45.5

cursus is van het type: string
lessen is van het type: number
prijs is van het type: number
document is van het type: object

Hier werd in het eerste document.write-instructie gebruik gemaakt van het escape-teken \. Je kunt op deze manier een enkele quote op het scherm plaatsen: \'.

Hoofdstuk 2
Javascript : Programmastructuren

1. Voorwaardelijke uitvoering: if en switch
2. Herhaaldelijke uitvoering: iteraties of lussen
3. Oefeningen

2.1 Voorwaardelijke uitvoering van instructie

if

Om instructies uit te voeren wanneer een voorwaarde is voldaan gebruik je het **if**-statement. Je kunt aan dit statement ook een **else**-tak koppelen: deze instructies worden uitgevoerd als de gestelde voorwaarde niet waar is.

In deze toepassing wordt een berichtvenster getoond met de *alert* -methode van het *window* -object. Je hoeft het *window* -object niet expliciet te vermelden bij het tonen van een alert.
\n zorgt in de alert voor een newline-karakter: het nemen van een nieuwe lijn.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Tsamransitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
 <script type="text/javascript">
<!--
naamcursus = "Javascript";
prijs = 45.5;
aantalleerlingen = 12;
leerkrachtgevonden = false;
// -->
</script>
</HEAD>
<BODY>
<script type="text/javascript">
<!--
document.write('<h3>Cursus ' +naamcursus +'</h3>');

// if gevolgd door 1 statement
if(prijs < 50) document.write('<p>De cursus is niet duur</p>');

// if gevolgd door meerdere statements
if(aantalleerlingen <= 15){
      document.write('<p>Een kleine klas</p>');
      alert('Er zijn nog te weinig leerlingen\nMaak wat meer reclame!');
}
```

Vervolg >>

```
// if met een else-tak
if(leerkrachtgevonden){
	document.write('<p>Er is reeds een leekracht voor deze cursus.</p>');
} else {

	document.write('<p>Er is nog <b>geen</b> leekracht voor deze
cursus.</p>');
}
// -->
</script>
</BODY>
</HTML>
```

Opmerkingen:

- na het **else**-sleutelwoord kan je onmiddellijk een nieuw **if**-statement plaatsen.
- Je kunt if-statements nesten: een if-statement binnen een ander if-statement.

switch

Om meerdere mogelijkheden te scheiden kan je gebruik maken van een **switch**-statement in plaats van meerder **if**-statements.

Bij het **switch** statement moet je tussen haakjes een expressie invoeren. Het programma springt dan binnen het opgegeven programmagedeelte naar een bepaald label (een naam gevolgd door een :)

Alle statements eindigen met **break**; . Wanneer u geen **break** plaatst worden ook de statements uitgevoerd van de labels eronder !

In Javascript mag u zoals uit het voorbeeld blijkt strings opgeven voor het **switch** statement, getalwaarden zijn natuurlijk ook toegestaan.

Wanneer een label met de naam **default** in het **switch** statement wordt opgenomen worden de statements onder dit label uitgevoerd wanneer geen enkele case beantwoord aan de **switch**-voorwaarde.

In dit voorbeeld wordt de methode *prompt* van het object *window* gebruikt om een invulvenster te tonen.

Voorbeeld op voklgende pagina >>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
 <script type="text/javascript">
<!--
naam = prompt("Wat is je naam ?","Naam ingeven");
// -->
</script>
</HEAD>
<BODY>
<script type="text/javascript">
<!--
switch(naam) {
case 'William' :
        document.write ('De naam is William');
        break;
case 'Pol' :
        document.write ('De naam is Pol');
        break;
default:
document.write ('De naam is niet William of Pol');
break;
}
// -->
</script>
</BODY>
</HTML>
```

2.2 Herhaaldelijke uitvoering: iteraties of lussen

While

Een **while** statement voert een opdracht of een groep opdrachten telkens opnieuw uit tot de voorwaarde voor de **while**-lus onwaar wordt.

```
<script type="text/javascript">
<!--
var teller=1;
while(teller<11) {
        document.write('<br>teller heeft de waarde '+teller);
        teller++;
}
// -->
</script>
```

Dit stukje code plaatst 10 keer het zinnetje 'teller heeft de waarde ' op de webpagina.

Het zinnetje wordt voorafgegaan door een
-tag en gevolgd door de waarde die in de variabele **teller** is opgenomen. De variabele **teller** heeft oorspronkelijk de waarde *1*, telkens de **while** lus wordt doorlopen wordt teller met 1 opgehoogd. Wanneer teller *11* wordt, worden de statements in de **while**-lus niet meer uitgevoerd.

teller++; is equivalent met teller = teller + 1;

Wanneer teller oorspronkelijk *11* of meer zou zijn, worden de statements binnen de **while**-lus niet uitgevoerd !

Do...While
Bij een **Do...While**-lus wordt de voorwaarde om verder te lussen aangegeven na de lusopdrachten.

Doordat deze voorwaarde pas na de statements wordt gecontroleerd worden de statements minstens 1 keer uitgevoerd!

De opdrachten zullen dus ook uitgevoerd worden wanneer de initiële waarde van de variabele teller *11* of meer is.

```
<script type="text/javascript">
<!--
var teller=1;
do{
        document.write('<br>teller heeft de waarde '+teller);
        teller++;
}
while(teller<11);
// -->
</script>
```

For
Een **for**-lus is lus-structuur waarbij je de **initialisatie, lusvoorwaarde** en **eindstatements** in een compact geheel plaatst.

Doordat de eindstatements als derde argument in de lus opgenomen zijn, is het niet nodig de variabele **teller** binnen de lus op te hogen.

```
<script type="text/javascript">
<!--
for(var teller = 1; teller < 11 ; teller++){
        document.write('<br>teller heeft de waarde '+teller);
}
// -->
</script>
```

for...in

Met een **for...in** lus kan je objecten doorzoeken. We zullen deze lusstructuur gebruiken wanneer we zelf javascript-objecten aanmaken. We kunnen de ingebouwde browser-objecten waarmee we reeds kennis hebben gemaakt nu reeds onderzoeken.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
</HEAD>
<BODY>
<script type="text/javascript">
<!--
document.write("<h2>Onderzoek van het object navigator</h2>");
for(var veld in navigator){
      document.write("<b>" +veld +"</b>: " +navigator[veld] +"<br>");
}
document.write("<hr>");

document.write("<h2>Onderzoek van het object window</h2>");
for(var veld in window){
      document.write("<b>" +veld +"</b>: " +window[veld] +"<br>");
}
document.write("<hr>");

document.write("<h2>Onderzoek van het object document</h2>");
for(var veld in document){
      document.write("<b>" +veld +"</b>: " +document[veld] +"<br>");
}
document.write("<hr>");
// -->
</script>
</BODY>
</HTML>
```

Maak je nog niet te veel zorgen over de concrete betekenis van deze code, deze objecten met hun eigenschappen zijn in feite voorbeelden van associatieve arrays - meer hierover later.

Break

Een lus kan je stoppen met het sleutelwoord **break**.
Dit is handig wanneer je een lus gebruikt om een item te zoeken. Je kan wanneer de lusdoorgang succesvol was de lus beïndigen.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
</HEAD>
<BODY>
<script type="text/javascript">
<!--
for(var teller = 1; teller < 11 ; teller++){
        document.write('<br>teller heeft de waarde '+teller);
        if(teller == 6) break;
}
// -->
</script>
</BODY>
</HTML>
```

Continue

Met het sleutelwoord **continue** kan je ervoor zorgen dat de eropvolgende opdrachten binnen de huidige lusdoorgang niet meer worden uitgevoerd.

Wanneer de voorwaarde voor het verder lussen nog is voldaan wordt de lus -anders dan bij het statement **break**- hernomen.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
</HEAD>
<BODY>
<script type="text/javascript">
<!--
for(var teller = 1; teller < 11 ; teller++){
        if(teller == 6) continue;
        document.write('<br>teller heeft de waarde '+teller);
}
// -->
</script>
</BODY>
</HTML>
```

Hoofdstuk 3
Javascript : Operatoren

1. Toekenning
2. Vergelijking
3. Getalwaarden: isNaN, parseInt en parseFloat
4. Compacte if: ... ? ... : ...
5. Wiskundige operatoren
6. Booleaanse operatoren
7. Bitniveau operatoren
8. Oefeningen

3.1 Toekenningsoperatoren

Het gelijkheidsteken = heeft in Javascript een heel andere betekenis dan in de wiskundelessen.
Deze operator gebruik je in Javascript voor het toekennen van een waarde aan een variabele:

```
var a = 6;
var b = 10;
var c = a + b;
```

Eerst wordt aan **a** de waarde 6 en aan **b** de waarde 10 toegekend. Daarna berekenen we de som van **a** en **b** en kennen die waarde toe aan de variabele **c**.

Je kan deze operator ook combineren met wiskundige operatoren:

```
var a = 6;
var a += 3;
var b = 10;
b *= a;
```

Eerst wordt aan **a** de waarde 6 toegekend, daarna hogen we de waarde van **a** met 3 op. Het statement **a** += 3 kan je eigenlijk voluit schrijven als **a** = **a** + 3 : we verhogen **a** met drie.

We stoppen de waarde 10 in variabele **b**. Daarna gaan we het product van **b** met **a** in de variabele **b** stoppen: **b** *= **a** is voluit: **b** = **b** * **a**.

3.2 Vergelijkingsoperatoren

Gelijk / niet gelijk
Een gelijkheid zoals we kennen uit de lessen wiskunde wordt in Javascript uitgedrukt met twee gelijkheidstekens (==):

```
var a = 6;
var b = 7;
if (a==b) document.write ('a en b zijn gelijk')
else document.write ('a en b zijn niet gelijk')
```

Om een ongelijkheid te testen gebruik je de operator != waarbij ! de not-operator is.

identiek / niet identiek
Het identiek zijn van twee variabelen of waarden betekent dat niet alleen hun waarde maar ook hun type gelijk is.

De operator om het identiek zijn van twee variabelen te testen is het trippele gelijkheidsteken === . De negatie is **!**== .

'42' is niet identiek aan *42* daar *'42'* een **String** is en *42* een **number**.

groter / kleiner
<, >, <=, >= : kleiner, groter, kleiner of gelijk, groter of gelijk.

Het resultaat van een dergelijke test is steeds *true* of *false.*

```
var a = 6;
var b = 7;
if (a<b) document.write ('a is kleiner dan b');
else if (a>b) document.write ('a is groter dan b');
else document.write ('a en b zijn gelijk');
```

Deze operatoren kan je ook toepassen op string-variabelen, je kan nu bvb. testen of een woord alfabetisch voor een ander woord komt.

3.3 Getalwaarden

isNaN
In toepassingen is het dikwijls nodig om na te gaan of bepaalde variabelen getallen bevatten.

De functie *isNaN* retourneert *true* als het argument **Not a Number** is, geen numerieke waarde. De functie *isNaN* retourneert *false* als het argument een numerieke waarde bevat.

```
document.write(isNaN("Een tekenreeks") +"<BR>");
document.write(isNaN("348") +"<BR>");
document.write(isNaN(348) +"<BR>");
document.write(isNaN(0/0) +"<BR>");
```

parseInt
Met de functie *parseInt* kan je cijfers van een tekenreeks afkappen. Begint de tekenreeks niet met cijfers dan retourneert *parseInt NaN* : **Not a Number**.

Deze functie wordt vaak gebruikt om kommagetallen af te kappen tot gehele getallen.

Voorbeeld op volgende pagina >>

```
document.write("<BR>" + parseInt("50"));
document.write("<BR>" + parseInt("50.12345"));
document.write("<BR>" + parseInt("32.00000000"));
document.write("<BR>" + parseInt("71.348  92.218  95.405"));
document.write("<BR>" + parseInt("         37 varkens"));
document.write("<BR>" + parseInt("Cursussen van het jaar 2005"));

var jaar = 120;
if ( parseInt(jaar / 4) == (jaar / 4 ) )
{
        document.write("<BR>jaar is deelbaar door 4");
}
```

parseFloat
Deze Javascript-functie zoekt het eerste (komma)getal in een tekenreeks. Begint de tekenreeks niet met een getal dan retourneert *parseInt NaN* : **Not a Number**.

```
document.write("<BR>" + parseFloat("50"));
document.write("<BR>" + parseFloat("50.12345"));
document.write("<BR>" + parseFloat("32.00000000"));
document.write("<BR>" + parseFloat("71.348  92.218  95.405"));
document.write("<BR>" + parseFloat("         37 varkens"));
document.write("<BR>" + parseFloat("Cursussen van het jaar 2005"));
```

3.4 Compacte if: … ? … : …

Een voorwaardelijke structuur kan met de operatoren ? en : op een compacte manier geschreven worden.

```
var a = 10;
var b = 12;
document.write( a>b ? "a groter dan b" : "a niet groter dan b");
```

3.5 Wiskundige operatoren

Het spreekt voor zich dat je met Javascript ook wiskundige bewerkingen kan uitvoeren. Een aantal van deze operatoren hebben we in de voorgaande voorbeelden reeds intuïtief gebruikt.

Betekenis	Operator	Voorbeeld
Optellen	+	Som = a + b
Aftrekken	-	verschil = a - b
Tegengestelde	-	tegengestelde = -b;
Vermenigvuldigen	*	product = a * b;
Delen	/	quotiënt = a / b;
Modulo	%	rest = a % b;
Verhogen met 1	++	a++ of ++a (a = a + 1)
Verlagen met 1	--	a-- of --a (a = a - 1)

Opgepast met de ++ en -- operatoren, er is een subtiel, doch belangrijk verschil wanneer deze operatoren als prefix of suffix worden gebruikt:

```
var aantal=1;
document.write(aantal++);
document.write('<br>');
document.write(aantal);

document.write('<hr>');

aantal=1;
document.write(++aantal);
document.write('<br>');
document.write(aantal);
```

In de eerste toepassing wordt de variabele **aantal** ingesteld op 1. De waarde van aantal++ wordt op het scherm gezet. Aantal wordt hier met 1 opgehoogd pas nadat de instructie werd uitgevoerd!

In de tweede toepassing wordt ++ als prefix gebruikt en wordt de waarde van **aantal** reeds opgehoogd vooraleer die op het document wordt geplaatst.

3.6 Booleaanse operatoren

In de Booleaanse logica kennen we twee waarden: waar en onwaar. In Javascript worden hiervoor de waarden *true* en *false* gebruikt.
We hebben deze Booleaanse waarden reeds gebruikt in voorwaardelijke structuren en lussen.

Met Booleaanse operatoren kan je meerdere voorwaarden ineens testen.

And: &&
Met de Booleaanse operator && kan je meerdere voorwaarden testen geschakeld met een logische 'en': alle voorwaarden moeten waar zijn vooraleer de totale expressie waar is.

```
var getal1 = 8;
var getal2 = 24;

if ( (getal1 < 10) && ( getal2 < 40) )
{
document.write("Beide voorwaarden zijn vervuld");
}
```

De binnenste koppels haken zijn hier optioneel, maar verhogen de leesbaarheid. De voorwaarden leveren telkens *true* waardoor de 'en'-geschakelde expressie *true* levert en de statements binnen de **if**-structuur worden uitgevoerd.

Als voorwaarde kan ook een getal dienen: alle numerieke waarde behalve 0 hebben een logische waarde *true*, 0 heeft de waarde *false*.
Or: ¦¦

Met de Booleaanse operator && kan je meerdere voorwaarden testen geschakeld met een logische 'en': alle voorwaarden moeten waar zijn vooraleer de totale expressie waar is.

```
var getal1 = 12;
var getal2 = 24;

if ( (getal1 < 10) ¦¦ ( getal2 < 40) )
{
document.write("Minstens 1 voorwaarde is vervuld");
}
```

De binnenste koppels haken zijn hier optioneel, maar verhogen de leesbaarheid. Wanneer 1 van de voorwaarden een logische *true* oplevert wordt de totale expressie *true* bij een 'of'-schakeling.

Als voorwaarde kan ook een getal dienen: alle numerieke waarde behalve 0 hebben een logische waarde *true*, 0 heeft de waarde *false*.

3.7 Bitniveau operatoren

Je kan in Javascript ook met de bitwaarde van een variabele werken. Deze bitwaarde van een variabele is de waarde van de variabele zoals die is opgeslagen in het interne geheugen van je computer : in het binaire talstelsel.

Bitoperator en: &

```
document.write( 14 & 7 );

Levert: 6.
Werkwijze:
14 binair:     1       1       1       0
7 binair       0       1       1       1
logische 'en' berekening - decimaal resultaat: 6       0       1       1       0
```

Bitoperator of: ¦

```
document.write( 14 ¦ 7 );

Levert: 15.
Werkwijze:
14 binair:     1       1       1       0
7 binair       0       1       1       1
logische 'of' berekening - decimaal resultaat: 15      1       1       1       1
```

Vervolg>>

Bitoperator exor: ^
document.write(14 ^ 7);

Levert: 9.
Werkwijze:
14 binair: 1 1 1 0
7 binair 0 1 1 1
logische 'exor' berekening - decimaal resultaat: 9 1 0 0 1

De exor of exclusieve or levert true als een van de deelwaarden true is, doch false als alle voorwaarden true zijn.

Concrete toepassingen:

```
<h3>Initialiseren van een variabele</h3>
<script type="text/javascript">
<!--
var a = 25;

document.write("<BR>" +a );
document.write("<BR>" +(a ^ a) );
// -->
</script>

<h3>Twee waarden omwisselen zonder hulpvariabele</h3>
<script type="text/javascript">
<!--
var a = 25;
var b = 10;

document.write("<BR>a = " +a );
document.write("<BR>b = " +b );

a = a ^ b;
b = a ^ b;
a = a ^ b;

document.write("<hr><BR>a = " +a );
document.write("<BR>b = " +b );
 </script>
```

Hoofdstuk 4
Javascript: functies

1. Functies maken
2. Functies aanroepen
3. Argumenten doorgeven
4. Waarden retourneren
5. Functies in een .js-bestand

4.1 Functies maken

Een functie kan je in Javascript gebruiken om een **groep** opdrachten te kunnen uitvoeren met een aangegeven naam. De opdrachten of statements die binnen de functie staan worden uitgevoerd wanneer de functie wordt aangeroepen.Een functie kan waarden ontvangen en retourneren naar de plaats van aanroep.
Eenvoudig voorbeeld:

```
function toonzin(){
        document.write('Een zin in een functie.<BR>');
 }
```

Het sleutelwoord **function** wordt gebruikt om een functie aan te maken. Dit woord wordt gevolgd door de naam van de functie, deze naam kan je zelf kiezen. Na de naam van de functie noteer je een paar ronde haken (en). Hier zal je later namen van argumenten noteren. De functie *toonzin* ontvangt geen argumenten, toch moeten de ronde haken er staan.

De statements die de ziel van de functie uitmaken worden genoteerd tussen accolades: { en }.

Wanneer deze functie wordt aangeroepen zal de zin in het document worden geplaatst.

Functies worden doorgaans in een script-blok in de hoofding van de pagina geplaatst. De informatie in de hoofding wordt geladen vooraleer de browser iets toont, zodoende ben je er zeker van dat de functie reeds geladen en dus gekend is door de browser vooraleer je ze aanroept.

4.2 Functies aanroepen

Een functie doet niks op zich, de statements binnen de functies worden pas uitgevoerd wanneer de functie wordt aangeroepen.

```
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
 <script type="text/javascript">
 <!--
 function toonzin(){
        document.write('Een zin in een functie.<BR>');
 }
 // -->
 </script>
</HEAD>
<BODY>
<script type="text/javascript">
<!--
toonzin()
// -->
</script>
</BODY>
</HTML>
```

Je merkt dat de functie-declaratie in de hoofding van de pagina staat. Het aanroepen van deze functie gebeurt binnen de **BODY** van het document, daar de *document.write*-opdracht die zal worden uitgevoerd informatie op het zichtbare gebied van de webpagina moet plaatsen.

Functies kunnen meerdere keren worden aangeroepen binnen hetzelfde document. Dit kan met een eenvoudige **for**-lus gedemonstreerd worden:

```
for (var teller = 1 ; teller < 11 ; teller ++){
        toonzin()
}
```

4.3 Argumenten doorgeven aan een functie

We zorgen ervoor dat we zelf kunnen kiezen welke zin we op het scherm zetten. Hiervoor doen we twee aanpassingen:

- we geven bij aanroep van de functie een argument mee: de weer te geven zin.
- we zorgen ervoor dat het argument bruikbaar is in de functie door het te stockeren in een variabele. De naam van deze variabele is in dit voorbeeld **tekst**, deze naam kan je zelf kiezen.

```
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
 <script type="text/javascript">
 <!--
 function toonzin(tekst){
        document.write(tekst +'<BR>');
 }
 // -->
 </script>
</HEAD>
<BODY>
<script type="text/javascript">
<!--

toonzin("Een eerste zin");
toonzin("Een tweede zin");
toonzin("Nog een zin");

// -->
</script>
</BODY>
</HTML>
```

Belangrijk: de scope (levensduur) van de variabele tekst is enkel de functie toonzin, buiten de functie is de variabele niet gekend.

4.4 Waarden retourneren

Een functie kan ook een waarde retourneren naar de plaats van aanroep.

Met het sleutelwoord **return** geef je aan welke waarde je wenst te retourneren

```
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
 <script type="text/javascript">
 <!--
 function dubbel(getal){
        if(!isNaN(getal)) return 2 * getal
        else return "geen getal !"

 }
 // -->
 </script>
</HEAD>
<BODY>
<script type="text/javascript">
<!--

document.write("<br>" +dubbel(50));
document.write("<br>" +dubbel(46.2));
document.write("<br>" +dubbel("aap"));

// -->
</script>
</BODY>
</HTML>
```

Het sleutelwoord **return** wordt vaak gebruikt om functies vroegtijdig (onder bepaalde voorwaarden) te beëindigen.

4.5 functies in een .js-bestand

Je zal tijdens het leren en werken met Javascript talloze nuttige functies schrijven die je wellicht op vele pagina's van je website zal kunnen gebruiken. Om de functie niet op elke pagina te moeten hernemen, met de nodige onderhoudsproblemen die hiermee gepaard gaan, kan je ervoor opteren om functies onder te brengen in aparte bestanden met een extensie .**js**.
functie5.html

```
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
 <script type="text/javascript" src="functie5.js"></script>
</HEAD>
<BODY>
<script type="text/javascript">
<!--

document.write("<br>" +dubbel(50));
document.write("<br>" +dubbel(46.2));
document.write("<br>" +dubbel("aap"));

// -->
</script>
</BODY>
</HTML>


functie5.js
function dubbel(getal){
        if(!isNaN(getal)) return 2 * getal
        else return "geen getal !"
 }
```

Hoofdstuk 5
Javascript: *String, Number, Math en Date - objecten*

1. Het object String
2. Het object Number
3. Het object Math
4. Het object Date
5. Oefeningen

5.1 Het object String

Een tekenreeks wordt in Javascript gezien als een *String*. In feite vormt een *String* een object die een eigenschap *length* heeft en tal van interessante methoden ondersteund:

methode	***beschrijving***
• *toUpperCase*	Zet de tekenreeks om in hoofdletters
• *toLowerCase*	Zet de tekenreeks om in kleine letters
• *substring*	Retourneert een deel uit een string met meeggeven begin- en eindpositie
• *substr*	Retourneert een deel uit een string met meeggeven beginpositie en het aantal tekens
• *concat*	Voegt tekenreeksen samen
• *indexOf*	Retourneert de eerste positie van een meegegeven argument, je kan als tweede argument de startpositie voor het zoeken meegeven
• *LastIndexOf*	Retourneert de laatste positie van een meegegeven argument
• *charAt*	Retourneert het teken op de aangegeven positie
• *split*	Splitst de tekenreeks in een Array, met een aangegeven delimeter of scheidingsteken

Voorbeeld:

```
var tekst = "Ik ben een stukje tekst";
document.write("<br><b>" +(tekst) +"</b>");
document.write("<br>type: <b>" +typeof(tekst) +"</b>");
document.write("<br>lengte: <b>" +tekst.length +"</b>");
document.write("<br>hoofdletters: <b>" +tekst.toUpperCase() +"</b>");
document.write("<br>kleine letters: <b>" +tekst.toLowerCase() +"</b>");
document.write("<br>substring 11,15: <b>" +tekst.substring(11,15) +"</b>");
document.write("<br>substr 11,4: <b>" +tekst.substr(11,4) +"</b>");
document.write("<br>concat: <b>" +tekst.concat(", met nog een stukje erbij.")
+"</b>");
document.write("<br>concat(+): <b>" +tekst +", met nog een stukje erbij."
+"</b>");
document.write("<br>indexOf(t): <b>" +tekst.indexOf("t") +"</b>");
document.write("<br>lastIndexOf(t): <b>" +tekst.lastIndexOf("t") +"</b>");
document.write("<br>charAt(21): <b>" +tekst.charAt(21) +"</b>");

document.write("<p><u>split:</u>");
```

Vervolg>>

```
var reeks = new Array();
reeks = tekst.split(" ");
document.write("<br>Eerste woord: <b>" +reeks[0] +"</b>");
document.write("<br>Tweede woord: <b>" +reeks[1] +"</b>");
```

Het gebruik van *Arrays* komt verderop in de cursus aan bod.

Het is belangrijk in te zien dat het toepassen van deze methoden op de *String* **tekst** enkel als retourwaarde van de gebruikte methode de gemanipuleerde tekenreeks bevat. Dit wil zeggen dat de originele *String* **tekst** in geen enkel geval blijvend werd aangepast. Hiervoor moet je het resultaat van de methode toekennen aan de variabele **tekst**.

Methoden om HTML-opdrachten te gebruiken:

Methode	***Beschrijving***
• *bold*	Zet de tekenreeks vet
• *italics*	Zet de tekenreeks schuin
• *strike*	Zet de tekenreeks doorstreept
• *blink*	Doet de tekereeks knipperen (indien ondersteund)
• *big*	Maakt het uitzicht van de tekenreeks groter
• *small*	Maakt het uitzicht van de tekenreeks kleiner
• *sup*	Zet de tekenreeks in superscript
• *sub*	Zet de tekenreeks in subscript
• *Link*	Maakt van de tekenreeks een hyperlink
• *Anchor*	Maakt van de tekenreeks een anchor

```
var tekst = "Ik ben een stukje tekst";
document.write("<br>" +tekst.bold());
document.write("<br>" +tekst.bold().italics());
document.write("<br>" +tekst.link("http://www.novacollege.nl").italics());
```

5.2 Het object Number

Het object *Number* heeft een vijftal eigenschappen:

Eigenschap	***Beschrijving***
• *NaN*	Geen geldig getal
• *MAX_VALUE*	Grootst geldige getal
• *MIN_VALUE*	Kleinst geldige getal
• *POSITIVE_INFINITY*	Oneindig
• *NEGATIVE_INFINITY*	Min oneindig

Het object *Number* heeft een methode: ***toString***: hiermee kan je een getalwaarde omzetten in een tekenreeks. Je kan als argument het tallenstelsel meegeven waarmee je wenst te werken.

Voorbeeld op volgende pagina >>

Voorbeeld:

```
document.write("<br>" +Number.NaN);
document.write("<br>" +Number.MAX_VALUE);
document.write("<br>" +Number.MIN_VALUE);
document.write("<br>" +Number.POSITIVE_INFINITY);
document.write("<br>" +Number.NEGATIVE_INFINITY);

document.write("<hr>");

var a = 56;
document.write("<br>toString:" +a.toString());
document.write("<br>toString(10): " +a.toString(10));
document.write("<br>toString(16): " +a.toString(16));
document.write("<br>toString(2): " +a.toString(2));
document.write("<br>toString(8): " +a.toString(8));
```

5.3 Het object Math

Het object Math kan je gebruiken voor wiskundige berekeningen, jaja :)

Eigenschappen van het object *Math* : (er zijn er meer).

Eigenschap	***Beschrijving***
• *PI*	Het getal pi
• *E*	Het getal e (getal van Euler)

Methoden van het object *Math* :

Methode	***Beschrijving***
• *round(x)*	Afronden tot een integer groter of gelijk aan .5: naar boven kleiner dan .5: naar beneden
• *random()*	Willekeurig getal tussen 0 en 1
• *abs(x)*	absolute waarde van een getal
• *abs(x)*	absolute waarde van een getal
• *sin(x)*	sinus
• *cos(x)*	cosinus
• *tan(x)*	tangens
• *asin(x)*	boogsinus
• *acos(x)*	boogcosinus
• *atan(x)*	boogtangens
• *log(x)*	natuurlijke logarithme: $\log_e(x)$
• *exp(x)*	e^x
• *sqrt(x)*	vierkantswortel
• *round(x)*	afronden tot een geheel getal
• *ceil(x)*	gehele waarde die groter of gelijk is
• *floor(x)*	gehele waarde die kleiner of gelijk is
• *pow(x,y)*	xy
• *min(x,y)*	kleinste waarde
• *max(x,y)*	grootste waarde

Voorbeeld op volgende pagina >>

Voorbeeld:

```
<HTML>
<HEAD>
 <TITLE>NovaCollege</TITLE>
 <script type="text/javascript">
 function radgrad(rad) {
        // pi radialen = 180 graden
    return (rad * 180) / Math.PI;
  }
 </script>
</HEAD>
<BODY>
<script type="text/javascript">
<!--
var getal1 = 8;
var getal2 = 2;
var getal3 = -2.89;
document.write("<br>getal1: " +getal1);
document.write("<br>getal2: " +getal2);
document.write("<br>getal3: " +getal3);
document.write("<br>abs(getal3): " +Math.abs(getal3));
document.write("<br>getal2 in graden: " +radgrad(getal2)); //eigen functie
document.write("<br>sin(getal1): " +Math.sin(getal1));
document.write("<br>pow(getal1,getal2): " +Math.pow(getal1,getal2));
document.write("<br>max(getal1,getal2): " +Math.max(getal1,getal2));
// -->
</script>
</BODY>
</HTML>
```

5.4 Het object Date

Het werken met datums levert steeds de nodige verwikkelingen op. Datums zijn nu eenmaal door de mens gemaakte eenheden die niet altijd de logica van de gewone getallen volgen.

Systeemtijd

Je kan de huidige systeemtijd (client) opvragen met:

```
var datum = new Date();
document.write("<br>datum: " +datum);
```

Resultaat:

datum: Wed Mar 24 11:00:10 UTC+0100 2010

Methoden voor opvragen van datumdetails:

Methode	***Beschrijving***
getTime	Tijd verstreken sinds 1 januari 1970 00:00:00 in milliseconden
getSeconds	aantal seconden (0-59)
getMinutes	aantal minuten (0-59)
getHours	aantal uren (0-23)
getDay	weekdag (0=zondag, 6=zaterdag)
getDate	dag van de maand (0-31)
getMonth	maand van het jaar (0=januari, 11=december)
getFullYear	jaar in 4 cijfers (niet in oude browsers)

Voorbeeld:

```
var datum = new Date();
document.write("<br>datum: " +datum);
document.write("<br>getTime: " +datum.getTime());
document.write("<br>getSeconds: " +datum.getSeconds());
document.write("<br>getMinutes: " +datum.getMinutes());
document.write("<br>getHours: " +datum.getHours());
document.write("<br>getDay: " +datum.getDay());
document.write("<br>getDate: " +datum.getDate());
document.write("<br>getMonth: " +datum.getMonth());
document.write("<br>getFullYear: " +datum.getFullYear());
```

Instellen van de datum

De methoden die je in bovenstaande tabel vindt voor het lezen van de datum kennen hun set-variant (uitgezonderd *getDay*: je kan de dag van de week niet zelf instellen, dit wordt berekend). Op deze manier kan je een datumvariabele volgens je wensen instellen.

Voorbeeld:

```
var datum = new Date();
datum.setDate(1);
datum.setMonth(3);
datum.setFullYear(1974);
document.write("<br>datum: " +datum);
document.write("<br>getDay: " +datum.getDay());
document.write("<br>getDate: " +datum.getDate());
document.write("<br>getMonth: " +datum.getMonth());
document.write("<br>getFullYear: " +datum.getFullYear());
```

Je kan een datum/tijd ook meteen instellen bij het aanmaken van het *Date*-object.

```
var datum = new Date(1974,3,1,10,30,0);
document.write("<br>datum: " +datum);
document.write("<br>getDay: " +datum.getDay());
document.write("<br>getDate: " +datum.getDate());
document.write("<br>getMonth: " +datum.getMonth());
document.write("<br>getFullYear: " +datum.getFullYear());
```

Hoofdstuk 6
Javascript: Arrays

1. Basisbegrippen
2. Methoden
3. Multidimensionele Arrays
4. Associatieve Arrays
5. Oefeningen

6.1 Basisbegrippen

Arrays worden vaak ook gegevensvelden of tabellen genoemd.

```
var mijnArray = new Array();
```

Aan een variabele kan je altijd maar één waarde tegelijk toewijzen. Een *Array* daarentegen kan verschillende waarden bevatten. Men spreekt daarbij van verschillende elementen, waarbij elk element een waarde heeft. De afzonderlijke elementen gedragen zich bijgevolg als variabelen. In plaats van tien variabelen kan je dus ook een *Array* van tien elementen gebruiken.

Arrays hebben nog een groot voordeel : de verschillende elementen worden doorlopend genummerd. In een lus kan je dan via een automatische teller de *Array* doorlopen.

In onderstaand voorbeeld maken we eerst een *Array* met 3 elementen. De *Array* is gekend onder de naam **namen**. Het opvullen van de Array doe je door aan elk element van de *Array* een waarde toe te kennen. Het eerste element is namen[0] en krijgt de waarde "William" toegewezen. Elementen kunnen we dus aanspreken door de naam van de *Array* te laten volgen door een indexcijfer tussen vierkante haakjes: [en]. Dit indexcijfer is *zero-based*: het eerste cijfer is een nul.

Het aantal elementen in een *Array* kan je opvragen via de eigenschap *length* van de *Array*.

Je kan met een eenvoudige **for**-lus alle elementen van de *Array* aflopen.
Voorbeeld:

```
var namen = new Array(3);
namen[0] = "William";
namen[1] = "Jos";
namen[2] = "Tine";

document.write("<br>Aantal namen: " +namen.length);
document.write("<br>Eerste naam: " +namen[0]);
document.write("<br>Tweede naam: " +namen[1]);
document.write("<hr>");

for(var i = 0; i < namen.length ; i++){
        document.write("<br>naam" +(i+1) +": " +namen[i]);
}
```

In dit tweede voorbeeld wordt een *Array* op een compacte manier gevuld met gemeenten.
Voorbeeld:

```
var gemeenten = new Array("Knokke","Brugge","Gent","Oostende");

document.write("<br>Aantal gemeenten: " +gemeenten.length);

for(var i = 0; i < gemeenten.length ; i++){
	document.write("<br>gemeente" +(i+1) +": " +gemeenten[i]);
}
```

Een alternatieve manier om een *Array* af te lopen is met een **for...in**-lus.
Hier neemt de variabele **item** de taak van de index over: voor elk element in de *Array* wordt het indexcijfer in de variabele **item** gestopt.

Voorbeeld:

```
var gemeenten = new Array("Knokke","Brugge","Gent","Oostende");

document.write("
Aantal gemeenten: " +gemeenten.length);

for(var item in gemeenten){
	document.write("<br>gemeente" +item +": " +gemeenten[item]);
}
```

6.2 Methoden

Arrays ondersteunen een aantal methoden:

Methode	***Beschrijving***
• *concat*	Voeg de elementen van twee (of meer) Arrays samen
• *join*	Plaatst alle elementen van een Array in een String. De elementen worden gescheiden door een aangegeven delimeter (standaard een komma).
• *pop*	Verwijdert en retourneert het laatste element van de Array
• *push*	Voegt een of meerdere elementen toe aan een Array en retourneert de nieuwe lengte van de Array
• *reverse*	Keert de volgorde van de Arrayelementen om
• *shift*	Verwijdert en retourneert het eerste element van de Array
• *slice*	Maakt een nieuwe Array van een aangegeven bereik in een Array. *Argumenten*: beginpositie en optioneel de eindpositie
• *sort*	Sorteert de elementen in de Array
• *splice*	Elementen verwijderen en toevoegen aan een Array. Deze methode retourneert de verwijderde elementen. *Argumenten:* startpositie voor verwijderen, aantal te verwijderen elementen, nieuwe elementen
• *unshift*	Voegt een of meerdere elementen toe aan het begin van de Array en retourneert de nieuwe lengte

Voorbeeld:

```
var dieren = new Array("koe","aap","ezel");
var beesten = new Array("paard","eend","kikker","hond");

document.write("<br>Dieren: <b>" +dieren.join() +"</b>");
// als je join() niet gebruikt levert de naam van de Array ook een String
document.write("<br>Nog eens de dieren: <b>" +dieren +"</b>");
document.write("<br>Beesten: <b>" +beesten.join("-") +"</b>");
document.write("<br>Dieren en beesten: <b>" +dieren.concat(beesten) +"</b>");
document.write("<br>Dieren omgekeerd: <b>" +dieren.reverse() +"</b>");
document.write("<br>Dieren gesorteerd: <b>" +dieren.sort() +"</b>");
document.write("<br>Beesten eruit: <b>" +beesten.splice(1,2) +"</b>");
document.write("<br>Beesten over: <b>" +beesten +"</b>");
document.write("<br>dieren.pop(): <b>" +dieren.pop() +"</b>");
document.write("<br>dieren.push('kat'): <b>" +dieren.push("kat") +"</b>");
document.write("<br>dieren: <b>" +dieren +"</b>");
```

Bij het toepassen van een methode op een *Array* wordt deze onmiddellijk gemanipuleerd en is dus de originele Array veranderd. Wil je de originele Array bewaren, dan moet je die eerst 'kopiëren' in een ander *Array* -object. Dit is anders dan bij de *String* -methoden uit het vorige hoofdstuk.

6.3 Multidimensionele Arrays

Een *Array* kan ook meerdere dimensies hebben.

Een tweedimensionele *Array* kunnen we ons als een tabel voorstellen:

```
var Adressen = new Array(2)
for ( var teller = 0; teller<Adressen.length;teller++) {
   Adressen[teller]=new Array(3);
}

Adressen[0][0]="Peters";
Adressen[0][1]="Pol";
Adressen[0][2]="Brugge";
Adressen[1][0]="Janssens";
Adressen[1][1]="Sofie";
Adressen[1][2]="Oostkamp";

Adressen.sort();

for (var n = 0;n<Adressen.length;n++) {
        for (var m = 0;m<Adressen[n].length;m++){
                document.write(Adressen[n][m] +" ");
        }
        document.write("<br>");
}
```

We declareren eerste een eendimensionele *Array* adressen. Elk element uit deze nieuwe *Array* wordt met behulp van een lus gedeclareerd als een *Array*.

Een driedimensionele n x m x l *Array* kunnen we ons voorstellen als een kubus bestaande uit l tabellen.

6.4 Associatieve Arrays

Associatieve Arrays geven je de mogelijkheid te werken met text-indices in plaats van numerieke waarden.

Het overlopen van de *Array* kan nu best met een **for...in**-lus.
Voorbeeld:

```
var arrAuto = new Array();
arrAuto["klein"] = "Smart";
arrAuto["stad"] = "Renault Twingo";
arrAuto["snel"] = "Porshe";

document.write("<br>Kleine auto:" +arrAuto["klein"]);
document.write("<hr>");
for( soort in arrAuto ){
        document.write("<br>" +soort +" :" +arrAuto[soort]);
 }
```

In feite worden objecten in Javascript op een analoge manier behandeld. Wil je de eigenschap *appName* van het object *navigator* opvragen dan tik je normaal:

```
        document.write("Uw browser: " +navigator.appName);
```

Resultaat: Uw browser: Microsoft Internet Explorer

Je kan eigenschappen van objecten ook benaderen door het object te behandelen als een associatieve *Array* :

```
        document.write("Uw browser: " +navigator["appName"]);
```

Resultaat: Uw browser: Microsoft Internet Explorer

Hoofdstuk 7
Javascript: Document Object Model DOM

1. Inleiding
2. DOM - versies
3. DOM 0
4. Tussenliggende DOM's
5. W3C DOM: elementen manipuleren, elementen toevoegen en verwijderen
6. Objectdetectie
7. Oefeningen

7.1 Inleiding

Het Document Object Model (DOM) beschrijft hoe alle elementen op een webpagina (figuren, links, textvakken,...) gerelateerd zijn ten opzichte van het document als hoofdelement.

Via het DOM kan je de elementen op een webpagina manipuleren: eigenschappen opvragen en zelfs veranderen.

Het is in dit hoofdstuk dat we afscheid nemen van het eerder theoretische eerste gedeelte van de cursus en ons meer zullen verdiepen in dynamische toepassingen.

7.2 DOM - versies

Het begrip DOM voor webpagina's werd meegeïntroduceerd met Javascript. We beschikken met Javascript over een taal waarmee we webpagina's dynamisch kunnen maken. In de voorbije hoofdstukken hebben we de basis van Javascript doorgenomen, het is nu tijd deze basiskennis in de praktijk te brengen met dynamische toepassingen.

Voor webontwikkelaars is het belangrijk de eigenschappen van elementen op een webpagina te kunnen beïnvloeden. Wil je een figuur laten veranderen wanneer de gebruiker er met de muis over beweegt, dan moet je de eigenschap *src* van het element **IMG** kunnen aanpassen.

Het DOM heeft ondertussen reeds een behoorlijk ontwikkeling doorgemaakt:

1. **DOM 0**: originele DOM ontworpen door Netscape. Wordt nog steeds door zowat alle browsers ondersteund.
2. **Tussenliggende DOM's:** met de introductie van de versie 4 browsers was **DHTML** het toverwoord op het internet. Plots moesten op webpagina's tal van dynamische effecten mogelijk zijn. Netscape en Microsoft gingen hier elk hun eigen weg. Netscape maakt gebruik van *document.layers* en Microsoft van *document.all*
3. **W3C DOM:** standaard voorgesteld door het W3C, reeds level 3 in ontwerp.

7.3 DOM 0

Daar nog tal van websites gebruik maken van de originele specificaties van DOM, is het onontbeerlijk dit model te bestuderen.

Niveau 0 van DOM is relatief eenvoudig: document is het hoofdobject en daaronder heb je toegang tot *images, links, forms* en *anchors.*

Images

Om de eigenschap *src* van een **IMG** aan te passen kan je de volgende opdracht gebruiken:

```
document.images[0].src = "figuurnaam"
```

Images vormt hier in feite een *Array* van alle figuren op de webpagina. Bovenstaande opdracht zal dus de eerste figuur op de pagina veranderen.

Op deze manier kunnen we een andere afbeelding laten verschijnen wanneer de muisaanwijzer boven de figuur komt. We moeten er wel voor zorgen dat de figuur weer in het origineel verandert wanneer de muisaanwijzer de figuur verlaat.

In het voorbeeld wordt het veranderen van de *src* van de figuur gedaan door vanuit de eventhandlers *onmouseover* en *onmouseout* telkens een functie aan te roepen.

```
<HTML>
<HEAD>
 <TITLE>Cursus Novacollege</TITLE>
 <LINK href="../../vb.css" rel="stylesheet" type="text/css">
 <script type="text/javascript">
<!--
        function over(){
                document.images[0].src = "../../images/schapen.jpg"
        }

        function uit(){
                document.images[0].src = "../../images/koeien.jpg"
        }

// -->
</script>
</HEAD>
<BODY>

<h1>Toepassing DOM 3</h1>
<img src="../../images/koeien.jpg" width="141" height="94" alt="" border="0"
   onmouseover="over()" onmouseout="uit()">
</BODY>
</HTML>
```

Aangezien je vaak niet zeker bent dat deze figuur steeds de eerste figuur zal zijn op de pagina kan je deze ook een uniek *id* of *name* toekennen.

```
<HTML>
<HEAD>
 <TITLE>NovaCollege</TITLE>
 <LINK href="../../vb.css" rel="stylesheet" type="text/css">
 <script type="text/javascript">
<!--
        function over(){
                document.images["dier"].src = "../../images/schapen.jpg"
        }

        function uit(){
                document.images["dier"].src = "../../images/koeien.jpg"
        }

// -->
</script>
</HEAD>
<BODY>

<h1>Toepassing DOM 0</h1>
<img id="dier" src="../../images/koeien.jpg" width="141" height="94" alt=""
border="0" onmouseover="over()" onmouseout="uit()">
</BODY>
</HTML>
```

Forms

Het gebruik van formulieren is heel belangrijk voor ontwikkelingen met Javascript.

Formulieren zijn op webpagina's de elementen waarmee interactie en input van de gebruiker kan ontvangen.

Hieronder enkele manieren om formulieren en elementen uit formulieren te adresseren:

```
   // eerste formulier op de pagina
  document.forms[0]

   // formulier met de naam f
  document.forms["f"]

   // derde element van het formulier met naam f
  document.forms["f"].elements[2]
```

Vervolg>>

```
  // element product van het formulier f
document.forms["f"].elements["product"]

  // waarde van het element product uit formulier f
document.forms["f"].elements["product"].value

  // verkorte notatie van dit laatste
f.product.value
```

Toepassing:

```
<HTML>
<HEAD>
 <TITLE>Cursus NovaCollege</TITLE>
 <LINK href="../../vb.css" rel="stylesheet" type="text/css">
 <script type="text/javascript">
<!--
        function bereken(){
                f.product.value = f.g1.value * f.g2.value
        }
// -->
</script>
</HEAD>
<BODY>

<h1>Toepassing DOM 0</h1>
<form name="f">
<table>
<tr>
      <td>Getal 1: </td>
      <td><input name="g1" type="text" value="10"></td>
</tr>
<tr>
      <td>Getal 2: </td>
      <td><input name="g2" type="text" value="50"></td>
</tr>
<tr>
      <td>Som: </td>
      <td>
                        <input name="product" type="text" value="">
                        <input type="button" value="Maak product"
onclick="bereken()">
           </td>
</tr>
</table>
</form>
</BODY>
</HTML>
```

7.4 Tussenliggende DOM's

Na DOM 0 ontwikkelden Netscape en Microsoft op eigen houtje een eigen DOM.

Netscape koos voor *document.layers*, waarbij ze hun eigen filosofie volgden en ook een nieuwe tag **LAYER** gingen gebruiken. Deze DOM zullen we niet verder bespreken daar zelfs Netscape nu is afgestapt van deze DOM.

Microsoft ging werken met *document.all* : één object-collectie van waaruit je alle elementen op de pagina kan bereiken.

Voorbeeld document.all:
In de functie verdubbel wordt eerst gecontroleerd of document.all wordt ondersteund:

```
if(document.all)
<HTML>
<HEAD>
 <TITLE>NovaCollege</TITLE>
 <LINK href="vb.css" rel="stylesheet" type="text/css">
 <script type="text/javascript">
 <!--
        function verdubbel(){
                if(document.all){
                        document.all["g1"].value = 2 * document.all["g1"].value;
                }
        }
 // -->
 </script>
</HEAD>
<BODY>

<h1>Tussendom: document.all</h1>
<form name="f">
<input id="g1" type="text" value="10">
<input type="button" value="Verdubbel" onclick="verdubbel()">
</form>
</BODY>
</HTML>
```

7.5 W3C DOM

Inleiding

De DOM voorgesteld door het W3C wordt in level 1 door Nescape en IE 5+ ondersteund, level 2 door de versie 6 - generatie.

In de W3C DOM is elk object, wat het ook is, een node:

```
<p>Dit is een alinea</p>
```

Hier maken we twee nodes: een elementnode voor het element P en een tekstnode voor de inhoud. De tekstnode zit in de elementnode en wordt beschouwd als een childnode.

```
<p>Dit is <b>een alinea</b></p>
```

Hier hebben we een elementnode P, deze node heeft als childnodes een tekstnode voor de inhoud 'Dit is' en een elementnode B. Deze elementnode heeft een childnode voor de inhoud 'een alinea'.

```
<p align="right">Dit is <b>een alinea</b></p>
```

De laatste nodes zijn attribuutnodes. De elementnode voor P heeft een attribuutnode align met een child-tekstnode 'right'.

Attribuutnodes worden niet als childnodes van een elementnode beschouwd.

Elementen manipuleren

Om een element te manipuleren moet je vanuit Javascript eenduidig kunnen opgeven met welk element je wenst te werken.

Methode	***Beschrijving***
• *getElementByID*	Referentie naar het element met aangegeven attribuut id
• *getElementsByName*	Referentie naar element(en) met aangegeven name
• *getElementsByTagName*	Referentie naar aangegeven elementen

Deze mogelijkheid van het DOM stelt je in staat de inhoud en attribuutwaarden van elk element te manipuleren.

De eenvoudigste manier om dit te bereiken is gebruik maken van de methode *getElementById* van het object *document*:

```
document.getElementById("kop")
```

deze opdracht vormt een referentie naar het element met *id kop*.

Inhoud veranderen:

We hebben bijvoorbeeld een element met *id tekst1*

Wil je de inhoud van dit element veranderen, dan kan je dit volgens de W3CDOM op volgende manier:

```
document.getElementById("tekst1").firstChild.nodeValue='De tekst is veranderd !';
```

Je kan ook met de *innerHTML*-eigenschap van een element werken:

```
document.getElementById("kop2").innerHTML ='Ook aangepast !';
```

Attribuutwaarde veranderen:
Hier hebben we een element met id uitlijn.

Op volgende manier kan je de uitlijning van deze alinea op 'right' instellen:

```
node = document.getElementById('uitlijn');
node.setAttribute('align','right');
```

Nodes maken en verwijderen
De mogelijkheden met DOM zijn van die aard dat je zelfs nieuwe elementen op een pagina kan plaatsen en bestaande elementen kan verwijderen.

We voegen in onderstaande toepassing een **HR**-element toe aan de pagina.

Het **HR**-element wordt **toegevoegd** als child-element van het element met *id alinea.*

```
var x = document.createElement('HR');
document.getElementById('alinea').appendChild(x);
```

We maken het **HR**-element met de methode *createElement* van het object *document* en plaatsen het in een variabele **x**.
We voegen het element toe als child van het element *alinea* met de methode *appendChild*.

Het **verwijderen** van dit element gebeurt als volgt:

```
var node = document.getElementById('alinea');
if(node.childNodes[1]) node.removeChild(node.childNodes[1]);
```

We plaatsen een referentie naar het element *alinea* in de variabele node.
We controleren of deze node reeds een tweede childNode heeft (de eerste childNode met index 0 is de inhoud van de alinea).
Als deze node bestaat, werd de lijn reeds op het scherm gezet en kunnen we ze verwijderen met de methode *removeChild* van het element. Deze methode ontvangt als argument de te verwijderen node.

7.6 Objectdetectie

Browserdetectie
Misschien werk je wel met een browser waarbij bepaalde van bovenstaande toepassingen niet functioneren. Als webontwikkelaar hebben we natuurlijk niet te kiezen met welke browser de gebruiker onze pagina's bezoekt.
Daarom moeten we ervoor zorgen dat deze gebruiker niet om de oren wordt geslagen met javascript-errors.

Met dit doel voor ogen is het dus interessant te weten met welke browser de gebruiker werkt, daar de implementatie van het DOM verschilt naargelang de gebruikte browser.

Met Javascript kan je eenvoudig nagaan welke instellingen de gebruiker heeft:

```
document.write(navigator.userAgent);
```

U werkt met: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; GTB6.4; .NET CLR 1.1.4322; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; InfoPath.2)

Met de geziene String-functies kan je nu heel eenvoudig testen welke browser er gebruikt wordt.

Objectdetectie

In feite is voor het ontwikkelen van cross-browser toepassingen niet zozeer van belang met welke browser de gebruiker surft, maar wel de wetenschap of de door jou gebruikte technieken door die browser worden ondersteund.

Het is heel vervelend voor toepassingen steeds te moeten uitzoeken welke browser de techniek ondersteund, en dan nog telkens uw code aan te passen wanneer een browser een nieuwe versie uitbrengt.

Vandaar dat het ten zeerste aan te raden is objectdetectie te gebruiken.

Controle **document.images** (DOM 0)

```
if(document.images){
        ... code met document.images ...
}
```

Controle op **document.getElementById** (DOM 1)

```
if(document.getElementById){
        ... code met document.getElementById ...
}
```

Cross browser toepassingen

Wanneer je een cross-browser toepassing maakt en je wenst een verwijzing naar een object te maken, dan ben je niet absoluut zeker dat *getElementById* ondersteund wordt door de browser.

Onderstaande functie detecteert welke de mogelijkheden van de browser zijn en bouwt een referentie op:

Voorbeeld op volgende pagina >>

```
function getObj(name)
{
  if (document.getElementById)
  {
        this.obj = document.getElementById(name);
  }
  else if (document.all)
  {
        this.obj = document.all[name];
  }
  else if (document.layers)
  {
        this.obj = document.layers[name];
  }
}
```

Deze functie is afgeleid van de DHTML API op de site van quirksmode.

Gebruik van deze functie:
Deze functie vormt eigenlijk een constructor-functie. We kunnen deze functie gebruiken als sjabloon om objecten aan te maken.

```
var x = new getObj('idvanelement');
```

Deze functie zorgt ervoor dat de variabele die we gebruiken een object zal bevatten met de eigenschap *obj*, met deze eigenschap hebben we toegang tot de html-eigenschappen van het element dat we meegeven aan de *getObj* -functie.

Voorbeeld op volgende pagina >>

```
<HTML>
<HEAD>
 <TITLE>NovaCollege</TITLE>
 <LINK href="vb.css" rel="stylesheet" type="text/css">
 <script type="text/javascript">
<!--
   function verandertekst(tekst)
   {
            var x = new getObj('kop');
            x.obj.innerHTML = tekst;
   }

      function getObj(name){
            if (document.getElementById){
                  this.obj = document.getElementById(name);
            }
            else if (document.all){
                  this.obj = document.all[name];
            }
            else if (document.layers){
                  this.obj = document.layers[name];
            }
      }
-->
</script>
</HEAD>
<BODY>
<h1 id="kop">Hoofding</h1>
<a href="javascript:verandertekst('test')">Test</a> |
<a href="javascript:verandertekst('Nog eens')">Nog eens</a>
</BODY>
</HTML>
```

Extra uitleg: werken met objecten en eigenschappen

```
<HTML>
<HEAD>
 <TITLE>NovaCollege</TITLE>
 <LINK href="vb.css" rel="stylesheet" type="text/css">
 <script type="text/javascript">
<!--

        function persoon(n,v,l){
                this.naam = n;
                this.voornaam = v;
                this.leeftijd = l;
        }

        var p1 = new persoon("Peters","Jan",45);
        var p2 = new persoon("Janssens","Sofie",23);

-->
</script>
</HEAD>
<BODY>
<script type="text/javascript">
<!--
        document.write("<br>" +p1.naam);
        document.write("<br>" +p2.voornaam);
-->
</script>
</BODY>
</HTML>
```

De functie persoon vormt hier een constructorfunctie voor *persoon* -objecten.

Een persoon maken we door een variabele te declareren als **new** *persoon*, we geven drie argumenten mee: de **naam**, **voornaam** en **leeftijd** van de persoon.
Deze drie argumenten worden in de constructor-functie *persoon* ontvangen onder de variabelenamen **n**, **v** en **l**. In deze constructorfunctie verwijst **this** naar de objectvariabele die je wenst te declareren: eerst **p1**, daarna **p2**.

Het resultaat van deze code is dat we 2 persoonsobjecten **p1** en **p2** hebben met elk drie eigenschappen: *naam*, *voornaam* en *leeftijd*.

Deze eigenschappen kunnen we opvragen door de naam van het object te noteren gevolgd door een punt en de naam van de eigenschap.

Hoofdstuk 8
Javascript: Events

8.1 Inleiding

Events vormen het hart van een Javascript-pagina. Event-handling staat toe dat de Javascript opdrachten op het juiste moment worden uitgevoerd, wanneer de juiste gebreurtenis heeft plaatsgevonden:
Een berekening wordt uitgevoerd wanneer een gebruiker op een knop klikt, een figuur verandert wanneer de gebruiker er met de muis over beweegt, een animatie wordt gestart wanneer de pagina geladen is ...

Een groot probleem met event-handling is dat browsers absoluut geen eenduidig meechanisme hebben om events af te handelen.

8.2 Events

Hier vind je een overzicht van de belangrijkste events die je met Javascript op een webpagina kan afhandelen.

<table>
<tr><th>Soort gebeurtenis</th><th>Beschrijving</th><th>Gebeurtenissen</th></tr>
<tr><td>Muis</td><td>Reageren op muisacties van de gebruiker</td><td>
<table>
<tr><td>click</td><td>Klikken</td></tr>
<tr><td>Mouseover</td><td>Muisaanwijzer</td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>
<table>
<tr><td></td><td>Boven element</td></tr>
<tr><td>Mouseout</td><td>Muisaanwijzer</td></tr>
<tr><td>Mousedown</td><td>Muisaanwijzer verlaat element</td></tr>
<tr><td>Mouseup</td><td>Muisknop wordt losgelaten</td></tr>
<tr><td>dblclick</td><td>Dubbelklik</td></tr>
<tr><td>mousemove</td><td>Muisaanwijzer beweegt</td></tr>
<tr><td>mouseenter</td><td>MS mouseover-variant, zonder bubbling</td></tr>
<tr><td>mouseleave</td><td>MS mouseout-variant, zonder bubbling</td></tr>
</table>
</td>
</tr>
<tr>
<td>Toets</td>
<td>Reageren op toetsaanslagen</td>
<td>
<table>
<tr><td>keypress</td><td>Toets wordt aangeslagen</td></tr>
<tr><td>keydown</td><td>Toets wordt ingedrukt</td></tr>
<tr><td>keyup</td><td>Toets wordt losgelaten</td></tr>
</table>
</td>
</tr>
<tr>
<td>Formulieren</td>
<td>Reageren op acties voor een formulier en formulierelementen</td>
<td>
<table>
<tr><td>click</td><td>Klikken</td></tr>
<tr><td>Mouseover</td><td>Muisaanwijzer</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>Interface</td>
<td>Acties als resultaat van gebruikersacties:

Vb. Wanneer de gebruiker klikt op een formulierelement wordt dit element actief: het krijgt de focus.</td>
<td>
<table>
<tr><td>focus</td><td>Een element krijgt de focus</td></tr>
<tr><td>Blur</td><td>Een element verliest de focus</td></tr>
<tr><td>load</td><td>Een element is geladen</td></tr>
<tr><td>unload</td><td>Een element is niet meer geladen</td></tr>
<tr><td>Resize</td><td>De afmetingen van een element worden veranderd</td></tr>
<tr><td>scroll</td><td>Er wordt gescrolld met een element</td></tr>
</table>
</td>
</tr>
</table>

Nog veel meer events vind je bij MSDN (Microsoft).

8.3 Inline event-handling

Nu we een heel aantal events hebben leren kennen moeten we die events ook kunnen afhandelen op onze pagina's.

Algemeen kunnen we stellen dat een event kan worden afgehandeld met een event-handler die als naam het event heeft en een prefix 'on': onmouseover, onmouseout, onload, onfocus, onsubmit, ...

Een eerste manier om events af te handelen is **inline**-afhandeling. Deze manier van gebeurtenisafhandeling bestaat erin dat een event-handler wordt geplaatst als attribuut in de (X)HTML-tag.

Deze manier van werken is de oorspronkelijke manier van gebeurtenisafhandeling en wordt door alle browsers ondersteund.

Een nadeel aan deze manier van werken is dat de event-handlers in de (X)HTML-tag ingebed zijn en je dus een mengeling hebt van logische programmacode in Javascript en inhoud van je document (X)HTML.
Voor grotere toepassingen en voor specifieke taken kan dit vervelend zijn, en moeilijk in onderhoud.

Voorbeelden

```
<img src="../images/geiten.jpg" width="134" height="101" alt="" border="0"
onmouseover="alert('over de figuur');">
```

```
<a href="http://www.novacollege.com" onclick="alert('We surfen niet');return
false">Surf naar novacollege</a>
```

Wanneer je klikt op deze hyperlink wordt een *alert* getoond. De hyperlink wordt niet gevolgd doordat we de waarde false retourneren in onze event-handler, waardoor de standaardactie -het volgen van de link- niet wordt uitgevoerd.

Wanneer je vanuit je eigen event-handler false retourneert, wordt de standaardeventhandler niet uitgevoerd. Een **uitzondering** op deze regel is het tonen van een bericht in de statusbalk bij een onmouseover-actie: hier moeten we true retourneren, al willen we niet dat de gekoppelde url in de statusbalk verschijnt (mysterieus...).

```
<a href=" http://www.novacollege.com "
onmouseover="window.status='Terug naar startpagina';return true"
onmouseout="window.status="">novacollege</a>
```

Voor hyperlinks bestaat zelfs de mogelijkheid javascript-code in het href-attribuut op te nemen:

```
<a href="javascript:alert('U heeft geklikt')">Klik me</a>
```

Gebruik van this

```
<div id="div1" style="width:100px;border:solid 1px"
  onmouseover="doeIets(this)">Divisie 1</div><br>
<div id="div2" style="width:100px;border:solid 1px"
  onmouseover="doeIets(this)">Divisie 2</div><br>

<script type="text/javascript">
<!--
function doeIets(obj)
{
        var divId = obj.id;
        alert('Muis over: ' + divId + '!')
}
-->
</script>
```

Divisie 1

Divisie 2

8.4 Traditionele event-handling

Event-handlers installeren en uitvoeren

Met de stormachtige opkomst van DHTML was er nood aan een nieuw model voor event-handling.

De grote browsers ondersteunen dit model vanaf hun versie 4.

In dit model zijn de events volledig te benaderen vanuit Javascript: je hoeft geen code meer toe te voegen aan HTML-tags.

```
<div id="divtrad" style="width:100px;border:solid 1px" >Divisie</div>

<script type="text/javascript">
<!--
mijndiv = document.getElementById('divtrad');
mijndiv.onclick = muisKlik;
function muisKlik()
{
        alert('Klik op divisie!');
}
-->
</script>
```

Divisie

Je kan de event-handler koppelen vanuit Javascript-code en hoeft geen HTML-attribuut meer te gebruiken.

In dit vernieuwde event-model zijn event-handlers zoals *onclick, onmouseover, onfocus,* ... eigenlijk eigenschappen van elementen.

Merk op dat je de functie koppelt aan de event-handler **zonder** haakjes na de naam van de functie. Zet je daar wel haakjes, dan wordt het resultaat van de functie gekoppeld aan de event-handler in plaats van de functie zelf, dit is niet de bedoeling.

Gebruiken we de functie *getObj* uit ***dom.js*** dan wordt dit:

```
<div id="divtrad" style="width:100px;border:solid" >Divisie</div>

<script type="text/javascript">
<!--
mijndiv = new getObj('divtrad');
mijndiv.obj.onclick = muisKlik;
```

Vervolg>>

```
function muisKlik()
{
        alert('Klik op divisie!');
}
-->
</script>
```

Voor ondersteuning in **Netscape4** is er nog een extra lijn code nodig om event-handlers te installeren. Je moet de methode captureEvents nog toepassen op het element.

```
mijndiv = new getObj('divtrad');
mijndiv.obj.onclick = muisKlik;
if (mijndiv.captureEvents) mijndiv.captureEvents(Event.CLICK)
```

Event-handlers initiëren

Je kan nu zelfs event-handlers initiëren zonder dat de gebruiker daarvoor iets hoeft te doen:

```
mijndiv.onclick();
```

Dit stukje code zorgt ervoor dat de onclick event-handler van **mijndiv** wordt uitgevoerd. Dit zal resulteren in het uitvoeren van de functie *muisKlik*.

```
<a href="javascript:mijndiv.onclick()">Probeer het</a>
```

Microsoft heeft hiervoor nog een eigen methode ontwikkeld:

```
element.fireEvent('onclick');
```

Gebruik van this

Het sleutelwoord **this** kunnen we nu gebruiken binnen de functie die het event afhandelt. Op deze manier kunnen we een functie schrijven die bruikbaar is voor meerdere elementen en binnen de functie nagaan welk element de functie heeft aangeroepen.

```
<div id="test1" style="width:150px;border:solid 1px">Een testdivisie</div><br>
<div id="test2" style="width:150px;border:solid 1px">Nog een testdivisie</div>
<script type="text/javascript">
<!--
t1 = document.getElementById("test1");
t2 = document.getElementById("test2");

t1.onclick = verandertekst;
t2.onclick = verandertekst;
```

Vervolg>>

```
function verandertekst(){
	this.innerHTML = "Tekst is aangepast.";
}

-->
</script>
```

Anonieme functies

In de volgende toepassing zorgen we ervoor dat de achtergrondkleur van elk h5-element wordt veranderd wanneer de gebruiker er met de muis over beweegt. De eerste toepassing maakt nog geen gebruik van anonieme functies.

We maken hiervoor gebruik van de *getElementsByTagName*-methode. Deze methode retourneert een *Array* van alle objecten met een aangegeven tagnaam. Met een lus kunnen we nu eventhandlers voor alle elementen in deze *Array* installeren.

In feite passen we hier een CSS-eigenschap van deze tag aan: veel meer hierover in de cursus DHTML.

```
<h5>Eerste hoofding</h5>
<h5>Tweede hoofding</h5>
<script type="text/javascript">
<!--
if (document.getElementsByTagName)
	var x = document.getElementsByTagName('h5');
else if (document.all)
	var x = document.all.tags('h5');
if(x){
	for (var i=0;i<x.length;i++)
	{
		x[i].onmouseover = over;
		x[i].onmouseout = out;
	}
}
function over()
{
	this.style.backgroundColor='#FFCC33'
}
function out()
{
	this.style.backgroundColor='#ffffff'
}
-->
</script>
```

Aangezien de functies *over* en *out* zo eenvoudig zijn is het eleganter hiervoor **anonieme functies** te gebruiken.

We herwerken het voorbeeld voor h6-elementen, met anonieme functies:

```
<h6>Eerste hoofding niveau 6</h6>
<h6>Tweede hoofding niveau 6</h6>
<script type="text/javascript">
<!--
if (document.getElementsByTagName)
        var x = document.getElementsByTagName('h6');
else if (document.all)
        var x = document.all.tags('h6');
if(x){
        for (var i=0;i<x.length;i++){
          x[i].onmouseover = function ()
           {this.style.backgroundColor='#CCFFCC'}

          x[i].onmouseout = function ()
           {this.style.backgroundColor='#ffffff'}
        }
}
-->
</script>
```

Een anonieme functie wordt gekenmerkt door het sleutelwoord **function** onmiddellijk gevolgd door een paar ronde haken en de functieopvulling tussen accolades. De functie heeft geen naam en is dus anoniem.

Problemen

Er rijzen problemen wanneer je twee functionaliteiten wenst te voorzien voor een eventhandler. Je kan natuurlijk alle functionaliteit die je wenst in een functie stoppen en die registreren bij de event-handler.

Programmeren verloopt echter een stuk gestructureerder en onderhoudsvriendeljiker wanneer je voor een functie slechts 1 taak voorziet.

Stel dat een gebruiker ergens kan klikken om een animatie te starten, maar hiervoor moeten eerst een reeks getallen worden gesorteerd. Je zou nu misschien geneigd zijn hetvolgende te doen (op voorwaarde dat de functies sorteerGetallen en startAnimatie bestaan):

element.onclick = sorteerGetallen;
element.onclick = startAnimatie;

Deze code levert geen fout op, maar de tweede registratie overschrijft echter de eerste, waardoor de getallen niet zullen worden gesorteerd.

Een mogelijke oplossing hiervoor is een functie te registreren die beide functie aanroept:

```
element.onclick = function () {sorteerGetallen();startAnimatie()}
```

8.5 Geavanceerde event-handling

W3C

De W3C aanbeveling stelt dat we een event-handler kunnen installeren met de methode *addEventListener*. Deze methode heeft drie argumenten:

1. *Event* waarop wordt gereageerd
2. Actie die moet worden uitgevoerd (functienaam, anonieme functie)
3. Boolean (false = bubbling, true= capturing - zie verderop)

In een afhandelingsfunctie refereert **this** naar het bronobject van de actie.

EventListeners verwijderen doe je met *removeEventListener*, met dezelfde argumenten als *addEventListener*.

Microsoft

Microsoft ondersteund de methode *attachEvent*

Deze methode ontvangt twee argumenten:

1. *Event* -handler
2. Actie die moet worden uitgevoerd (functienaam, anonieme functie)

Events worden uitgevoerd met bubbling (zie verderop)

Het sleutelwoord **this** in een afhandelingsfunctie verwijst steeds naar het object *window*, en is dus niet bruikbaar.

Een functie loskoppelen doe je met de methode *detachEvent*, met dezelfde argumenten als *attachEvent*.

Voorbeeld

Een algemeen probleem rijst bij het koppelen van meerder functies aan een event-handler: je weet niet welke routine eerst zal worden uitgevoerd.

```
<div class="actie">
<div id="divgeav">Testdivisie</div>
</div>

<script type="text/javascript">
<!--
var d = document.getElementById('divgeav');

//W3C
if(d.addEventListener) {
        d.addEventListener('click',toonBericht,false);
        d.addEventListener('click',toonNogEenBericht,false);
}
```

Vervolg>>

```
//Microsoft
else if(d.attachEvent){
        d.attachEvent('onclick',toonBericht);
        d.attachEvent('onclick',toonNogEenBericht);
}

function toonBericht(){
        alert('klikken gedetecteerd');
}
function toonNogEenBericht(){
        alert('Nog een bericht');
}
-->
</script>
```

8.6 Toegang tot event-eigenschappen

Wanneer je toepassingen maakt heb je vaak toegang nodig tot de eigenschappen van een event. Wanneer je bijvoorbeeld een functie maakt die kan aangeroepen worden vanaf verschillende elementen, dan wil je dikwijls weten welk element de actie heeft veroorzaakt.

Denk maar aan een online winkel, het is interessant te weten als de gebruiker klikt op een knop om een artikel te kopen dat je kan weten welk artikel de klant wenst te kopen :)

W3C
De afhandelingsfunctie ontvangt automatisch een argument, dit argument bevat een event-object. Dit object bevat hetgeen we nodig hebben.

Je mag het argument in een variabele stoppen met een naam naar keuze, meestal wordt de naam e gebruikt.

Microsoft
Het object *window* heeft een eigenschap *event* die steeds informatie bevat over het laatste event dat heeft plaatsgehad.

Voorbeeld
Bij W3C-browsers ontvangen we het eventobject als argument in de afhandelingsfuncties, voor Microsoft IE vinden we die informatie in *window.event.* We zorgen ervoor dat we een variabele e hebben die in beide situaties de *event*-informatie bevat.

De eigenschap *type* bevat voor beide browsers het type event.

Voorbeeld op volgende pagina >>

```
<div id="divtoegang1">Testdivisie</div>

<script type="text/javascript">
<!--
var d = document.getElementById('divtoegang1');

//W3C event-handler
if(d.addEventListener) {
	d.addEventListener('click',toonBericht,false);
}

//Microsoft event-handler
if(d.attachEvent) {
	d.attachEvent('onclick',toonBericht);
}


// argument e ontvangen voor W3C/Netscape
function toonBericht(e){
	// e opvullen voor Microsoft IE
	if (!e) var e = window.event;

	//statements
	alert('Event gedetecteerd van het type: ' +e.type);
}

-->
</script>
```

Dit werkt ook bij **traditionele event-handling**.

```
<p id="toegang2">Een alinea om op te klikken</p>

<script type="text/javascript">
<!--
t = document.getElementById('toegang2');
t.onclick = tradKlik;
function tradKlik(e)
{
	if(!e) e = window.event;
	alert(e.type);
}
-->
</script>
```

Gebruik je **inline event-handlers** dan geef je *event* mee als argument naar de functie. *window.event* is de correcte eigenschap in het Microsoft model, de andere browsers ondersteunen dit ook in dit speciale geval.

```
<p onclick="inlineKlik(event)">Een alinea om op te klikken<p>
<script type="text/javascript">
<!--
function inlineKlik(e)
{
        alert(e.type);
}
-->
</script>
```

8.7 Event-eigenschappen

Nu je weet hoe je toegangkrijgt tot de eigenschappen van een event kan je volgende informatie opvragen:

Eigenschap	Beschrijving
type	Het soort event
target	W3C / Netscape: element waarop het event plaatshad
srcElement	Microsoft: element waarop het event plaatshad
which	oude eigenschap voor Netscape 4: aangeslagen toets of muisknop Muisknop: • Links: 1 • Midden(wiel): 2 • Rechts: 3
keyCode	W3C / Microsoft: aangeslagen toets

<table>
<tr><td>button</td><td>Ingedrukte muistoets

<table><tr><th>Knop</th><th>WC3</th><th>Microsoft</th></tr><tr><td>Links</td><td>0</td><td>1</td></tr><tr><td>Midden</td><td>1</td><td>4</td></tr><tr><td>Rechts</td><td>2</td><td>2</td></tr></table></td></tr>
<tr><td>screenX, screenY</td><td>Positie van muisaanwijzer t.o.v. het scherm</td></tr>
<tr><td>pageX, pageY</td><td>Positie van muisaanwijzer t.o.v. het document</td></tr>
<tr><td>clientX, clientY</td><td>Positie van muisaanwijzer t.o.v. het document</td></tr>
<tr><td>relatedTarget</td><td>W3C: doelelement bij onmouseout en bronelement bij onmouseover</td></tr>
<tr><td>toElement, fromElement</td><td>Microsoft: doelelement bij onmouseout en bronelement bij onmouseover</td></tr>
</table>

8.8 Element-detectie

In dit voorbeeld zie je twee knoppen die een event-afhandelingsfunctie *toonElement* delen. Binnen de functie *toonElement* kunnen we nagaan welke knop werd aangeklikt.

De functie bevat een extra statement om een bug in de browser Safari te omzeilen: als een event plaatsgrijpt op een element dat text bevat wordt de tekstnode gezien als bron in plaats van het element waarin de tekst vervat is.

Het *nodeType* van een tekstnode is *3*.Als het target-element van dit type is gebruiken we de *parentNode* van dit element.

Voorbeeld op volgende pagina >>

```
<input id="knop1" type="button" value="Knop1"><br><br>
<input id="knop2" type="button" value="Knop2">
<script type="text/javascript">
<!--
var k1 = document.getElementById('knop1');
var k2 = document.getElementById('knop2');

//W3C event-handler
if(k1.addEventListener) {
	k1.addEventListener('click',toonElement,false);
	k2.addEventListener('click',toonElement,false);
}

//Microsoft event-handler
if(k1.attachEvent) {
	k1.attachEvent('onclick',toonElement);
	k2.attachEvent('onclick',toonElement);
}

function toonElement(e)
{
	var targ;
	if (!e) var e = window.event;
	if (e.target) targ = e.target;
	else if (e.srcElement) targ = e.srcElement;
	if (targ.nodeType == 3) // opvangen Safari bug
		targ = targ.parentNode;
	alert('U hebt geklikt op: ' +targ.id);
}
-->
</script>
```

8.9 Toetsdetectie

Wil je weten welke toets werd ingedrukt, dan kan dit met de eigenschap *keyCode* van het object *event*. Netscape 4 werkt met de eigenschap *which*.

Deze eigenschap bevat een numerieke waarde (ASCII) voor het teken, deze waarde kan je met de methode *fromCharCode* van het object *String* omzetten in een letterteken.

```
<input id="teksttoets" type="text" value="">

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

Vervolg>>

```
var t = document.getElementById('teksttoets');

//W3C event-handler
if(t.addEventListener) {
	t.addEventListener('keypress',toonToets,false);
}

//Microsoft event-handler
if(t.attachEvent) {
	t.attachEvent('onkeypress',toonToets);
}

function toonToets(e)
{
	var code;
	if (!e) var e = window.event;
	if (e.keyCode) code = e.keyCode;
	else if (e.which) code = e.which
	var character = String.fromCharCode(code);
	alert('Karakter was ' + character);
}
-->
</script>
```

Opgepast, bij bepaalde browsers moet je onderscheid maken tussen *keydown* en *keypress* om specifieke toetsen te detecteren.
Meer over bvb. de backspace-toets.

8.10 Muisknopdetectie

Het indrukken van een muisknop wanneer de muisaanwijzer boven een element staat kan je met de eigenschap *button*. Netscape 4 gebruikt *which*.

Gebruik beter *mousedown* of *mouseup* in plaats van *click*.

Gezien de rechtermuisknop in W3C en Microsoft afhandeling *button* steeds de waarde 2 heeft is het de eenvoudigste knop om op te testen.

Wel nog oppassen voor Netscape 4, daar is de waarde van *which* *3*.

```
<div id="divmuis" style="width:100px;height:50px;border:solid 1px;
background:#CCFFCC;padding:10px">Klik me</div>

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

Vervolg>>

```
var m = document.getElementById('divmuis');

//W3C event-handler
if(m.addEventListener) {
	m.addEventListener('mousedown',toonMuisKnop,false);
}

//Microsoft event-handler
if(m.attachEvent) {
	m.attachEvent('onmousedown',toonMuisKnop);
}

function toonMuisKnop(e)
{
	var rk;
	if (!e) var e = window.event;
	// rk wordt true als de rechtermuisknop wordt ingedrukt
	// anders wordt rk false
	if (e.which) rk = (e.which == 3);
	else if (e.button) rk = (e.button == 2);
	alert('Rechts geklikt ? \n\n' + (rk ? "ja" : "nee"));
}
-->
</script>
```

8.11 Muispositiedetectie

Het bepalen van de positie van de muisaanwijzer ten opzichte van de linkerbovenhoek van het **scherm**:

```
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
</HEAD>
<BODY>
<h2>Klik ergens op het document</h2>
<script type="text/javascript">
<!--
//W3C event-handler
if(document.addEventListener) {
	document.addEventListener('mousedown',toonMuisPositie,false);
}
//Microsoft event-handler
if(document.attachEvent) {
	document.attachEvent('onmousedown',toonMuisPositie);
}
```

Vervolg>>

```
function toonMuisPositie(e)
{
        if (!e) var e = window.event;
        alert("X: " +e.screenX +"\n\nY: " +e.screenY);
}
-->
</script>
</body>
</html>
```

Om de positie te bepalen ten opzichte van het document heb je *pageX, pageY, clientX, clientY* en eventueel de *srollLeft* en *scrollTop* - eigenschappen van BODY.

8.12 Volgorde van Events

Stel dat je binnen het ene element een ander element onderbrengt. Beide elementen hebben een click-event. Welk event wordt dan uitgevoerd? Of allebei? En in welke volgorde?

Bubbling
Bubbling is de term die wordt gebruikt wanneer events afgehandeld worden van binnen naar buiten.
Dit model wordt gebruikt door Microsoft.

Capturing
Bij capturing worden de events afgehandeld van buiten naar binnen.
Dit model wordt gebruikt door Netscape.

W3C
Voor browsers die de W3C-eventregistratie ondersteunen kan je als derde argument meegeven of er bubbling (false) of capturing (true) moet worden gebruikt.

```
element1.addEventListener('click',doIets,true);
element2.addEventListener('click',doIets2,false);
```

Dit is enkel van belang als je overkoepelende elementen hebt met een zelfde event-handler.

Gebruik je het traditionele model:

```
element1.onclick = doIets;
```

Dan wordt event-bubbling gebruikt

Voorbeeld
We maken een document met twee divisies die binnen elkaar gelegen zijn, we registreren het click-event voor beide divisies en voor het document.

De functie klik wordt aan de drie event-handlers gekoppeld. Wanneer we op de binnenste divisie klikken zien we drie berichtvensters verschijnen: we klikken op de binnenste maar ook op de buitenste divisie, we klikken ook op het document.

```
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
</HEAD>
<BODY>
<h2>Klik</h2>
<div id="kopdiv" style="width:200px;height:200px;padding:50px;
background:#FFFF99;border:solid 1px">
        <div id="subdiv" style="width:100px;height:100px;padding:50px;
background:#CCFFCC;border:solid 1px">
        </div>
</div>
<script type="text/javascript">
document.id = "Hetdocument"   // voor currentTarget.id
k = document.getElementById('kopdiv');
s = document.getElementById('subdiv');

<!--
//W3C event-handler
if(document.addEventListener) {
        document.addEventListener('click',klik,false);
        k.addEventListener('click',klik,false);
        s.addEventListener('click',klik,false);
}

//Microsoft event-handler
if(document.attachEvent) {
        document.attachEvent('onclick',klik);
        k.attachEvent('onclick',klik);
        s.attachEvent('onclick',klik);
}
function klik(e)
{
        if (!e) var e = window.event;
        if (e.target) targ = e.target;
        else if (e.srcElement) targ = e.srcElement;
        if (targ.nodeType == 3) // opvangen Safari bug
                targ = targ.parentNode;
        alert('U hebt geklikt op: '
        +(e.currentTarget ? e.currentTarget.id : targ.id));
}
```

Vervolg>>

```
-->
</script>
</body>
</html>
```

De eigenschap *currentTarget* werkt niet bij Microsoft, deze eigenschap is nochtans bijzonder interessant daar deze een referentie retourneert naar het element waarvoor de klikactie nu wordt afgehandelt.

De eigenschap *srcElement* voor Microsoft bevat steeds een verwijzing naar het binnenste element.

Ik geef document hier een id doordat ik voor browsers die *currentTarget* wel ondersteunen het *id* van het element toon.

Bubbling onderdrukken
Bij W3C browsers kan je bubbling onderdrukken met:

```
e.stopPropagation()
```

Voor Microsoft-browsers gebruik je:

```
window.event.cancelBubble = true
```

Cross-browser code:

```
function doIets(e)
{
	if (!e) var e = window.event;
	e.cancelBubble = true;
	if (e.stopPropagation) e.stopPropagation();
}
```

8.13 Muisacties in detail

Je hebt zopas geleerd hoe je kan detecteren met welke muisknop een element werd aangekikt.

Wanneer de gebruiker de muis gebruikt kan deze echter heel wat verschillende acties uitvoeren.

Klikken
Wanneer de gebruiker klikt op een element worden in feite drie events uitgevoerd:

1. *mousedown*: muisknop naar beneden
2. *mouseup*: muisknop naar boven
3. *click*: mousedown en mouseup gedetecteerd

On klik-event wordt dus pas geraised wanneer de gebruiker de muis indrukt boven een element en de muis loslaat boven hetzelfde element.

Wees bewust van het feit dat wanneer je deze drie event-handlers koppelt aan een element ze alle drie worden uitgevoerd als de gebruiker klikt.

Dubbelklikken
Deze actie valt uiteen in de volgende events:

1. mousedown: muisknop naar beneden
2. mouseup: muisknop naar boven
3. click: mousedown en mouseup gedetecteerd
4. mouseup: muisknop naar boven
5. dblclick: er werd gedubbelklikt

Het is te vermijden een *dblclick* en *click* event op hetzelfde element te definiëren.

Bewegen met de muis
mousemove vangt muisbewegingen boven een element op. Wees op je hoede wanneer je dit event voorziet van code, elke beweging van 1 pixel boven het element resulteert in het uitvoeren van de code.

Elementen binnengaan en verlaten
Met *mouseover* kan je detecteren of de muisaanwijzer boven een element komt, onmouseout onderschept het verlaten van een element.

Door event-bubbling kunnen meerdere events ineens afgevuurd worden.

Het W3C voegt de eigenschap *relatedTarget* toe aan het *event*, deze eigenschap bevat een referentie naar:

- Element waarvan de muisaanwijzer kwam bij *mouseover*
- Element waarnaar de muisaanwijzer gaat bij *mouseout*

Microsoft heeft daarvoor de twee eigenschappen *fromElement* en *toElement*

Microsoft heeft voor muisbeweging *mouseover* en *mouseout* ook twee events beschikbaar die niet 'bubblen' (geen events afvuren van overkoepelende elementen): *mouseenter* en *mouseleave*.

Hoofdstuk 9
Javascript: Formulieren

1. Inleiding
2. Tekstvakken, tekstgebieden en verborgen elementen
3. Keuzelijsten : select
4. Aankruisvakjes: checkbox
5. Keuzerondjes: radiobutton
6. Formulieren versturen - formuliercontrole
7. Formulierelementen uitschakelen - disable / enable
8. Oefeningen

9.1 Inleiding

In de hoofdstukken DOM en Events heb je geleerd hoe je elementen op een webpagina kan benaderen en kan reageren op acties die worden ondernomen.

In dit hoofdstukken focussen we ons op formulierelementen zoals tekstvakken, keuzelijsten, aankruisvakken en keuzerondjes. Hoe kan je met andere woorden nagaan welke keuzes de gebruiker heeft gemaakt, en hoe kan je de keuzes codematig instellen. Dit hoofdstuk maakt veelvuldig gebruik van de cross-browser functie getObj.

Vanzelfsprekend ben je niet verplicht deze functie te gebruiken en kan werken volgens de oudere manier om formuliervelden aan te spreken:

```
document.form[0].elementnaam.eigenschap
```

9.2 Tekstvakken

De waarde van een tekstvak, tekstgebied en een verborgen veld manipuleer je via de eigenschap *value* van het element.

b.v kan je een stukje tekst intikken in het tekstvak. Druk je op 'Toevoegen', dan wordt de tekst toegevoegd aan het tekstgebied.

9.3 Keuzelijsten

Waarden uitlezen

Een keuzelijst of SELECT-element heeft een aantal belangrijke eigenschappen:

- *options*: een *Array* met de opties van de keuzelijst.
  De eigenschap *length* van deze *Array* levert het aantal elementen in de keuzelijst.
  Het aantal elementen vind je ook in de eigenschap *length* van de keuzelijst (je hoeft in principe de options-Array niet te gebruiken)
- *selectedIndex*: indexcijfer van de geselecteerde optie

Een optie uit een keuzelijst is een element uit de *Array* options van de keuzelijst, maar heeft op zich ook een aantal eigenschappen:

- *text*: de tekstwaarde van de optie
- *value*: de waarde van het attribuut value voor deze optie
- *selected*: boolean die aangeeft of het element geselecteerd is

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
<HEAD>
 <TITLE>Cursus Javascript - </TITLE>
 <link rel="stylesheet" type="text/css" href="vb.css">
<script type="text/javascript">
//<!--
var talen;
var resultaat;

function initieer(){
        talen = new getObj("talen");
        resultaat = new getObj("resultaat");
}
function toonEerste(){
        resultaat.obj.value = talen.obj.options[0].text +
                              " (" +talen.obj.options[0].value +")";
}

function toonGeselecteerd(){
        resultaat.obj.value = talen.obj.options[talen.obj.selectedIndex].text +
                              " (" +talen.obj.options[talen.obj.selectedIndex].value
+")";
}
function toonAantal(){
        resultaat.obj.value = talen.obj.options.length;
}

function getObj(name)
{
  if (document.getElementById)
  {
        this.obj = document.getElementById(name);
  }
  else if (document.all)
  {
        this.obj = document.all[name];
  }
```

Vervolg>>

```
  else if (document.layers)
  {
        this.obj = document.layers[name];
  }
}

-->
</script>
</HEAD>
<BODY onload="initieer()">
<h2>Formulieren: keuzelijst</h2>


<form name="f">
<select id="talen" name="talen" size="5">
   <option value="Csharp" selected="selected">C#</option>
        <option value="J">Java</option>
        <option value="JS">Javascript</option>
        <option value="VB">VB.net</option>
</select>
</textarea>

<p>
<input type="button" value="Toon eerste taal" onclick="toonEerste()">
<input type="button" value="Toon geselecteerde taal"
onclick="toonGeselecteerd()">
<input type="button" value="Toon aantal elementen" onclick="toonAantal()">
<p>
<input id="resultaat" name="resultaat" type="text" value="">
</form>
</body>
</html>
```

Opties bijmaken en verwijderen

Keuzelijsten en opties kan je bijmaken met de methode *createElement(),* of door de eigenschap *innerHTML* (van een select-lijst of overkoepelend element) aan te passen.

Naast deze mogelijkheden kan je ook als volgt werken:

Een optie toevoegen aan een keuzelijst waarnaar wordt verwezen met de variabele lijst kan als volgt:

```
lijst.options[lijst.options.length]= new
Option('tekstwaarde','valuewaarde');
```

of:

```
lijst.options[lijst.options.length]= new Option('tekstwaarde');
```

Een element uit een keuzelijst verwijderen kan door deze optie op *null* in te stellen:

```
lijst.options[index] = null;
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
	<TITLE>Cursus Javascript - </TITLE>
	<link rel="stylesheet" type="text/css" href="vb.css">
<style type="text/css">
	select.hoofd
	{
		width:100px;
		background-color:#CCCCFF;
		font-weight:bold;
	}
	select.detail
	{
		width:200px;
		background-color:#CCFFCC;
		color:red;
	}
</style>
<script language= "JavaScript">
<!--
var hoofd;
var detail;

var inhoud = new Array();

inhoud[0] = new Array('Krokus','Roos','Tulp');
inhoud[1] = new Array('Eik','Es','Populier');
inhoud[2] = new Array('Aap','Beer','Hond','Schaap');

function initieer()
{
	hoofd = new getObj("hoofd");
	detail = new getObj("sub");

}
```

Vervolg>>

```
function vulDetail()
{
        var nr = hoofd.obj.selectedIndex;
        detail.obj.options.length = 0;
        for (var i=0; i < inhoud[nr].length; i++)
        {
                detail.obj.options[i] = new Option(inhoud[nr][i]);
        }
        detail.obj.options[0].selected = true;
}
function getObj(name)
{
  if (document.getElementById)
  {
        this.obj = document.getElementById(name);
  }
  else if (document.all)
  {
        this.obj = document.all[name];
  }
  else if (document.layers)
  {
        this.obj = document.layers[name];
  }
}
//-->
</script>
</head>

<body onload="initieer()">
<h1>Dynamische lijst</h1>
<form name="f">
<table border="0">
<tr><td>
   <select class="hoofd" id="hoofd" name="hoofd" size="10"
onchange="vulDetail()">
                <option value="0">Bloemen</option>
                <option value="1">Bomen</option>
                <option value="2">Dieren</option>
   </select><br>
</td><td align=center>
   <select class="detail" id="sub" name="sub" size="10">
   </select><br>
</td></tr></table>
</form>
</body>
</html>
```

Meervoudige selecties

Door een lus te gebruiken en naar de eigenschap selected van elke optie te kijken kan je lijsten manipuleren waar meerdere selecties mogelijk zijn. Hiervoor is in de select-tag de optie multiple geactiveerd.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
	<TITLE>Cursus Javascript - </TITLE>
	<link rel="stylesheet" type="text/css" href="vb.css">
<style type="text/css">
	select.lijst
	{
		width:200px;
	}

	select.leerkracht
	{
		background-color:#FFFFCC;
	}

	select.deelnemer
	{
		background-color:#FFCC33;
	}
</style>
<script language= "JavaScript">
<!--
var lkr;
var dln;

function initieer()
{
	lkr = new getObj("leerkrachten");
	dln = new getObj("deelnemers");
}

function doorgeef(van, naar)
{
   vanlen = van.obj.options.length ;

	//geselecteerden in 'van' toevoegen aan 'naar'
   for ( i=0; i<vanlen ; i++)
   {
```

Vervolg>>

```
  if (van.obj.options[i].selected == true )
        {
          naarlen = naar.obj.options.length;
          naar.obj.options[naarlen]= new
Option(van.obj.options[i].text,van.obj.options[i].value);
        }
    }

         //geselecteerden in 'van' verwijderen
    for ( i = (vanlen -1); i>=0; i--)
    {
       if (van.obj.options[i].selected == true )
       {
          van.obj.options[i] = null;
       }
    }
}

function getObj(name)
{
  if (document.getElementById)
  {
        this.obj = document.getElementById(name);
  }
  else if (document.all)
  {
        this.obj = document.all[name];
  }
  else if (document.layers)
  {
        this.obj = document.layers[name];
  }
}


//-->
</script>
</head>

<body onload="initieer()">
<h1>Studiedag</h1>
<form name="f">
<table border="1">
<tr><td>
```

Vervolg>>

```
<select class="lijst leerkracht" id="leerkrachten" name="leerkrachten" size="10"
multiple="multiple">
          <option value="ap">Aarnouts Peter</option>
          <option value="bp">Baele Peter</option>
          <option value="bi">Bonne Ilse</option>
      <option value="jdd">De Deurwaerder Jan</option>
      <option value="sw">Schokkelé William</option>
      <option value="sk">Somers Karina</option>
   </select><br>
   <p align=center><input type="button" onClick="doorgeef(lkr,dln)" value=" >>
"></p>

</td><td align=center>
   <select class="lijst deelnemer" id="deelnemers" name="deelnemers" size="10"
multiple="multiple">
   </select><br>
   <p align=center><input type="button" onClick="doorgeef(dln,lkr)" value=" << "
></p>

</td></tr></table>
</form>
</body>
</html>
```

9.4 Checkbox

Je kan nagaan of een aankruisvakje is aangevinkt door de eigenschap *checked* te controleren. Je kan ook codematig deze eigenschap instellen.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
        <TITLE>Cursus Javascript - </TITLE>
        <link rel="stylesheet" type="text/css" href="vb.css">
<script language= "JavaScript">
<!--

var boxen = new Array();


function initieer()
{
        //kijken hoeveel checkboxen er zijn die beginnen met 'cb'
        var doorgaan = true;
        var telbox = 1;
        //als het element bestaat telbox ophogen
```

Vervolg>>

```
// -> test met de functie eval
        //telbox is uiteindelijk 1 te groot
        while(doorgaan)
        {
                if(eval("f.cb" +telbox)) telbox++;
                else doorgaan = false;
        }

        //een referentie naar elke checkbox stoppen in de Array boxen
        for(var i = 0; i < telbox - 1 ; i++)
        {
                boxen[i] = new getObj("cb" +(i+1));
        }
}

function toonKeuze()
       {
        var strRes = "U koos voor:\n\n";
        for(var i = 0; i < boxen.length ; i++)
        {
                if(boxen[i].obj.checked)
                {
                        strRes += boxen[i].obj.value +"\n";
                }
        }
        alert(strRes);
}
function getObj(name)
  {
  if (document.getElementById)
  {
        this.obj = document.getElementById(name);
  }
  else if (document.all)
  {
        this.obj = document.all[name];
  }
  else if (document.layers)
  {
        this.obj = document.layers[name];
  }
}
//-->
</script>
</head>
```

Vervolg>>

```
<body onload="initieer()">
<h1>Leuke sporten</h1>
<form id="f" name="f">
<input type="checkbox" id="cb1" name="cb1" value="badminton">Badminton<br>
<input type="checkbox" id="cb2" name="cb2" value="tennis">Tennis<br>
<input type="checkbox" id="cb3" name="cb3" value="voetbal">Voetbal<br>
<input type="button" value="Toon keuze" onclick="toonKeuze()">
</form>
</body>
</html>
```

9.5 Keuzerondjes: radiobuttons

Radiobuttons vormen een *Array* van elementen. We kunnen deze *Array* doorlopen met een lus om te zien welke optie aangeduid is: de eigenschap *checked* van dit aankruisvakje is *true*.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
	<TITLE>Cursus Javascript - </TITLE>
	<link rel="stylesheet" type="text/css" href="vb.css">
<script language= "JavaScript">
<!--
function toonKeuze(){
	//eenvoudiger werken via DOM0 ipv de functie getObj
	var strRes = "Uw geslacht: ";
	for(var i = 0; i < f.geslacht.length ; i++){
		if(f.geslacht[i].checked){
			strRes += f.geslacht[i].value;
		}
	}
	alert(strRes);
}

//-->
</script>
</head>

<body>
<h1>Wat is uw geslacht ?</h1>
<form id="f" name="f">
<input type="radio" name="geslacht" value="V" checked>Vrouwelijk
<input type="radio" name="geslacht" value="M">Mannelijk
<p><input type="button" value="Toon keuze" onclick="toonKeuze()">
</form>
</body>
</html>
```

9.6 Formulieren versturen

Actie ondernemen voor het formulier wordt verstuurd

Dikwijls wens je programmatorisch nog een aantal acties te ondernemen wanneer een gebruiker een formulier verstuurt. Je wil formuliervelden controleren, verborgen velden aanvullen,...

Hier kan je handig gebruik maken van de *onSubmit* event-handler van het element **FORM**.

```
<form action="doeiets.asp" onsubmit="return controle()">
```

Hier roepen we de functie *controle* aan wanneer de gebruiker het formulier wenst te versturen. Wanneer de functie *controle true* retourneert wordt het formulier verstuurd. Krijg je *false* terug uit de functie *controle*, dan wordt het formulier niet verstuurd.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
	<TITLE>Cursus Javascript - </TITLE>
	<link rel="stylesheet" type="text/css" href="vb.css">
<script language= "JavaScript">
<!--

function controle(){
	if(f.naam.value == "" || f.voornaam.value == "") {
		alert('Vul uw naam en voornaam in aub.');
		return false;
	}
	return true;
}

//-->
</script>
</head>

<body>
<h1>Wat is uw geslacht ?</h1>
<form id="f" name="f" action="../../formscript.asp" method="POST"
onsubmit="return controle()">
<table>
<tr>
	<td>Naam:</td>
	<td><input name="naam" type="text" value=""></td>
</tr>
```

Vervolg>>

```
<tr>
     <td>Voornaam:</td>
     <td><input name="voornaam" type="text" value=""></td>
</tr>
<tr>
     <td>Geslacht:</td>
     <td>
          <input type="radio" name="geslacht" value="V" checked>Vrouwelijk
          <input type="radio" name="geslacht" value="M">Mannelijk
          </td>
</tr>
</table>

<p><input type="submit" value="Toon keuze">
</form>
```

Formulieren handmatig versturen en resetten

Wens je een formulier te versturen vanuit code, onafhankelijk van een eventuele submitbutton, dan kan dit op een eenvoudige manier:

```
document.forms[0].submit()
```

Het is echter niet aan te raden, wanneer je een formulier wenst te versturen, een gewone button te gebruiken die het formulier op bovenstaande manier verstuurd in plaats van een submit-button: gebruikers zonder javascript kunnen het formulier nu onmogelijk versturen.

Wanneer je de *submit*-functie gebruikt wordt de onsubmit event-handler niet uitgevoerd.

Een formulier resetten kan als volgt:

```
document.forms[0].reset()
```

9.7 Formulierelementen uitschakelen

Bepaalde formuliervelden mogen soms slechts onder specifieke voorwaarden door de gebruiker ingevuld of aangepast worden. Vandaar dat het met Javascript mogelijk is om formulierelementen uit te schakelen en in te schakelen.
Uitschakelen kan je doen door de eigenschap *disabled* van het formulierelement op *true* te plaatsen. Door deze eigenschap op *false* in te stellen wordt het element weer actief.

Voorbeeld op volgende pagina>>

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
<head>
	<TITLE>Cursus Javascript - </TITLE>
	<link rel="stylesheet" type="text/css" href="vb.css">
<script language= "JavaScript">
<!--
function initieer()
{
	//codematig het textvak disablen
	//werken met het attribuut disabled in de input-tag
	//zorgt ervoor dat in Firefox het element niet meer enabled wordt
	f.aantaldieren.disabled = true;
}

function controle()
{
	if(f.naam.value == "" || f.voornaam.value == "")
	{
		alert('Vul uw naam en voornaam in aub.');
		return false;
	}
	return true;
}

function CHKhuisdier()
{
	if(f.huisdier.checked)
	{
		f.aantaldieren.disabled = false;
		f.aantaldieren.value = "1";
	} else
	{
		f.aantaldieren.disabled = true;
		f.aantaldieren.value = "";
	}
}
//-->
</script>
</head>

<body onload="initieer()">
<h1>Huisdieren</h1>
<form id="f" name="f" action="../../formscript.asp" method="POST"
onsubmit="return controle()">
```

Vervolg>>

```
<table>
<tr>
	<td>Naam:</td>
	<td><input name="naam" type="text" value=""></td>
</tr>
<tr>
	<td>Voornaam:</td>
	<td><input name="voornaam" type="text" value=""></td>
</tr>
<tr>
	<td><input type="checkbox" name="huisdier" value="HDOK"
onclick="CHKhuisdier()">Ik heb huisdieren.</td>
	<td>
		Aantal:
		<input style="width:30px;text-align:right" type="text" name="aantaldieren"
value="" disabled="disabled">
		</td>
</tr>
</table>
<p><input type="submit" value="Toon keuze">
</form>
</body>
</html>
```

Hoofdstuk 10
Javascript: Frames en vensters

1. Basisbegrippen
2. Frames
3. Popups

10.1 Basisbegrippen

Het verdelen van een pagina in frames werd aangeleerd in de cursus HTML.

Om met Javascript frames te benaderen is het handig dat je elk frame een duidelijke naam hebt gegeven. Dit doe je door het attribuut *name* van elke **FRAME**-tag in te stellen.

10.2 Frames benaderen

Als je elementen op een document dat zich in een andere frame bevindt wil aanspreken, dan bouw je eerste en correcte verwijzing op naar het andere frame.

Hierbij kan je volgende sleutelwoorden gebruiken:

- **parent**: frame-element dat in de hiërarchie 1 niveau hoger ligt.
- **top**: meest overkoepelende frame
- **self**: verwijzing naar het frame waarin het huidige document is vervat. Wordt gebruikt om verwijzingen te maken naar onderverdelingen binnen de huidige frame.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
      "http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
 <title>Cursus Javascript - </title>
<script type="text/javascript">
//<!--
var klikteller = 0;
function geklikt()
{
        klikteller++;
        //verwijzen naar een formulierelement , DOM0
        self.onder.document.f.aantal.value = klikteller;

        //verwijzen met de functie getElementById
        var divAantal = self.boven.document.getElementById("aantal");
        divAantal.innerHTML = "Teller: <B>" +klikteller +"</B>";
}
-->
</script>
</head>
```

Vervolg>>

```
<frameset cols="20%,*">
	<frame name="links" src="nav.html">
	<frameset rows="20%,*">
		<frame name="boven" src="titel.html">
		<frame name="onder" src="intro.html">
	</frameset>
</frameset>
</html>

nav.html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
	"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>Cursus Javascript - </title>
 <link rel="stylesheet" type="text/css" href="../vb.css">
<script type="text/javascript">
//<!--
function toonPagina(pagina)
{
	if(pagina != "intro.html")
	{
		document.getElementById('verhoog').disabled = true;
	} else
	{
		document.getElementById('verhoog').disabled = false;
	}
	parent.onder.location.href = pagina;
}
-->
</script>
</head>
<body>
<h2>nav</h2>
<input id="verhoog" type="button" value="Verhoog teller"
onclick="parent.geklikt()">
<p>
<input type="button" value="Toon aantal" onclick="alert(parent.klikteller)">
<p><input type="button" value="Toon test" onclick="toonPagina('test.html')">
<p><input type="button" value="Toon intro" onclick="toonPagina('intro.html')">
</body>
</html>
```

Titel

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>Cursus Javascript - </title>
 <link rel="stylesheet" type="text/css" href="../vb.css">
</head>
<body>
<h1>titel</h1>
<div id="aantal"></div>
</body>
</html>
```

Intro

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>Cursus Javascript - </title>
 <link rel="stylesheet" type="text/css" href="../vb.css">
</head>
<body>
<h2>intro</h2>
<form name="f">
      Teller: <input name="aantal" type="text" value="0" style="text-
align:right;width:30px" disabled>
</form>
</body>
</html>
```

Test

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
      "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
 <title>Cursus Javascript - </title>
 <link rel="stylesheet" type="text/css" href="../vb.css">
</head>
<body>
<h2>test</h2>
</body>
</html>
```

Popups

Pop-up vensters heb je wellicht al vele malen tegengegekomen bij je surftochten op het internet. Meestal onder de vorm van irritante reclamevensters bij het ophalen (load) of verlaten (unload) van een pagina. Je kan popup vensters echter ook op een nuttige manier aanwenden op een website.

Een popup maken

Volgende functie stelt je in staat een pop-up venster te openen, als argument geef je de URL mee van de te openen pagina:

```
<SCRIPT LANGUAGE="JavaScript" TYPE="text/javascript">
<!--
function popitup(url)
{
        newwindow=window.open(url,'name','height=200,width=150');
        if (window.focus) {newwindow.focus()}
        return false;
}

// -->
</SCRIPT>
```

De eerste opdrachtlijn leert je hoe je een popup kan openen:

```
newwindow=window.open(url,'name','height=200,width=150');
```

Een popup open je met de methode *open* van het object *window*. Deze methode ontvangt drie argumenten:

1. **URL** van de te openen pagina
2. **naam** voor de popup
   Deze eigenschap voegt in feite niets extra toe aan de popup, maar zorgt er in bepaalde browsers voor dat dezelfde popup geen twee keer naast elkaar kan worden geopend.
3. Met het derde argument kan je allerhande waarden instellen:
   - *width*: breedte
   - *height*: hoogte
   - *top*: afstand t.o.v. bovenkant van het scherm
   - *left*: afstand t.o.v. linkerkant van het scherm
   - *sceenX / screenY*: positionering NS4 en Safari
   - *dependent*: als de pagina die de popup opende wordt gesloten, wordt ook de popup gesloten (Mozilla, NS). (*yes / no*)
   - *directories*: weergeven van de 'directories' (IE: werkbalk koppelingen) (*yes / no*).
   - *fullscreen*: popup opent in volledige schermmodus (*yes / no*).
   - *location*: adresbalk weergeven (*yes / no*).

Vervolg>>

- *menubar*: menubalk weergeven (*yes / no*).
- *resizable*: afmetingen zijn aanpasbaar (*yes / no*).
- *scrollbars*: de popup kan scrollbalken hebben
- *status*: weergeven van de statusbalk (*yes / no*).
- *toolbar*: weergeven van de knoppenbalk (*yes / no*).

In de eerst opdracht merk je dat we het popupvenster stockeren in een variabele (**newwindow**), zodanig dat we er verderop in de code kunnen naar verwijzen.

De tweede opdracht geeft het popupvenster de focus, zodat het venster zeker wordt getoond en niet verscholen zit achter een ander venster. Doordat niet alle browsers de methode *focus* ondersteunen testen we eerst of deze methode gekend is door de browser.

```
if (window.focus) {newwindow.focus()}
```

De derde opdracht retourneert *false*, zodanig dat wanneer we de code oproepen vanuit een hyperlink, er geen koppeling wordt gevolgd, maar enkel de pop-up wordt geopend:

```
return false;
```

Een popup openen

Bovenstaande code geeft je een functie op een popup te maken, deze functie kunnen we nu eenvoudig koppelen aan bv. een hyperlink.

```
<a href="vb/popupvb.html" onClick="return popitup('vb/popupvb.html')"
        >Open popup</a>
```

Een popup sluiten

Wanneer je popup hebt toegekend aan een variabele, dan kan je deze eenvoudig sluiten met de methode *close*. Vooraleer we de methode *close* uitvoeren testen we eerst of de popup reeds is geopend door na te gaan of de variabele newwindow bestaat.

```
<A HREF="javascript:if (newwindow) newwindow.close()"
        >Sluit popup.</A>
```

Schrijven in een popup

Wanneer je bij het maken van een popup het eerste attribuut leeglaat kan je zelf dynamisch de inhoud van de popup bepalen. Het popupdocument-vullen doe je eenvoudig met de methode *write* van het object *document* van de popup.

De opdracht **newwindow2.document.close();** duidt voor de browser aan dat er niet meer zal geschreven worden in de popup. Deze opdracht sluit de popup niet af, dit zou je kunnen met **newwindow2.close();**

```
function popitup2()
{
	newwindow2=window.open('','name','height=200,width=250');
	var nu = new Date();
	var tijd = nu.getHours() +":" +nu.getMinutes() +":" +nu.getSeconds()
	var strHTML = "<html><head><title>Dynamische popup</title></head>";
	strHTML += "<body>";
	strHTML += "Popup gemaakt om: " +tijd;
	strHTML += "</body>";
	strHTML += "</html>";
	newwindow2.document.write(strHTML);
	newwindow2.document.close();
	if (window.focus) {newwindow2.focus()}
	return false;
}
```

Opener manipuleren

Wil je vanuit de popup de bronpagina beïnvloeden, dan kan door het object opener aan te spreken vanuit de popup.

popupvb2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
<head>
	<title>Javascript: popups</title>
<script type="text/javascript">
<!--
function verander_opener(url)
{
	opener.location.href = url;
}

-->
</script>
</head>

<body>
Ik ben een popup
<p>
<a href="javascript:verander_opener('../dom.asp')">DOM</a>
<br><a href="javascript:verander_opener('../events.asp')">Events</a>
<br><a href="javascript:verander_opener('../vensters.asp#popup')">Vensters -
popup</a>
</body>
</html>
```

popup openen met de functie popitup:

```
<a href="vb/popupvb2.html" onClick="return popitup('vb/popupvb2.html')">Open popup</a>
```

Opgepast, wanneer de popup frames bevat is de opener te manipuleren vanuit de top-locatie:

```
top.opener.location.href = url;
```

Opener sluiten

Je kan de bronpagina sluiten met volgende opdracht:

```
opener.close();
```

In sommige browsers wordt nu een waarschuwing getoond, en moet de gebruiker het sluiten van deze pagina bevestigen. Dit komt doordat de bronpagina door de gebruiker werd geopend, en niet door een ander venster: de bronpagina heeft geen opener.

Dit kan je in bepaalde browsers eenvoudig omzeilen door voor het sluiten van de bronpagina een waarde in te stellen voor de opener van de bronpagina:

```
opener.opener = top; //willekeurige waarde instellen
opener.close()
```

Hoofdstuk 11
Javascript: reguliere expressies

1. Basisbegrippen
2. Werken met reguliere expressies
3. Argumenten
4. Tekens en symbolen

11.1 Basisbegrippen

Een reguliere expressie vormt een patroon dat je kan gebruiken om tekenreeksten te doorzoeken.
Met reguliere expressies kan je antwoorden krijgen op vragen zoals:

- op de tweede positie van de tekenreeks moet een letter 'A' staan
- in de uitdrukking moet een '@'-symbool en een '.' voorkomen
- De tekenreeks mag enkel bestaan uit cijfers tussen 2 en 6
- ...

Dit kan je handig toepassen bij bijvoorbeeld de controle van formuliergegevens.

Om een reguliere expressie te definiëren declareer je een variabele en plaats je de uitdrukking tussen twee slashes.

Je kan ook de constructor van een *RegExp* gebruiken, nu geef je de expressie als String mee. Het voordeel aan deze werkwijze is dat het patroon variabel kan worden ingesteld, en zoekwoorden dus bijvoorbeeld aan de gebruiker kunnen worden gevraagd. Hierbij zal je voor complexere uitdrukkingen vaak rekening moeten houden met het escapeteken (\).

```
var patroon = /RexExp/;
of:
var patroon = new RegExp("RexExpString");
```

11.2 Werken met reguliere expressies

De methode test van een Reguliere Expressie

De methode *test* van een Reguliere Expressie retourneert *true* als de expressie voorkomt in de meegegeven tekenreeks.

```
var patroon = /javascript/;
var tekst = "Reguliere expressies met javascript";
if(patroon.test(tekst)) document.write("Het patroon werd gevonden");
```

Het patroon werd gevonden

String-methoden met reguliere expressies

De krachtigste methoden voor het werken met reguliere expressies zijn echter methoden die op de tekenreeks worden toegepast.

Methode	Omschrijving
search(regex)	Retourneert de positie waar de expressie het eerst voorkomt in de tekenreeks. Komt de expressie niet voor, dan retourneert search -1
replace(regex,string)	Vervangt het tekstdeel dat beantwoord aan regex door een tekenreeks
split(regex)	Splitst een tekenreeks op basis van regex in een Array

```
var patroon = /javascript/;
var tekst = "Werken met reguliere expressies in javascript";
document.write("Zin: <b>" +tekst +"</b><br>");
document.write("Expressie gevonden op positie: <b>" +tekst.search(patroon)
+"</b><br>");
document.write("Nieuwe zin: <b>" +tekst.replace(patroon,"Java-Script") +"</b>");

var talen = "javascript,C#,Java,HTML,CSS,Cobol,Fortran";
var arrTalen = talen.split(/,/);
document.write("<h5><u>Talen</u></h5>");
for(i=0;i<arrTalen.length;i++){
        document.write(arrTalen[i] +"<br>");
}
```

Zin: **Werken met reguliere expressies in javascript**
Expressie gevonden op positie: **35**
Nieuwe zin: **Werken met reguliere expressies in Java-Script**

Talen
javascript
C#
Java
HTML
CSS
Cobol
Fortran

11.3 Argumenten

Je kan de volgende argumenten gebruiken om de zoekopdracht uit te voeren:

Argument	Omschrijving
i	case-insensitive: de zoekopdracht is niet hoofdlettergevoelig
g	global: de volledige tekenreeks wordt doorzocht - bij een replace wordt niet

```
var zin = "Werken met Javascript - JAvAscript is leuk";

document.write("Originele zin: <b>" +zin +"</b><br>");

var patroon = /java/;
document.write("Zonder argumenten: <b>" +zin.replace(patroon,"JAVA")
+"</b><br>");

var patroon = /java/i;
document.write("Argument i: <b>" +zin.replace(patroon,"JAVA") +"</b><br>");

var patroon = /java/ig;
document.write("Argument i en g: <b>" +zin.replace(patroon,"JAVA")
+"</b><br>");

var patroon = new RegExp("java","ig");
document.write("RegExp-constructor i en g: <b>" +zin.replace(patroon,"JAVA")
+"</b><br>");
```

Originele zin: **Werken met Javascript - JAvAscript is leuk**
Zonder argumenten: **Werken met Javascript - JAvAscript is leuk**
Argument i: **Werken met JAVAscript - JAvAscript is leuk**
Argument i en g: **Werken met JAVAscript - JAVAscript is leuk**
RegExp-constructor i en g: **Werken met JAVAscript - JAVAscript is leuk**

11.4 Tekens en symbolen

Om een patroon op te stellen zijn er heel wat geavanceerde mogelijkheden:

Teken	Beschrijving	Voorbeeld
Normaal teken	Het opgegeven teken zelf wordt gezocht	**java**: er wordt gezocht naar het woord java met de tekens exact in deze volgorde
[...]	Een van de tekens moet worden gevonden	**[jJ]ava**: java met een kleine of grote J zal worden gevonden. (de rest moet in kleine letters geschreven zijn)
[^...]	Een teken dat niet tussen de haakjes voorkomt	**[^aA]001**: 001 voorafgegaan door een teken dat niet a of A is.
[A-Z]	Een teken van A tot Z	**[B-F]111**: 111 voorafgegaan door een letter B-F.
(...)	Een bereik waarmee je tekens groepeert	
?	Het voorafgaande teken of bereik is optioneel	**varkens?:** het woord varken of varkens - de s is optioneel **java(script)?**: java of javascript - script is optioneel
+	Het voorafgaande teken of bereik moet een of meerdere keren voorkomen	
*	Het voorafgaande teken of bereik kan een willekeurig aantal keren of helemaal niet voorkomen	

{n,m}	Het voorafgaande teken of bereik moet n keer (en hoogstens m keer gevonden worden – optioneel)	
|	Onderscheid tussen verschillende mogelijkheden	**html|css|javascript:** html of css of javascript
^	Caret: het begin van de tekenreeks	**^Een**: het woord Een aan het begin van de tekenreeks
$	Dollar: het einde van de tekenreeks	**a$**: de letter a op het einde van de tekenreeks **^[a-z]{4}$**: exact 4 kleine letters
\	Backslash: escape-teken om speciale karakters te vormen	****: een backslash **\/**: een slash **\w**: een word karakter: [a-zA-Z0-9_] **\W**: geen word karakter: [^a-zA-Z0-9_] **\d**: digit - een cijfer [0-9] **\D**: geen cijfer [^0-9] **\n**: regelterugloop **\t**: tabsprong **\b**: woordgrens **- \bjava\b**: zoeken naar het woord java, javascript wordt niet gevonden. **\B**: geen woordgrens