1

2 # Inhoudsopgave

3 JavaScript

4 Wat is Javascript?

JavaScript is een programmeertaal die gebruikt wordt om HTML-documenten interactief te maken. Een pagina die enkel uit HTML bestaat is statisch. Dit wil zeggen dat de browser de pagina laadt en enkel toont, er zijn geen extra effecten of interacties. De gebruiker kan de pagina lezen en eventueel op verwijzingen klikken. In combinatie met JavaScript wordt een pagina dynamisch: de pagina reageert op input van de gebruiker. Zo kan je bijvoorbeeld controleren of formulieren correct zijn ingevuld, kleine animaties starten als de gebruiker met de muis ergens over een plaats in de pagina beweegt of nieuwe browservensters op commando openen en sluiten.

JavaScript is een zogenaamde **scriptingtaal**. Programma's hoeven niet gecompileerd te worden (zoals bijvoorbeeld een C-programma), maar ze kunnen onmiddellijk worden uitgevoerd. Binnen de browser is hiervoor een component, de vertolker (engels: interpreter), aanwezig die de programma's uitvoert. Het grote voordeel van vertolkte talen zoals JavaScript is dat de programmeur snel kan testen of een programma werkt, gewoon openen in een browser volstaat. Een nadeel is dat fouten in het programma soms moeilijk te vinden zijn, omdat er geen compiler is die foutmeldingen in de broncode aanduidt.

JavaScript werd in 1995 ontwikkeld door de programmeurs van Netscape, maar werd daarna goedgekeurd door de ECMA (European Computer Manufacturers Association) wat een naamswijziging tot ECMAScript tot gevolg had. Momenteel is de taal ISO-gestandaardiseerd door standaard ISO 16262. De huidige versie is JavaScript ECMAScript2015 of ES6.

1. 1995: JavaScript wordt voorgesteld (eerst onder de naam LiveScript)
2. 1997: ECMAScript standaard wordt goedgekeurd
3. 1999: ES3
4. 2000–2005: XMLHttpRequest, a.k.a. AJAX wordt populair op het web: Outlook Web Access (2000), Gmail (2004) and Google Maps (2005).
5. 2009: ES5 komt uit (meest gebruikt op dit ogenblik) met `forEach`, `Object.keys`, `Object.create` en JSON
6. 2015: ES6/ECMAScript2015 komt uit (geen grote veranderingen, problemen om consensus te vinden over een aantal onderwerpen)

De JavaScript taal is niet beperkt tot browsers of internettoepassingen alleen. Je kan bijvoorbeeld de taal uitstekend gebruiken om binnen een Windows omgeving administratieve scripts te schrijven. De Windows Scripting Host is de vertolker die Microsoft in zijn besturingssysteem heeft geïntegreerd om dit te realiseren. Applicaties kunnen ook de mogelijkheid bieden om scripts uit te voeren om op die manier de applicatie zelf uit te breiden en flexibel te maken. Een voorbeeld is Flash dat ook JavaScript beheerst. Een softwareproducent die dergelijke mogelijkheden voor zijn applicatie wil aanbieden dient niet zelf JavaScript te implementeren, maar kan gebruik maken van de engines die door de Mozilla-organisatie in open source worden aangeboden.

Wanneer er gebruik gemaakt wordt van JavaScript is het niet de bedoeling om een server continu te connecteren om extra gegevens op te halen. Alle code moet dus voorradig zijn in de webpagina zelf. Het gebruik van JavaScript in een webbrowser wordt aangeduid als **client-side** JavaScript. Typisch surft een gebruiker immers met een PC of laptop (een client machine) op het internet. Eigenlijk is deze benaming niet zo goed gekozen, want je kan natuurlijk ook nog andere netwerkprogramma's op een client draaien. Men had beter gesproken over **browser-side**

JavaScript, maar deze term bestaat niet. Denk er dus aan dat als je leest over client-side JavaScript, je eigenlijk de taal beschouwt zoals die in een webbrowser ingebed is.

De Core van de taal definieert enkele standaardobjecten (bv. het String object) die je als programmeur altijd en overal kan gebruiken. Aan de client-side bestaan er nog enkele andere objecten, die het mogelijk maken om vanuit JavaScript alle belangrijke onderdelen van een webapplicatie aan te spreken.

Tenslotte willen we nog even vermelden dat JavaScript helemaal niet gerelateerd is aan de taal Java. Beide zijn totaal verschillend qua historiek en bedoeling. Het enige wat je kan zeggen is dat de syntax van beide talen erg gelijkend is, omdat ze allebei afstammen van C-taal.

5 De <script> tags en je eerste javascript

De <script>-tag wordt gebruikt om de client-side scripts toe te voegen aan je html document.

```
<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <title>Demo 1: first javascript</title>
</head>
<body>
<output id="demo"></output>
<script>
    document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
</body>
</html>
```

Het <script> element bevat de programma-logica, of kan verwijzen naar een extern bestand met het script.

```
<!DOCTYPE html>
<html lang="nl”>
<head>
    <meta charset="UTF-8">
    <title>Demo 1: first javascript</title>
    <script src="info.js" async></script>
</head>
<body>
<div>
    <output id="demo"></output>
</div>
<div>
    <output id="demo2"></output>
</div>
<script>
    document.getElementById("demo").innerHTML = "Hello JavaScript!";
</script>
</body>
</html>
```

En het bestand info.js:

```
window.addEventListener("load", displayInfo);
```

```
function displayInfo() {
    document.getElementById("demo2").innerHTML = "I'm enjoying JavaScript";
}
```

Indien we de JavaScript code in een extern document schrijven plaatsen we de <script> tag binnen de <head> tag en gebruiken we async of defer. Dit omwille van de performantie van onze webpagina.

- Bij gebruik van **async**: Het script wordt asynchroon gedownload en uitgevoerd, dus tegelijkertijd met het verwerken van de rest van de html-pagina (het script wordt dus uitgevoerd terwijl de pagina verder wordt geparsed).
- Bij gebruik van **defer**: Het script wordt gedownload en uitgevoerd zodra de html-pagina verwerkt is (dus de browser klaar is met het parsen (verwerken) van de html-code.
- Zonder defer of async: Het script wordt direct opgehaald en uitgevoerd, daarna gaat de browser pas verder met het verwerken van de rest van de webpagina. Dit heeft dus grotere impact op de laadtijd van de webpagina. Daarom werd vroeger de script-tag pas aan het einde van de pagina geplaatst.

6 Lexicale structuur

Bij het schrijven van JavaScript moeten een aantal regels in het oog gehouden worden: Hoofdletters moeten gerespecteerd worden, statements correct gescheiden en commentaar op de juiste plaats gezet.

❖ Hoofdletters

Aangezien JavaScript hoofdlettergevoelig (case sensitive) is, dien je op te letten wanneer je variabelen of functies definieert met hoofdletters. Indien je deze variabelen of functies nadien wil gebruiken moet de schrijfwijze exact overeenkomen. Voor namen van variabelen en functies wordt vaak de camelCase notatie gehanteerd. In deze notatie wordt alles aan elkaar geschreven, maar beginnen alle woorden met een hoofdletter, uitgezonderd het eerste woord.

Ook bij gebruik van methodes, bijvoorbeeld `getElementById()`, moet de naam correct ingegeven worden.

```
var mijnVariabele = 0; //declareer een variabele

Mijnvariabele = 10; //Dit is fout
MijnVariabele = 10; //Dit is fout

mijnVariabele = 10; //Dit is correct
```

❖ Scheiden van statements

Statements (één enkele instructie) worden gescheiden door puntkomma's (;). Hoewel je ze soms kan weglaten, is het aangeraden om ze consistent en overal te schrijven om fouten te vermijden. Het is gemakkelijker ze steeds te plaatsen dan aan te leren wanneer ze niet nodig zijn.

Je kan meerdere statements op één lijn ingeven, maar tracht slechts één statement per lijn te gebruiken, dit verhoogt de leesbaarheid van je programma.

```
a = 10; b = 20;
```

of beter

```
a = 10;
```

```
b = 20;
```

❖ Commentaar

De volgende regel geeft aan hoe je commentaar binnen JavaScript kan gebruiken, dit is dezelfde methode als binnen CSS gebruikt wordt.

```
//deze functie toont een boodschap
```

Als je C++ of Java kent, dan ben je hier al mee vertrouwd. Twee schuine strepen (//) dienen voor commentaar op één lijn, terwijl meerdere lijnen commentaar worden afgebakend door de tekens /* en */.

```
/*
Dit zijn meerdere lijnen commentaar
Commentaar kan de code verduidelijken
*/
```

❖ Voorbehouden woorden

Er zijn enkele woorden die je beter niet gebruikt om variabelen en functies te benoemen. Dit omdat ze al in gebruik zijn in de JavaScript-taal zelf en omdat dit anders verwarring zou scheppen. Ter volledigheid volgt hier het lijstje.

Woorden gemarkeerd met een * zijn toegevoegd in ES (ECMAScript) 5 en 6.

```
abstract    arguments         await*boolean
break       byte              case        catch
char        class*      const       continue
debugger    default           deletedo
doubleelse              enum*       eval
export*     extends*          false       final
finally     float             for         function
goto        if                implements  import*
in          instanceof        int         interface
let*        long              nativenew
null        package           private     protected
publicreturn      short       static
super*synchronized      switchthis
throw       throws      transient   true
try         typeof      var         void
volatile    while             with        yield
```

❖ Strict mode

De strict mode is toegevoegd sinds ES5 en geeft aan dat de code “streng” en “nauwkeurig” wordt uitgevoerd. We zorgen ervoor dat deze directive ALTIJD aan ons script wordt toegevoegd!

Met strenge uitvoering bedoelen o.a. dat:

- variabelen steeds gedeclareerd moeten worden
- hergebruik van parameternaam niet is toegestaan

- octaalgetallen en escape karkaters niet zijn toegestaan
- …

```
<!DOCTYPE html>
<html lang="nl”>
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<output id="output"></output>
<script>
    "use strict";

    document.getElementById("output").innerHTML = double(5, 6);

    function double(value1, value1) {
        var oct = 010; // 1 * 8  = 8
        twice = 2;
        return value1 * twice * oct;
    }
</script>
</body>
</html>
```

Bovenstaand code-voorbeeld geeft bij niet-strikte uitvoering 96 als resultaat, bij strikte uitvoering krijg je dus verschillende foutmeldingen:

- Uncaught SyntaxError: Duplicate parameter name not allowed in this context
- Uncaught SyntaxError: Octal literals are not allowed in strict mode.
- Uncaught ReferenceError: twice is not defined

7 Variabelen

❖ Getallen

JavaScript gebruikt 64 bits om een nummer op te slaan. Je kan dus niet oneindig grote nummers maken.

```
var decGetal = 10;           // decimale waarde is 10
var hexGetal = 0x10;         // decimale waarde is 16
var octGetal = 0o10;         // decimale waarde is 8
var hexGetal = -0x10;        // decimale waarde is -16
var valversnelling = 9.81;   // komma-getallen
var grootGetal = 2.998e8;    // 2.998 × 10^8 = 299 800 000
```

❖ Wiskundig operatoren

Rekenkundige operatoren worden gebruikt voor wiskundige bewerkingen op numerieke waarden.

operator	bewerking	voorbeeld	commentaar
+	optellen	i=j+3;	
-	aftrekken	i=j-3;	

*	vermenigvuldigen	i=j*3;	
/	delen	i=j/3;	
%	modulus	i=j%3;	rest van de deling
++	plus 1	i++;	idem aan i=i+1;
--	min 1	i--;	idem aan i=i-1;

❖ Toekenningoperatoren

Bij het toekennen van variabelen kunnen nog meerdere operatoren gebruikt worden. Buiten het gelijk-aan teken, kunnen al deze operatoren ook voluit geschreven worden met de rekenkundige operatoren, zoals in de laatste kolom getoond wordt.

operator	bewerking	voorbeeld	commentaar
=	gelijk aan	i=j	
+=	optellen met	i+=j	idem aan i=i+j
-=	aftrekken met	i-=j	idem aan i=i-j
*=	vermenigvuldigen met	i*=j	idem aan i=i*j
/=	delen met	i/=j	idem aan i=i/j
%=	modulus met	i%=j	idem aan i=i%j

❖ Speciale getallen

Er zijn drie speciale waarden in JavaScript die we als getallen beschouwen, maar die zich niet gedragen als normale getallen.

De eerste twee zijn `Infinity` en `-Infinity`, die stellen dus positief oneindig en negatief oneindig voor. `Infinity - 1` geeft `Infinity`. De implementatie hiervan is niet volledig wiskundig onderbouwd en je kan er dus best niet te veel op vertrouwen als je oneindig-gebaseerde bewerkingen wil doen. Bewerkingen met `Infinity` en `-Infinity` leiden als snel tot het volgende speciale getal: `NaN`.

`NaN` staat voor “not a number”, ondanks het feit dat het een waarde voor een getal is. Je krijgt die als resultaat voor bijv. `0 / 0` (nul delen door nul) of `Infinity - Infinity`, ...

```
<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <title>Speciale getallen</title>
</head>
<body>
<output id="output"></output>
<script>
    var waarde1 = 120;
    var waarde2 = 0;
    var result1 = waarde1 / waarde2;
    var result2 = waarde1 / "abc";
    document.getElementById("output").innerText =
                `Deling door nul geeft:  ${result1}.
                 Deling door string geeft: ${result2}`;
</script>
</body>
</html>
```

We bekijken een concreet programma. Dit voorbeeld leest een getal in en rondt het af naar de dichtstbijzijnde gehele waarde.

```
function roundNumber() {
    var getal = prompt("Geef een getal in: ");
    if (isNaN(getal)) {
        return "U heeft geen getal ingegeven: " + getal;
    } else {
        var afrond = parseInt(getal + 0.5);
        return "Het getal " + getal + " is afgerond naar " + afrond;
    }
}
```

In bovenstaand programma wordt getest of een ingegeven waarde een nummer is of niet. Dit gebeurt met de functie `isNaN()` die `true` geeft indien de waarde geen getal is of `false` indien het wel een getal is.

```
if (isNaN(getal))
```

❖ String

Dit is een reeks van ASCII-karakters. Ze worden tussen enkele of dubbele aanhalingstekens geplaatst.

```
var welkomtekst = "Hello, World!";
var welkomtekst = 'Hello, Planet!';
```

```
<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <title>Haiku</title>
</head>
<body>
<output id="output"></output>
<script>
    var webSite = "Web site";
    var haiku = `The ${webSite} you seek
                 cannot be located but
                 endless others exist`;
    document.getElementById("output").innerText = haiku;
</script>
</body>
</html>
```

♦ *Operatoren voor tekstvariabelen*

Bij strings krijgt het plusteken een heel andere betekenis, namelijk de concatenatie of het samenvoegen van de tekst.

operator	bewerking	voorbeeld	commentaar
=	toekenning	woord=woord2	
+	concatenatie	woord=woord+woord2	idem aan woord += woord2

+=	concatenatie	woord+=woord2	idem aan woord=woord+woord2

Een klein voorbeeld maakt dit duidelijk.

```
var woord = "Hallo"
var woord2 = "Wereld"
woord += woord2                // woord bevat nu de string "HalloWereld"
```

- ***Property length***

Deze (alleen lezen) property geeft aan uit hoeveel karakters de string bestaat.

```
var tekst = "Hogeschool PXL";
var lengte = tekst.length;                              // lengte = 14
```

- ***String.charAt(pos)***

Deze methode retourneert het karakter op positie `pos`. Let op: net zoals bij arrays begin je te tellen vanaf nul!

```
var tekst = "Hogeschool PXL";
var karakter = tekst.charAt(0);                         // karakter = 'H'
```

- ***String.indexOf(char, [start])***

Deze methode retourneert de eerste index waar het karakter of substring `char` binnen de string voorkomt. Je kan deze methode gebruiken om te zoeken binnen een string. Als het karakter of de substring niet gevonden wordt, retourneert de functie -1.

Optioneel kan een beginindex `start` meegegeven worden. Dit duidt aan vanaf welke positie er moet begonnen worden met zoeken.

```
var tekst = "Piet@myComp.be";
var i = tekst.indexOf('@');                          // i = 4
var j = tekst.indexOf('.nl');                        // j = -1
```

- ***String.substring(from, [to])***

Deze methode isoleert een substring uit de gegeven string. De eerste letter van de substring bevindt zich op index `from` en het argument `to` de positie van het eerste karakter dat *niet* meer tot de substring behoort. Als het wordt weggelaten, gaat het gewoon tot het einde van de string.

```
var tekst = "Hogeschool PXL";
var s1 = tekst.substring(0, 9);           // s1 = 'Hogeschool'
var s2 = tekst.substring(10);             // s2 = 'PXL'
```

- ***String.toUpperCase()en String.toLowerCase()***

Converteert een string naar hoofdletters of naar kleine letters.

❖ boolean

Indien je slechts 2 mogelijke waarden nodig hebt zoals: 'ja' en 'nee', of 'aan' en 'uit', dan kan je het boolean type gebruiken. De waarden zijn dan **true** en **false**. Deze waarden zijn ook het resultaat bij het gebruik van een vergelijkende operator.

var aan = true;

var geslaagd = punten >= 10;

var test = “aaa” < “abc”;

❖ Vergelijkende operatoren

Vergelijkende operatoren worden binnen controlestructuren (zie verder) toegepast en geven de waarde “waar” (true) of “niet waar” (false).

operator	bewerking	voorbeeld	commentaar
==	is gelijk aan	i==5	
===	waarde en type gelijk	i===5	dus enkel gelijk aan de integer 5, niet aan de string “5”
!=	is niet gelijk aan	i!=5	
!==	Waarde en type is niet gelijk	i !== 5	
>	is groter dan	i>5	
<	is kleiner dan	i<5	
>=	is groter of gelijk aan	i>=5	
<=	is kleiner of gelijk aan	i<=5	

JavaScript heeft 2 sets van operatoren om gelijkheid (of ongelijkheid) te testen: === en !==, en de slechtere varianten == en !=. De eerste (goede) werken zoals je zou verwachten: indien twee operanden hetzelfde type en waarde hebben, dan geeft === true en !== false als resultaat. De “evil twins” werken hetzelfde indien beide operanden van hetzelfde type zijn, maar indien ze een verschillend type hebben, zullen ze de waarden naar andere types dwingen volgens regels die erg ingewikkeld (en dus ook niet onthoudbaar) zijn. Hier enkele interessante voorbeeldjes:

```
'' == '0'           // false
0 == ''             // true
0 == '0'            // true

false == 'false'    // false
false == '0'        // true

false == undefined  // false
false == null       // false
null == undefined   // true

' \t\r\n ' == 0     // true
```

Het gebrek aan transitiviteit is een ernstig probleem. **Daarom: gebruik ALTIJD === en !==.**

❖ Logische operatoren

Dit zijn de Booleaanse operatoren and, or en not om een logische relatie tussen operandi te bepalen. Deze operatoren worden vaak in controlestructuren gebruikt. Bij de and-operator moeten beide vergelijkingen waar (`true`) zijn. Bij de or-operator moet één van beide operatoren waar (`true`) zijn, de andere mag zowel waar (`true`) of niet-waar (`false`) zijn. De not-operator mag vertaald worden als dit niet-waar (`false`) is.

operator	bewerking	voorbeeld	commentaar
&&	and	i ==0 && j < 5	controleert de en-relatie
||	or	i==0 || i>=100	controleert de of-relatie
!	not	!(i==j)	is eigenlijk een negatie en kan ook als i!=j genoteerd worden

❖ null

Dit gereserveerde woord betekent zoveel als een lege variabele die dus geen inhoud of waarde heeft. Dit mag niet verward worden met de waarde van het getal nul. Door bij de declaratie de waarde null mee te geven maak je duidelijk dat dit om een object gaat, dat nog geen waarde heeft toegewezen gekregen. Bij stringvariabelen is er dus ook een duidelijk verschil met het toewijzen van een lege string met “” als waarde. Hiermee definieer je de string duidelijk als leeg, terwijl je met null aangeeft dat de stringwaarde nog aangemaakt moet worden.

❖ Undefined

Een variabele waaraan geen waarde is toegekend heeft type undefined. Je kan op onderstaande manieren testen of een variabele x undefined is:

```
var x;
if (typeof x === 'undefined') {
   // these statements execute
}
OF
if (x === undefined) {
   // these statements execute
}
```

Nog een voorbeeldje met berekeningen met null en undefined. Een berekening met undefined geeft NaN, bij berekeningen met null wordt null naar 0 geconverteerd.

```
<!DOCTYPE html>
<html lang="nl”>
<head>
    <meta charset="UTF-8">
    <title>Null en undefined</title>
</head>
<body>
<output id="outputCirkel1"></output><br/>
<output id="outputCirkel2"></output>
<script>
    var straalCirkel1;
    var omtrekCirkel1 = straalCirkel1 * 2 * Math.PI;
    document.getElementById("outputCirkel1").innerHTML =
        "De omtrek van een cirkel met straal "
        + straalCirkel1 + " is " + omtrekCirkel1;
    var straalCirkel2 = null;
    var omtrekCirkel2 = straalCirkel2 * 2 * Math.PI;
    document.getElementById("outputCirkel2").innerHTML =
        "De omtrek van een cirkel met straal "
        + straalCirkel2 + " is " + omtrekCirkel2;
</script>
</body>
</html>
```

❖ Zwakke typering

Het type van de variabele is niet opgegeven, maar wordt door de uitvoeringsomgeving afgeleid uit de context. JavaScript is dus een **zwak getypeerde** scripttaal in tegenstelling tot bijvoorbeeld C en Java wat sterk getypeerde programmeertalen zijn. In deze talen dient een typedeclaratie te

gebeuren en wordt het type meegegeven, bijvoorbeeld `int` voor integers. In onderstaand C-voorbeeld declareer je een variabele i van het type `integer`.

```
/* C voorbeeld: gebruik van variabelen */
int i = 10;    // type: integer
i = i * 5;     // gebruik integers enkel volgens de regels
```

In JavaScript is dit niet nodig en wordt altijd `var` gebruikt (of let of const). Je moet variabelen dus nog altijd declareren (vandaar het woord `var`), maar hun type wordt automatisch door de vertolker bepaald uit de context, dit noemt men **automatische typeconversie**.

```
/* JavaScript voorbeeld: automatische types */
var i = 10;                    // type is duidelijk uit context
i = "Dit is een getal: " + i;  // conversie
```

In het voorbeeld ken je eerst het getal 10 toe aan de variabele i. Op dat moment is i van het type integer. Bij het optellen van i bij een string, zal i omgezet worden naar een string en zal het optelteken geïnterpreteerd worden als een concatenatie. Het eindresultaat is dus dat i een stringvariabele is.

❖ Bereik van een variabele

Het bereik (scope) bepaalt waar en wanneer een bepaalde variabele beschikbaar is. Het eenvoudigst zijn **globale variabelen**: zij worden gedeclareerd buiten de functies maar binnen de `script`-tag en zijn bijgevolg voor alle functies toegankelijk. Binnen de functies kunnen ook **lokale variabelen** gedeclareerd worden die dan ook enkel beschikbaar zijn binnen deze functie. Het declareren van een variabele is overal mogelijk binnen de functie.

```
<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <title>Globale en lokale variabelen</title>
</head>
<body>
<output id="output"></output>
<script>
    var getal1 = 5; // globale variabelen
    var getal2 = 10;

    function vermeningvuldig() {
        var getal2 = 7; // lokale variabele
        document.getElementById("output").innerHTML = getal1 * getal2;
    }

    vermeningvuldig()
</script>
</body>
</html>
```

❖ Let en const (ES6)

De **`const` declaratie** wordt gebruikt voor een read-only variabele, een constante. Wanneer aan een constante een waarde is toegekend, kan je er geen andere aan toewijzen.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Const</title>
```

```
</head>
<body>
<output id="output"></output>
<script>
    "use strict";
    const MINUTES_IN_HOUR = 60;

    MINUTES_IN_HOUR = 40; //Uncaught TypeError: Assignment to constant variable.

    function aantalMinuten(aantalUren) {
       return hour * MINUTES_IN_HOUR;
    }

    document.getElementById("output").innerHTML =
                                          "3 uur = " + aantalMinuten(3) + " min";
</script>
</body>
</html>
```

De `let` **declaratie** wordt gebruikt voor variabelen waarvan de waarde wel mag veranderen tijdens de uitvoering van het programma (zoals een teller in een herhaling, …). Verder zorgt let er ook voor dat de variabele enkel gebruikt kan worden in het code-blok waarin de variabele wordt gedefinieerd. Dit laatste is niet het geval bij gebruik van var zoals onderstaand voorbeeld laat zien.

```
<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <title>Var en let</title>
</head>
<body>
<output id="output"></output>
<script>
    "use strict";

    function optellen() {
        let resultaat = "";
        for (let teller = 0; teller <= 5; teller++) {
            resultaat += teller + "<br/>";
        }
        for (var andereTeller = 6; andereTeller <= 10; andereTeller++) {
            resultaat += andereTeller + "<br/>";
        }
        resultaat += "teller=" + teller + "<br/>"; // ReferenceError
        resultaat += "andereTeller=" + andereTeller + "<br/>";
        return resultaat;
    }
    document.getElementById("output").innerHTML = optellen();
</script>
</body>
</html>
```

Je krijgt nu een foutmelding “Uncaught ReferenceError: teller is not defined”, terwijl andereTeller wel gekend is na de for-lus (en de waarde 11 heeft).

Gebruik dus steeds let bij for-lussen!

In strict mode zal het bij gebruik van let ook niet mogelijk zijn een variabele met dezelfde naam te herdeclareren.

```
'use strict';
let me = 'foo';
let me = 'bar'; // SyntaxError: Identifier 'me' has already been declared

var me = 'foo';
var me = 'bar'; // No problem, `me` is replaced.
```

❖ Arrays

Een variabele kan telkens maar één waarde bevatten, maar soms is het nuttig om bepaalde variabelen te koppelen in een reeks van gerelateerde data. Net zoals in de programmeertaal Java (en andere programmeertalen) kan hiervoor een array of reeksvariabele gebruikt worden. De reeksvariabele is dus een rij van objecten, maar wordt opnieuw voorgesteld door een object. Het declareren van een array kan op verschillende manieren.

```
var leeg = new Array();              // geen bepaalde grootte, leeg
var weekdag = new Array(7);           // grootte 7, leeg
var cijfers = new Array(10);          // grootte 10, leeg
var cijfers = new Array(0,1,2,3,4,5,6,7,8,9);   // grootte 10, bepaald
```

De eerste reeks is leeg, de tweede reeks weekdag heeft plaats voor 7 elementen, maar die zijn nog niet gedefinieerd. De derde reeks cijfers heeft 10 ongedefinieerde elementen, terwijl deze in de vierde reeks wel bepaald zijn. De ingevulde variabelen hoeven niet noodzakelijk van hetzelfde type te zijn.

Het benaderen van de elementen van de reeks gebeurt met vierkante haakjes. Let op: de indexen beginnen te tellen vanaf nul!

```
var maand = new Array(12);
maand[0] = "januari";
maand[1] = "februari";
…
maand[11] = "december";
```

Dit kan ook korter in gecondenseerde opmaak:

```
var maand = new Array("januari", “februari", …, "december");
```

Of in letterlijke opmaak, gebruik makend van vierkante haken.

```
var maand = new Array();
     var maand = ["januari", “februari", …, "december"];
```

Het aantal elementen in een array wordt bijgehouden in de property `length`.

```
var n = maand.length;
maand[n-1] = "december";           // laatste element
```

Let erop dat je de juiste indexen gebruikt, want als je een foutieve (te grote) index gebruikt, wordt de reeks gewoon stilzwijgend groter gemaakt. Hierin verschilt JavaScript van Java waar er een `ArrayIndexOutOfBoundException` zou worden teruggegeven.

```
maand[12] = "Leve de dertiende maand!";  //maand bevat nu 13 elementen
```

8 Controlestructuren

Controlestructuren zorgen ervoor dat er in een programma beslissingen genomen worden en berekeningen uitgevoerd. Deze constructies komen min of meer letterlijk voor in de meeste programmeertalen. We bespreken ze daarom bondig.

❖ if-else

Deze structuur laat toe om op basis van een voorwaarde een beslissing te nemen. De voorwaarde staat tussen haakjes en maakt gebruik van een vergelijkingsoperator of een boolean. Ze geeft als resultaat een booleaanse waarde (`true` of `false`) terug. Indien er meerdere statements uitgevoerd moeten worden na de `if` of `else`, dan groepeer je de statements met accolades.

```
if (voorwaarde) {
  statements;
}
```

Bijvoorbeeld:

```
if (leeftijd >= 18) {
   document.getElementById(“output”).innerHTML = “+ 18”;
}
```

In dit voorbeeld wordt getest of de variabele `leeftijd` groter is dan 18.

Om te controleren of een variabele een geldige waarde heeft kan je de volgende voorwaarde gebruiken:

```
if( value ) {
}
```

Dit wordt geëvalueerd naar `true` indien `value` niet een van de volgende waarde is:

- null
- undefined
- NaN
- empty string ("")
- 0
- false

Je kan meerdere voorwaarden samenvoegen door de operatoren `&&` en `||`. Negatie van een test doe je met `!` (zie logische operatoren). Let erop dat je de haakjes correct gebruikt!

De `if`-structuur kan uitgebreid worden met een `else`-gedeelte.

```
if (voornaam && achternaam) {
  document.getElementById(“output”).innerHTML =
         "Hallo, " + voornaam + " " + achternaam;
}
else {
  document.getElementById(“output”).innerHTML = "Hallo, John Doe!";

}
```

De if-structuur kan verkort weergegeven worden door middel van de voorwaardelijke operator, die geschreven wordt als een vraagteken (?).

```
var teken = (i >= 0)? 'positief': 'negatief';
```

In bovenstaand voorbeeld zal de stringvariabele dus het woord positief bevatten als i groter of gelijk is aan nul, in het andere geval bevat ze het woord negatief.

❖ switch

De `switch`-constructie is een veralgemening van de `if-else` structuur en laat toe om op een overzichtelijke manier verschillende mogelijke beslissingen te evalueren. Dit is veel leesbaarder en handelbaarder dan een geneste `if-else` structuur.

In het voorbeeld is de uitvoering van het programma afhankelijk van de waarde van de variabele `n`. Als `n` gelijk is aan 1 wordt het bijhorende stuk code dat volgt op `case 1` uitgevoerd. Bemerk het `break`-statement dat ervoor zorgt dat er niet verder gegaan wordt met `case 2`, maar dat er correct uit de `switch` wordt gesprongen. Een typische fout is het vergeten van het `break`-statement. Tenslotte staat een `default`-gedeelte ter beschikking voor alle andere gevallen.

```
switch(n) {
  case 1:              //voer code uit horende bij n == 1
    statement1;
    break;             //spring uit het switch statement
  case 2:              //voer code uit horende bij n == 2
    statement2;
    break;             //spring uit het switch statement
  case 3:              //voer code uit horende bij n == 3
    statement3;
    break;             //spring uit het switch statement
       default:             //voer code uit horende bij alle andere gevallen
    statementn;      //spring uit het switch statement
}
```

❖ while

Een lus kan gemaakt worden met behulp van een `while`-statement.

```
while (voorwaarde) {
  statement(s);
}
```

Zolang aan de voorwaarde voldaan wordt `(true)`, worden de statements uitgevoerd. Het volgende voorbeeld drukt de even getallen kleiner dan 10 af.

```
function printEven() {
      var tekst ="Even getallen: ";
      var i = 0;
      while (i < 10) {
      if (i % 2 === 0) {
            tekst += i + " ";
      }
      i++;
      }
      document.getElementById("output").innerHTML = tekst;
}
```

❖ do-while

De `do-while`-constructie is een alternatief voor de `while`-constructie. Het verschil is dat de statements altijd minstens éénmaal worden uitgevoerd.

```
do {
  statement(s);
} while (voorwaarde);
```

De statements worden uitgevoerd zolang de expressie waar is. Let ook op de puntkomma (;) na de voorwaarde. Hier volgt de aanpassing om de even getallen te printen.

```
function printEven () {
     var tekst ="Even getallen: ";
     var i = 0;
     do  {
     if (i % 2 === 0) {
      tekst += i + " ";
     }
     i++;
     } while (i < 10);
     document.getElementById("output").innerHTML = tekst;
}
```

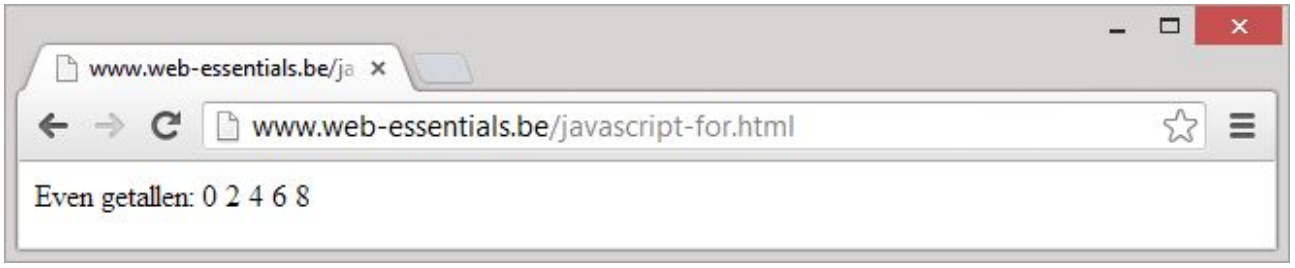
❖ for

De `for`-lus maakt het werken met tellers wat eenvoudiger en is de aangewezen constructie wanneer het aantal herhalingen van te voren gekend is. Tussen de haakjes komt eerst de initialisatie waarbij de teller op een waarde ingesteld wordt, daarna volgt de voorwaarde. Zolang er aan de voorwaarde voldaan wordt en dus de waarde `true` terugkomt, blijft het programma in de `for`-lus, anders (`false`) springt ze eruit. Als laatste komt de definitie van de stappen, meestal een ophoging in de vorm van `i++`.

```
for (initialisatie; voorwaarde; stappen) {
  statement(s);
}
```

We herhalen het voorbeeld met de even getallen.

```
function printEven () {
     var tekst ="Even getallen: ";
     for (let i = 0; i < 10; i++) {
     if (i % 2 === 0) {
      tekst += i + " ";
      }
     }
     document.write(tekst);
}
```



9 Functies

❖ Functie expressie en functie declaratie

Er zijn twee manieren om functies te definiëren in JavaScript. De eerste manier is de functie expressie.

Een **functie expressie** is eigenlijk een gewone variabele declaratie waarbij de waarde van de variabele een functie is. In het onderstaande voorbeeld hebben we een variabele *square* waaraan een functie (die een kwadraat berekent) is toegekend.

```
<!DOCTYPE html>
<html lang="nl”>
<head>
    <meta charset="UTF-8">
    <title>Demo9: functie</title>
</head>
<body>
<output id="square"></output>
<output id="cube"></output>
<script>
    "use strict";
    var getal1 = 5;
    var getal2 = 4;

    document.getElementById("square").innerHTML = square(getal1);
    // bovenstaande regel geeft: "Uncaught TypeError: square is not a function"
    document.getElementById("cube").innerHTML = cube(getal1);

    var square = function(x) {
        return x * x;
    };

    function cube(x) {
        return x * x * x;
    }

    document.getElementById("square").innerHTML = square(getal2);
    document.getElementById("cube").innerHTML = cube(getal2);
</script>
</body>
</html>
```

Voor de functie cube gebruiken we een **functie declaratie**. Deze vorm van definiëren van functies heeft één groot verschil met de **functie expressie**: de functie declaratie geeft je meer vrijheid over de plaats waar je ze plaats, omdat de functie ook beschikbaar is in code die voorafgaat aan de functie declaratie.

❖ Geneste functies

```
<!DOCTYPE html>
<html lang="nl”>
<head>
    <meta charset="UTF-8">
    <title>Demo10: nested functions</title>
</head>
<body>
```

```
<output id="output"></output>
<script>
    var landscape = function() {
        var result = "";
        var flat = function(size) {
            for (let count = 0; count < size; count++)
                result += "_";
        };
        var mountain = function(size) {
            result += "/";
            for (let count = 0; count < size; count++)
                result += "'";
            result += "\\";
        };

        flat(3);
        mountain(4);
        flat(6);
        mountain(1);
        flat(1);
        return result;
    };

    document.getElementById("output").innerHTML = landscape();
</script>
</body>
</html>
```

❖ Immediately-Invoked Function Expression (IIFE)

IIFE (uitspraak: “Iffy”) is de afkorting voor Immediately-Invoked Function Expression, en de syntax ziet er als volgt uit:

```
(function () {
    // do fun stuff
})();

of

(function () {
    // do fun stuff
}());
```

Het is een anonieme functie die onmiddellijk wordt uitgevoerd. Deze constructie wordt gebruikt om te vermijden dat variabelen in de global scope gedeclareerd worden. Globale variabelen kunnen soms vreemde neven-effecten veroorzaken in je script (zeker bij concurrency (=het gelijktijdig uitvoeren van bepaalde stukken code)). Het is daarom aangeraden om je volledige script als een IIFY te schrijven.

```
<!DOCTYPE html>
<html lang="nl”>
<head>
```

```
    <meta charset="UTF-8">
    <title>Demo 11: IIFE</title>
    <style>
        output {
            display: block;
        }
    </style>
</head>
<body>
<output id="output1"></output>
<output id="output2"></output>
<output id="output3"></output>
<script>
    (function () {
            (function () {
                var now = new Date();
                var start = new Date(now.getFullYear(), 0, 0);
                var diff = now - start;
                var oneDay = 1000 * 60 * 60 * 24;
                var days = Math.floor(diff / oneDay);
                document.getElementById("output1").innerHTML =
                        "Dag van het jaar:" + days;
            })();

            (function (bedrag) {
                var incBtw = 1.21;
                document.getElementById("output2").innerHTML =
                        "Bedrag inc. btw: " + (incBtw * bedrag);
            }(120));

            var min = Math.min(15, (function () {
                return new Date().getDate();
            })());

            document.getElementById("output3").innerHTML = min;
    }());
</script>
</body>
</html>
```

❖ Functie met return-waarde functie

```
<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <title>Demo12: functie return functie</title>
</head>
<body>
<output id="output"></output>
<script>
    (function() {
        var btw = function (procent) {
            return function (bedrag) {
                return bedrag * (1 + procent / 100);};
        };

        var btwEenEnTwintig = btw(21);
        var btwZes = btw(6);
```

```
        document.getElementById("output").innerHTML =
                    btwEenEnTwintig(120) + " " + btwZes(120);
    })();
</script>
</body>
</html>
```

10 Objecten

Objectgeoriënteerd programmeren wordt door veel talen ondersteund, ook door JavaScript. In dit deel zie je kort hoe JavaScript objecten aanbiedt en hoe je als programmeur ermee overweg kan.

❖ Wat is een Object?

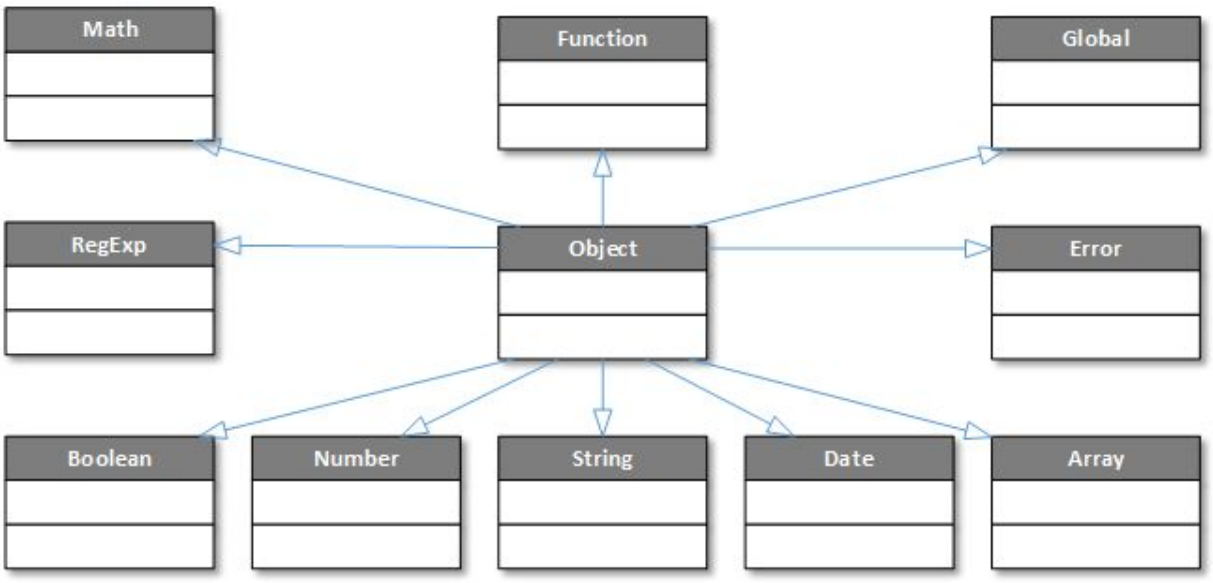
Het is niet de bedoeling om hier volledig uit te leggen wat een object is. Daarover bestaan er meerdere handboeken. In een programma bestaat een **object** uit een set van waarden (**properties)** en **methods**. Een method kan de toestand (property) van een object wijzigen.

Een `String`, een `Array`, een `Window`, een `Frame`, een `Form`, . . . zijn allemaal objecten die eigenschappen bevatten en zo een **entiteit** vormen.

Een object heeft **properties** of eigenschappen die je kunt lezen en/of schrijven. Bijvoorbeeld een `window` heeft een `name` die je kunt zetten of opvragen. De property van een object kan opnieuw objecten opleveren, bijvoorbeeld een `window` heeft een property `frames[]` die een array van frame-objecten oplevert.

```
object.property
object.method()
```

Een object heeft **methodes** die je kan oproepen. Deze kunnen de toestand van dat object wijzigen. Bijvoorbeeld een `window` heeft een methode `close()` om dit venster te sluiten**.** Onderstaande figuur geeft een overzicht van de objecten die JavaScript ter beschikking stelt aan de programmeur. Je kan ze dus gebruiken in eender welke omgeving die de JavaScript standaard ondersteunt.



- ***Object***

Dit is de basisklasse. Elk object binnen JavaScript erft de properties en methods van dit object over. Bijvoorbeeld de methode `Object.toString()` geeft een string voorstelling van het betreffende object.

- ***Boolean***

Deze objecten fungeren als omhulsel (eng: wrapper) voor de booleaanse waarden `true` en `false`.

- ***Number***

Objecten van het type `Number` stellen getallen voor. Deze klasse heeft enkele belangrijke properties, bijvoorbeeld `Number.NaN, Number.MAX_VALUE` en `Number.MIN_VALUE`.

- ***String***

Een `String` is een karaktervoorstelling van tekstuele informatie.

- ***Date***

Objecten die dienen om te werken en te rekenen met datums.

- ***Array***

Dit is een datastructuur die bestaat uit een lijst van elementen (objecten).

- ***RegExp***

De klasse `String` biedt reeds heel wat methodes die helpen met het zoeken van woorden en patronen. Deze klasse gaat hierop verder en biedt de programmeur *reguliere expressies*. Oorspronkelijk afkomstig uit de Unix-wereld en de taal Perl, is het hier mogelijk om ingewikkelde zoekpatronen op te bouwen.

- ***Math***

Dit een bibliotheek van de meest gebruikte wiskundige functies en constanten.

- ***Global***

Een verzameling van alle globale objecten. Als je zelf een globale variabele declareert erft die de properties en methods over van de klasse `Global`.

- ***Error***

Net zoals Java en C++, biedt JavaScript een exception mechanisme. We gaan hier niet verder op in, maar vermelden dat de exceptions die gegooid worden van het type `Error` zijn.

❖ Objecten maken

```
var vandaag = Date.now();
var kerstmis = new Date(2009, 12, 25);
```

Meestal hoef je geen objecten aan te maken, maar gebruik je reeds beschikbare objecten. Bijvoorbeeld binnen een browser omgeving beschik je over de objecten `document` en `window`.

❖ Properties en methodes

Properties benader je door middel van de zogenaamde *dot-notatie*. Aan de linkerkant van de punt is een referentie naar het object, de rechterkant is de beschouwde property. Een methode is een functie die hoort bij een object. Ze wordt net zoals properties opgeroepen door middel van de *dot-notatie*. Een voorbeeld maakt veel duidelijk.

```
var myField = document.getElementById("field"); // methode
myField.style.backgroundColor = "white";        // properties
```

11 Document Object Model

12 Browser Object Model (BOM)

Het **window**-object wordt door alle browsers ondersteund en is een referentie naar het browserscherm. Alle globale variabelen en functies zijn beschikbaar op het window-object. Globale variabelen zijn eigenschappen van het window-object. Globale functies zijn methoden van het window-object.

```
<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <title>Demo: BOM</title>
    <script>
        var x = 120;

        var isEven = function(waarde) {
            return waarde && waarde % 2 == 0;
        }
    </script>
</head>
<body>
<output id="output" />

<script>
    var info = "x=" + window.x + " of " + x + "\n";
    info += "isEven(window.x)=" + window.isEven(x) + "\n";
    info += "afmetingen browserscherm (px)= " + window.innerHeight + " X " +
window.innerWidth + "\n";
    info += "url=" + window.location.href + "\n";
    info += "os=" + navigator.platform + "\n";

    document.getElementById("output").innerText = info;
</script>
</body>
</html>
```

13 Document Object Model (DOM)

Wanneer je een webpagina opent in de browser, dan haalt de browser de inhoud van de pagina op van de server en verwerkt vervolgens de inhoud van de pagina (dit noemen we “parsen”). Tijdens het parsen van de pagina bouwt de browser een model van de structuur van de webpagina. Dit model wordt dan gebruikt om de pagina op het scherm (of een ander medium) weer te geven.

Dit model, de voorstelling van de webpagina, kan je ook manipuleren in een Javascript programma. Je kan gegevens uit het model ophalen en aanpassen. Het model is een “levende” datastructuur: als je het model aanpast, zal de voorstelling van de webpagina op het scherm aangepast worden en je aanpassing dus zichtbaar zijn.

14 W3C DOM

Hier is een html-document:

```
<!DOCTYPE html>
<html lang="nl”>
<head>
    <meta charset="UTF-8">
    <title>Demo 14: DOM</title>
</head>
<body>
<form id="myForm"
      action="mailto:nele.custers@pxl.be?subject=myAnswer"
      enctype="text/plain" method="post">
    <label for="naam">Naam:</label>
    <input type="text" id="naam" name="naam" />
    <fieldset id="vraag1">
        <legend>Inside which HTML element do we put the JavaScript?</legend>
          <input type="radio" name="scriptTag" value="js" id="js"/>
          <label for="js">&lt;js&gt;</label>
          <input type="radio" name="scriptTag" value="script" id="script"/>
          <label for="script">&lt;script&gt;</label>
          <input type="radio" name="scriptTag" value="javascript"
                 id="javascript"/>
          <label for="javascript">&lt;javascript&gt;</label>
          <input type="radio" name="scriptTag" value="code" id="code"/>
          <label for="code">&lt;code&gt;</label>
    </fieldset>
</form>
<ul class="hinder">
    <li>Weet je het antwoord al?</li>
</ul>
</body>
</html>
```

De bijhorende DOM ziet er als volgt uit (gegenereerd met http://software.hixie.ch/utilities/js/live-dom-viewer/):

```
DOCTYPE: html
HTML lang="en"
  HEAD
    #text:
    META charset="UTF-8"
    #text:
    TITLE
      #text: Demo 14: DOM
    #text:
  #text:
  BODY
    #text:
    FORM id="myForm" action="mailto:nele.custers@pxl.be?subject=myAnswer" enctype="text/plain" method="post"
      #text:
      LABEL for="naam"
        #text: Naam:
      INPUT name="naam" id="naam" type="text"
      #text:
      FIELDSET id="vraag1"
        #text:
        LEGEND
          #text: Inside which HTML element do we put the JavaScript?
        #text:
        INPUT name="scriptTag" id="js" type="radio" value="js"
        LABEL for="js"
          #text: <js>
        #text:
        INPUT name="scriptTag" id="script" type="radio" value="script"
        LABEL for="script"
          #text: <script>
        #text:
        INPUT name="scriptTag" id="javascript" type="radio" value="javascript"
        LABEL for="javascript"
          #text: <javascript>
        #text:
        INPUT name="scriptTag" id="code" type="radio" value="code"
        LABEL for="code"
          #text: <code>
        #text:
      #text:
    #text:
    UL class="hinder"
      #text:
      LI
        #text: Weet je het antwoord al?
      #text:
    #text:
```

De gebruikte methode om een door een `id`-attribuut benoemd element te benaderen is

```
var tekst = document.getElementById('naam').value;
```

Twee andere methodes zijn `getElementsByTagName()`en `getElementsByName()`. Let erop dat er hier in veelvoud (`Elements`) gesproken wordt aangezien er natuurlijk meerdere objecten met dezelfde naam of tagnaam kunnen zijn. Alle HTML-elementen die voldoen aan deze naam komen in een array terecht. Bij `getElementById()`is dit niet zo aangezien een `id`-attribuut uniek moet zijn.

```
var tekst = document.getElementsByTagName('input').item(0).value;
```

Indien er aan het `input`-element naast een `id` ook een `name`-attribuut was meegegeven, kan ook de volgende code.

```
var tekst = document.getElementsByName('naam').item(0).value;
```

of

```
var tekst = document.getElementsByName('naam')[0].value;
```

Zoals in de bovenstaande voorbeeldcodes is weergegeven kan van elk geadresseerd element een attribuutwaarde opgevraagd worden, door de naam van het attribuut mee te geven.

```
var titel = document.getElementById('identificatie').title;
var waarde = document.getElementById('identificatie').value;
var klasse = document.getElementById('identificatie').class;
var bron = document.getElementById('identificatie').src;
```

Bij het benaderen van een stijlregel van een HTML-element moet de term `style` opgegeven worden, gevolgd door de benaming van de stijldeclaratie. Indien deze benaming uit meerdere woorden bestaat dient ze aan elkaar vast geschreven te worden, waarbij het tweede woord met een hoofdletter begint.

```
var hoogte = document.getElementById('naam').style.height;
var kleur = document.getElementById('naam').style.color;
var achtergrond = document.getElementById('naam').style.backgroundColor;
var bovenmarge = document.getElementById('naam').style.marginTop;
```

Binnen het W3C-DOM kunnen deze methods verder aangevuld worden met de volgende formuleringen:

- `childNodes[]`, een reeks van alle knooppunten onder dat bepaalde element, sequentieel genummerd in de volgorde waarin ze in de HTML-code voorkomen.
- `firstChild`, het eerste onderliggend HTML-element.
- `lastChild`, het laatste onderliggend HTML-element.

Om met behulp hiervan terug de waarde van het `input`-element met de `id` naam op te vragen, kan dit met de volgende code. Ondanks het feit dat er maar één `body`-element mag aanwezig zijn in het `document`, moet hier toch het volgordenummer binnen de reeks meegegeven worden.

```
var tekst =
     document.getElementsByTagName('body')[0].firstChild.firstChild.value;
```

of

```
var tekst =
     document.getElementsByTagName('body')[0].childNode[0].childNode[0].value;
```

Vanzelfsprekend kan ook een mengvorm van beide methoden.

```
var tekst =
     document.getElementsByTagName('body')[0].firstChild.childNode[0].value;
```

of

```
var tekst =
     document.getElementsByTagName('body')[0].childNode[0].firstChild.value;
```

Nieuw vanaf HTML5 zijn de functies `document.getElementsByClassName()`, `document.querySelector()` en `document.querySelectorAll()`.

Met `document.getElementsByClassName()`kan je alle elementen kiezen waaraan je een bepaalde `class` hebt toegekend. Dit kan handig zijn om de stijl of inhoud van dergelijke elementen te veranderen.

```
<script>
    function maakOranje () {
        var groep = document.getElementsByClassName('geel');
        for (let i = 0; i < groep.length; i++) {
            groep[i].style.color='orange';
            }
    }
</script>
```

Met `document.querySelector()` kunnen CSS-selectors worden meegegeven om een element te definiëren. Enkel de eerste hit wordt opgenomen.

```
<script>
    function maakOranje () {
        document.querySelector('p + p').style.color='orange';
    }
</script>
```

Indien alle elementen die matchen met de opgegeven selector moeten geselecteerd worden in een reeks, dan dien je `document.querySelectorAll()` te gebruiken.

```
<script>
    function maakOranje () {
        var groep = document.querySelectorAll('p + p');
        for (var i = 0; i < groep.length; i++) {
            groep[i].style.color='orange';
            }
    }
</script>
```

We gebruiken nu de bovenstaande methodes om de DOM van onze pagina te gaan manipuleren:

```
<!DOCTYPE html>
<html lang="nl">
<head>
    <meta charset="UTF-8">
    <title>Demo 14: DOM</title>
</head>
<body>
<form id="myForm"
      action="mailto:nele.custers@pxl.be?subject=myAnswer"
      enctype="text/plain" method="post">
    <label for="naam">Naam:</label>
    <input type="text" id="naam" name="naam" />
    <fieldset id="vraag1">
        <legend>Inside which HTML element do we put the JavaScript?</legend>
          <input type="radio" name="scriptTag" value="js" id="js"/>
          <label for="js">&lt;js&gt;</label>
          <input type="radio" name="scriptTag" value="script" id="script"/>
          <label for="script">&lt;script&gt;</label>
          <input type="radio" name="scriptTag" value="javascript"
                 id="javascript"/>
          <label for="javascript">&lt;javascript&gt;</label>
          <input type="radio" name="scriptTag" value="code" id="code"/>
```

```
            <label for="code">&lt;code&gt;</label>
    </fieldset>
</form>
<ul class="hinder">
    <li>Weet je het antwoord al?</li>
</ul>
<script>

    (function() {
        var scriptTags = document.getElementsByName("scriptTag");
        for (let counter = 0; counter < scriptTags.length; counter++){
            scriptTags.item(counter).value = "code";
        }
    }());

    var scriptTagsVerdwijn = function() {
        var scriptTags = document.getElementsByName("scriptTag");
        for (let counter = 0; counter < scriptTags.length; counter++){
            scriptTags.item(counter).style.visibility = "hidden";
        }
    }

    var submitForm = function() {
        document.getElementById("myForm").submit();
    }

    var hinder = function() {
        var newElement = document.createElement("li");
        var hinderLijst =  document.getElementsByClassName("hinder")[0];
        hinderLijst.appendChild(newElement);
        hinderLijst.lastChild.innerText = "Weet je het al?";

    }

    setTimeout(scriptTagsVerdwijn, 9000);
    setTimeout(submitForm, 6000);
    setInterval(hinder, 1000);
</script>

</body>
</html>
```

15 DOM scripting

16 Inleiding: Wat is DOM-scripting?

Om browser- en platformonafhankelijk te werken is het noodzakelijk om enkel met algemeen erkende technologieën te werken, nl. **CSS**, **Javascript** en het **W3C DOM** (Document Object Model). Al deze standaarden worden door de huidige generatie browsers ondersteund.

Voor deze browseronafhankelijke combinatie wordt vaak de term DOM-scripting gebruikt. Er wordt gebruik gemaakt van het W3C DOM om van bepaalde HTML-elementen het uitzicht te wijzigen. Veelgebruikte toepassingen zijn onder andere:

- het zichtbaar en onzichtbaar maken van elementen of lagen door bijvoorbeeld de CSS-declaratie *visibility:visible* te wijzigen naar *visibility:hidden*.
- De kleur van bepaalde elementen wijzigen.
- De volledige opmaak van een pagina wijzigen.
- Het creëren van een roll-over afbeelding die dus wijzigt wanneer de muis erover gaat.
- Het programmeren van een diavoorstelling.
- Het valideren van dynamische formulieren.

Al deze functies worden in JavaScript geprogrammeerd en opgeroepen door een bepaalde trigger.

17 Event handling

Het gebruik van JavaScript binnen een webbrowser is hoofdzakelijk eventbased. De gebruiker leest de webpagina en reageert erop: formulieren verzenden, op links klikken, scrollen, animaties starten, ...

De gebeurtenis die de gebruiker initieert noemt men een **event**. Nu kan je aan deze gebeurtenis gehoor geven door een bepaald stuk programmacode uit te voeren. Dit noemt men **event handling**.

❖ Event handling via JavaScript

```
<type type="button" id="knop>…</button>
```

We kunnen op twee manieren events toevoegen aan html-elementen. De oudste manier voegt een event als methode toe, waarbij je dit aan een functie koppelt.

```
document.getElementById("knop").onclick = doeIets;

function doeIets(){
     …
}
```

Bij de nieuwere gebruik je de methode “addEventListener”. Onze voorkeur gaat uit naar deze nieuwere methode.

```
window.addEventListener('load',function() {
     document.getElementById("knop").addEventListener('click', doeIets);
});

function doeIets(){
     …
```

```
}
```

❖ Event handlers voor documenten en vensters

♦ ***onload***

Dit event treedt op nadat een object is geladen. Dit wordt zeer veel gebruikt bij het laden van een document en is zelfs onmisbaar als nieuwe pagina's dynamisch dienen opgebouwd te worden. In onderstaand voorbeeld wordt bij het laden een pop-upvenster geopend.

```
window.addEventListener('load',function() {
    alert('Welkom op de site!');
});
```

♦ ***onunload***

Dit event treedt op indien een object niet langer is geladen. Indien dit toegepast wordt op het body-element dan geeft dit de waarde `true` terug als de gebruiker de pagina verlaat.

♦ ***onresize***

Dit event treedt op als de afmetingen van het browservenster of van een frame worden aangepast.

❖ Event handlers voor alle elementen

♦ ***onclick***

Dit event wordt meestal toegepast bij knopachtige elementen, afbeeldingen en hyperlinks, maar kan eigenlijk bij alle elementen gebruikt worden. De bijhorende programmacode wordt uitgevoerd wanneer de gebruiker op het element klikt. Als de handler (de JavaScript functie die hoort bij dit attribuut) de waarde `false` teruggeeft, dan voert de browser de standaardactie die bij dit element hoort *niet* uit. Er wordt bijvoorbeeld niet gesprongen naar een URL (voor het `<a>`-element), of het formulier wordt niet verstuurd wanneer op de submitknop geklikt wordt.

Ondanks het feit dat `onclick` bij alle elementen gebruikt kan worden moet hier spaarzaam mee omgesprongen worden aangezien gebruikers niet de gewoonte hebben overal op te gaan klikken.

♦ ***onmousedown, onmouseup***

Deze events treden op als de gebruiker met de muis op een element staat en de muisknop respectievelijk induwt en loslaat. De meeste elementen die `onclick` ondersteunen, ondersteunen ook deze events.

♦ ***onmouseout, onmouseover***

Deze events treden op als de gebruiker met de muis over een element beweegt (`onmouseover`), of zich niet langer boven het element bevindt (`onmouseout`). Ze worden zeer veel met afbeeldingen gebruikt om een rollover afbeelding te maken, maar kunnen ook zeer nuttig zijn bij imagemaps en hyperlinks. Indien `onmouseover` bij een `<a>`-element gebruikt wordt dan zal de URL die bij deze link hoort *niet* in de statusbalk van de browser weergegeven worden als dit event de waarde `true` teruggeeft.

♦ ***onmousemove***

Dit event treedt op als de gebruiker de muis beweegt als ze zich boven het element bevindt.

❖ Event handlers voor formulieren

♦ ***onchange***

Dit event hoort enkel bij de elementen `<input>`, `<select>` en `<textarea>`. Het treedt op als de gebruiker de waarde van deze elementen wijzigt.

- ***onsubmit, onreset***

Deze events horen bij het `<form>`-element en treden op als het formulier verstuurd (`submit`) of gewist (`reset`) wordt. Als de handler de waarde `false` teruggeeft, dan wordt de form *niet* verstuurd of gewist.

- ***onfocus***

Dit event treedt op als een element wordt geselecteerd en kan toegepast worden bij hyperlinks `<a>`, bij afbeeldingsgebieden `<area>` en de formulierelementen `<input>, <select>` en `<textarea>`. Je kan de focus verplaatsen met de tab toets of met de muis.

- ***onblur***

Dit event treedt op als de selectie van een element ongedaan wordt gemaakt en is dus de tegenhanger van `onfocus`.

- ***onkeydown***

Dit event treedt op als een toets ingedrukt wordt op het toetsenbord.

- ***onkeyup***

Dit event treedt op als de toets van het toetsenbord terug wordt losgelaten.

- ***onkeypress***

Dit event treedt op na het `onkeydown`-event, dus als de toets ingedrukt gehouden wordt. Keypress werkt enkel voor printbare karakters, dus niet voor bijvoorbeeld de alt- of ctrl-toets.

18 Pagina's dynamisch maken

❖ Methods en properties van window

Als je een webbrowser start, opent slechts één venster (window). Terwijl je surft op het net, kan het gebeuren dat nieuwe vensters geopend worden en andere gesloten. Dit kan je bekomen met de `window.open()` en `window.close()`-methodes van het `window` object. Deze methodes worden dus gebruikt voor nieuwe vensters, dit in tegenstelling tot `document.open()` en `document.close()` die ingrijpen op het huidige venster.

Een nieuw venster openen kan met de `open()`-methode.

```
window.open()
```

Het sluiten of beëindigen van een venster kan met methode `close()`.

```
window.close()
```

Naast het openen en sluiten kan je met JavaScript ook vensters verschuiven en van grootte veranderen. Dit kan naar een absolute waarde of met een relatieve aanpassing ten opzichte van de huidige waarde.

```
moveTo(x,y)
moveBy(dx,dy)
resizeTo(width,height)
resizeBy(dwidth, dheight)
```

Bij het openen van een nieuw browservenster kan je verschillende parameters meegeven.

```
window.open(url, name, features, replace)
```

Argument	Betekenis
url	De URL van de locatie die moet geopend worden. Als een lege string wordt meegegeven ("") wordt een leeg document getoond.
name	De naam van het venster. Deze naam kan gebruikt worden als de waarde van een `target`-attribuut bij een `<a>` of een `<form>`-tag.
features	Een string die de eigenschappen van het venster bevat. • height: de hoogte van het venster • left,top: de X-Y coördinaten van het venster (voor Internet Explorer) • location: de adresbalk (yes/no) • menubar: de menubalk (yes/no) • resizable: mag het venster van grootte veranderen (yes/no) • screenX, screenY: de X-Y coördinaten van het venster (voor Firefox). • scrollbars: de scrollbars (yes/no) • status: de statusbalk (yes/no) • toolbar: de toolbar met de Back en Forward knoppen enz. (yes/no) • width: de breedte van het venster
replace	Indien true, dan wordt een extra entry in de history lijst van de browser aangemaakt.

Je kan ook testen of een bestaand `window`-object reeds gesloten is, door te controleren wat de waarde is van de property `window.closed`. Dit uitgezonderd, kan je op een gesloten venster niet langer methodes of properties oproepen.

Beschouw het volgende voorbeeld. De pagina bevat een link die een nieuw venster opent en een andere link die het venster kan sluiten. Let op in de code hoe de eigenschappen van het venster bepaald worden.

```
<!DOCTYPE html>
<html lang="nl”>
<head>
    <meta charset="UTF-8">
    <title>Dynamische pagina's</title>
</head>
<body>
<h1 id="dummy">Welkom op de site</h1>
<p>Meer informatie, klik <a href="#dummy" id="linkOpen">hier</a>.</p>
<p>Klik <a href="#dummy" id="linkSluit">hier</a> om het venster te sluiten.</p>
<script>
    document.getElementById("linkOpen").addEventListener("click", show);
    document.getElementById("linkSluit").addEventListener("click", dispose);

    var nieuwVenster;

    function show() {
        var eigenschappen = "width=300, height=300, left=150";
        nieuwVenster = window.open("", "info", eigenschappen, false);
        var nieuweBody = nieuwVenster.document.getElementsByTagName("body")[0];
        var title = document.createElement("h1");
        title.innerHTML = "Hallo!";
        var paragraph = document.createElement("p");
        paragraph.innerHTML = "Dit venster bevat allerhande info";
        nieuweBody.appendChild(title);
        nieuweBody.appendChild(paragraph);
    }

    function dispose() {
        if (nieuwVenster) {
            nieuwVenster.close();
```

```
        }
    }
</script>
</body>
</html>
```

De functie show() opent een nieuw venster met de naam “info” en de eigenschappen `"width=300, height=300, left=150"`. Dit venster wordt toegekend aan de globale variabele `nieuwVenster`. Er worden nieuwe html-elementen aangemaakt en aan de body van het nieuwe venster toegevoegd.

De twee links hebben een `href`-attribuut dat wijst naar een anker binnen de pagina. Dit anker zit op het begin van de pagina, dus eigenlijk doet dit niets. Interessanter zijn de twee event handlers op `onclick`.

De tweede functie is de `dispose()`-operatie die de `close()`-methode van het venster oproept.

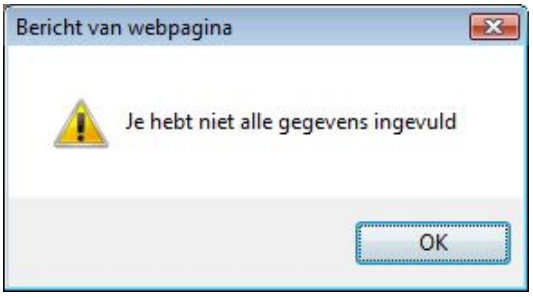
❖ pop-upvensters: alert, prompt, confirm

Je hebt in JavaScript ook de mogelijkheid om pop-upvensters te gebruiken om te communiceren met de gebruiker.

♦ *Het alert venster*

Het *alert* venster toont een korte boodschap die de gebruiker enkel kan bevestigen door op OK te klikken. Het wordt meestal gebruikt voor waarschuwingen. De methode heeft als argument een `string` die de boodschap of waarschuwing bevat.

```
window.alert('je hebt niet alle gegevens ingevuld');
```



♦ *Het confirm venster*

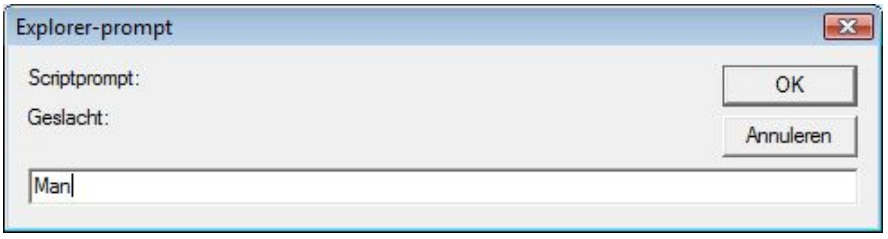
Indien een vraag moet gesteld worden waarop bevestigend of ontkennend dient geantwoord te worden, is een confirm venster de geëigende keuze. In het pop-up venster kan de gebruiker enkel op “OK” of “Annuleren” klikken. De methode retourneert `true` of `false`, waarop je dan beslissingen kan baseren. Het argument van de functie bevat de gestelde vraag.

```
var status = window.confirm('Wil je werkelijk afsluiten?');
// status bevat true of false
if (status) {
   doe_iets();
} else {
   doe_iets_anders();
}
```

♦ *Het prompt venster*

Als je input wil vragen doe je dat met de `prompt` methode van het `window` object. Het eerste argument van de functie is de gestelde vraag, het (optionele) tweede argument de standaardwaarde als de gebruiker op “Annuleren” zou klikken. De functie retourneert de ingetikte waarde of de standaardwaarde waarop je dan beslissingen kan baseren. Indien die er niet zou zijn wordt null teruggegeven.

```
var antwoord = prompt('Geslacht: ','Man');
if (antwoord !== "Man") {
   document.getElementById("greeting").innerHTML = "Welkom!";
} else {
   document.getElementById("greeting").innerHTML = "Gegroet!";
}
```



Toch is hier een waarschuwing op zijn plaats. Hoewel deze pop-upvensters toelaten om te communiceren met de gebruiker, moet je hiermee niet overdrijven. Gebruikers ervaren teveel pop-ups immers als erg hinderlijk, wat de gebruiksvriendelijkheid van de site niet ten goede komt. Bovendien kan de gebruiker de browser zo configureren dat pop-upvensters niet meer getoond worden, zodat je (belangrijke) boodschap toch verloren gaat.

19 DOM-scripting

In DOM-scripting wordt vooral gebruik gemaakt van de `getElementById()`-methode uit het W3C-DOM om HTML-elementen te benaderen. Elk HTML-element wordt als een knooppunt (node) beschouwd binnen het model.

❖ Benaderen van eigenschappen (properties)

De typische properties die kunnen benaderd worden zijn:

- `x.innerHTML` : de HTML-code binnen de HTML-container
- `x.nodeName`: het `name`-attribuut van een HTML-element
- `x.nodeValue`: het `value`-attribuut van een HTML-element
- `x.parentNode`: de parent node van een HTML-element
- `x.childNodes`: de child nodes van een HTML-element

- `x.firstChild`: het eerste kind van een knooppunt
- `x.lastChild`: het laatste kind van een knooppunt
- `x.attributes`: de attributen van een HTML-element

In deze opsomming staat x voor een knooppunt (bijvoorbeeld: `document.getElementById('knop').`)

De `innerHTML`-eigenschap is een zeer handige manier om HTML-elementen aan te passen. Hiermee kan je immers de inhoud van een HTML-container opvragen en aanpassen. Het opvragen van de inhoud van een knooppunt, kan als volgt.

```
tekst=document.getElementById("paragraaf").innerHTML;
```

Voor het aanpassen gebruik je de volgende code:

```
document.getElementById("paragraaf").innerHTML="Hallo, <big>Wereld</big>";
```

De tekstuele inhoud van een knooppunt kan opgevraagd worden met de property `nodeValue`. Dit werkt enkel voor pure tekst en attribuutknooppunten. In tegenstelling tot innerHTML wordt hier dus geen opmaakinformatie (HTML-codes) mee opgenomen en wordt de tekstuele inhoud ook als een child beschouwd. Een voorbeeld zal veel duidelijk maken.

```
<ul id="lijst"><li>een</li><li>twee</li><li>drie</li><li>vier</li></ul>
```

In bovenstaand voorbeeld is elk `li`-element een child van de lijst. De teksten een, twee, drie en vier zijn op hun beurt ook weer een child van het list item. Als je bijvoorbeeld de waarde drie wenst terug te krijgen, dan dien je volgende code te gebruiken.

```
tekst = document.getElementById('lijst').childNodes[2].firstChild.nodeValue;
```

Wil je de waarde vier terugkrijgen kan dat bijvoorbeeld met:

```
tekst = document.getElementById('lijst').lastChild.firstChild.nodeValue;
```

Aangezien de witruimtes (spaties en enters) ook knooppunten kunnen zijn in de DOM-tree is het nodig in de HTML-code alles aan elkaar te schrijven om dit werkend te krijgen. Deze werkwijze wordt dan ook voornamelijk bij XML gebruikt.

❖ Het gebruik van DOM Methods

Zoals al besproken in het onderdeel over het W3C DOM zijn er verschillende methodes beschikbaar om een HTML-element te benaderen.

- `x.getElementById(id)`: selecteert op het `id`-attribuut
- `x.getElementsByName(name)`: selecteert op het `name`-attribuut
- `x.getElementsByTagName(tag)`: selecteert op elementnaam
- `x.getElementsByClassName(class)`: selecteert op classnaam
- `x.querySelector(selector)`: selecteert één css-selector
- `x.querySelectorAll(selector)`: selecteert alle css-selectors
- `x.appendChild()`: voegt een onderliggend knooppunt toe
- `x.removeChild()`: verwijdert een onderliggend knooppunt
- `x.createElement()`: voegt een HTML-element toe

- `x.createTextNode()`: voegt een tekstueel knooppunt toe

Als we bij het voorbeeld van de lijst bijvoorbeeld een vijfde item wensen toe te voegen kan dat met onderstaande code:

```
var knooppunt=document.getElementById('lijst');
knooppunt.appendChild(document.createElement("li"));
knooppunt.lastChild.innerHTML="vijf";
```

Met de `appendChild`-methode wordt een kind toegevoegd, maar aangezien dit kind opnieuw een HTML-element is wordt de `createElement`-methode gebruikt, waarna hierin met de `innerHTML`-property de tekst vijf geplaatst wordt.

Indien enkel de tekst vijf aan de webpagina moest toegevoegd worden, dan kan je dat doen met de `creatTextNode`-methode.

20 Bronnen

- Eloquent JavaScript, second edition: A Modern introduction to programming. Marijn Haverbeke (http://eloquentjavascript.net/index.html)
- W3Schools JavaScript tutorial (https://www.w3schools.com/js/)