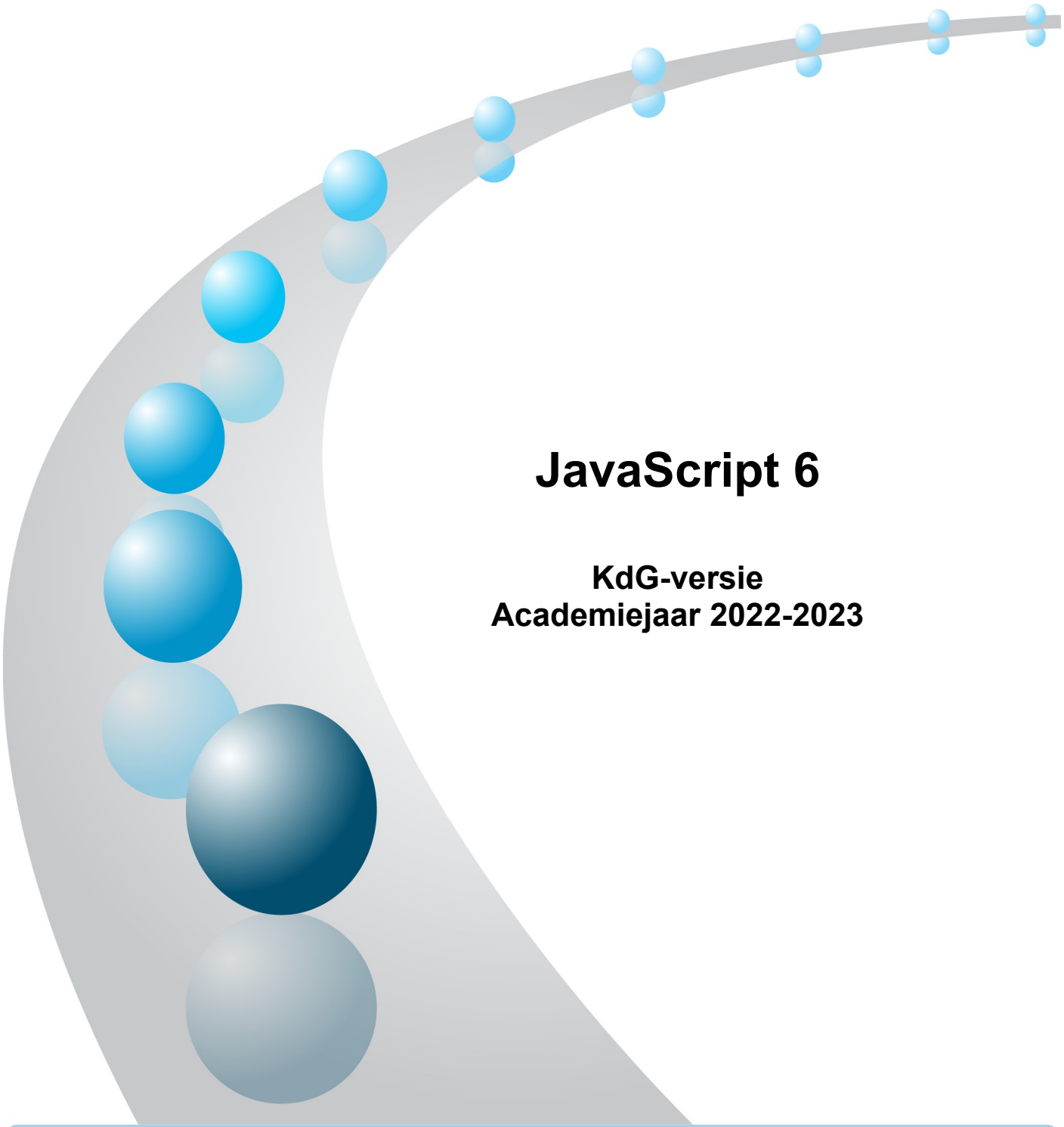
Noël Vaes
Java Trainer & Consultant



JavaScript 6

KdG-versie
Academiejaar 2022-2023

Roode Roosstraat 5
3500 Hasselt
België

+32 474 38 23 94
noel@noelvaes.eu
www.noelvaes.eu

Correspondentie inzake overnemen of reproductie kunt u richten aan:

Noël Vaes
Roode Roosstraat 5
3500 Hasselt
België

Tel: +32 474 38 23 94

noel@noelvaes.eu
www.noelvaes.eu

27/09/2022

Inhoudsopgave

Hoofdstuk 1: Inleiding in JavaScript

1.1 Inleiding

JavaScript is een scripttaal die destijds door *Netscape* ontwikkeld werd en die veel gebruikt wordt voor onder andere internettoepassingen. Deze scripttaal werd intussen gestandaardiseerd door de Europese instelling ECMA (www.ecma-international.org) en draagt officieel de naam *ECMAScript*. Er zijn inmiddels verschillende versies van *ECMAScript.* Momenteel wordt *ECMAScript 6 (ES 6)* door de meeste hedendaagse browsers ondersteund. Deze versie is uitgegeven in 2015 en wordt soms ook *ECMAScript 2015* genoemd. In deze cursus behandelen deze versie. Aangezien de taal in de volksmond gekend is als *JavaScript*, zullen we deze naam gebruiken.

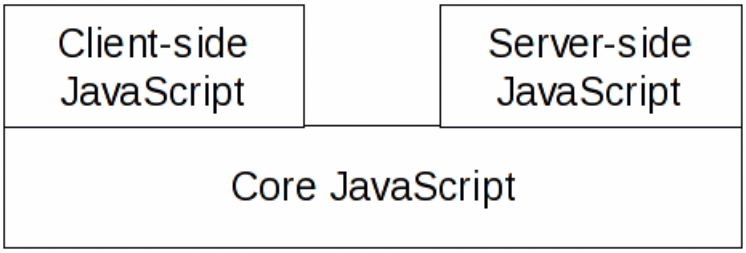
JavaScript heeft de volgende algemene kenmerken:

- Objectgeoriënteerd.
- Platformonafhankelijk.
- Geschikt voor *client*- en *server*-toepassingen.

Het internet is gebaseerd op een *client server*-technologie. *JavaScript* kan hierbij zowel op de *servers* als op de *clients* gebruikt worden.

JavaScript in deze internetomgeving kan onderverdeeld worden in drie delen:

1. *Core JavaScript.*
2. *Client Side JavaScript.*
3. *Server Side JavaScript.*



Afbeelding 1.1: JavaScript in verschillende omgevingen

Core JavaScript wordt doorgaans gebruikt in een of andere gastomgeving. Dat kan een internetbrowser of een internetserver zijn. Maar ook andere toepassingen maken gebruik van *JavaScript*. Zo kan men in de Windows-omgeving de aloude *batch*-bestanden vervangen door scriptbestanden die veel meer mogelijkheden bieden. *Adobe* Acrobat ondersteunt inmiddels ook *JavaScript* in PDF-bestanden.

Bij al deze gastomgevingen wordt gebruikt gemaakt van *Core JavaScript* gecombineerd met een aantal objecten en functies die eigen zijn aan de omgeving.

1.2 Core JavaScript

Core JavaScript vormt de kern van *JavaScript* en bevat de kernelementen van de scripttaal. Ze is gemeenschappelijk voor alle gastomgevingen waarin *JavaScript* gebruikt wordt.

Core JavaScript bestaat uit de volgende onderdelen:

1. **Keywords**: Dit zijn voorgedefinieerde woorden van de programmeertaal.
2. **Syntax van** *statements*: de programmaregels van *JavaScript* volgen een bepaalde syntax en grammatica. Deze worden gedefinieerd in *Core JavaScript*.
3. **Regels voor uitdrukkingen, variabelen en *literals***: *Core JavaScript* bepaalt de syntax van uitdrukkingen, de declaratie en gebruik van variabelen en het gebruik van *literals*.
4. **Het objectmodel**: *Core JavaScript* beschikt over een standaardobjectmodel.
5. **Voorgedefinieerde objecten en functies**: *Core JavaScript* stelt een aantal standaardobjecten en functies ter beschikking die kunnen gebruikt worden in zowel *Client Side JavaScript* als *Server Side JavaScript*.

1.3 Client Side JavaScript

Het internet is gebaseerd op een *client-server*-model. De *client* is in dit geval de internet-browser en de server is de webserver op het internet of intranet. De *client* vraagt aan de webserver een bepaalde webpagina aan die hij vervolgens aan de gebruiker toont. Deze pagina's zijn opgemaakt in HTML[1]. De browser interpreteert de HTML-opmaakcodes en genereert een document met de juiste lay-out.

HTML bepaalt slechts de opmaak van het document. Het is niet mogelijk om extra's toe te voegen zoals het reageren op muisbewegingen, het indrukken van muisknoppen en toetsen, het controleren van de invoer van de gebruiker enzovoort.

Om HTML uit te breiden met deze mogelijkheden maakt men gebruik van een scripttaal. Deze scripttaal is geïntegreerd in de HTML-code. Enkel browsers die deze scripttaal ondersteunen, zullen ze interpreteren en de juiste instructies uitvoeren. Andere browsers worden geacht het script te negeren. De combinatie HTML – *JavaScript* – *Cascading StyleSheets* (CSS) noemt men ook wel *Dynamic HTML (DHTML).*

De verschillende elementen vormen een drielagenmodel waarbij iedere laag zijn eigen verantwoordelijkheid heeft:

1. **HTML**: is verantwoordelijk voor de structuur van de inhoud.
2. **CSS**: is verantwoordelijk voor de vormgeving van de inhoud.
3. **JavaScript**: is verantwoordelijk voor het dynamische gedrag.

De meest gekende scripttalen zijn *JavaScript* en *Visual Basic Script*. *JavaScript* is echter de meest gebruikte *Client-Side*-scripttaal. Ze wordt door de meeste browsers ondersteund. *Visual Basic Script* wordt enkel ondersteund door *Microsoft*-producten.

Aangezien er op het internet een grote verscheidenheid is aan browsers, is het daarom aangewezen om *JavaScript* te gebruiken.

Client Side JavaScript is volledig gebaseerd op *Core JavaScript*. Hieraan worden een aantal standaardobjecten en functies toegevoegd die specifiek zijn voor de omgeving van een browser.

Hoewel *JavaScript* zelf gebaseerd is op een min of meer gestandaardiseerde scripttaal, geldt dit niet altijd voor de objecten en functies die aan de zijde van de *client* ter beschikking worden gesteld. Dit maakt het programmeren in *Client Side JavaScript* soms tot een frustrerende onderneming. Scripts die werken onder *Firefox* werken niet altijd onder *Internet Explorer* en omgekeerd. Vaak moet men op basis van het type browser verschillende scripts schrijven.

In een intranetomgeving is dit minder problematisch omdat daar meestal een grotere

1 HTML is de afkorting van *HyperText Markup Language*.

uniformiteit is in het gebruik van het type browser.

Gelukkig werd het objectmodel van de browsers in de loop van de tijd gedeeltelijk gestandaardiseerd door *W3C* en heet DOM: ***Document Object Model***. We spreken van *DOM level 1*, *DOM level 2* en *DOM level 3*. Voor HTML-documenten bestaat de bijzondere versie HTML-DOM. Browsers die DOM ondersteunen kunnen met *JavaScript* op dezelfde wijze werken.

In deze cursus zullen we ons houden aan de specificaties van HTML-DOM en ECMAScript (en CSS).

1.4 Server Side JavaScript

HTML-pagina's kunnen dus *Client Side JavaScript* code bevatten. Daarnaast is het ook mogelijk om *Server Side JavaScript* code te integreren in het HTML-document. In dit geval wordt deze *JavaScript* code niet doorgestuurd naar de browser maar wordt ze eerst door de webserver geïnterpreteerd. De webserver kan op basis van deze code bepaalde acties ondernemen. Zo kan hij gegevens uit een databank halen om weer te geven in de HTML-pagina. Op die manier worden dynamische HTML-pagina's gegenereerd.

Server Side JavaScript wordt bijvoorbeeld gebruikt bij *Node.js* (www.nodejs.org).

Server Side JavaScript valt buiten het bestek van deze cursus.

1.5 JavaScript versus Java

Java en *JavaScript* worden vaak verward. Het zijn wel degelijk twee verschillende programmeertalen die ook wel een aantal dingen gemeenschappelijk hebben.

Java is een platformonafhankelijke programmeertaal die destijds ontwikkeld werd door *SUN*. De enorme populariteit van Java heeft *Netscape* ertoe gebracht om hun scripttaal te noemen naar Java, vandaar de naam *JavaScript.*

De overeenkomst is echter niet beperkt tot het woord *Java*. *JavaScript* lijkt in de syntax heel sterk op Java. In feite is *JavaScript* gebaseerd op Java, wat op zijn beurt afgeleid is van de programmeertaal C++.

JavaScript code vertoont dus heel veel gelijkenissen met Java code maar toch zijn er ook belangrijke verschillen:

1. *JavaScript* is een **geïnterpreteerde** programmeertaal. Dit wil zeggen dat de broncode stap voor stap wordt geïnterpreteerd en uitgevoerd. In het geval van *Client Side JavaScript* is het de browser die de *JavaScript*-code interpreteert en uitvoert. Geïnterpreteerde programmeertalen zijn doorgaans trager omdat de interpretatie op het moment zelf gebeurt en dit vraagt enige processortijd.
   Java daarentegen is zowel een gecompileerde als een geïnterpreteerde programmeertaal. De broncode wordt vooraf gecompileerd tot *bytecode* die ten slotte geïnterpreteerd wordt. Dit maakt Java tot een relatief snelle en tevens platformonafhankelijke programmeertaal.

2. Zowel Java als *JavaScript* zijn objectgeoriënteerde programmeertalen. Java is echter op **klassen gebaseerd** terwijl *JavaScript* op **prototypes gebaseerd** is. Dit zal later duidelijk worden in deze cursus.

3. *JavaScript* maakt gebruik van **dynamische datatypen.** Dit wil zeggen dat het datatype van variabelen niet expliciet gedeclareerd wordt. De interpreter bepaalt op dynamische wijze het juiste datatype. In Java moeten datatypes strikt gedefinieerd

worden.

JavaScript werd een tijdje stiefmoederlijk behandeld als een taal waarmee men enkel wat dynamisch gedrag aan een website kon toevoegen. Inmiddels wordt de taal geherwaardeerd en vormt ze de kern van bibliotheken en *frameworks* zoals *AngularJS, JQuery, NodeJS* en vele andere. Er worden momenteel heel wat krachtige toepassingen met *JavaScript* gemaakt, dit zowel aan de kant van de browser als aan de kant van de server. Ontwikkelaars die dergelijke toepassingen maken in de browser worden sinds kort *Frontend Developers* genoemd.

1.6 Mijn eerste script

In deze paragraaf gaan we ons eerste script in *JavaScript* schrijven. Vooreerst moeten we een aantal benodigdheden op onze computer installeren. Voor deze cursus gaan we uit van een Windows-omgeving. Maar uiteraard kan *JavaScript* ook in andere omgevingen gebruikt worden (Linux, Apple etc...).

1.6.1 Benodigdheden

Alvorens ons eerste script te schrijven moeten we dus over een aantal werkmiddelen beschikken: een browser, een HTML-editor en de *JavaScript*-documentatie.

1.6.1.1 Internetbrowser

Deze cursus behandelt *Core JavaScript* en *Client Side JavaScript*. Daarom zullen we voor de praktische oefeningen steeds werken met *Client Side JavaScript*. De omgeving die we daarvoor nodig hebben, is een webbrowser die *JavaScript* ondersteunt. De meest gebruikte browsers op het internet zijn *Google Chrome, Firefox* en *Microsoft Internet Explorer* of *Microsof Edge.* In mindere mate komen ook *Opera* en *Safari (Apple*) voor. Voor de meest recente statistieken kan je een kijkje nemen op http://www.w3schools.com/browsers/browsers_stats.asp.

Gezien de verschillen in het *Object Model* tussen deze browsers is het aangewezen er meerdere te installeren en alles uit te testen op de verschillende browsers.

Ontwikkelaars maken doorgaans gebruik van *Firefox* omwille van de krachtige *tools* voor het ontwikkelen van *JavaScript*-toepassingen. Ook wij zullen in deze cursus *Firefox* gebruiken als ontwikkelomgeving.

Opdracht 1: De browser installeren
In deze opdracht gaan we de nodige browsers installeren. Naargelang de omgeving is er doorgaans een bepaalde browser standaard aanwezig:

- *Windows*: *Microsoft Internet Explorer* of *Microsoft Edge*
- *Linux: Firefox*
- *Apple:* Safari

- Installeer de laatste versie van *Firefox:* www.firefox.com
- Optioneel: installeer de laatste versie van *Chrome*: www.google.com

1.6.1.2 De JavaScript-documentatie

Om te programmeren in *JavaScript* moet je onder andere goede documentatie ter beschikking hebben.

De documentatie voor *Core JavaScript* vinden we op :

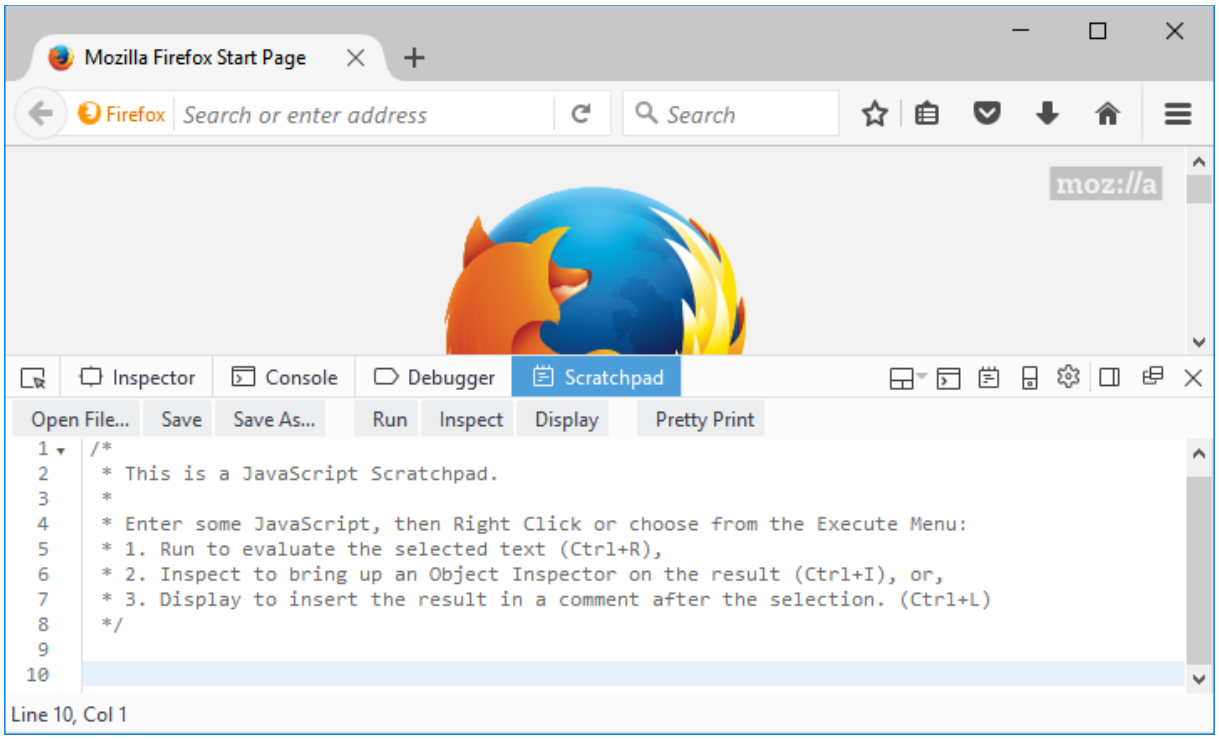
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

Tevens vinden we een goed overzicht van *JavaScript,* HTML en CSS bij
http://www.w3schools.com

1.6.1.3 JavaScript-ontwikkelomgeving

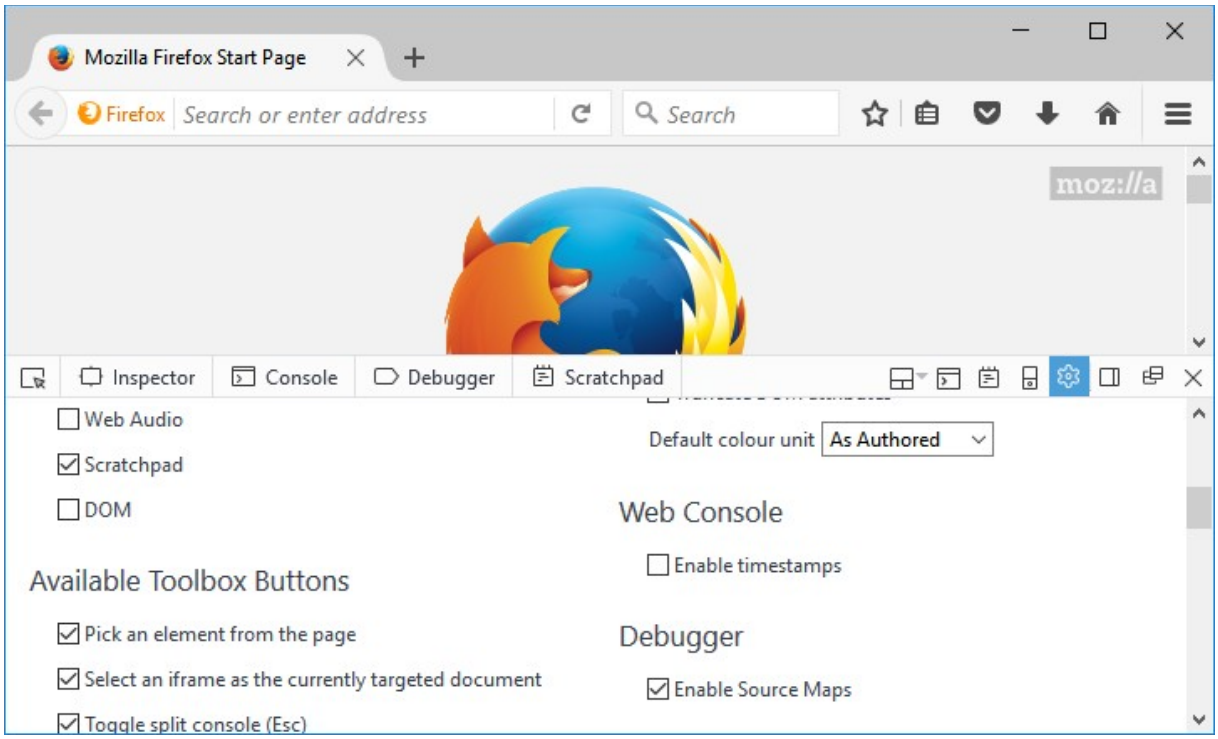
JavaScript is meestal ingebed in een gastomgeving: doorgaans is dat de browser, een internetserver of een andere toepassing zoals *Adobe Acrobat*. Aangezien deze cursus vooral gericht is naar het gebruik van *JavaScript* binnen een browser, gaan we van in het begin ook al gebruikmaken van de browser als ontwikkelomgeving. *Firefox* leent zicht daar bijzonder goed toe omwille van de uitgebreide ontwikkelhulpmiddelen. Zo heeft *Firefox* een debugger, een consolevenster en een kladblok (*scratchpad*) waarin we makkelijk *JavaScript*-bestanden kunnen aanmaken, uitvoeren, bewaren en opnieuw inladen.

Deze hulpmiddelen kunnen eenvoudig geopend worden met de toets F12.



De inhoud van het ontwikkelvenster kunnen we aanpassen via het icoontje .

We kunnen hier de tabbladen selecteren die we willen zien in het ontwikkelvenster. *Scratchpad* of kladblok moeten we eventueel nog toevoegen om het zichtbaar te maken.



1.6.2 Hello World!

Het is een aloude traditie om bij het leren van een nieuwe programmeertaal het eerste programma de woorden "*Hello World!"* op het scherm te laten verschijnen. We willen ons uiteraard bij deze traditie aansluiten. Ons eerste script zal daarom deze legendarische woorden tonen.

Via *Scratchpad* in *Firefox* kunnen we makkelijk een script aanmaken en uitvoeren, zonder dat we dit script reeds moeten opnemen in een HTML-pagina.
Tevens kunnen we het script opslaan op ons systeem zodat we het later opnieuw kunnen inladen en uitvoeren.

1.6.2.1 Het script aanmaken

Ons eerste script ziet er als volgt uit:

```
console.log("Hello World!");
```

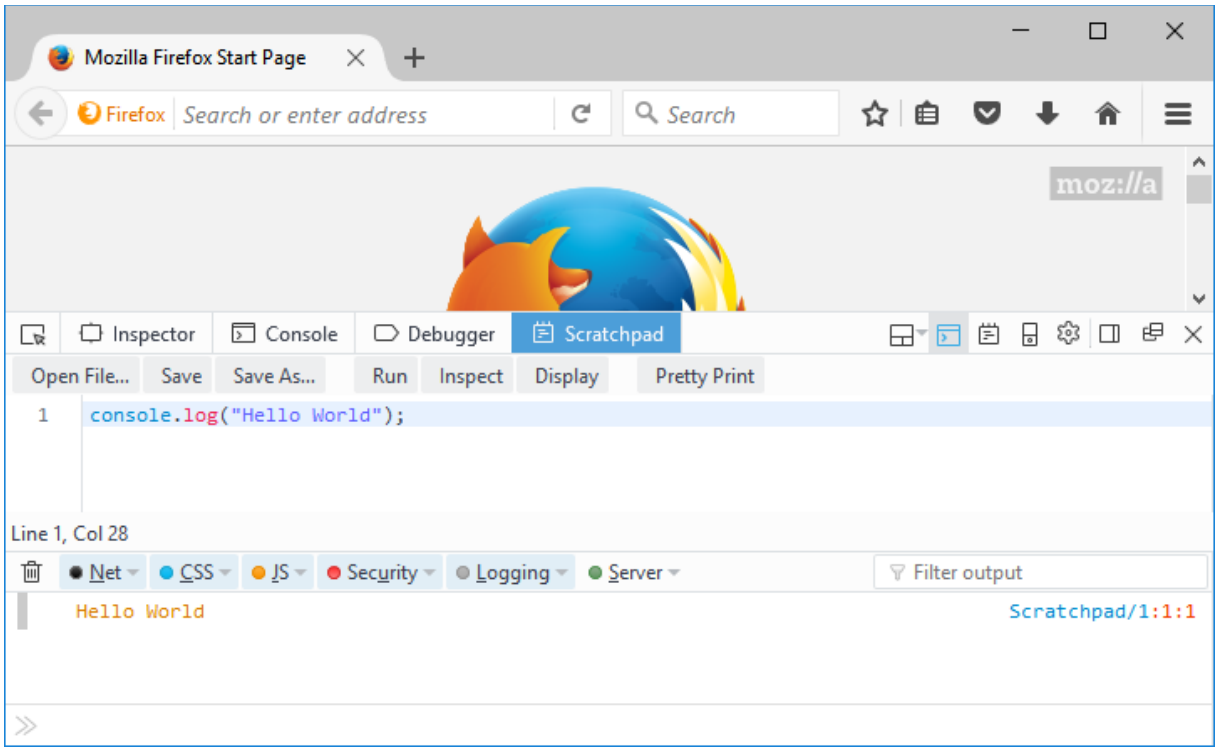
Met deze scriptregel wordt de tekst *Hello World!* naar de console weggeschreven. De console is een extra venster in de browser dat enkel bedoeld is voor ontwikkelaars en is beschikbaar op de meeste browsers. We kunnen hier boodschappen naartoe schrijven.

We kunnen dit bestand wegschrijven door op de knop ***Save*** te klikken. *JavaScript*-bestanden hebben als extensie **.js**.

We kunnen het bestand opnieuw inladen via de knop ***Open File***.

1.6.2.2 Het script uitvoeren

We kunnen het script uitvoeren door op de knop ***Run*** te klikken.



Het resultaat verschijnt in het consolevenster. Om zowel de code als de uitvoer te zien, kunnen we het consolevenster onder het werkvenster positioneren. We gebruiken hiervoor het icoontje met de vermelding *Toggle split console*.

Opdracht 2: Mijn eerste script

- Open *Firefox* en open de ontwikkelomgeving door op F12 te drukken. Pas eventueel deze omgeving aan zodat je het *Scratchpad* te zien krijgt. Positioneer het consolevenster onder het *Scratchpad*.
- Voeg de volgende regel code toe:

```
console.log("Hello World!");
```

- Voer het script uit via de knop ***Run.***
- Bewaar het script via de knop ***Save***. Geef het script de naam ***HelloWorld.js***.

1.7 Samenvatting

In dit hoofdstuk hebben we kennisgemaakt met *JavaScript* als geïnterpreteerde programmeertaal die vooral in een internetomgeving gebruikt wordt. We hebben gezien dat er een onderscheid is tussen de kern van de taal (*Core JavaScript)* en de concrete toepassing ervan in een bepaalde gastomgeving: *Client Side JavaScript* of *Server Side JavaScript*.

Om *Client Side JavaScript* uit te voeren hebben we een internetbrowser nodig. Door de vele verschillen tussen de browsers is het nodige verschillende versies te installeren om de scripts uit te testen.

Ten slotte hebben we ons eerste script aangemaakt en uitgevoerd.

Hoofdstuk 2: Core JavaScript

2.1 Inleiding

In dit hoofdstuk beschrijven we de basiselementen van *Core JavaScript*. We gaan hierbij de *JavaScript*-syntax systematisch onder de loep nemen. We zullen ons hier richten op versie 6 van *JavaScript*, ook wel *ECMAScript 6 (ES6)* of *ECMAScript 2015* genoemd.

Core JavaScript staat volledig los van de gastomgeving waarin *JavaScript* gebruikt wordt. Maar om dergelijke code uit te voeren, hebben we doorgaans toch een gastomgeving nodig. Deze gastomgeving zal de browser zijn.

Core JavaScript heeft zelf geen mogelijkheden voor communicatie met de gebruiker: de invoer van gegevens en het tonen van resultaten. Hiervoor dienen we gebruik te maken van de mogelijkheden van de concrete gastomgeving. Bij de browser is dat het consolevenster en de dialoogvensters. Deze invoer en uitvoer is geen *Core JavaScript* meer maar maakt het ontwikkelen wel interactiever. Het gevolg is evenwel dat de ontwikkelde scripts enkel kunnen werken binnen een browseromgeving. Bij andere gastomgevingen zal er dus een alternatief voor deze invoer en uitvoer gezocht moeten worden.

2.2 Commentaar

Het is een goede programmeertechniek om je broncode te doorspekken met heel wat commentaar. Zo kan je zelf achteraf beter achterhalen wat die code nu ook al weer betekent en bovendien maak je het je collega's heel wat makkelijker als zij aanpassingen moeten doen aan jouw code terwijl jij op de Canarische Eilanden ligt.

JavaScript kent twee soorten commentaar:

1. **`/* commentaar */`**
   Dit is de standaardcommentaar zoals we die ook kennen in Java en C/C++. Alles tussen **`/*`** en **`*/`** wordt genegeerd door de interpreter. Deze methode wordt vooral gebruikt bij langere blokken commentaar, verspreid over meerdere tekstregels.

2. **`// commentaar`**
   In dit geval negeert de interpreter alles wat achter `//` komt tot aan het einde van de regel. Deze methode wordt veel gebruikt bij korte stukjes commentaar achter een programmaregel.

```
/* dit is een blok commentaar
verspreid over meerdere regels. */
console.log("Hello World!"); // dit is een regel commentaar
```

2.3 Literals en variabelen

In een programma wordt er gewerkt met bepaalde gegevens. De gegevens worden in een programma voorgesteld door middel van *literals* en variabelen.

2.3.1 Literals

Literals zijn letterlijke waarden die we opnemen in de scriptcode. We kunnen ze onder andere gebruiken om waarden toe te kennen aan variabelen.
In ons eerste script hebben we een *literal* gebruikt om de tekst *"Hello World"* op het scherm te brengen. De tekst tussen aanhalingstekens is in dit geval een string-*literal.* We brengen deze tekst letterlijk op het scherm.

We onderscheiden de volgende soorten *literals:*

Gehele getallen	
`1234`	Decimaal getal 1234.
`0b``101` **`0B`**`101`	Binair getal 101 (= 5 decimaal). Binaire getallen hebben als grondtal 2. De notatie begint met `0b` of `0B`.
`0``35` **`0o`**`35` **`0O`**`35`	Octaal getal 35 (= 29 decimaal). Octale getallen hebben als grondtal 8. De notatie begint met een `0, 0o` of `0O`.
`0x``22` **`0X`**`22`	Hexadecimaal getal 22 (= 54 decimaal). Hexadecimale getallen hebben als grondtal 16. De notatie begint met `0x` of `0X`.
Reële getallen	
`34.235`	Getal met kommanotatie. Met gebruikt hiervoor steeds een punt.
`34235E-3`	Getal met wetenschappelijke notatie. 34235 is de *mantisse* en -3 de *exponent*. 34235 x 10^{-3} = 34.235.
Strings of tekenreeksen	
`"Hello World"` `'Hello World'`	Reeks van karakters tussen enkelvoudige of dubbele aanhalingstekens.
`` `Hello World` ``	Reeks van karakters tussen *accent graves* (*backticks*). Deze tekenreeks kan verspreid zijn over meerdere regels en kan sjabloonvariabelen (*template variables*) bevatten.
Voorgedefinieerde waarden	
`true`	Booleaanse waarde ‘waar’.
`false`	Booleaanse waarde ‘vals’.
`NaN`	*Not a Number* : geeft aan dat een waarde geen getal is.
`undefined`	Ongedefinieerd: onbepaalde waarde.
`null / NULL`	Niets: referentie naar niets (geen enkel object).

Tabel 2.1: Literals in JavaScript

Strings of tekenreeksen kunnen ook speciale tekens bevatten. Deze worden voorafgegaan door het teken `\` (*backslash*).

De speciale tekens zijn weergegeven in de volgende tabel:

Teken	***Betekenis***
`\b`	*Backspace*
`\f`	*Formfeed*
`\n`	*New line*
`\r`	*Carriage return*
`\t`	*Tabulator*
`\'`	Enkelvoudig aanhalingsteken gebruikt in de string zelf.
`\"`	Dubbel aanhalingsteken gebruikt in de string zelf.

Teken	*Betekenis*
`\\`	Het *backslash*-karakter zelf.
`\XXX`	De octale waarde van een karakter in de *Latin-1* codering aangeduid door drie octale cijfers.
`\0xXX`	De hexadecimale waarde van een karakter in de *Latin-1* codering aangeduid door twee hexadecimale cijfers.
`\uXXXX`	Een UNICODE-karakter aangeduid door vier hexadecimale cijfers.

Tabel 2.2: Speciale tekens in tekenreeksen

UNICODE is een universele standaard om karakters uit verschillende talen weer te geven. In tegenstelling tot ASCII werkt UNICODE met 16-bits-karakters en kan daardoor duizenden verschillende letters bevatten. Voor meer informatie over de UNICODE-standaard kan je terecht op de volgende website: www.unicode.org.

2.3.2 Variabelen

Een variabele is een plek in het geheugen waarin men een waarde kan bewaren. Deze plek heeft een naam zodat men elders deze plek met de naam kan aanduiden en zo de waarde kan gebruiken. Een variabele is dus een gegevenseenheid met een naam.

Een variabele heeft de volgende kenmerken:

1. Een **naam**.
2. Een **datatype**.
3. Een **bereik**.

We bekijken achtereenvolgens de naam, de declaratie, het datatype en het bereik van variabelen.

2.3.2.1 De naam van een variabele

De naam van een variabele bestaat uit een aaneengesloten reeks van letters en cijfers. De naam moet steeds **beginnen** met een **letter,** het **$**-teken of het **_**-teken (*underscore*).

Voorbeeld: `value, number1, number2, _text, $value`

De naam van een variabele mag **geen gereserveerd woord** zijn. Gereserveerde woorden zijn onder andere programmeerwoorden van de scripttaal, voorgedefinieerde waarden enzovoort.

JavaScript is **hoofdlettergevoelig**: `value` en `Value` zijn daarom twee verschillende variabelen.

Het is aangewezen een variabele een naam te geven die overeenkomt met de betekenis van die variabele. Dit maakt de scriptcode veel beter leesbaar en onderhoudbaar.

Doorgaans wordt de *CamelCase*-notatie aanbevolen: dit wil zeggen dat de naam van een variabele steeds in kleine letters is maar dat bij samengestelde woorden, ieder nieuw woord (behalve het eerste) begint met een hoofdletter.

Bijvoorbeeld:

`first`**`N`**`ame`

```
lastName
myFirstCar
```

2.3.2.2 De declaratie van variabelen

Alvorens variabelen gebruikt kunnen worden, moeten ze gedeclareerd worden. Variabelen kunnen op drie manieren gedeclareerd worden in een script.

De meest aangewezen manier om een variabele te declareren is met het sleutelwoord **`let`**:

```
let number = 5;
```

Hier declareren we een geheugenlocatie met de naam *number* en we vullen deze geheugenlocatie onmiddellijk met de letterlijke waarde 5.

Vervolgens kunnen we de inhoud van deze geheugenlocatie elders gebruiken:

```
let number = 5;
console.log(number);
```

Hier wordt de waarde van de variabele *number* in de console afgedrukt.

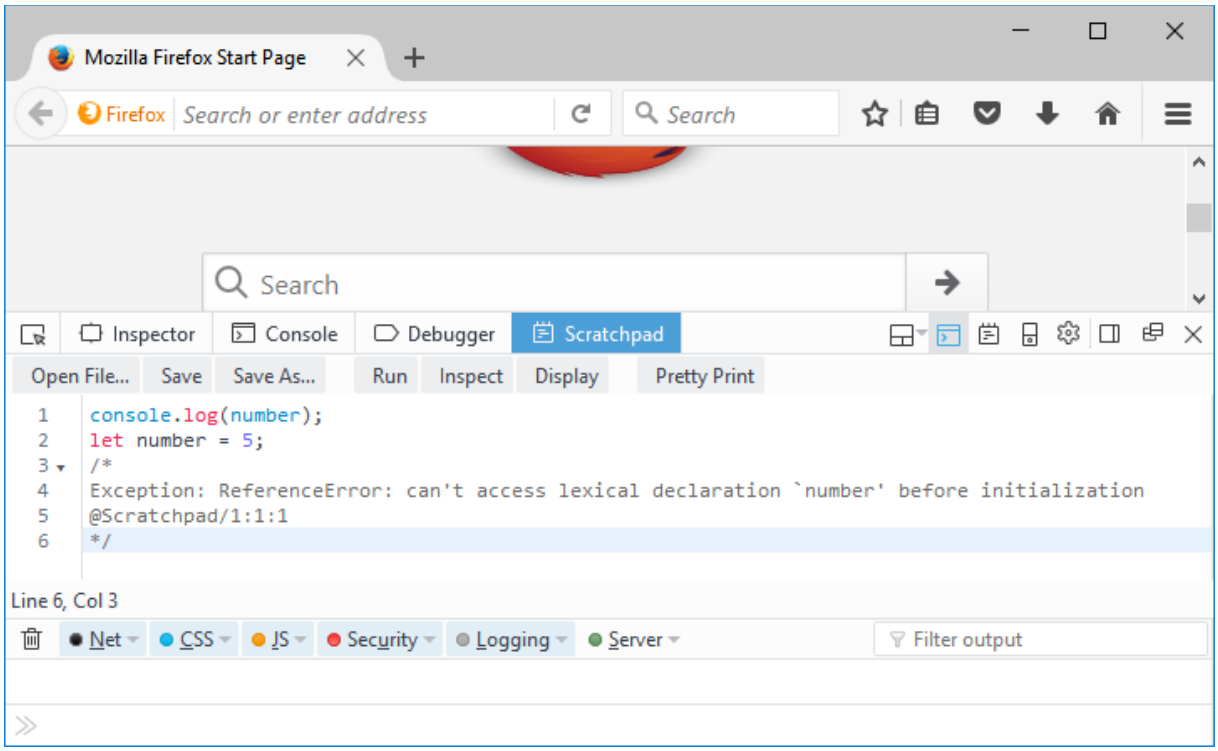
Indien we de variabele niet onmiddellijk invullen tijdens de declaratie, zal deze tijdelijk de voorgedefinieerde waarde `undefined` krijgen totdat we deze wel expliciet invullen.

```
let number;
console.log(number); // undefined
number = 5;
console.log(number); // 5
```

Indien we een variabele toch willen gebruiken voordat hij gedeclareerd is, krijgen we een foutmelding:

```
console.log(number); // ReferenceError
let number = 5;
```

Foutmeldingen worden in *Scratchpad* van *Firefox* weergegeven in commentaarblokken:



We kunnen deze commentaar nadien gewoon verwijderen. Om voorheen gedefinieerde waarden te verwijderen doen we best een *refresh* van de browser. Zo beginnen we telkens met een schone lei bij een script.

Indien de variabele een waarde bevat die na de initiële toekenning niet meer mag wijzigen, kunnen we gebruikmaken van **`const`**. Het gaat hier om constanten.

```
const PI= 3.14;
```

Indien we deze variabele nadien toch wijzigen, krijgen we een foutmelding:

```
const PI= 3.14;
PI = 3.15; // SyntaxError
```

We kunnen ook gebruikmaken van `var`. Dit sleutelwoord heeft ongeveer dezelfde betekenis als `let`. Het subtiele verschil zal duidelijk worden zodra we het hebben over de *scope* van een variabele.

```
var number = 5;
```

`let` en **`const`** zijn nieuwigheden in *JavaScript 6*. Voorheen bestond enkel **`var`**. Dat maakt dat in vele scripts vooral `var` gebruikt wordt. Bij nieuwe scripts is het evenwel aanbevolen telkens `let` en `const` te gebruiken, en dit omwille van enkele nadelen van `var` die we later zullen uitleggen.

Verder vermelden we nog dat je een variabele ook impliciet kan declareren door er gewoon een waarde aan toe te kennen:

```
number = 5;
```

In feite declareren we hier een globale variabele. Het gebruik van dergelijke globale variabelen is evenwel niet aanbevolen. Het is daarom beter iedere variabele eerst expliciet te declareren met `let`, `const` of `var`.

De waarde die we toekennen aan een variabele hoeft ten slotte niet altijd een letterlijke waarde te zijn. We kunnen ook gewoon de waarde van een andere variabele gebruiken:

```
let number1 = 6;         // Toekenning door middel van een literal
let number2 = number1;   // Toekenning door middel van een variabele
```

We kunnen meerdere variabelen op dezelfde regel declareren. De variabelen worden dan gescheiden door een komma:

```
let number1 = 6, number2 = 8;
```

Het is evenwel minder overzichtelijk en daarom ook minder gebruikelijk.

We vatten het nog even samen: variabelen worden steeds gedeclareerd met het woord `let`. Indien de waarde niet meer mag veranderen gebruiken we `const`.

2.3.2.3 Het datatype van variabelen

Een variabele bevat gegevens van een bepaald type. Dit noemt men het datatype. We kunnen de datatypes in *JavaScript* onderverdelen in primitieve datatypes en het referentietype.

2.3.2.3.1. Het primitieve datatype

Het primitieve datatype bevat een enkelvoudige waarde van een bepaald type. *JavaScript* kent volgende soorten primitieve datatypes:

- `Number`: alle getallen, zowel met als zonder komma.
- `String`: tekenreeks zoals "Hello World".
- `Boolean`: logische waarde: `true` en `false`.
- `null`: een lege waarde, 'niets'.
- `undefined`: een onbepaalde waarde. Niet-geïnitialiseerde variabelen krijgen automatisch deze waarde.
- `NaN`: een waarde die geen getal is. *Not A Number.*
- `Symbol`: een unieke en onveranderlijke waarde (nieuw in *JavaScript 6*). Een dergelijke waarde wordt aangemaakt met de *factory*-functie `Symbol()`. We komen later nog terug op het nut en gebruik van dit datatype.

We kunnen deze datatypes als volgt gebruiken:

```
let aNumber = 5;
let aString = "Hello World";
let aBoolean = true;
let somethingEmpty = null;
let somethingUndefined= undefined;
let noNumber = NaN;
let symbol = Symbol("mysymbol");
```

Bij de definitie van een string kan de tekst lang zijn. In dat geval kunnen we een nieuwe regel

beginnen om het geheel beter leesbaar te maken. We moeten dit op het einde van iedere regel dan wel aangeven met het teken `\` zodat de interpreter op de hoogte is dat er geen nieuwe programmeerregel volgt.

Een voorbeeld:

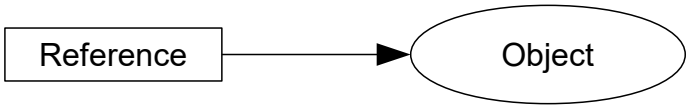
```
let longText = "Lorem ipsum dolor sit amet,\
consectetur adipiscing elit. Ut dolor dolor,\
bibendum id ex eget, molestie hendrerit nunc.\
Nunc ac augue sed nulla pellentesque molestie.";
```

2.3.2.3.2. Het referentietype

Het referentietype is een verwijzing of referentie naar een object in het geheugen. Een object is een uitgebreide datastructuur in het geheugen. En daarin ligt het onderscheid met primitieve datatypes die slechts een "enkelvoudige" waarde bevatten.

In de volgende code wordt een nieuw object gecreëerd waarin de gegevens van een persoon zitten. De referentievariabele `person` is een verwijzing naar dit object.

```
let person = new Person("Homer","Simpson");
```



Afbeelding 2.1: Een referentie naar een object

In het verdere verloop van de cursus zullen we uitvoerig aandacht besteden aan het gebruik van het referentietype. Voorlopig volstaat het te weten dat een referentie naar een object niet hetzelfde is als een primitief datatype.

2.3.3 *Template literals* en *multi-line strings*

Sinds *JavaScript 6* kunnen we tekenreeksen of strings ook definiëren tussen *accent graves (*`). We noemen dit een *template literal* omdat deze strings een sjabloon (*template*) vormen met variabele waarden. We kunnen in de tekenreeks de inhoud van een variabele eenvoudig opnemen door deze te plaatsen tussen de volgende tekens: `${}`.

We illustreren dit met een voorbeeld:

```
let name = "World";
let text = `Hello ${name}`;
console.log(text);  // Hello World
```

De inhoud van de variabele `name` wordt ingeplakt in de tekenreeks.

Op deze manier is het veel makkelijker tekenreeksen samen te stellen die bestaan uit vaste en veranderlijke gegevens.

De notatie met *accent graves* heeft een tweede voordeel: we kunnen tekenreeksen definiëren waarin het teken voor een nieuw regel (`\n`) automatisch opgenomen wordt telkens we in de definitie een nieuwe regel beginnen. We noemen dit *multi-line strings*.

```
let longText = `Lorem ipsum dolor sit amet,
consectetur adipiscing elit. Ut dolor dolor, bibendum id ex eget,
molestie hendrerit nunc. Nunc ac augue sed nulla pellentesque
molestie.`;
console.log(longText);
```

Dit geeft als resultaat:

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Ut dolor dolor, bibendum id ex eget, molestie hendrerit nunc.
Nunc ac augue sed nulla pellentesque molestie.
```

Het equivalent met aanhalingstekens ziet er als volgt uit:

```
let longText = "Lorem ipsum dolor sit amet,\nconsectetur adipiscing
elit. Ut dolor dolor, bibendum id ex eget,\nmolestie hendrerit
nunc. Nunc ac augue sed nulla pellentesque\nmolestie.";
console.log(longText);
```

2.3.4 Typeconversie

In tegenstelling tot de meeste andere programmeertalen moet het datatype van een variabele in *JavaScript* niet vooraf vastgelegd worden. De interpreter beslist op het moment zelf om welk datatype het gaat en zal de nodige conversies doen. *JavaScript* gebruikt dynamische datatypen. Dit maakt het de programmeur makkelijker maar de uitvoering van het programma is daardoor wel trager. De interpreter moet telkens nagaan om welk datatype het gaat.

Het gevolg is dat we aan een variabele achtereenvolgens gegevens van verschillende datatypes kunnen toekennen:

```
let variable = 5;
let variable = "Hello World";
let variable = true;
let variable = null;
```

Soms wordt er wel een bepaald datatype verwacht, bijvoorbeeld omdat er een berekening moet gebeuren met een getal of omdat er een beslissing genomen moet worden op basis van een voorwaarde die waar of vals is. In dat geval zal de interpreter de nodige conversies doen. Het ene datatype wordt daarbij automatisch omgezet naar een ander datatype. Voor dergelijke conversies hanteert de interpreter bepaalde regels die worden weergegeven in de onderstaande tabel: