

Grammatica en datatypen

Syntaxis

JavaScript – afgekort als JS – leent zijn **syntaxis** (*Eng. syntax*) uit de programmeertalen C, C++ en Java, met bepaalde invloeden uit Awk, Perl en Python. JS is **hoofdlettergevoelig** (*Eng. case-sensitive*) en gebruikt de **Unicode** [☑](#)-**tekenreeks** (*Eng. character set, tekens, leestekens en symbolen*). Bijvoorbeeld het woord Noël (wat “Kerstmis” betekent in het Frans) kan gebruikt worden als naam van een variabele.

```
1 const Noël = "Kerstmis";
```

Maar de variabele `noël` is niet hetzelfde als `Noël`, omdat JavaScript hoofdlettergevoelig is.

Het identificeren van keywords, variabelen, eigenschappen en labels moet gebeuren via een unieke naam. Deze namen fungeren als unieke identifiers. Een identifier moet starten met een letter, underscore (`_`) of een dollar teken `$`. De tekens hierna kunnen ook een getal zijn. Gereserveerde identifiers (*Eng. keywords*) zoals: `var`, `let`, `const`, `function` ... kunnen niet als eigen identifier gebruikt worden. Extra **witruimte** (*Eng. white space*) wordt genegeerd door JavaScript, maar maakt de code leesbaarder. **Spatie** (*Eng. space*), tabs en **nieuwe lijn** (*Eng. newline*) tekens worden gezien als witruimte. Tekst in een string kan verdergezet worden op een volgende regel met een backslash `\`.

```
1 alert ( "Deze zin\  
2   loopt door");
```

Unicode is een internationale standaard voor de codering van grafische tekens en symbolen in binaire codes, vergelijkbaar met de ASCII-standaard. Unicode voorziet alle tekens van alle geschreven talen van een naam (in de standaard in hoofdletters geschreven) en een nummer (hexadecimale waarde voorafgegaan door `U+`).

Verzameling van Unicode tekens:

Naam	Unicode	Rendering
AMPERSAND	U+26	&

Naam	Unicode	Rendering
COPYRIGHT	U+A9	©
GREEK CAPITAL LETTER OMEGA	U+3A9	Ω
PILE OF POO	U+1F4A9	
SMILING FACE WITH OPEN MOUTH	U+1F603	😊
ALIEN MONSTER	U+1F47E	
THUMBS UP	U+1F44D	
SUN WITH FACE	U+1F31E	

In JavaScript zijn **instructies** (*Eng. statements*) commando's die door een "JavaScript engine" (een computerprogramma) worden uitgevoerd. Alle moderne webbrowsers bevatten zo'n JS-engine. De meest bekende is de "**Chrome V8**" engine. JavaScript-code is een opeenvolging of **sequentie** (*Eng. sequence*) van instructies, uitgevoerd door onder andere een webbrowser in een welbepaalde volgorde (van boven naar beneden, van links naar rechts).

Instructies worden bij voorkeur gescheiden door een **puntkomma** (*Eng. semi-colon*) (;). Dit is echter niet noodzakelijk indien slechts één statement op één regel wordt geschreven. Indien meer dan één statement op dezelfde regel wordt geschreven, dan moeten deze gescheiden worden door een puntkomma. ECMAScript heeft regels om automatische puntkomma's te injecteren in de code, maar om de kans tot een **fout** (*Eng. bug*) te vermijden is het aan te raden om steeds puntkomma's te gebruiken.

Code blocks bevatten gegroepeerde statements, die samen uitgevoerd zullen worden. Een code block start met een **linker accolade** (*Eng. left curly bracket*, {) en eindigt met een **rechter accolade** (*Eng. right curly bracket*, }).


Een **functie** (*Eng. function*) is een van de meest gekende codeblokken binnen JavaScript.

Camel Case

Er bestaan verschillende soorten afspraken voor de **naamgeving** van identifiers, zoals: Kebab case, Snake Case, Camel case ...

Kebab case gebruikt een **koppelteken** (*Eng., dash*) tussen elk woord en alle tekst wordt geschreven met **kleine letters** (*Eng., lower case*). Kebab case is ook gekend als Dash case of Hyphen case.

```
1  var first-name = "Philippe";
2  var sur-name = "De Pauw - Waterschoot";
3  var day-of-birth = "12/12/1985";
```

Snake case  gebruikt een **liggend streepje** (*Eng., underscore*) tussen elk woord en alle tekst wordt geschreven met kleine letters.


```
1  var first_name = "Philippe";
2  var sur_name = "De Pauw - Waterschoot";
3  var day_of_birth = "12/12/1985";
```

JavaScript programmeurs hebben de voorkeur om **camel case** , startend met een kleine letter, te gebruiken voor de naamgeving van identifiers die bestaan uit meerdere woorden.

```
1  var firstName = "Philippe";
2  var surName = "De Pauw - Waterschoot";
3  var dayOfBirth = "12/12/1985";
```

In JavaScript worden constanten, dit zijn variabele die niet van type kunnen veranderen (maar het liefst ook niet van waarden), geschreven met de Upper case Snake case notatie.

```
1  const LOAD_REPOS_ACTION = "Load Repo's Action";
```

In JS worden **functieobjecten** (*Eng., functional objects*) en **klassen** (*Eng., classes*) geschreven via de **Pascal case**  notatie. Dit is gelijkaardig met de Camel case notatie waarvan de eerste letter een **hoofdletter** (*Eng., capital letter*) is.

```
1  // Definition of a ElectricCar via functional declaration
2  const ElectricCar = function () {
3      ...
4  }
5
6  // Definition of a ElectricCar via Arrow function
7  const ElectricCar = () => {
8      ...
9  }
```

Commentaar

Om de code begrijpbaar te maken voor jezelf en voor anderen is het interessant om commentaar bij code te plaatsen. De syntaxis van **commentaar** (*Eng.* *comment*) is dezelfde als in C++ en in andere programmeertalen. Commentaar gedraagt zich als witruimte en wordt genegeerd tijdens het uitvoeren van JavaScript. Een goede gewoonte is om commentaar steeds te schrijven in het Engels, dit geldt trouwens ook voor identifiers. Aan de hand van commentaar moet je kunnen afleiden wat je wil bereiken met specifieke code.

Enkele regel commentaar (*Eng.* *single-line comments*) start met `//`.

```
1 // The variable gameState contains the current game state. Possible values: loading, pl
2 let gameState = 'loading';
```

Meerdere regels commentaar (*Eng.* *multiline comment*) start met `/*` en eindigt met `*/`. De tekst tussen deze symbolen wordt tijdens het uitvoeren genegeerd.

```
1 /*
2     1. Get references to all the articles on the page in an array format.
3     2. Loop through all the articles and add a click event listener to each one.
4
5     When any article is pressed, the gotoArticleDetail() function will be run.
6 */
7 const articleElelements = document.querySelectorAll('.article');
8
9 for (let i = 0; i < articleElelements.length ; i++) {
10     articleElelements[i].addEventListener('click', gotoArticleDetail);
11 }
```

Extra tekens mogen toegevoegd worden voor de duidelijkheid.

Bijvoorbeeld:

```
1 /*****
2  * @author Voornaam Naam
3  * @created 14/09/2020
4  * @modified 18/09/2020
5  * @copyright Copyright © 2020-2021 Artevelde University of Applied Sciences
```

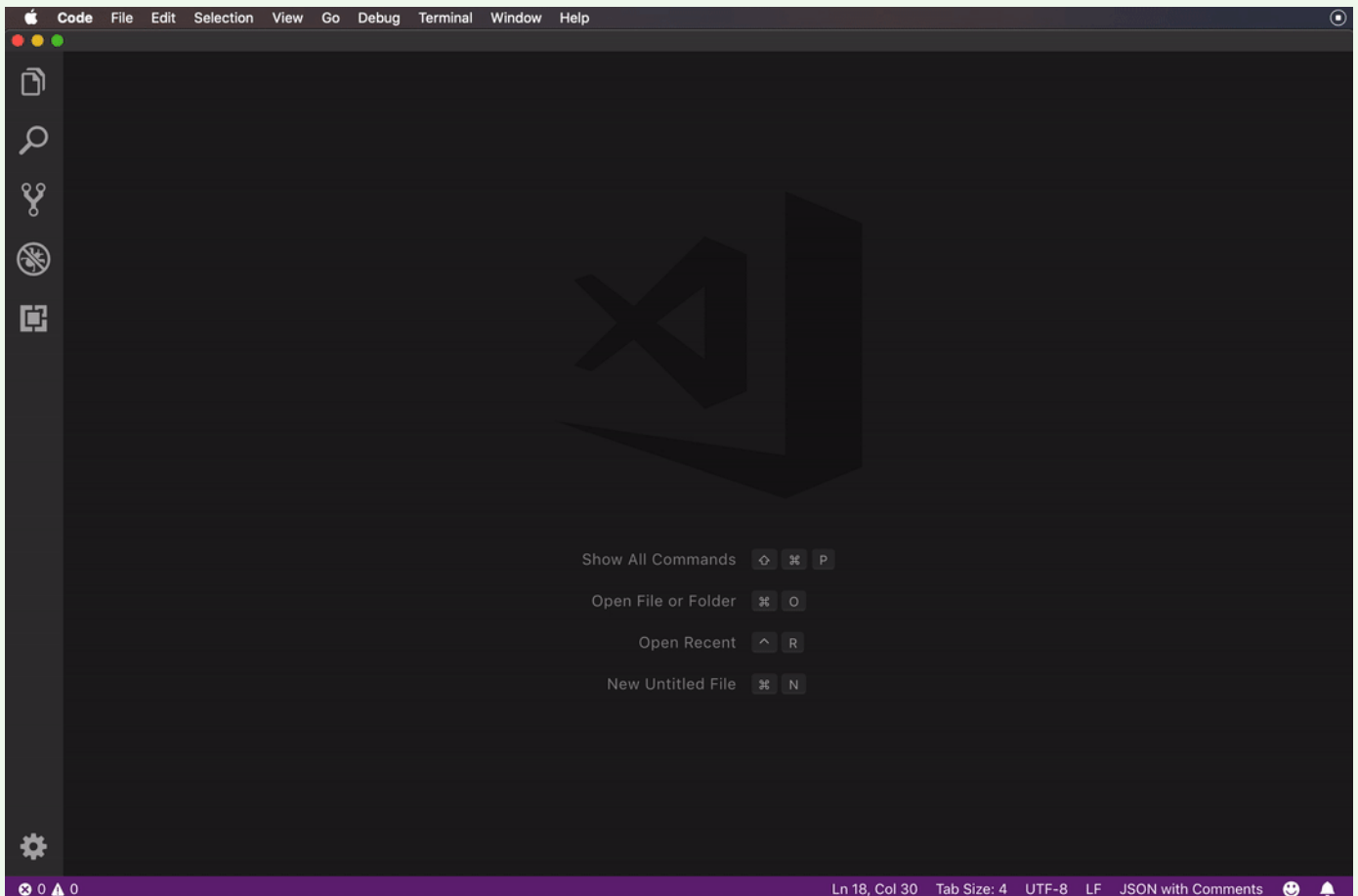
```
6 * @function Plaats hier - een korte samenvatting van jouw script
7 *****/
```



Tips

In VsCode kan je code in commentaar plaatsen door gebruik te maken van **sneltoetsen** [↗](#).

- MacOS: `command + /` voor 1 lijn in of uit commentaar te plaatsen (`//foo`), `shift + option + a` voor een selectie in of uit multi-line comment te plaatsen (`/*foo*/`).
- Windows: `ctrl + /` voor 1 lijn in of uit commentaar te plaatsen (`//foo`), `shift + alt + a` voor een selectie in of uit multi-line comment te plaatsen (`/*foo*/`).



Variabelen

Variabelen worden gebruikt als container voor **waarden** (*Eng., values*). De symbolische naam van de variabele geeft toegang tot deze waarde. De naam van de variabele is ook gekend als **identifier**. De naamgeving van een variable

moet beantwoorden aan bepaalde regels, die eerder beschreven zijn in het hoofdstuk [syntaxis](#).

Voorbeelden van legale naamgevingen: `String_fullname` , `temp404` , `$money` , `_dayOfBirth` ...

Declaratie

Een variabele kan gedeclareerd worden op twee manieren:

- Met het **trefwoord** (*Eng. keyword*) `var` . Bijvoorbeeld: `var firstName = "Philippe"` . Het trefwoord wordt gebruikt om **lokale** en **globale variabelen** te declareren.
- Met de trefwoord `const` of `let` . Bijvoorbeeld: `let i = 99` . Worden gebruikt om **block-scoped** variabelen te definiëren, zie hoofdstuk [scope](#).

```
1  var a; // Declaratie van een variabele met de naam a, bevat de standaardwaarde undefined
2  a = 36; // Toewijzing van de waarde 36 aan de variabele met de naam a
3
4  var b = 72; // Declaratie van de variabele met de naam b inclusief toewijzing van de waarde 72
```

Een `let` variabele is een variabele waarbij de identifier slechts één keer gedeclareerd mag worden. De scope is beperkt tot het block waarbinnen het gedefinieerd is.

```
1  let b;
2  let b = 6; // Syntaxis-fout, herdefinitie is niet mogelijk binnen dezelfde scope --> SyntaxError: Identifier 'b' has already been declared
3
4  var c;
5  let c = 7; // Syntaxis-fout, herdefinitie is niet mogelijk binnen dezelfde scope met trefwoord var
```

Een `const` variabele is een constante variabele waarbij de declaratie en toewijzing in één regel moet gebeuren en de waarde binnenin kan niet van type veranderen. Net zoals bij een `let` variabele is de scope beperkt tot het block.

```
1  const d; // Syntaxis-fout --> SyntaxError: Missing initializer in const declaration
2  const e = 6;
3  e = 12; // Syntaxis-fout --> TypeError: Assignment to constant variable.
```

⚠ Opmerking

Constant in `const` slaat op de toewijzing, niet noodzakelijk de waarde van de variabele.

Dat de toewijzing van een constate variabele maar één keer kan gebeuren wil niet zeggen dat de waarde **onveranderlijk** (*Eng. immutable*) is.

```
1  const f = [1];
2  ++f[0];
3  console.log(f); // Outputs: [2]
4  f.push(3); // No Error
5  f = 2; // SyntaxError: Assignment to constant variable.
```

Evaluatie van variabelen

Bij een declaratie zonder toewijzing wordt er toch een initiële waarde toegekend aan deze variabele, namelijk de waarde `undefined`. Veronderstel dat we formulier willen verzenden met één veld geïdentificeerd met een variabele `txtEmail`, dan kunnen we nagaan of dit veld al dan niet is ingevuld:

```
1  let txtEmail;
2  if (txtEmail === undefined) {
3    showError();
4  } else {
5    sendForm();
6  }
```

Een `undefined` waarde gedraagt zich als `false` wanneer deze gebruikt wordt in condities (boolean context):

```
1  let myData = [];
2  if (!myData) {
3    showData();
4  }
```

In een numerieke context wordt de `undefined` waarde geconverteerd naar `NaN` (Not-a-Number).

```
1  let a;
2  let b = a + 2; // The value of b is NaN
```

Een `null` waarde wordt geconverteerd naar een `0` in een numerieke context en naar een `false` in een boolean context.

```
1 let k = null;
2 let l = k * 6; // The value of l is 0
```

Scope

Wanneer we een variabele declareren buiten alle functies dan wordt deze een **globale variabele** (*Eng. global variable*) genoemd, omdat deze beschikbaar is voor alle code binnen hetzelfde document. Declareren we een variabele binnen een functie, dan noemen we deze een **lokale variabele**, omdat deze enkel beschikbaar is binnen deze functie.

```
1 if (true) {
2   var e = 16; // Global context
3 }
4 console.log(e); // Output: 16
```

```
1 if (true) {
2   let f = 16;
3 }
4 console.log(f); // Uncaught ReferenceError: f is not defined
```

```
1 let b = 6;
2 {
3   let b = 8;
4   console.log(b); // Output: 8
5 }
6 console.log(b); // Output: 6
```

```
1 var c = 6;
2 {
3   var c = 8;
4   console.log(c); // Output: 8
5 }
6 console.log(c); // Output: 8
```



```
1  var d = 1; // Variabele in Global Scope
2  {
3      let d = 2; // Variabele in Block Scope
4      console.info(d); // Output: 2
5  }
6
7  console.info(d); // Output: 1
```

Hoisting

In JavaScript worden `var` en `function` declaraties **opgehesen** (*Eng. hoisted*) bovenaan het document of indien binnen een block gedeclareerd bovenaan binnen dit block (bovenaan binnen zijn eigen scope). Tijdens compilatie worden `var` en `function` declaraties toegevoegd aan het geheugen, zodat deze beschikbaar worden binnen de scope.

```
1  console.log('The value of x is ' + x); // ReferenceError: x is not defined
2
3  console.log('The value of y is ' + y); // The value of y is undefined --> variable hoisting
4  var y;
5
6  console.log('The value of z is ' + z); // ReferenceError: z is not defined
7  let z;
8
9  let v;
10 console.log('The value of v is ' + v); // The value of v is undefined
```

Toewijzingen worden niet opgehesen.

```
1  console.log('The value of l is ' + l); // Output: The value of l is undefined
2  var l = 6;
3
4  var l;
5  console.log('The value of l is ' + l); // Output: The value of l is undefined, equivalent to undefined
6  l = 6;
7
8  m = 9;
9  console.log('The value of m is ' + m); // Output: 9
10 var m;
```

Functie declaraties worden opgehesen maar niet functie expressies.

```
1  foo(); // Outputs: bar because it's a function declaration
2
3  // Function declaration
4  function foo() {
5      console.log('bar');
6  }
7
8  foo2(); // TypeError: foo2 is not a function
9
10 // Function expressions
11 var foo2 = function() {
12     console.log('bar 2');
13 }
```

Globale variabelen

Globale variabelen zijn in feite **eigenschappen** (*Eng. properties*) van het globale object. In Webpagina's is het globale object het object `window`. Dit betekent dat we een globaal variabele kunnen definiëren voor een webpagina die gebruikt maakt van meerdere JavaScript-bestanden. Binnen deze gekoppelde bestanden kunnen we de globale variabelen aanspreken.

```
1  window.database = {
2      preferences: {
3          ui: 'dark',
4      }
5  }
```

In alle gekoppelde JavaScript-bestanden kunnen we deze globale variabele aanspreken via `window.database`. We kunnen bijvoorbeeld de **gebruikersinterface** (*Eng. user interface*) veranderen daar de heldere versie:

```
window.database.preferences.ui = 'light'
```

 . Deze wijziging is van toepassing op alle gekoppelde bestanden.

Samenvatting

Keyword	Declaratie	Toewijzen	Initialiseren	Hoisted
---------	------------	-----------	---------------	---------

Keyword	Declaratie	Toewijzen	Initialiseren	Hoisted
<code>var</code>	∞	∞	Mag	Ja
<code>let</code>	1	∞	Mag	Nee
<code>const</code>	1	1	Moet	Nee

Datastructuren en datatypen

Datatypen

ECMAScript2018 standaard definieert acht datatypen:

- **primitieve** (*Eng. primitives*) **datatypen** (*Eng. data types*):
 - `Boolean`
`true` of `false`
 - `null`
Trefwoord met `null` als waarde.
 - `undefined`
Een niet gedefinieerde waarde.A top-level property whose value is not defined.
 - `Number`
Een **geheel getal** (*Eng. integer*) of **kommagetal** (*Eng. floating point number*). Bijvoorbeeld: `3.14159`
 - `BigInt`
Een geheel getal met enorme precisie. Bijvoorbeeld: `9007199254740992n` .
 - `String`
Een verzameling van tekens die een tekstwaarde representeren. Bijvoorbeeld: `"Greetings Earthling"` .
 - `Symbol`
Datatype waarvan de instanties uniek en **onveranderlijk** (*Eng. immutable*) zijn.
- **Object**

Boolean

Het Boolean datatype bevat twee mogelijke waarden, namelijk: `true` of `false` . Het Boolean object is een wrapper rond het primitieve Boolean datatype.

```

1  const isRunning = false;
2  const isGameOver = new Boolean(false); // Boolean object
3  if (isGameOver) {
4      console.log('The game is finished!');
5  }

```

Omdat in het bovenstaand voorbeeld de variabele `isGameOver` een Boolean object is, zal de stelling waar zijn, ondanks de waarde `false` binnenin dit Boolean object.

Numeriek

In JavaScript kunnen numerieke waarden uitgedrukt worden in gehele getallen en kommagetallen. Een geheel getal kunnen we implementeren via de datatypen `Number` en `BigInt`. Kommagetallen realiseren we via een “Floating-point” literal of primitieve waarde (er bestaat **geen** `Float` **object**).

`Number` en `BigInt` datatypen kunnen uitgedrukt worden met verschillende talstelsels, namelijk: hexadecimaal (base 16), decimaal (base 10), octaal (base 8) en binair (base 2). Een decimale waarde bestaat uit een sequentie van **cijfers** (*Eng. digit*) waarvan het **eerste cijfer** (*Eng. leading digit*) geen 0 (*Eng. zero*) is, bijv.: 6, -111, 123456789123456789n ... Een octale waarde start met een 0, 0o of 0O en bevat digits met een waarde tussen 0 en 7. Een hexadecimale waarde start met 0x en de digits kunnen een waarde bevatten tussen 0 en 9, a en f of A en F, bijv.: 0x1134, 0xFFFFF, 0x123456789ABCDEFn ... Een binaire waarde start met 0b of 0B en de digits kunnen een waarde 0 of 1 bevatten, bijv.: 0b11, 0B1100, 00b111010010101010101n ...

Een kommagetal bestaat uit:

- een positief of negatief geheel getal (decimaal talstelsel)
- een punt (.)
- een geheel getal
- een exponent De exponent begint met een e of E gevolgd door een + of een - en eindigt met een geheel getal.

Voorbeelden:

```

1  const pi = 3.1415926;
2  const a = 965E+4; // 9650000
3  const b = 965E-4; // 0.0965

```

String

Een string bestaat uit 0...n tekens (Eng. *characters*) die omsloten worden door **dubbele aanhalingstekens** (Eng. *double quotation marks*) of **enkele aanhalingstekens** (Eng. *single quotation marks*).

js

```
1 'john'
2 "doe"
3 'B-9000'
4 "I'm a programmer"
5 'I\'m a developer'
6 'first rule\nsecond rule'
7 'what is my length?'.length // 18
```

In strings kunnen we ook gebruik maken van speciale tekens, waaronder: \n (nieuwe regel), \t (tab)... Volledige lijst van speciale tekens:

Teken	Omschrijving
\0	Null Byte
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\'	Apostrophe or single quote
\"	Double quote
\\	Backslash character
\000	Latin-1 encoding met drie octale digits 000 tussen 0 en 377 . Bijvoorbeeld: \251 komt overeen met het copyright symbool (©).

Teken	Omschrijving
<code>\x00</code>	Latin-1 encoding met twee hexadecimale digits <code>00</code> tussen <code>00</code> en <code>FF</code> . Bijvoorbeeld: <code>\xA9</code> komt overeen met het copyright symbool (©).
<code>\u0000</code>	Unicode teken gespecificeerd door vier hexadecimale digits <code>0000</code> . Bijvoorbeeld: <code>\u00A9</code> komt overeen met het copyright symbool (©).

Wanneer we een string beginnen en eindigen met dubbele aanhalingstekens met binnenin de string ook dubbele aanhalingstekens, dan moeten deze interne dubbele aanhalingstekens **backslash escaped** worden, kortom voorzien van een backslash: `\` . Hetzelfde geldt voor enkele aanhalingstekens.

```

1  const msg = "I'm reading \"Steve Jobs: The Man Who Thought Different: A Biography\"";
2  const msg2 = 'I\'m reading "Steve Jobs: The Man Who Thought Different: A Biography"';
js

```

Om een backslash `\` te integreren in een string, dan moeten we deze ook voorafgegaan worden door een backslash `\` .

```

1  const path = 'C:\\backup\\';
2  console.log(path); // Output: 'C:\backup\';
3
4  const fechtUrl = 'https://www.api.com/';
js

```

We kunnen een string waarde in meerdere code-regels schrijven door vermelding van een backslash `\` op het einde van iedere regel:

```

1  var str = 'this string \
2  is splitted \
3  across multiple \
4  lines.'
5  console.log(str); // this string is splitted across multiple lines.
js

```

Dit resulteert in één output-regel. Wensen we als output meerdere regels te voorzien, dan moeten we extra het return symbool `\n` voorzien.

```

1  const poem =
2  'Roses are red,\n\
3  Violets are blue.\n\
js

```

```
4   Sugar is sweet,\n5   and so is foo.');
```

Template literal

Sinds ECMAScript2015 kunnen we gebruik maken van **template literals** of template strings. Template literals of waarden zijn strings met daarbinnen expressies. Ze worden gebruikt om tekst te separeren in **meerdere regels** en laten ook toe om binnenin **string interpolaties** of expressies te voorzien. Een geavanceerde vorm van template literals zijn de **tagged template literals**. Hiermee kan de template literal verder gemanipuleerd worden via een functie.

Template literals beginnen en eindigen met een **back tick** (accent grave symbool: ```) in plaats van de gewone aanhalingstekens of apostrof. Speciale tekens, zoals `\n`, zijn niet nodig om tekst te separeren in meerdere regels. Dit verhoogt de leesbaarheid van de code.

```
1   const poem =
2   `Roses are red,
3   Violets are blue.
4   Sugar is sweet,
5   and so is foo.`
```

String of expressie interpolatie is één van de key features van template literals. Ze laten toe om expressies te implementeren binnenin de string, dit in tegenstelling tot gewone strings (`""` of `' '`) waar we expressies kunnen toevoegen via string **concatenering** (*Eng. concatenation*).

```
1   const firstName = 'Philippe';
2   const age = 36;
3   const msg1 = 'My name is ' + firstName + ',\ni\'m ' + age + ' years old.';
4   console.log(msg1);
5   const msg2 = `My name is ${firstName},
6   i\'m ${age} years old`;
7   console.log(msg2);
```

De expressies binnen een template literals kunnen eender welke **expressies** bevatten. Ze worden aangegeven met een dollar-teken (`$`) en accolades (`{ }`). Binnen de accolades vermelden we de expressie. Indien deze variabelen bevat, zal de engine op zoek gaan naar de initialisaties van deze variabelen. De waarden die toegekend zijn aan deze variabelen worden geïnjecteerd in de template literal.

```

1  const w = 1920;
2  const h = 1080;
3  const msg3 = 'My computer screen have a dimension of ' + w + 'x' + h + '.\n The aspect
4  console.log(msg3);
5  const msg4 = `My computer screen have a dimension of ${w}x${h}.
6  The aspect ratio is ${w/h}.`;
7  console.log(msg4);

```

Met tagged template literals kunnen we een template literal verder manipuleren via een functie. Het eerste argument of parameter van de functie is een array van stringwaarden, de volgende parameters zijn gerelateerd aan de expressies.

```

1  function bmi(strings, name, weight, height) {
2    const bmi = (weight/(height*height)).toFixed(1);
3    let condition = 'healthy weight';
4    if(bmi < 18.5) {
5      condition = 'underweight';
6    } else if (bmi >= 18.5 && bmi <= 25) {
7      condition = 'healthy weight';
8    } else if (bmi > 25 && bmi <= 30) {
9      condition = 'overweight';
10   } else if (bmi > 30) {
11     condition = 'obesity';
12   }
13   return strings[0] + name + strings[1] + bmi + '\nCondition: ' + condition
14 }
15 const name = 'Philippe';
16 const weight = 165;
17 const height = 1.72;
18 const msg5 = bmi`The person ${name} has a bmi of ${weight}${height}`;
19 console.log(msg5);
20 // "The person Philippe has a bmi of 22.0
21 // Condition: healthy weight"

```

Bij een tagged template literal wordt de string literal voorafgegaan door de naam van de tag-functie, in dit geval `bmi`. De tekst `The person` en `has a bmi of` behoren tot de eerste parameter van de functie, wat een array van strings bevat. `strings[0]` bevat de waarde `The person` en `strings[1]` bevat de waarde `has a bmi of`. De volgende parameters in de functie corresponderen met de expressies in de string literal, respecteer wel de volgorde!

Datatype conversie

In JavaScript kunnen we het datatype niet specificeren tijdens de declaratie. Het datatype wordt automatisch toegekend (*Eng. dynamically typed language*) tijdens het uitvoeren van het script.

```
1 let age = 89;
2 console.log(typeof age); // Output: 'number'
3
4 age = 'I\'m 89';
5 console.log(typeof age); // Output: 'string'
```

Omdat JavaScript **dynamically typed** is kunnen we eerst bijvoorbeeld een numerieke waarde toekennen aan de variabele en vervolgens een tekstwaarde toekennen aan diezelfde variabele.

```
1 let msg = 'My age is ' + 89;
2 console.log(msg); // Output: 'My age is 89'
```

In expressies, die bestaan uit de combinatie van numerieke- en tekstwaarden, met de `+` operator, zullen alle numerieke waarde geconverteerd worden naar corresponderende tekstwaarden.

```
1 const x = '89' - 6;
2 console.log(x); // Output: 83
3 const y = 'Howdy' - 6;
4 console.log(y); // Output: NaN
```

Met alle andere operatoren resulteert dit in de waarde `NaN`. Zoals eerder vermeld onder het hoofdstuk **evaluatie van variabelen** gedraagt een `0`, `null` en `undefined` zich in een Boolean context als de waarde `false`.

```
1 const msg = 'Howdy';
2 function isItOk(m) {
3   return m ? 'ok' : 'nok';
4 }
5 console.log(isItOk(msg)); // Output: 'ok'
6 console.log(isItOk(0)); // Output: 'nok'
7 console.log(isItOk(null)); // Output: 'nok'
```

Conversie van strings naar numbers

Zoals reeds vermeld kan een string geconverteerd worden naar een numerieke waarde via de `parseFloat` operator. Als alternatief kunnen we ook de ingebouwde functies `parseInt` en `parseFloat` gebruiken. `parseInt` geeft een geheel getal terug, `parseFloat` een kommagetal.

De `parseInt` functie bevat 2 verplichte argumenten, namelijk de string (die we willen converteren) en het numeriek systeem of talstelsel (dat we wensen te gebruiken). Gekende talstelsels zijn: binair (2), octaal (8), decimaal (10) en hexadecimaal (16).

```
1 parseInt('101', 2); // 5: 1*2^0+0*2^1+1*2^2
2 parseInt('101', 8); // 65
3 parseInt('-101', 10); // -101
4 parseInt('0xF', 16); // 15
5 parseInt('56757657657867988090989089866746546456465', 10); // 5.675765765786799e+40
```

Een string met een kommagetal als waarde wordt in het 10-talig stelsel geconverteerd naar geheel getal met afronding naar beneden. In het binair talstelsel wordt deze geconverteerd naar de waarde `NaN`.

```
1 parseInt('3.84', 10); // 3
2 parseInt('3.84', 2); // NaN
```

In het geval dat een waarde die een getal voorstelt als een string in het geheugen zit, zijn er methoden voor omzetting van dat getal.

```
1 parseFloat('3.84'); // 3.84
2 parseFloat('3.84 is my age'); // 3.84
3 parseFloat('My age is 3.84'); // NaN
4 parseFloat(NaN); // NaN
5 parseFloat('5675765765786798809.80989089866746546456465'); //5675765765786799000
6 parseFloat('Infinity'); // Infinity
```

Arrays

Arrays aanmaken

Een array is een speciale variabele, die meer dan 1 waarde kan bevatten.

De eenvoudigste manier om een array aan te maken is met een array literal.

```
1  const arrayName = ['waarde 1', 'waarde 2', 3, ...]
```

Een array kan dus ook verschillende datatypen bevatten en maken we meestal aan met het `const` keyword.

We kunnen ook altijd eerst de array aanmaken en dan waarden toekennen aan de verschillende posities.

```
1  const cars = [];  
2  cars[0] = "Saab";  
3  cars[1] = "Volvo";  
4  cars[2] = "BMW";
```

Een andere manier om een array aan te maken is met het `new` keyword.

```
1  const cars = new Array("Saab", "Volvo", "BMW");
```

Deze methode wordt echter niet veel gebruikt.

Elementen opvragen

Een element opvragen uit een array kunnen we doen door de index te gebruiken.

```
1  const cars = ["Saab", "Volvo", "BMW"];  
2  let car = cars[0]; // Output: Saab
```

Elementen aanpassen

Elementen van de array aanpassen kunnen we doen met de index en een toewijzing.

```
1  const cars = ["Saab", "Volvo", "BMW"];  
2  cars[0] = "Opel";
```

