Javascript – deel 05

Deze les behandelt de number en string types in Javascript.

Verslag

In dit deel van de cursus staan verschillende vragen die je moet beantwoorden en opdrachten om iets te maken of uit te proberen. Het is belangrijk dat je alle opdrachten zorgvuldig uitvoert!

Documenteer je werk in een verslag document 'javascript deel 05' waarin je

- voor elke uitprobeer opdracht een entry maakt met screenshots ter staving van wat je deed
- je antwoorden op de gestelde vragen neerschrijft

Oplossingen van grotere opdrachten (met veel code) bewaar je aparte folders in een Webstorm project.

Types

In Javascript kun je in de source code geen datatype opgeven bij de declaratie van een variabele, functie parameter of functie return value. Het is dus geen statisch getypeerde programmeertaal.

Tijdens de uitvoering zijn er echter wel degelijk types voor de waarden die een programma men noemt Javascript dan ook een dynamisch getypeerde programmeertaal.

Er bestaat in javascript trouwens een **typeof** operator die een tekstvoorstelling produceert van wat voor soort waarde iets voorstelt. Bijvoorbeeld :

let s1 = "blablabla";
typeof s1 geeft "**string**"

let s2 = new String("blablabla");
typeof s2 geeft "**object**"

Merk op dat er zowel een 'string' als een 'String' type bestaat (idem voor number/Number), het verschil tussen die beiden is voor deze cursus echter niet van belang.

Op onderstaande link kun je zien wat voor soort waarden er in Javascript bestaan, het zijn er minder dan je wellicht verwacht :

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

Opdracht

Wat voor soort waarden bevatten onderstaande dikgedrukte variabelen ?

```
let leeftijd = 34;
let intrest = 0.12;
let isGevaarlijk=true;
let vandaag = new Date();
const print = (message) => {
  console.log(message);
};
```


functies zijn ook waarden, van het type 'function'!

Je kan dit bv. snel op de console in de Chrome Developer Tools uitproberen.

Wat is het type van de waarden in de volgende globale variabelen?

- window
- document (eigenlijk window.document)
- console (eigenlijk window.console)

Functies zijn waarden

Dat functies wel degelijk waarden zijn kun je zien aan het volgende stukje code :

```
const print = (message) => {
  console.log(message);
};
```

```
const aantalSpelers = 11;
const url = "http://www.example.com";
const isTestMode = true;
```

Dit definieert een aantal (constante) variabelen die elk een waarde krijgen, m.a.w. 'print' is precies zoals 'aantalSpelers', 'url' en 'isTestMode'.

We declareren dus een (constante) variabele met naam 'print' en stoppen er een waarde in. Deze waarde is een verwijzing naar een functie met een parameter die deze parameter op de console plaatst.

Vermits (verwijzingen naar) functies waarden zijn, kunnen we ze kopiëren :

```
print("Hello");                  // dit toont "Hello" op de console
const show = print;
show("Goodbye");                 // dit toont "Goodbye" op de console
```

Een show() oproep zal nu dezelfde function uitvoeren als een print() oproep.

Dit illustreert ook dat er niet echt een *'print' functie* bestaat, er is enkel een 'print' (constante) variabele wiens waarde naar een arrow function verwijst. Voor de eenvoud noemen we deze function dan de *'print' functie*, maar zo te zien mogen we ze voortaan ook de *'show' functie* noemen 😉

We kunnen (een verwijzing naar) een function ook doorgeven als parameter aan een andere function. Het koppelen van een event listener is hiervan een typisch voorbeeld :

```
const setup = () => { … };
window.addEventListener("click", setup);
```

De (constante) variabele 'setup' verwijst naar een function en die verwijzing geven we mee als tweede parameter aan de addEventListener() oproep.

Number

Alle getallen worden in javascript voorgesteld door number waarden en zijn steeds kommagetallen met dubbele precisie, zie

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

Er zijn dus geen aparte voorstellingen zoals int, float, double, etc. in C# of Java.

Je kan natuurlijk ook een getal als tekst in een string stoppen maar als je berekeningen wil maken heb je een number nodig. In de "rekenmasjien" demonstratie zagen we reeds hoe we een number kunnen bekomen uit een string m.b.v. **Number.parseInt** :

```
let getalAlsTekst="123";
let getalAlsNumber = Number.parseInt(getalAlsTekst, 10);
```

Let op de meegegeven radix 10, zodat de functie weet in welk talstelsel de cijfers moeten geïnterpreteerd worden. Denk aan het T-shirt grapje 😉

Naast de eerdere geziene parseInt bestaat er ook een **Number.parseFloat** method. Deze produceert ook een number op basis van een meegegeven string, maar houdt ook rekening met cijfers na de komma.

Merk op dat de regio/taal instellingen van de browser een rol kunnen spelen bij de voorstelling van een kommagetal. Namelijk, of het geheel en de fractie door een komma dan wel een punt moeten gescheiden worden (bv. 13,45 versus 13.45).

De parseFloat method werkt echter uitsluitend met een punt als scheidingsteken (bv. 13.45)!

Ons programma moet dus in principe de browser regio/taalinstellingen raadplegen om eventueel nog een omzetting te doen van het scheidingsteken. Dit is niet zo eenvoudig, zeker niet als er ook nog een scheidingsteken op de duizendschaal (bv. 1,000,000 voor 1 miljoen) kan voorkomen.

Gelukkig biedt de browser ons een eenvoudige oplossing, op voorwaarde dat we met een <input> element werken en het type attribuut instellen op 'number' :

type = 'number' ⟵ speciaal voor getallen

De browser zorgt er dan voor dat de .value waarde van dit <input> element, altijd netjes een string met een punt oplevert zoals bv. 13.45 (zelfs als de werkelijke gebruikersinput een komma bevat, bv. 13,45).

Kort gezegd : als we type='number' gebruiken, zal de .value property altijd een string opleveren in het juiste formaat voor Number.parseFloat().

Beide methods negeren extra karakters die na het getal komen. Het parsen van "123Eur" met parseInt levert het getal 123 op.

Indien de string helemaal niet als een getal kan geïnterpreteerd worden, dan produceren parseInt en parseFloat de waarde NaN.

NaN is een speciale number waarde om aan te geven dat het geen geldig getal is, NaN staat dan ook voor Not a Number.

Om te testen of een bepaalde number waarde al dan niet NaN is, kun je Number.isNaN(…) gebruiken :

```
let getal = Number.parseInt( txtInput.value, 10 );
if ( Number.isNaN(getal) ) {
        // het is geen getal, doe iets anders
} else {
        // het is wel een getal, bereken iets met 'getal'
}
```

Als je de tekstvorm wil bekomen van een number kun je diens .toString() method gebruiken,

```
let aantal = 10;
let aantalAlsTekst = aantal.toString();                    // aantalAlsTekst is nu "10"
```

maar als je die tekst wil samenplakken met andere tekst kan het eenvoudiger met '+' voor Strings :

```
let tekst = "U kocht " + aantal + " producten";
```

of met een template literal (let op de speciale symbolen rond de tekst, dit zijn backticks of back quotes)

```
let tekst = `U kocht ${aantal} producten`;
```

Om getallen (visueel) te beperken tot een bepaald aantal cijfers na de komma, kun je de **.toFixed(n)** method gebruiken. Deze zal een string produceren van het afgeronde cijfer met slechts 'n' cijfers na de komma.

Bijvoorbeeld :

```
let getal=1234.56789
console.log( getal.toFixed(2) );                    // toont "1234.57" (afronding!)
```

Merk op dat we bij de console.log parameter geen .toString() moeten doen, de log method kan dat zelf.

Random getallen

Je kan een willekeurig getal bekomen met de volgende method

```
let getal = Math.random();
```

De variabele 'getal' zal een kommagetal bevatten in [0,1[

Indien je een getal in [0,15[wil, doe je

```
let getal = Math.random() * 15;
```

Indien je een getal in [10,25[wil, doe je

```
let getal = 10 + Math.random() * 15;
```

Gehele getallen bekomen

Indien je een geheel getal wil bekomen op basis van een kommagetal (dus zonder fractie), kun je de volgende Math methods gebruiken :

Math.floor(kommaGetal)
dit zal naar onderen afronden
bv. Math.floor(7.67) geeft 7
bv. Math.floor(-7.34) geeft -8!

Math.ceil(kommaGetal)
dit zal naar boven afronden
bv. Math.ceil(7.34) geeft 8
bv. Math.ceil(-7.67) geeft -7!

Math.round(kommaGetal)
dit zal afronden
bv. Math.round(7.34) geeft 7
bv. Math.round(7.67) geeft 8
bv. Math.round(7.5) geeft 8
bv. Math.round(-7.5) geeft -7!

Let goed op met waarden waarvan je niet op voorhand weet of ze positief dan wel negatief zijn!

Vraag : Waarom is het geen goed idee om het rollen van een dobbelsteen te simuleren met

```
Math.round( Math.random()*5 ) + 1
```

Als je het niet meteen ziet, probeer dan eens het onderstaande programma uit. Dit telt hoe vaak elk aantal ogen voorkomt als we de virtuele dobbelsteen 600 000 keer laten rollen.

```
let aantalKeer = [0,0,0,0,0,0];

// genereer random getallen van 1 t.e.m. 6 en tel hoe vaak elk voorkomt
for (let i=0 ; i<600000 ; i++) {
  let getal = Math.round( Math.random() * 5 ) + 1;
  aantalKeer[ getal-1 ]++;                                    // tel er voor dit getal eentje bij
}

// toon het resultaat van de telling
for (let getal=1 ; getal<=6 ; getal++) {
  console.log( `getal ${ getal } kwam ${ aantalKeer[getal-1] } keer voor` );
}
```

Merk op dat het aantal keer dat getal i voorkwam, op positie i-1 staat in het 'aantalKeer' array. Dit komt omdat we in ons array de posities tellen vanaf 0 maar het aantal ogen tellen vanaf 1.

Op 600 000 gooien zou je verwachten dat elk van de zes getallen ongeveer 100 000 keer voorkomt.

Welke aantallen krijg je als je het programma een paar keer uitvoert?

Welke getallen komen minder vaak voor dan je zou verwachten?

Hoe komt dit en waarom precies bij deze getallen?

Vraag : zijn de volgende berekeningswijzen fair voor het rollen van een dobbelsteen?

- Math.floor(Math.random() * 6) + 1
- Math.ceil(Math.random() * 6)

Opdracht Producten

Maak een webpagina met volgende inhoud (een tabel en een button) :

producten	prijs	aantal	btw	subtotaal
product1	10.00 Eur	0	6%	0.00 Eur
product2	15.00 Eur	0	21%	0.00 Eur
product3	12.20 Eur	0	21%	0.00 Eur
totaal				0.00 Eur

Herbereken

De cellen in de 'aantal' kolom zijn numerieke invoervelden die de gebruiker kan wijzigen :

```
<input type="number" value="0">
```

De waarden in de 'prijs' en 'btw' kolommen zijn hardgecodeerd in de HTML, bijvoorbeeld :

```
<td>10.00 Eur</td>
```

Na het klikken op de 'Herbereken' knop, vult het programma de subtotalen en het totaal in (rekening houdend met de 'prijs' en het 'btw' tarief) :

producten	prijs	aantal	btw	subtotaal
product1	10.00 Eur	1	6%	10.60 Eur
product2	15.00 Eur	1.5	21%	27.22 Eur
product3	12.20 Eur	2	21%	29.52 Eur
totaal				67.35 Eur

Herbereken

Voor alle duidelijkheid : de gebruiker kan enkel de kolom 'aantal' invullen en op de knop klikken.

Zorg ervoor dat je programma ook correct werkt indien de hardgecodeerde waarden in de HTML code veranderd worden, m.a.w. je mag er niet zomaar van uitgaan dat het 'btw'-tarief voor product1 altijd 6% zal zijn. Je zult dus in je programma die waarden uit de <td> elementen in de DOM-tree moeten halen!

> *Je krijgt uit dergelijke cellen een string als "10.00 Eur" die je moet omzetten naar een number om ermee te kunnen rekenen. Gelukkig negeren de parseInt en parseFloat functies alles wat niet op een getal lijkt, dus je hoeft je geen zorgen te maken om het " Eur" achtervoegsel in die string!*

Tip : geef alle elementen van eenzelfde soort dezelfde class (bv. **'prijs', 'aantal', 'btw' en 'subtotaal'**).

De HTML-code voor twee rijen in je tabel zien er dan ongeveer zo uit :

```
<tr>
  <td>product1</td>
  <td class='prijs'>10.00 Eur</td>
  <td><input class='aantal' type="number"></td>
  <td class='btw'>6%</td>
  <td class='subtotaal'>0.00 Eur</td>
</tr>

<tr>
  <td>product2</td>
  <td class='prijs'>15.00 Eur</td>
  <td><input class='aantal' type="number"></td>
  <td class='btw'>21%</td>
  <td class='subtotaal'>0.00 Eur</td>
</tr>
```

Als je nu alle elementen met class 'aantal' opvraagt, krijg je een lijst van alle <input> elementen uit de kolom met hoofding 'aantal'.

Doe hetzelfde voor de andere classes en je krijgt in totaal 4 lijsten met elk 3 elementen (want : 3 rijen).

De vraag is nu, staan de elementen van een rij in elk van die 4 lijsten op dezelfde positie?

Het antwoord is JA. Bijvoorbeeld, op positie 1 vind je in de 4 lijsten de 4 elementen uit rij 2.

Je kan dus gewoon een for loop maken die per iteratie een rij uit de tabel afwerkt : namelijk de 'prijs', 'aantal' en 'btw' uitlezen, berekeningen maken en het 'subtotaal' invullen.

Strings

Een javascript programma kan teksten voorstellen a.d.h.v. *strings*, net als in andere programmeertalen.

Er zijn 2 manieren om een String aan te maken :

```
let s1 = "blablabla";                 // dit is de aanbevolen manier
let s2 = new String("blablabla");     // dit kom  je maar zelden tegen
```

Technisch gezien zijn beide mogelijkheden (string vs. String) niet 100% gelijk maar voor deze cursus is het verschil niet zo belangrijk.

Een string literal kan ofwel met aanhalingstekens (") ofwel apostrofen (') afgebakend worden. Dit is handig als je eens HTML-code met attributen in een string literal wil stoppen :

- let imgHTML = '<img src="image.png">'; // je mag zelf kiezen welke je gebruikt 😊
- let imgHTML = "<img src='image.png'>";

Beide regels verschillen enkel in de afbakening van de string literals en de src-attribuut waarden.

De lengte van een string kun je opvragen met de .length property, bv.

```
let tekst="Hello world";
console.log( tekst.length );          // toont 11
```

Individuele karakters uit een string kun je opvragen met de .charAt() method die een index als parameter vereist.

```
console.log( tekst.charAt(1) );       // toont "e"
```

Ook in javascript nummeren we de posities in een string beginnend vanaf 0. Het tweede karakter vind je dus op indexpositie 1.

Let op, soms kom je wel eens code tegen die hiervoor de [] operator gebruikt ipv de .charAt() method, bv. tekst[1] , maar dit werkt niet op alle browser(versies). Bovendien wekt het de illusie dat een string een array van karakters, maar dit is NIET het geval! De analogie gaat bv. niet op voor toekenningen :

bv. tekst[1]="a" zal NIET werken

Gebruik dus liever de .charAt() method in je eigen code.

Vraag : wat is het type van het resultaat van de .charAt() method? Is het iets als 'char' in C#? Heeft Javascript eigenlijk wel een type voor individuele karakters?

Om teksten aaneen te plakken kun je ofwel de .concat() method gebruiken of wel de '+' operator, bv.

```
let s1 = "Hello ";
let s2 = "world!";
let s3=s1.concat(s2);          // s3 bevat nu "Hello world!"
let s4=s1+s2;                  // s4 bevat nu "Hello world!"
```

Doorgaans gebruikt men de '+' operator.

Om de waarde van een variabele middenin een string te stoppen kun je tekst stukjes aaneenplakken

```
let naam = "Jan";
let leeftijd = 19;
…
let tekst = "De persoon "+naam+" is "+leeftijd+" jaar oud";
```

maar vaak is het leesbaarder om een **template literal** te gebruiken (vgl. met string interpolatie in C#) :

```
let naam = "Jan";
let leeftijd = 19;
…
let tekst = `De persoon ${naam} is ${leeftijd} jaar oud`;
```

Let op : template literals worden afgebakend door *backquote* (`) symbolen.

Template literals kunnen trouwens ook meerdere regels in beslag nemen.

Bijvoorbeeld

```
console.log(`dit is regel 1 van de tekst
en regel 2 eronder`);
```

toont op de console

```
dit is regel 1 van de tekst
en regel 2 eronder
```

Bedenk dat de tabs en spaties die voor de insprongen in de source code zorgen, ook in de string terecht zullen komen. Vaak is dit geen probleem, bv. als de string gebruikt wordt om HTML-code in de DOM-tree te stoppen : de browser negeert sowieso extra tabs en spaties in HTML-code.

Je kunt een zoektekst opzoeken in een string met de volgende twee methods :

- .indexOf(zoektekst) // begint zoektocht vooraan de tekst
- .lastIndexOf (zoektekst) // begint zoektocht achteraan de tekst

Beide methods geven als resultaat de positie waarop de deeltekst gevonden werd, of -1 indien deze niet gevonden werd.

Bijvoorbeeld, als je je afvraagt of een bepaalde tekst het woordje 'hond' bevat :

```
let tekst = "Man bijt hond met valse tanden";
if ( tekst.indexOf("hond") != -1 ) {
        // tekst bevat het woordje "hond"
} else {
        // tekst bevat het woordje "hond" niet
}
```

De return value van indexOf (en lastIndexOf) geeft dus de karakter positie waarop het woord werd gevonden, in bovenstaand voorbeeld is dit index positie 9. Die exacte positie is bv. handig als je woorden in een tekst wil vervangen (zie verderop).

Je kan deze methods ook oproepen met een tweede parameter idx :

- .indexOf(zoektekst, **idx**) // begint zoektocht vooraan de tekst
- .lastIndexOf (zoektekst, **idx**) // begint zoektocht achteraan de tekst

De tweede parameter idx is dus optioneel en kan gebruikt worden om vanop een andere startpositie te beginnen zoeken. Dit is bv. handig als je in een tekst verder wil blijven zoeken nadat je al iets gevonden hebt. De nieuwe startpositie is dan doorgaans het karakter na de treffer van de eerdere zoekactie.

Vaak is het nodig om een nieuwe String te bemachtigen die gebaseerd is op een stukje van een andere String. Bijvoorbeeld, als je in een tekst een zoekwoord wil vervangen door een vervangwoord moet je bij een treffer, de tekst voor en na die treffer gebruiken om de nieuwe tekst te bouwen.

Om zo'n deeltekst te verkrijgen heb je drie ingredienten nodig : de String waarop je je baseert, de start indexpositie en tenslotte de eind indexpositie (of evt. het aantal karakters geteld vanaf de startpositie).

Om deelteksten uit een tekst te kunnen gebruiken, bestaan er maar liefst drie verschillende methods :

- .substring(idx1, idx2)
- .substr(idx, length)
- .slice(idx1, idx2) // dit is de betere keuze

De methods .slice() en .substring() werken min of meer op dezelfde manier, .substr() werkt echter iets anders. Zie onderstaande link voor een bespreking :

http://www.jacklmoore.com/notes/substring-substr-slice-javascript/

In je eigen code gebruik je liefst de .slice() method, die werkt altijd zoals je zou verwachten. Bijvoorbeeld

```
let s1="Hello world";
console.log( s1.slice(3, 8) ); // dit toont lo wo
```

Merk op dat het karakter 'r' op indexpositie 8 niet in de deeltekst zal zitten, de rechtergrens index is niet inclusief.

Om letterlijke string waarden (i.e. string literals) in source code te plaatsen hebben we hierboven steeds de tekst tussen aanhalingstekens (") geplaatst, bv.

```
let tekst="deze tekst staat tussen dubbele aanhalingstekens";
```

Zoals eerder werd vermeld, je mag echter ook apostrofen gebruiken :

```
let tekst='deze tekst staat tussen enkele aanhalingstekens';
```

Je kunt ook beide door elkaar mengen, dit is handig in de tekst zelf aanhalingstekens moet bevatten

```
let message= "U drukte op 'stop'.";
let message ='U drukte op "stop".';
```

of uit het eerdere voorbeeld met HTML-code :

```
let imgHTML = '<img src="image.png">';
let imgHTML = "<img src='image.png'>";
```

Sommige speciale karakters moeten in een string literal door een backspace \ karakter voorafgegaan worden :

- \n newline
- \' single quote
- \" double quote
- \\ backslash
- \u89ab voor unicode karakter 89ab

Om de hoofdletter (of kleine letter) versie van een string te verkrijgen kun je de volgende twee methods gebruiken :

- .toUpperCase()
- .toLowerCase()

Het resultaat van deze methods is een nieuwe string met inhoudelijk dezelfde tekst als de originele string maar met enkel hoofdletters (of kleine letters).

Soms wil je zeker zijn dat een string geen lege ruimte (whitespace) bevat aan het begin of einde, dit is bv. handig om extra spaties of tab-karakters uit user-input te verwijderen. Je kunt hiervoor de .trim() method gebruiken :

```
let tekst="    Jan Janssens  ";
let proper=tekst.trim();                    // bevat "Jan Janssens"
```

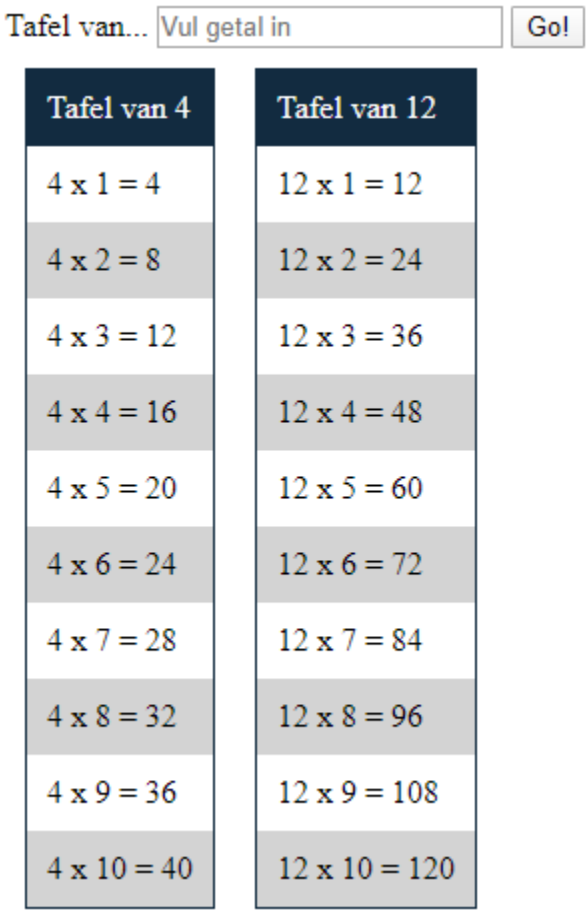
Deze method produceert een nieuwe string waarin vooraan en achteraan geen whitespace meer voorkomt.

Opdracht Tafels

Schrijf een webpagina met een invoerveld en een “Go!” knop.

Indien de gebruiker een geheel getal intypt en op de knop drukt, komt de tafel van dat getal in de pagina erbij. Als de input geen geheel getal is komt er natuurlijk niks bij. Het programma maakt telkens het invoerveld leeg nadat er op de knop gedrukt is.

Bijvoorbeeld, als de gebruiker 4 invult en op ‘Go!’ klikt en vervolgens hetzelfde doet met 12, dan ziet de pagina er als volgt uit :



Een tafel heeft een hoofding in kleur #122b40, wat gelijk ook de kleur is van het 1px brede kader. De achtergrondkleur van de even rijen is #d3d3d3 en de oneven rijen zijn wit (doe dit met CSS en niet JS!).

Je kunt dit met een <table> element maken, maar tables stylen is niet zo simpel en deze lessenreeks gaat niet zozeer over CSS. Maak dus gewoon een met daarin bv. voor elke rij een <p> element. Om de tafels naast elkaar te krijgen kun je hun CSS 'display' property op 'inline-block' instellen en hen een 10px margin geven. In elke rij is een 10px padding voorzien.