Javascript – deel 09

Deze les behandelt Javascript objecten.

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 09' 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

Objecten

Objecten bundelen gegevens (datavelden, properties) en gedrag (methods). Als je al eens met objecten gewerkt hebt in een andere programmeertaal (bv. C#), dan zullen Javascript objecten heel snel vertrouwd aanvoelen.

> *Een kleine waarschuwing : veel zaken in Javascript lijken vertrouwd en eenvoudig als je reeds C# kent, op voorwaarde dat je niet te diep delft. In werkelijkheid zijn er zéér grote verschillen in de mogelijkheden van objecten in C# en Javascript. Gelukkig zullen we daar in deze cursus geen last van hebben : voor onze DOM-tree manipulaties en eenvoudige programma's, kunnen we ons beperken tot heel simpele objecten.*

Javascript objecten werken met een *prototype chain* die gebruikt wordt om properties en methods van een object terug te vinden. Laten we hier vooral heel vaag over blijven en snel overgaan naar het volgende thema… 😉 voor de geïnteresseerde lezer

Javascript objecten zijn ook heel erg flexibel qua welke properties en methods ze ondersteunen.

- In een programmeertaal als C# moet je een *class* definiëren en dan zien alle objecten van die *class* er hetzelfde uit. Er zijn weliswaar oneindig veel mogelijke objecten van een Student klasse, maar ze zullen allen altijd exact dezelfde properties en methods hebben.
- In Javascript kan elk object in principe een uniek pareltje van properties en methods zijn. Je kan zelfs bij een bestaand object een nieuwe property toevoegen of een bestaande property verwijderen.

In de praktijk zul je deze doorgedreven flexibiliteit niet vaak gebruiken. Het is gewoon veel eenvoudiger om een programma te structureren volgens gelijkaardige (lees : voorspelbare) objecten, i.p.v. met allerlei uitzonderingen rekening te moeten houden.

Dus ook in je Javascript programma's zullen alle objecten die studenten voorstellen, steeds dezelfde properties en methods hebben.

Tot recent kwam het 'class' concept niet eens expliciet voor in Javascript. Er waren wel manieren om dit te faken met constructor functies en hun 'prototype' property, maar die zijn nogal technisch en vergen behoorlijk wat voorkennis en discipline van de programmeur.

Sinds ECMAscript 6 heeft Javascript wel classes, al was het vooral een cosmetische ingreep om bovenstaand gefake te uniformiseren.

Wij zullen ons in deze cursus beperken tot hele **simpele objecten**. Deze objecten zullen enkel properties bevatten en geen methods, ze bundelen dus **enkel data en geen gedrag**. Men noemt dit wel eens *dataholders* of *databags*.

We zullen ook geen enkele soort van (al dan niet fake) classes of overerving gebruiken.

Date objecten

Om het werken met Javascript objecten te introduceren, bekijken we hieronder eerst eens Date objecten. Daarna wordt besproken hoe je zelf je eigen simpele objecten kunt maken.

Een Date object stelt een tijdstip voor (datum + tijd). Het huidig tijdstip kun je achterhalen met

```
let now = new Date();
```

De variabele 'now' zal dan naar een Date object wijzen dat het tijdstip voorstelt ten tijde van de uitvoering van die opdracht.

De documentatie voor Javascript Date objecten vind je op

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

Bekijk die pagina en let daarbij vooral op de verschillende constructoren om Date objecten aan te maken, alsook de diverse get en set methods waarmee je de maand/jaar/jaar/uur/... onderdelen van zo'n tijdstip kunt opvragen of aanpassen.

Bijvoorbeeld,

```
let now = new Date();
let uren = now.getHours();                                  // uren
let minuten = now.getMinutes();                             // minuten
console.log( `het is nu ${ uren } uur en ${ minuten } minuten` );
console.log( `de datum is ${ now.getDate() } / ${ now.getMonth() } ` );   // dag en maand
console.log( now.toISOString() );                           // ISO 8601 formaat
```

Vraag : Stel het is vandaag 2 mei, wat is dan het resultaat van now.getMonth() ?

Een Date object maken op basis van een string is een beetje problematisch, gebruik best de constructor met een string parameter en zorg dat de tekst in ISO-8601 formaat staat, bv.

```
new Date('2014-12-25')
new Date('2011-12-25T23:59:59')
```

Als je een goeie tekstvoorstelling van een Date object nodig hebt, bv. om aan de gebruiker te tonen in de user interface, dan zul je die tekst zelf moeten opbouwen (of beter : een library gebruiken), zie

https://stackoverflow.com/questions/3552461/how-to-format-a-javascript-date

Om snel iets uit te proberen kun je echter de tekstvoorstellingen gebruiken die de methods .toString(), .toISOString(), .toDateString() of .toTimeString() produceren.

Bijvoorbeeld :

```
console.log( now.toString() );
    Sun May 09 2021 23:52:43 GMT+0200 (Central European Summer Time)     // output
console.log( now.toDateString() );
    Sun May 09 2021                                                      // output
console.log( now.toTimeString() );
    23:52:43 GMT+0200 (Central European Summer Time)                     // output


console.log( now.toISOString() );
    2021-05-09T21:52:43.296Z                                             // output
console.log( now.toISOString().substr(0,10) );    // het datum stukje eruit halen met substr()
    2021-05-09                                                           // output
console.log( now.toISOString().substr(11,8) );    // het tijd stukje eruit halen met substr()
    21:52:43                                                             // output
```

Let erop dat het tijdsdeel van .toISOString() als UTC-tijd is uitgedrukt en dus wellicht niet klopt voor de eindgebruiker (verkeerde tijdzone of zomer vs wintertijd).

Vandaar, gebruik dit enkel om snel iets uit te proberen of voor schoolse opdrachten.

Al bij al laat de Date mogelijkheid in Javascript wat te wensen over. Om 'serieus' met datums te werken gebruik je best een library zoals

- Luxon op https://moment.github.io/luxon/
- Day.js op https://day.js.org/

Wat je zeker niet doet is het wiel opnieuw uitvinden in 2021 😉

Zelf simpele objecten maken

In Javascript kun je heel eenvoudig eigen objecten aanmaken, je hoeft zelfs geen class te definiëren :

```
let student = { };                          // maak een nieuw leeg object
student.voornaam = "Jan";                   // voeg 'voornaam' property toe
student.familienaam = "Janssens";           // voeg 'familienaam' property toe


let locatie = { };                          // maak een nieuw leeg object
locatie.straat = "Koekoekstraat 70";        // voeg 'straat' property toe
locatie.postcode = "90210";                 // voeg 'postcode' property toe
locatie.gemeente = "Melle";                 // voeg 'gemeente' property toe

locatie.postcode = "9090";                  // wijzig waarde van 'postcode' property

student.adres = locatie;                    // voeg 'adres' property toe

console.log(student.voornaam+" woont in "+student.adres.postcode);
    Jan woont in 9090                       // output
```

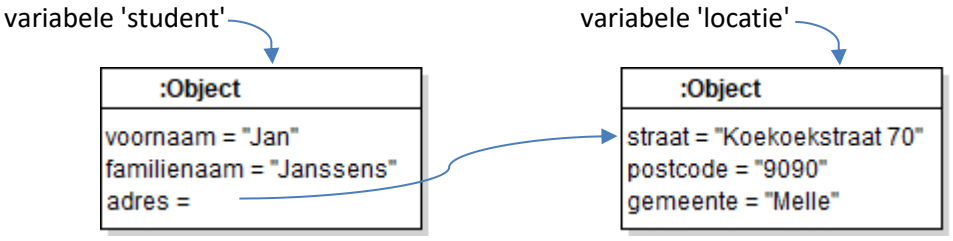
Je ziet dus dat het geen probleem is om een waarde toe te kennen aan een onbestaande property. De toekenning zal de property alsnog toevoegen aan het object en dan diens waarde instellen!

Een property verwijderen kan trouwens met delete, bv.

```
delete student.voornaam;
```

maar dit zul je niet zo vaak tegenkomen in applicatiecode (wel in sommige library code).

We kunnen de toestand in het geheugen illustreren met een UML object diagram :



Elk blokje is een object en toont de waarden van de properties van dat object. De blauwe pijltjes stellen verwijzingen voor. We hadden de strings ook als objecten kunnen tonen (incl. blauwe pijltjes ernaar!), maar dat zou het diagram niet interessanter maken, enkel drukker.

Verwijzingen zijn (net als getallen en teksten) gewoon waarden die je kunt kopiëren, bv.

```
student.adres = locatie;
```

deze regel kopieerde de verwijzing in variabele 'locatie' naar de (nieuwe) 'adres' property van het linkse object. Het resultaat hiervan is dat de 'adres' property in het linkse object, nu wijst naar het rechtse object. Achteraf is het linkse object niet blijvend afhankelijk van de 'locatie' variabele, enkel de verwijzing werd gekopieerd (het is immers een doodnormale toekenningsopdracht).

We kunnen verwijzingen naar objecten ook gewoon in een array bewaren, bv.

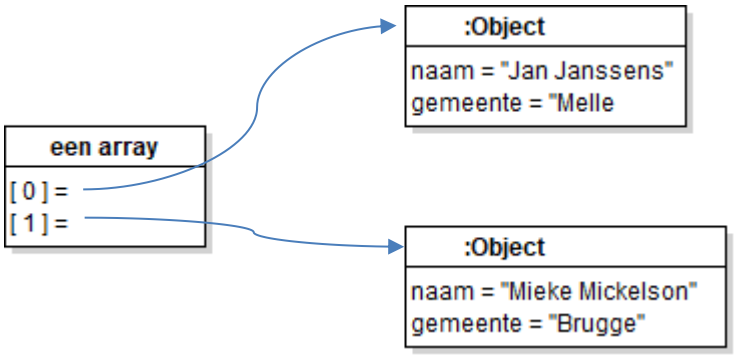
```
let p1 = {};                          // een nieuw leeg object
p1.naam = "Jan Janssens";
p1.gemeente = "Melle";

let p2 = {};                          // een nieuw leeg object
p2.naam = "Mieke Mickelson";
p2.gemeente = "Bruhhe";

let personen = [];                    // een leeg array
personen.push( p1 );                  // voeg p1 achteraan toe aan array (*)
personen.push( p2 );                  // voeg p2 achteraan toe aan array (*)
```

(*) in het array komen verwijzingen terecht naar de objecten waar 'p1' en 'p2' naar verwijzen

Een UML object diagram dat de situatie toont na de uitvoering van deze code:



De variabelen werden niet getoond op het diagram (want : niet interessant) en de voorstelling van het array is geen officiële UML notatie.

We kunnen natuurlijk ook de array[index] notatie gebruiken :

```
personen[0] = p2;          // verwijzing uit variabele 'p2' kopiëren naar positie 0 in array 'personen'
let mieke = personen[1];   // verwijzing op positie 1 in array 'personen' kopiëren naar variabele 'mieke'
```

het is immers gewoon een array (waarvan de elementen, verwijzingen zijn naar objecten)

Verwijzingen kunnen ook als parameter worden doorgegeven bij een functieoproep, bv.

```
const toonGroet = ( persoon ) => {                                    // functie 'toonGroet'
   console.log( `Hallo ${ persoon.naam } uit ${ persoon.gemeente }!` );
}

let p = {};                                                           // een nieuw leeg object
p.naam = "Jan Janssens";
p.gemeente = "Melle";

toonGroet( p );                    // functieoproep met de verwijzing uit 'p' als parameter
   Hallo Jan Janssens uit Melle!                    // output
```

Indien je een property probeert te gebruiken die een object niet heeft, dan levert dit de speciale 'undefined' waarde op. Bijvoorbeeld, deze opdracht :

```
console.log( student.postcode );
   undefined                                        // output
```

toont `undefined` op de console plaatsen.

*Als je 9090 had verwacht, daarvoor schrijven we student.**adres**.postcode*

Meestal duidt dit op een programmeerfout, bv. je maakte een schrijffout in de naam van een property of je vergiste je van object (zoals hierboven).

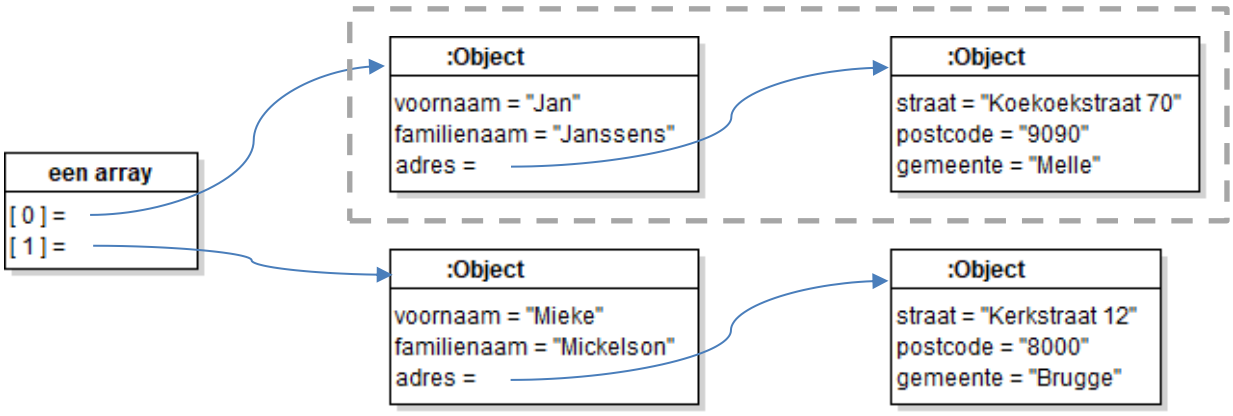
Deze 'undefined' waarde is net als 'null' een speciaal geval, waar heel wat over te vertellen valt maar dat valt buiten het bestek van deze cursus.

Er bestaan ook mogelijkheden om met constructor functies te werken, methods toe te voegen aan objecten, classes te definiëren en je kan zelfs een soort overerving gebruiken. Zoals eerder al werd aangehaald, valt dit buiten de opzet van deze cursus.

Als je zelf gaat experimenteren met nieuwe Javascript topics, hou altijd in het achterhoofd dat het een HEEL ander soort programmeertaal is dan bv. C# of Java.

Opdracht 1

Welke Javascript code moet je schrijven om de volgende situatie te creëren?



Voor de twee objecten in het grijze kader (stippellijn) kun je de code kopiëren die onder het kopje "Zelf simpele objecten maken" staat, enkel bladzijden terug.

De twee gelijkaardige objecten eronder (voor Mieke), kun je bekomen door de code voor het grijze kader nog eens te kopieren en aan te passen met Miekes gegevens.

Terzijde (niet zo belangrijk) :

Je kan de properties van een object ook aanspreken met de []-operator, waarbij dan

```
student.voornaam
student["voornaam"]
```

net hetzelfde betekenen.

Deze []-notatie voor properties is eigenlijk alleen maar nuttig in 2 gevallen

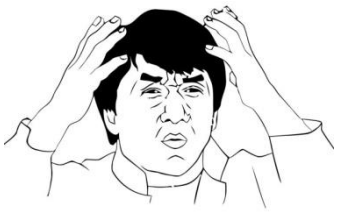
1. de naam vd property bevat speciale tekens, bv. spaties

   student.thuis of kot adres is niet toegelaten maar *student["thuis of kot adres"]* wel. Mini tip van Flip : gebruik liever *student.thuisOfKotAdres* ;)

   Dit komt soms voor in gegenereerde code, maar zelden in handgeschreven code.

2. de naam van de property komt uit een variabele, bv.

```
let propertyNaam;
if (...) {
    propertyNaam="voornaam";
} else {
    propertyNaam="geboortedatum";
}
console.log(  student[propertyNaam] );
```



Mind blown!

Let erop dat er geen quotes rond propertyNaam staan, we bedoelen immers de naam van die variabele en niet de letterlijke tekst.

Dit is bijzonder handig als je heel flexibele code moet schrijven zoals in een herbruikbare library, maar voor gewone applicatie code ga je dit niet vaak nodig hebben

Object Literals

Als je een ingewikkeld object moet maken met veel properties plus nog wat verwijzingen naar andere nieuwe objecten, dan zul je behoorlijk veel moeten typen. Bijvoorbeeld :

```
let student={};
student.voornaam="Jan";
student.familienaam="Janssens";
student.geboortedatum=new Date("1993-12-31");
enz.
```

Dit kan veel korter door de **object literal notatie** te gebruiken, bv.

```
let student = {
    voornaam : "Jan",
    familienaam : "Janssens",
    geboortedatum : new Date("1993-12-31")
};
```

Let op de dubbele punten, de komma's tussen de properties en de puntkomma op het einde!

Als je een object literal definieert hoeven diens property values niet automatisch ook literals te zijn, het kunnen bv. ook gewoon waarden zijn die uit andere variabelen komen :

```
// enkele variabelen met verwijzingen naar de DOM-tree nodes voor <input> elementen
let txtVoornaam = …
let txtFamilienaam = …
let txtGeboortedatum = …

let student={
    voornaam : txtVoornaam.value,
    familienaam : txtFamilienaam.value,
    geboortedatum : new Date(txtGeboortedatum.value)
};
```

Je ziet dat we de values van de <input> elementen gebruiken als de property waarden voor het student object.

Je kan deze notatie ook uitbreiden met objecten en arrays, bv.

```
let student={
    voornaam : "Jan",                                          // een string
    familienaam : "Janssens",
    geboortedatum : new Date("1993-12-31"),                    // een Date
    kotAdres : {                                               // een object
        straat : "Kerkstraat 12",
        postcode : "8000",
        gemeente : "Brugge"
    },
    isIngeschreven : true,                                     // een boolean
    namenVanExen : ["Sofie", "Berta", "Philip", "Albertoooo"], // een array
    aantalAutos : 0                                            // een number
};
```

Je ziet dat een property value eender welk soort waarde kan bevatten : strings, Dates, booleans, **andere objecten(!)**, numbers, **arrays(!)**, etc.

Het is belangrijk dat je deze object literal notatie gaandeweg vlot leert lezen en gebruiken, je zult ze nog vaak tegenkomen in je Javascript carrière.

Soms zul je een array van vaste objecten willen definiëren in je code. Het voorbeeld hieronder toont hoe je een 'spelers' array literal kunt definiëren die twee object literals bevat.

Voorbeeld :

```
let namen = ["Jan", "Piet", "Joris"];                  // een simpele array literal

let spelers = [ {    naam : "Lelie",                   // een complexere array literal
   score : 37
} , {
   naam : "Maja",
   score : 33
} ];
```

Kijk goed waar de onderdelen van het array beginnen en eindigen (groene kleur). De opsomming begint en eindigt met [ ] en de array elementen worden gescheiden door een komma.

Elk element van dit array is een object literal (blauwe kleur) die begint en eindigt met { }, en properties worden gescheiden door komma's.

Opdracht 2

Herschrijf de oplossing van opdracht 1 zodat je code de object literal notatie gebruikt om de twee persoon objecten te bouwen die Jan en Mieke voorstellen.

De bijbehorende adres objecten kun je in die persoon objecten stoppen, zoals bij de 'kotAdres' property op de vorige bladzijde.

Opdracht 3

Begin met een leeg project en voorzie een setup() functie voor het window *load* event.

Maak een globale variabele 'personen' met daarin een array dat de volgende info bevat :

voornaam	familienaam	leeftijd
Jan	Janssens	29
Mieke	Mickelson	31
Donald	Duck	86
Piet	Piraat	54

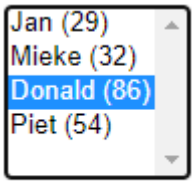
Gebruik voor elke persoon 1 object en stop al die objecten in hetzelfde 'personen' array.

In de HTML-pagina stop je een <select> element met class 'person-list'. Dit select element toont max. 5 keuzes en laat slechts 1 geselecteerd element toe (geen 'multiple' attribuut dus).

Daaronder plaats je de elementen voor de details van de geselecteerde persoon :

```
<h1>Details</h1>
<ul class="person-info">
    <li class="firstname"></li>          <!-- hier moet de voornaam komen -->
    <li class="lastname"></li>           <!-- hier moet de familinaam komen -->
    <li class="age"></li>                <!-- hier moet de leeftijd komen -->
</ul>
```

Het eindresultaat zal er zo uitzien, als Donald (86) geselecteerd werd :



Details

- Donald
- Duck
- 86

Bovenaan is er de mogelijkheid om een persoon te selecteren, momenteel is dit Donald.

Daaronder worden de details van die persoon getoond.

Dit is een typische toepassing van het **master/detail UI patroon** (ook : two panel selector) :

- de gebruiker een (master) lijst tonen met (oppervlakkige info over) veel elementen
- de gebruiker kan er eentje selecteren waarvan dan alle details getoond worden

Als je de lijst vervangt door de details (i.p.v. ze samen op het scherm te tonen), dan bekom je het one window drilldown UI patroon. Wij gaan dat hier niet toepassen, maar moeilijk is het niet : we hebben al vaker code geschreven die op de juiste momenten elementen toont of verbergt.

Eenmaal de pagina is ingeladen, wordt het <select> element opgevuld met gegevens over alle personen (te vinden in de globale variabele 'personen').

- je programma moet alle persoon objecten in het 'personen' array overlopen
- voor elke persoon moet er een <option> bijkomen in het <select> element

Per persoon wordt er een <option> element voorzien dat de voornaam toont met de leeftijd tussen haakjes. Bijvoorbeeld, voor Donald wordt deze <option> toegevoegd aan het <select> element :

```
<option> Donald (86) </option>
```

Telkens de gebruiker een andere persoon selecteert in het <select> element, moeten de details van die andere persoon in het <ul> element verschijnen :

- je programma moet reageren op het 'change' event van het <select> element
- bij elke 'change' moet het geselecteerde <option> element worden opgevraagd (of geen!)
    - op basis van deze option gekend zijn over welk persoon object het gaat (*)
    - het juiste object moet opgezocht worden in het 'personen' array
    - de informatie in dit object moet in de drie <li> elementen gezet worden
        - voornaam, familienaam en leeftijd

Je zult dus een antwoord moeten bedenken op deze vraag :

> ***Welk object uit het 'personen' array hoort bij een bepaald <option> element?***

Op het moment dat je de <option> elementen toevoegt aan het <select> element (in de setup functie), is dit makkelijk : je weet immers met welk persoon object je bezig bent op het moment dat je diens <option> toevoegt.

Als de gebruiker later een <option> selecteert moeten we nog steeds kunnen achterhalen, bij welk persoon object die <option> hoort. We kunnen extra informatie in de <option> stoppen bij het toevoegen, zodat we er later het juiste object mee kunnen terugvinden!

De makkelijkste manier om dit te doen, is elk <option> element een custom data-attribuut te geven waarin we die extra informatie kwijt kunnen.

Vermits alle objecten sowieso al in het 'personen' array staan, volstaat het om per object de indexpositie in het array te onthouden als data-attribuut in de corresponderende <option>.

Bijvoorbeeld, de persoon met voornaam "Donald" staat op indexpositie 2 in het array, dus voegen we voor hem het volgende <option> element toe :

> <option data-index="2"> Donald (86) </option>
>
> *Omdat het een <option> is hadden we ook het 'value' attribuut kunnen gebruiken hiervoor, maar een custom data-attribuut is de algemenere oplossing die altijd werkt.*

Als de selectie wijzigt naar "Mieke (32)" en we zien dat de geselecteerde <option> een data-index waarde **1** heeft, dan vinden we het corresponderende persoon object terug op index positie **1** in het personen array. Anders gezegd, we moeten `personen[1]` hebben en daar vinden we het object dat Mieke beschrijft.

Mocht je je afvragen :

- waarom gebruiken we niet gewoon de voornaam om het juiste object op te zoeken?
    - we gaan ervan uit dat voornamen niet uniek zijn, da's realistischer en interessanter 😉

- waarom onthouden we niet gewoon het object zelf in de <option> ?
    - we kunnen in een custom-data attribuut enkel een string bewaren, geen object

Een andere mogelijkheid, die eenvoudig maar niet erg praktisch is : we stemmen de onderlinge volgorde van de <option> elementen af op de volgorde van de objecten in het array.

- De tweede option komt dan overeen met het tweede object in het 'personen' array, simpel!
- dit lukt maar het zal in praktijk niet handig zijn want :
    - als we de mogelijkheid willen geven om de lijst te sorteren op basis van verschillende criteria (bv. door een tabel met klikbare kolommen te gebruiken), dan wordt het lastig om de volgorde in de lijst en het array gelijk te houden. Zeker als we een library gebruiken voor de sortering die daar niet op voorzien is.
    - je zult voor de geselecteerde <option> moeten achterhalen het hoeveelste kind dit is in het <select> element, wat iets omslachtiger is dan je zou denken
        - zie https://stackoverflow.com/questions/5913927/get-child-node-index

Nog een andere mogelijkheid, die veel praktischer is : we voorzien in elk persoon object een speciale property (bv. 'id') die voor elke object een unieke waarde heeft.

> Vaak hebben objecten dit sowieso al omdat ze langs de serverkant in een databank met primary keys zitten.
> Of je kan zelf aan de browserkant een tijdelijk uniek id getal toevoegen die niet voor de server bestemd is.
> Het persoon object voor jan krijgt dan bv. id=3456 en dat voor **Donald** krijgt **id=36731**.

We stoppen die unieke waarde ook in het corresponderende <option> element via een custom data-attribuut :

> <option **data-id="36731"**> Donald (86) </option>

Zo kunnen we makkelijk het juiste persoon object terugvinden in het array : we overlopen alle persoon objecten totdat we een match vinden met id waarde 36731.

Opdracht 4

Start met een leeg project dat **een setup functie** zodra de pagina is ingeladen.

Voeg in de Javascript file **een functie toonPersoon** toe.

Deze functie heeft een parameter 'persoon', dit is een verwijzing naar een object met 5 properties :

- voornaam (string)
- familienaam (string)
- geboortedatum **(Date)**
- email (string)
- aantalKinderen **(number)**

Ze toont voor elk van deze properties een geschikte tekstvoorstelling op de console. Deze komen alle 5 naast elkaar op dezelfde regel, gescheiden door komma's.

Probeer je toonPersoon functie uit met het onderstaande code fragment (bv. op de console in de Chrome Developer Tools) :

```
let p = {
   voornaam : "Jan",
   familienaam : "Janssens",
   geboortedatum : new Date("2010-10-15"),
   email : "jan@example.com",
   aantalKinderen : 0
};


toonPersoon( p );
   Jan,Janssens,2010-10-15,jan@example.com,0          // output
```

Als dit goed werkt kun je dit fragment weer wissen, we hebben het verder niet meer nodig.

Definieer **een globale variabele 'personen'** bovenaan je Javascript file. Geef deze variabele een leeg array als beginwaarde.

Schrijf **een functie vulMetDemoData**.

Deze functie heeft geen parameters en vult het 'personen' array met de volgende gegevens :

voornaam	familienaam	geboortedatum	email	aantal kinderen
Jan	Janssens	15 oktober 2010	jan@example.com	0
Mieke	Mickelson	1 januari 1980	mieke@example.com	1
Piet	Pieters	31 januari 1970	piet@example.com	2

Doe dit door 3 objecten aan het globale 'personen' array toe te voegen, één object per persoon.

Schrijf **een functie toonAllePersonen**.

Deze functie heeft een 'personen' parameter, dit is een array van persoon objecten. Of correcter maar saaier : dit is een verwijzing naar een array dat verwijzingen naar objecten bevat. Elk van die objecten heeft dezelfde 5 properties van hiervoor.

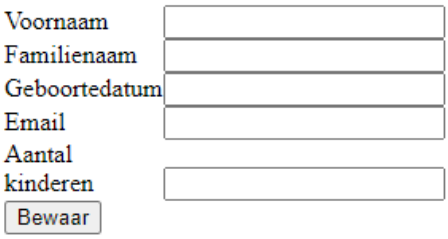
De functie overloopt het 'personen' array en roept voor elke persoon object, de toonPersoon(...) functie op.

Stop dit codefragment in je **setup functie** :

```
vulMetDemoData();
toonAllePersonen();
  Jan,Janssens,2010-10-15,jan@example.com,0                  // output
  Mieke,Mickelson,1980-01-01,mieke@example.com,1             // output
  Piet,Pieters,1970-12-31,piet@example.com,2                 // output
```

Herlaad nu de pagina in de browser, als je output klopt dan heb je hoogstwaarschijnlijk alle voorgaande stappen correct uitgevoerd.

Bewerk nu de HTML pagina zodat ze de volgende inhoud toont (de CSS-opmaak is niet zo belangrijk)



De tekstvelden hebben de volgende id attribuutwaarden :

txtVoornaam, txtFamilienaam, txtGeboortedatum, txtEmail, txtAantalKinderen

en de button heeft id-waarde btnBewaar.

Schrijf **een bewaarPersoon functie**.

Deze functie heeft geen parameters.

De functie zal uitgevoerd worden telkens de gebruiker op de "Bewaar" knop drukt. Registreer ze dus in je setup functie als click event listener bij de "Bewaar" knop.

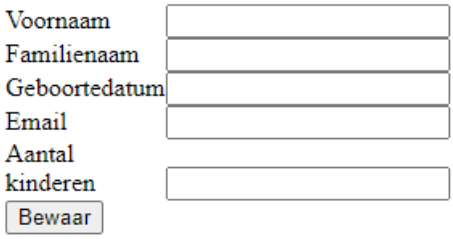
Telkens de gebruiker op de 'Bewaar' knop klikt, doet het programma het volgende :

1. maak een nieuw persoon object, op basis van de teksten in de invoervelden.
    - denk eraan : geboortedatum (Date) en aantalKinderen (Number), geen strings!
2. voeg dit nieuwe object toe, achteraan het 'personen' array
3. roep toonAllePersonen() op
    - dit toont alle bewaarde personen op de console, inclusief het nieuwe persoon object
4. maak alle invoervelden leeg

Voeg de nodige code toe aan je bewaarPersoon functie om dit voor elkaar te krijgen.

Als je klaar bent kun je je oplossing met de volgende stappen uitproberen :

Stap 1
Laad de pagina opnieuw in, er wordt een leeg invulformulier getoond



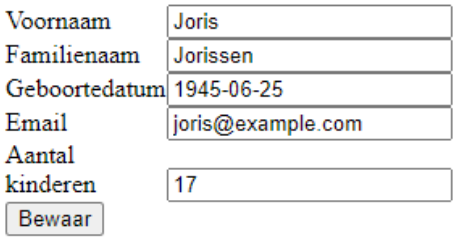
op de console staat nu :

```
Jan,Janssens,2010-10-15,jan@example.com,0
Mieke,Mickelson,1980-01-01,mieke@example.com,1
Piet,Pieters,1970-12-31,piet@example.com,2
```

} output na inladen pagina

Stap 2

Vul nu het formulier als volgt :



en klik op "Bewaar".

Op de console zou nu deze output moeten staan :

```
Jan,Janssens,2010-10-15,jan@example.com,0
Mieke,Mickelson,1980-01-01,mieke@example.com,1
Piet,Pieters,1970-12-31,piet@example.com,2
Jan,Janssens,2010-10-15,jan@example.com,0
Mieke,Mickelson,1980-01-01,mieke@example.com,1
Piet,Pieters,1970-12-31,piet@example.com,2
Joris,Jorissen,1945-06-25,joris@example.com,17
```

output na inladen pagina

output na klik op "Bewaar"

Je ziet hopelijk dat telkens alle personen uit het 'personen' array getoond worden. Op de laatste regel staat Joris erbij en zijn informatie is correct overgenomen uit de invoervelden.

Check tot slot nog eens je oplossing, bevat het nieuwe object voor Joris daadwerkelijk

- een verwijzing naar een Date object in de geboortedatum property? (geen string!)
- een number waarde in de aantalKinderen property? (geen string!)

Je kunt dit makkelijk nagaan in de Chrome Developer Tools (bv. bij de debugger).