Javascript – deel 08

Deze les behandelt javascript objecten en hun tekstuele voorstelling in het JSON formaat.

Permanente evaluatie

In de volgende les kan je docent je oplossingen van de opdrachten opvragen en laten meetellen voor je permanente evaluatie.

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 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 apart in een .zip file.

Date objecten

Het huidige tijdstip (datum en tijd) kun je achterhalen met

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

De documentatie voor javascript Date objecten vind je op

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

Let hier vooral op de verschillende constructoren om Date objecten aan te maken alsook de diverse get en set methods voor uren, maanden, jaren, etc.

Om 'serieus' met data te werken gebruik je best een library, bv.

- moment.js op http://momentjs.com/
- xdate op http://arshaw.com/xdate/

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

Syntax:

```
new Date(value);
new Date(dateString);
new Date(year, month[, day[, hour[, minutes[, seconds[, milliseconds]]]]]);
```

Voorbeelden:

```
var verjaardag = new Date('December 17, 2020 03:24:00');
var verjaardag = new Date('2020-12-17T03:24:00');
var verjaardag = new Date(2020, 11, 17);
var verjaardag = new Date(2020 11, 17, 3, 24, 0);
```

Als je een goeie tekstvoorstelling van een Date object nodig hebt (bv. voor in een UI), zul je die zelf moeten opbouwen (of beter : een library gebruiken). Om snel iets uit te proberen kun je echter de toString, toISOString, toDateString en toTimeString methods gebruiken

Voorbeelden

toString():

```
const event = new Date('2020-04-10T23:15:30');

console.log(event.toString());
// expected output: "Fri Apr 10 2020 23:15:30 GMT+0200 (Midden-Europese zomertijd)"
// (note: your timezone may vary)
```

toISOString()

```
const event = new Date('2020-04-10T14:48');
console.log(event.toString());
// expected output: "Fri Apr 10 2020 14:48:00 GMT+0200 (Midden-Europese zomertijd)"
// (note: your timezone may vary)

console.log(event.toISOString());
// expected output: "2020-04-10T12:48:00.000Z"
// The timezone is always zero UTC offset, as denoted by the suffix "Z".
```

toDateString()

```
const event = new Date("2020-04-10T10:10:40");

console.log(event.toString());
// expected output: "Fri Apr 10 2020 10:10:40 GMT+0200 (Midden-Europese zomertijd)"
// (note: your timezone may vary)

console.log(event.toDateString());
// expected output: "Fri Apr 10 2020"
```

toTimeString()

```
const event = new Date("2020-04-10T12:10:40");

console.log(event.toTimeString());
// expected output: "12:10:40 GMT+0200 (Midden-Europese zomertijd)"
// (note: your timezone may vary)
```

Zelf de datum opbouwen

```
let event = new Date('2020-04-10T12:10:30');

console.log(event.getDate() + "-"+ (event.getMonth()+1) + "-" + event.getFullYear() + "  " + event.getHours()
+":"+event.getMinutes() +":"+event.getSeconds());
// expected output: "10-4-2020  12:10:30"
// getMonth() geeft de maand als een getal tussen 0 en 11 (januari is maand 0 daarom + 1)
```

```
let event = new Date(); // huidige datum

console.log(event.getDate() + "-"+ (event.getMonth()+1) + "-" + event.getFullYear() + "  " + event.getHours()
+":"+event.getMinutes() +":"+event.getSeconds());;
// expected output: "15-4-2020  16:37:26"
```

Opdracht

Bereken het aantal dagen tussen jouw verjaardag en de huidige datum. Het aantal dagen wordt weergegeven in de console.

Eigen objecten maken

Tot voor kort kwam het 'class' concept niet expliciet voor in javascript. Er zijn 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.

Vanaf ECMAscript 6 ondersteunt javascript gelukkig wel klassen, maar het zal nog enige tijd duren eer deze versie overal zijn intrede heeft gedaan. Voorlopig houden we het dus op rechtstreeks met objecten werken, zonder klassen.

Je kan heel eenvoudig eigen objecten aanmaken, i.t.t. bv. Java hoef je nml. geen klassen te definieren.

```
let student1 = {}; // een leeg object
student1.voornaam="Jan";
student1.geboorteDatum=new Date("1993-12-31");
student1.adres={};
student1.adres.gemeente="8500";

console.log(student1.voornaam+" woont in "+student1.adres.gemeente);
```

Je ziet dus dat je 'zomaar' een property kan toevoegen aan een object!

Een property verwijderen kan trouwens met delete, bv.

```
delete student1.voornaam;
```

maar dit zul je zelden nodig hebben.

Indien je een property probeert te gebruiken die niet aanwezig is, levert dit de speciale waarde 'undefined' op, bv.

```
console.log(student1.postcode)
```

zal undefined op de console plaatsen.

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

Je kan de properties van een object ook aanspreken met de []-operator, merk op dat bv.

```
student1["voornaam"]
student1.voornaam
```

net hetzelfde betekenen.

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

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

   student1.thuis of kot adres is niet toegelaten maar *student1["thuis of kot adres"]* wel.
   Tip van Flip : gebruik liever *student1.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( student1[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

Er bestaan ook mogelijkheden om met constructor functies te werken, methods toe te voegen aan objecten en je kan zelfs een soort overerving gebruiken. Dit valt echter buiten de opzet van deze cursus. Als je zelf wat gaat experimenteren, hou altijd in het achterhoofd dat javascript een HEEL anders soort programmeertaal is dan bv. Java of C#

Object Literals

In de marge:lLiterals represent values in JavaScript. These are fixed values—not variables—that you literally provide in your script.

Als je een ingewikkeld object moet maken met veel properties, en eventueel nog wat verwijzingen naar andere nieuwe objecten, zul je nogal veel moeten typen. Bv.

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

Dit kan korter door de object literal notatie te gebruiken:

An object literal is a list of zero or more pairs of property names and associated values of an object, enclosed in curly braces ({}).

bv. let student = {
voornaam : "Jan"**,**
familienaam : "Janssens"**,**
geboorteDatum : new Date("1993-12-31")
}

Let op de dubbele punten en de komma na elke property value!

Merk op dat het object als literal gedefinieerd wordt, maar dat de property values daarom geen literals hoeven te zijn. Het kunnen ook gewoon waarden zijn die uit andere variabelen komen, bv.

```
let student={
    voornaam : txtVoornaam.value,
    familienaam : txtFamilienaam.value,
    geboorteDatum : new Date(txtGeboorteDatum.value)
}
```

(de txtXYZ variabelen wijzen bv. naar DOM-tree nodes van tekstvelden uit een invulformulier)

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

```
let student={
    voornaam : "Jan",
    familienaam : "Janssens",
    geboorteDatum : new Date("1993-12-31"),
    adres : {                                                    // een object
        straat : "Kerkstraat 13",
        postcode : "8500",
        gemeente : "Kortrijk"
    },
    isIngeschreven : true,
    namenVanExen : ["Sofie", "Berta", "Philip", "Albertoooo"], // een array
    aantalAutos : 2
}
```

JSON

JSON staat voor **J**ava**S**cript **O**bject **N**otation en is een tekstvoorstelling voor objecten. Kort gezegd, het is eigenlijk gewoon een String met daarin de object literal notation van hierboven!

Deze JSON tekstvoorstelling wordt vaak gebruikt om

1. gegevens uit te wisselen
2. gegevens te bewaren

JSON als **data uitwisselingsformaat**

- bv. in een webapplicatie tussen de client code in de browser en de server code op de server
  dus als string verpakt in een http request of http response
- bv. als uitwisselingsformaat tussen verschillende programma's
  export/import functionaliteit
- bv. als input formaat voor de (gevorderde) eindgebruiker
  dikwijls voor programma configuratie

JSON als **data opslagformaat**

- bv. in een cookie, localstorage, etc.

Een alternatief voor JSON is XML, beide hebben echter hun sterktes en zwaktes.

Het voordeel van JSON is dat het veel toegankelijker is : het is leesbaarder, makkelijker in te typen en elke javascript programmeur kent het al omdat het gewoon de *object literal notation* is.

XML is dan weer beter voor zeer gestructureerde data (i.e. met een achterliggend schema) of waarop complexe bewerkingen moeten gebeuren (bv. XSLT).

Je kan makkelijk een object omzetten naar een JSON String :

```
let tekst = JSON.stringify(student1);
```

En in de omgekeerde richting, een object maken op basis van een JSON tekst

```
let student1 = JSON.parse(tekst);
```

Opdracht

Schrijf een kort programma waarin je

1. de variabele student1 definieert met een complex object zoals hierboven
2. een JSON string voor dit object bouwt en op de console zet

Kopieer de bekomen tekst vanop de console (CTRL-C) voor het vervolg

Maak een tweede programma waarin je

1. een object maakt a.d.h.v. deze gekopieerde JSON String
2. een property van dit object op de console zet

Ga na dat het object qua properties en property values identiek is aan het object uit het eerste programma.

Vraag :

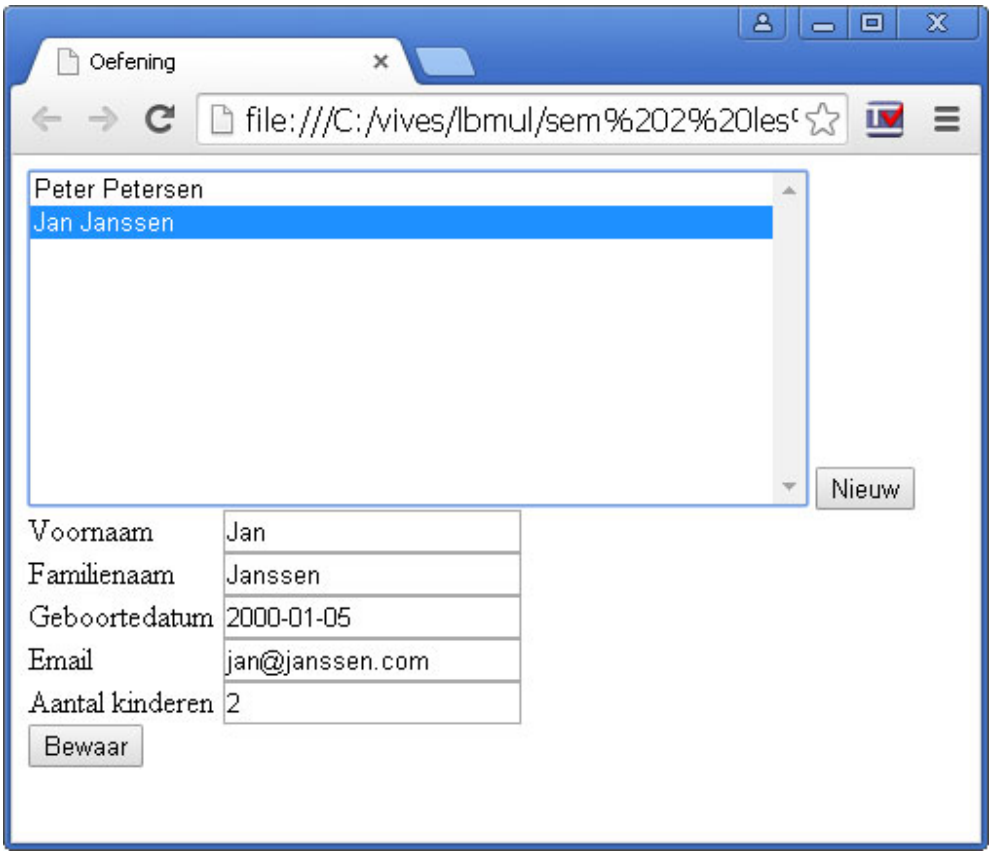
Welk formaat wordt er eigenlijk gebruikt om het Date object?

Vraag :

Wat gebeurt er als eenzelfde object meermaals voorkomt bij het opbouwen van de JSON tekst?

Opdracht contact manager

Breid de oplossing van de form oefening uit zodat je de data van meerdere personen kunt bijhouden.



UITLEG: https://kuleuven.mediaspace.kaltura.com/media/uitleg+Contact+Manager/1_0466x30g (video)

Start bestand: “start contact manager opdracht.zip” zie Toledo.

Maak een UI met het invulformulier, een lijst met personen en 2 knoppen : Bewaren en Nieuw.

Voor de UI-lijst gebruik je een <select> element met bv. attribuut size="10", je krijgt dan een lijst ipv een dropdown element.

Voor elke persoon in de lijst zul je een <option> element moeten toevoegen. Geef dit <option> element een id attribuut wiens waarde de index van de persoon in een ‘personen’ array is. Normaliter zou je hiervoor een custom attribuut gbruiken ipv het id-attribuut.

Wanneer een persoon in de lijst geselecteerd wordt, worden diens gegevens in de form getoond. Registreer hiervoor een change event listener bij het select element.

Als er op Nieuw wordt geklikt, maken we het formulier leeg.

Als er op Bewaren geklikt wordt moet volgende gebeuren :

igv een nieuwe persoon

maak een nieuw persoon object
stop de ingevulde gegevens (na validatie!) in dit object
voeg dit nieuwe object toe, achteraan in de personen lijst

igv een bestaande persoon uit de lijst

pas de properties van het bijbehorende object aan
in overeenstemming met de ingevulde data (na validatie!)
toevoegen aan het array is natuurlijk niet nodig in dit geval

Je zult dus in je programma op een of andere manier moeten achterhalen welke persoon in de form getoond wordt. Een makkelijke manier is gewoon de index van het persoon object in de personen array bij te houden.

Indien er een nieuwe persoon bewerkt wordt gebruik je bv. een ongeldige index zoals -1, dan kan je bij het bewaren deze index controleren om te weten wat je moet doen (indien -1 moet je een object toevoegen en anders niet).