Javascript – deel 10

Deze les behandelt het JSON-tekstformaat.

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 10' 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

JSON

Bij een webapplicatie is het soms nodig om wat data uit een Javascript programma als tekst voor te stellen. Deze tekst is niet voor de eindgebruiker bestemd, maar maakt deel uit van de interne keuken van de applicatie. Bijvoorbeeld om de data mee te sturen in een HTTP response, of te bewaren in een "permanente" opslagplaats in de browser (bv. *local storage*).

Het is dan de bedoeling deze tekst later (of elders) te gebruiken om de data te reconstrueren.

Een zeer bekend tekstformaat hiervoor is JSON. Deze afkorting staat voor **J**ava**S**cript **O**bject **N**otation en het is **een tekstvoorstelling voor objecten** en andere waarden (array, string, number, boolean, …)

Kort gezegd, het is gewoon een string met daarin de object literal notatie uit het vorige deel!

Je kan makkelijk een object omzetten naar een JSON-tekst met **JSON.stringify()** :

```
let persoon = {
    voornaam : "Jan",
    familienaam : "Janssens",
    aantalKinderen : 0
};

let tekst = JSON.stringify( persoon );

console.log( tekst );
  {"voornaam":"Jan","familienaam":"Janssens","aantalKinderen":0}          // output
```

Zoals je kunt zien aan de output, is de JSON-tekst gewoon de Javascript code voor een object literal!

Let op de aanhalingstekens rond de propertynamen! In Javascript is dit niet verplicht bij een object literal, maar in een correcte JSON-tekst wel. Ook mag je de aanhalingstekens (") niet vervangen door accenten ('), zelfs niet rond een string literal.

Met **JSON.parse()** kunnen we een Javascript object reconstrueren op basis van een JSON-tekst :

```
let tekst = '{"voornaam":"Mieke", "familienaam":"Mickelson", "leeftijd":32}';
                                                    // let op de " en ' afwisseling!
let p = JSON.parse( tekst );

console.log( p.leeftijd );
  32                                                                    // output
```

De return value van de JSON.parse() oproep is hier een object met drie properties (voornaam, familienaam en leeftijd) dat de info voor Mieke bevat.

In dit voorbeeld staat de JSON-string letterlijk in onze broncode, in een echt programma is dat natuurlijk niet het geval.

Een JSON-string komt doorgaans binnen in ons programma via een HTTP-response (als antwoord op een AJAX call) of uit een opslagplaats (zoals localstorage in de browser).

We kunnen ook arrays en eenvoudige waarden omzetten naar tekst en reconstrueren, bijvoorbeeld

```
let array = [1, "hallo", false];
console.log( JSON.stringify( array ) );
  [1, "hallo", false]                                          // output

let tekst = '[1, "hallo", false]';                             // let op de " en ' afwisseling!
let reconstructed = JSON.parse( tekst );
console.log( reconstructed[1] );
  hallo                                                        // output
```

Vraag : Wat is die "JSON" eigenlijk in een opdracht als JSON.stringify(…) of JSON.parse(…) ?

Zoals gezegd, wordt de JSON-tekstvoorstelling wordt vaak gebruikt

1. als data uitwisselingsformaat
    - bv. de browser en server sturen elkaar gegevens als tekst
2. als data opslagformaat
    - bv. een browserspelletje bewaart de instellingen en hoogste score als tekst

JSON als **data uitwisselingsformaat**

- bv. in een webapplicatie wisselen de client code (in de browser) en de server code (op de webserver) gegevens met elkaar uit, als JSON-tekst in een HTTP request of response
- bv. als uitwisselingsformaat tussen verschillende programma's
  export/import functionaliteit

JSON als **data opslagformaat**

- bv. om gegevens tussen twee sessies te onthouden, bv. in een cookie, local storage, etc.
- bv. als formaat voor de (gevorderde) eindgebruiker/programmeur
  in het configuratiebestand van een programma, in build scripts, …

Een alternatief voor JSON is XML (een voorbeeld), beiden hebben hun sterktes en zwaktes.

Het voordeel van JSON is dat het zeer toegankelijk is : het is leesbaar, makkelijk in te typen en elke Javascript programmeur kent het al omdat het gewoon de *object literal notatie* is.

XML is minder vlot te lezen en gebruikt veel meer tekst om informatie voor te stellen, maar het is wel beter geschikt voor zeer complexe data (die met een XML schema kan gevalideerd worden) of als er complexe bewerkingen op de data moeten uitgevoerd worden (XSLT is hiervoor heel handig).

JSON-tekst voor samengestelde waarden

In de voorgaande voorbeelden hebben we de JSON-tekst gemaakt voor eenvoudige waarden :

- een object met wat string en number properties
- een array met string, number en boolean waarden

Wat zou er gebeuren als we een object nemen waarvan sommige properties naar andere objecten wijzen? Wat met een array van objecten? Of een array van arrays (*yep, dat bestaat*)?

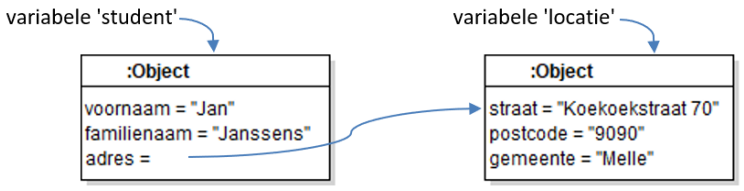
De volgende demonstratie probeert dit uit met een objecten dat naar een ander object verwijst.

Demonstratie

Veronderstel de onderstaande code uit een vorig deel :

```
let student = { };
student.voornaam = "Jan";
student.familienaam = "Janssens";


let locatie = { };
locatie.straat = "Koekoekstraat 70";
locatie.postcode = "9090";
locatie.gemeente = "Melle";

student.adres = locatie;
```

Deze code produceert de volgende situatie in het geheugen :



Kijken we nu eens naar de JSON-tekstvoorstelling van 'student' :

```
const tekst = JSON.stringify( student );
console.log( tekst );
{"voornaam":"Jan","familienaam":"Janssens","adres":{"straat":"Koekoekstraat 70","postcode":"9090","gemeente":"Melle"}}   // output
```

Als we de output mooier formatteren en inkleuren, wordt de structuur van de JSON-tekst duidelijk :

```
{
   "voornaam":"Jan",
   "familienaam":"Janssens",
   "adres": {
      "straat":"Koekoekstraat 70",
      "postcode":"9090",
      "gemeente":"Melle"
   }
}
```

Je ziet dat het '**locatie**' object automatisch werd opgenomen in de JSON-tekst van het '**student**' object!

Indien dit 'locatie' object ook properties had die naar andere objecten wijzen, dan zouden die andere objecten automatisch ook in de JSON-tekst van 'student' opduiken.

De JSON.stringify(P) oproep produceert een JSON tekst waarmee de meegegeven parameter P kan gereconstrueerd worden, inclusief alle waarden die bereikbaar zijn vanuit P !

Dit "bereikbaar" zijn mag je ruim interpreteren : de JSON-tekst zal ook waarden beschrijven die bereikbaar zijn via deze bereikbare waarden. En ook alle waarden die bereikbaar zijn via waarden die bereikbaar zijn via waarden die bereikbaar zijn.

- De JSON-tekstvoorstelling van een **array** zal dus ook **alle elementen** bevatten in dat array,
  - maar **ook alle waarden die vanuit die elementen bereikbaar zijn** enz.

- de JSON-tekstvoorstelling van een **object** zal dus ook **alle property waarden** bevatten
  - maar **ook alle waarden die van daaruit bereikbaar zijn** enz.

Kortom, als je een diagram zou tekenen dan wordt alles meegenomen dat bereikbaar is door de blauwe pijltjes te volgen.

Opdracht 1

Voorspel de output van de onderstaande code :

```
let p1 = {
    naam : 'Jan Janssens',
    gemeente : 'Melle',
};

let p2 = {
    naam : 'Mieke Mickelson',
    gemeente : 'Bruhhe',
};

let personen = [];
personen.push( p1 );
personen.push( p2 );

console.log(    JSON.stringify( personen )    );
```

Probeer dit eerst zelf te voorspellen door je JSON-tekst in een document te schrijven.

Voer de code pas daarna uit op de console en vergelijk je voorspelling met de output. Let daarbij vooral op de aanhalingstekens en accenten!

Opdracht 2

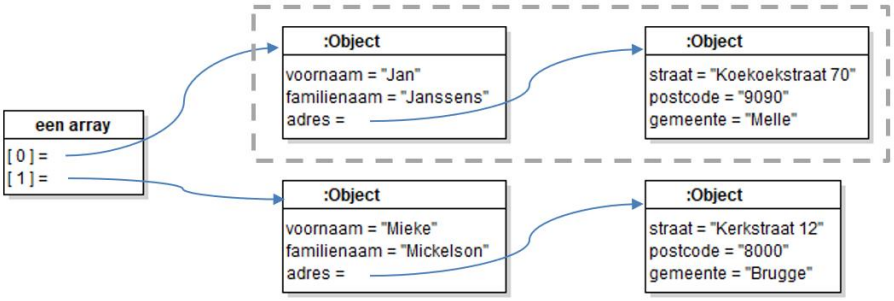
Voorspel de output van onderstaande code :

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

console.log(    JSON.stringify( persoon )    );
```

Voer de code pas daarna uit op de console en vergelijk je voorspelling met de output.

Opdracht 3

Wat is de JSON-tekst die overeenkomt met het linkse array uit deze situatie :



Als je je oplossing wil controleren a.d.h.v. een experiment op de console : je vindt de code voor deze situatie in de oplossing van een opdracht uit 'Javascript deel 09'.

Opdracht 4 – colorpicker uitbreiding met data-color

Neem de oplossing van opdracht 'Colorpicker uitbreiding' uit 'Javascript deel 08' erbij.

Pas deze oplossing aan, zodat elke b-swatch nog maar één enkel data-attribuut 'data-color' meer gebruikt. Dit ene attribuut vervangt de drie aparte data-red, data-green en data-blue attributen.

Bijvoorbeeld, als de originele oplossing dit in de DOM-tree stopte :

```
<div class="swatch" data-red="34" data-green="123" data-blue="90" style="…">…</div>
```

dan moet er nu dit in de DOM-tree terechtkomen :

```
<div class="swatch" data-color="???" style="…">…</div>
```

Op de plaats van de **???** komt er een JSON-tekst die een object beschrijft met een 'red', 'green' en 'blue' property.

De wijzigingen die nodig zijn :

- Op het moment dat ons programma een b-swatch toevoegt, maakt het eerst een simpel kleur object met drie properties waarin de RGB waarden terechtkomen. Daarna wordt van dit kleur object een JSON-tekst gemaakt en die string wordt als waarde voor het **data-color** attribuut gebruikt.
- Als de gebruiker op een b-swatch klikt, haalt ons programma de JSON-tekst uit het custom data-color attribuut. Daarmee reconstrueert het programma vervolgens het kleur object met de RGB waarden en tenslotte worden die waarden gebruikt om de sliders in te stellen.

Opdracht 5

In deze opdracht belichten we twee potentiële problemen bij het gebruik van het JSON formaat.

Opdracht 5a

Schrijf code die het onderstaande 'origineel' object omzet naar een JSON-tekst en die tekst bewaart in een variabele 'tekst' (TODO 1).

Reconstrueer er vervolgens een nieuw object 'kopie' mee in onderstaande code (TODO 2)

```
let origineel  = {
   naam : "Jan",
   geboortedatum : new Date("1993-12-31")
};

let tekst = TODO 1 : hier aanvullen

let kopie = TODO 2 : hier aanvullen

console.log( origineel.geboortedatum );
   Sun May 09 2021 23:52:43 GMT+0200 (Central European Summer Time)        // output
console.log( kopie.geboortedatum );
   2021-05-09T21:52:43.296Z                                                // output
console.log( origineel.geboortedatum.getFullYear() );
   2021                                                                    // output
console.log( kopie.geboortedatum.getFullYear() );
   Uncaught TypeError: kopie.getFullYear is not a function                 // output
```

Wat loopt er fout bij de onderste opdracht, waarom krijgen we een foutmelding?

Er is blijkbaar een belangrijk verschil tussen het origineel en de gereconstrueerde kopie.

Probeer de bovenstaande code in het console venster van je browser. Typ daarna eens de volgende twee expressies in :

typeof origineel.geboortedatum
typeof kopie.geboortedatum

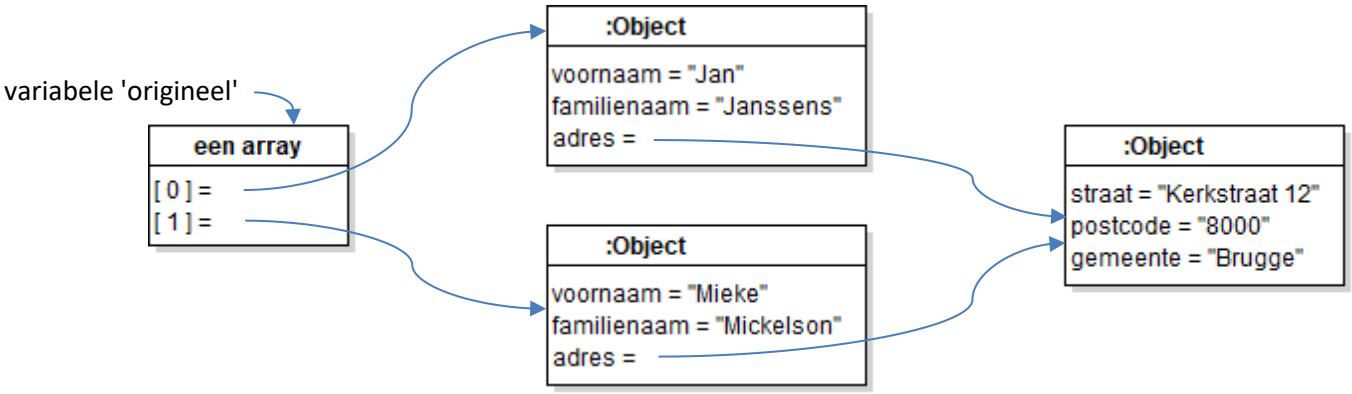
De **typeof** bewerking vertelt ons wat voor soort waarde een expressie voorstelt tijdens de uitvoering

Wat is het type van (de waarde in) de geboortedatum property, voor 'origineel' en 'kopie'?

Wat is dus het verschil tussen het origineel en de kopie?

Opdracht 5b

Veronderstel deze situatie :



Het 'origineel' array verwijst naar twee objecten, die Jan en Mieke voorstellen. Rechts op het diagram zie je dat deze twee objecten hun adres object delen, Jan is blijkbaar bij Mieke ingetrokken?

> *Terzijde, we hadden ook gewoon de straat / postcode /gemeente waarden uit Miekes adres object kunnen kopiëren naar Jans adres object. Dan hadden ze elk een eigen adres object bekomen, maar met dezelfde inhoud.*

Neem de oplossing van opdracht 1 uit Javascript deel 9 erbij, waarin Jan en Mieke een eigen adres object hebben. Veronderstel dat de variabele 'origineel' van hierboven, naar het array verwijst uit die oplossing.

Pas de code uit die oplossing aan, zodat ze bovenstaande situatie creëert.

Je kon makkelijk Jan bij Mieke laten intrekken met een toekenning als

```
origineel[0].adres = origineel[1].adres;          // origineel [0] wijst naar Jans object
                                                  // origineel [1] wijst naar Miekes object
```

De adres property van Jans object zal dan dezelfde verwijzing bevatten als Miekes object, m.a.w. ze wijzen dan beiden naar hetzelfde adres object!

Probeer nu eens de volgende code uit :

```
origineel[1].adres.straat = "Kerkstraat 14";      // Mieke verhuist naar de buren

console.log( origineel[1].adres.straat );
   Kerkstraat 14              // output           // Mieke woont bij de buren

console.log( origineel[0].adres.straat );
   Kerkstraat 14              // output           // Jan woont nu blijkbaar ook bij de buren!
```

Jans en Miekes objecten delen nu hetzelfde adres, er is immers maar 1 adres object voor beide. Als we origineel[0].adres.straat zouden aanpassen om Jan te doen verhuizen, zal ook Mieke automatisch mee verhuisd worden.

Laten we nu eens kijken naar de gereconstrueerde objecten in de kopie.

Voer de volgende code uit :

```
let tekst = JSON.stringify( personen );      // we maken via de JSON-tekst een kopie
let kopie = JSON.parse( tekst );

console.log( kopie[0].adres.straat );        // kopie[0] is een kopie van Jans object
   Kerkstraat 14           // output          // Jans kopie woont bij de buren
console.log( kopie[1].adres.straat );        // kopie[1] is een kopie van Miekes object
   Kerkstraat 14           // output          // Miekes kopie woont bij de buren
```

Zo te zien wonen beide kopies dus bij de buren in Kerkstraat 14.

Dat was bij het origineel ook het geval, dus op het eerste zicht lijkt het een perfecte kopie te zijn?

(spoiler alert : het is geen perfecte kopie ☹)

Probeer onderstaande code eens uit :

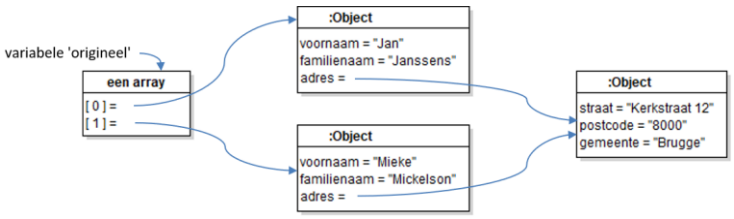
```
kopie[1].adres.straat = "Markt 3";                  // Miekes kopie verhuist naar de markt

console.log( kopie[0].adres.straat );
  Kerkstraat 14          // output                  // Jans kopie is NIET mee verhuisd!
```

Zo te zien is verhuist Jans kopie **niet** automatisch mee met Miekes kopie.

Nochtans was dit bij de originele **wel** het geval! Blijkbaar is de kopie dan toch niet perfect, helaas.

Teken hieronder een UML object diagram dat de situatie toont voor het 'kopie' array, Miekes kopie, Jans kopie en de bijhorende adres objecten.

In de originele situatie deelden de objecten voor Jan en Mieke eenzelfde adres object :



Wat is er met dit gedeelde adres gebeurd bij het kopiëren, hoe is die informatie voorgesteld in de gereconstrueerde kopie?

Onthou :

De automatische omzetting van een **kluwen objecten** naar een JSON-tekst en terug, verloopt niet altijd vlekkeloos :

1. Sommige objecten worden als string teruggezet i.p.v. als object van de juiste soort
    - bv. voor Date objecten wordt de toISOString() tekst gebruikt

2. als het **kluwen** meermaals een verwijzing naar hetzelfde object bevat, dan zal dit object in de gereconstrueerde versie meermaals worden teruggezet

Je kan deze problemen oplossen door op het JSON omzettingsmechanisme in te grijpen

- we kunnen een object een .toJSON() method geven
    - om de geproduceerde JSON-tekst aan te passen
    - JSON.stringify() zal deze gebruiken bij het opstellen van de JSON-tekst
- we kunnen een optionele *reviver* parameter meegeven aan JSON.parse()
    - om het juiste soort object te reconstrueren op basis van de JSON-tekst

maar dat is niet zo eenvoudig en valt buiten het bestek van deze cursus.

In een eenvoudige applicatie kun je het eerste probleem aanpakken, door simpelweg een nieuw Date object te maken op basis van de teruggezette string en daarna de string te vervangen.

Bijvoorbeeld, Jans kopie heeft een string waarde voor de geboortedatum property i.p.v. een verwijzing naar een Date object. We kunnen dit als volgt rechtzetten :

```
let datumAlsTekst = kopie[0].geboortedatum;   // de teruggezette geboortedatum ISO string in kopie

let datum = new Date( datumAlsTekst );        // een nieuw Date object op basis van die ISO string

kopie[0].geboortedatum = datum;               // teruggezette ISO string vervangen door nieuw Date object
```