Javascript – deel 03

Deze les introduceert de DOM-tree, toont hoe je elementen in een pagina kunt opvragen en hoe je hun tekst kunt manipuleren.

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 03' waarin je

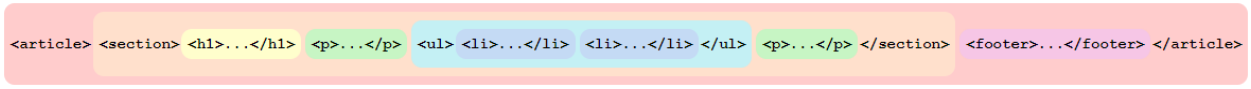
- voor elke uitprobeer opdracht een entry maakt met screenshots ter staving van wat je deed
- je antwoorden op de gestelde vragen neerschrijft

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

De DOM-tree

De DOM-tree is een datastructuur die de browser opbouwt op het moment dat de HTML code gescand en geanalyseerd wordt. Deze structuur beschrijft de HTML-elementen uit het HTML-document en hoe ze in elkaar genest (i.e. vervat) zitten.

Hieronder staat een voorbeeld van hoe elementen in een HTML-document in elkaar genest zijn :



Zo bevat het <ul> element twee <li> elementen; we zeggen dat de <li> elementen in het <ul> element *genest* zijn.

DOM is een afkorting voor Document Object Model, het is een beschrijving van document-gerelateerde objecten. Elk object beschrijft een HTML element : z'n attributen en tekstuele inhoud maar ook verwijzingen naar de andere elementen die het bevat (anders gezegd : naar de elementen die erin vervat zitten).

Eenmaal de browser klaar is met het inladen en analyseren van een HTML-document, gebruikt de browser enkel nog de DOM-tree die werd opgebouwd. *De originele HTML code wordt niet meer gebruikt!*

Kort gezegd, een DOM-tree is de fundamentele representatie van een HTML document in de browser. Alle styling met CSS alsook alle pagina manipulaties met Javascript, gebeuren ten opzichte van deze structuur.

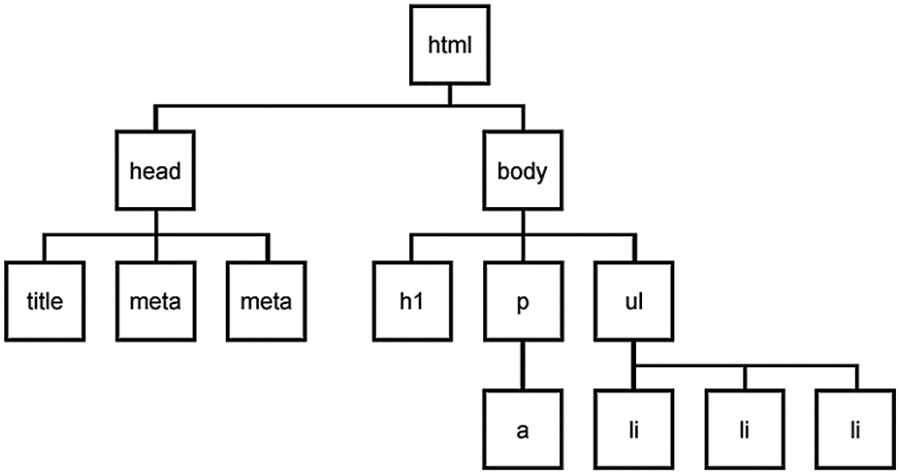
Je kunt trouwens de DOM-tree bekijken in Chrome op het Elements tabblad van de Chrome Developer Tools. Normaliter heb je hiermee leren werken in de HTML en CSS lessen van de opleiding, wie een opfrisser wil, kan onderstaande video bekijken :

> Chrome DevTools - Elements Panel
> https://www.youtube.com/watch?v=DO54CzdVrBQ

Bij wijze van voorbeeld nemen we eens het volgende HTML document :

```
<!DOCTYPE html>
<html>
  <head>
    <title>White Russian FTW!</title>
    <meta charset="UTF-8">
    <meta name="author" content="Jeff Lebowski ">
  </head>
  <body>
    <h1>The dude abides!</h1>
    <p>Making a <a href="https://en.wikipedia.org/wiki/White_Russian_(cocktail) ">White
russian</a> is easy man, you only need</p>
    <ul>
      <li>Vodka</li>
      <li>Coffee liqueur</li>
      <li>Cream</li>
    </ul>
  </body>
</html>
```

De DOM-tree die hiermee correspondeert kunnen we schematisch als volgt weergeven :



De hokjes stellen de HTML-elementen voor en de lijnen ertussen tonen hoe ze in elkaar genest zijn. Bijvoorbeeld, het <ul> element bevat drie <li> elementen.

Merk op dat dit schema enkel de nesting relatie tussen de elementen weergeeft, de eigenlijke DOM-tree bevat veel meer details dan wat er getoond wordt.

De DOM-tree is een boomstructuur. Boomstructuren zijn zeer algemene datastructuren die je in IT wel vaker tegenkomt. We kunnen hun belangrijkste kenmerken uitleggen a.d.h.v. het eerdere DOM-tree schema :

- Elk onderdeel van een boomstructuur noemen we een '**node'** van de **tree**.
    - de hokjes in het schema stellen de nodes van de tree voor.

- De “html” node bovenaan is speciaal, we noemen ze de '**root'** van de boom.

- Als we in het schema de “ul” node als voorbeeld nemen, dan is de “body” node z’n **parent** en vormen de “li” nodes samen z’n **children**.
    - **Elke node heeft hoogstens één parent en kan zelf meerdere children hebben.**

- De “a” node heeft “p”, “body” en “html” als **ancestors**.
- De **descendants** van “body” zijn “h1”, “p”, “ul”, “a” en de drie “li” elementen.

Vraag : welke node heeft geen parent?

Opdracht 01

Open de volgende pagina in je browser

https://www.htmldog.com/examples/headings1.html

Kijk op het Elements tabblad naar de DOM-tree en teken op een blad papier een schematische weergave ervan.

--

Merk op dat een node tegelijkertijd parent en child kan zijn in een boomstructuur. In de eerdere DOM-tree is het <ul> element een parent voor de <li> elementen en tegelijk ook een child van het <body> element.

De HTML-standaard legt vast welke parent-child relaties zijn toegelaten. Sommige elementen mogen geen children hebben (bv. een <img> element) en sommige elementen mogen alleen heel specifieke kinderen hebben (bv. een <ul> element mag enkel <li> children hebben).

Telkens we in onze Javascript code iets in de getoonde webpagina willen veranderen, moeten we hiervoor met de DOM-tree werken. Een aanzienlijk deel van deze cursus gaat dan ook over wat we zoal met de DOM-tree kunnen doen (en welke code we daarvoor moeten schrijven!).

Terzijde :

Een andere IT-context waarin je boomstructuren tegenkomt is het filesysteem van je computer : de files en folders op je schijf vormen samen een boomstructuur.

- de files en folders vormen de nodes in deze boomstructuur
- elke folder de parent is van z'n inhoud (zijnde files en mogelijks ook andere folders).
- elke folder en elke file zitten vervat in precies één 'parent' folder
    - de enige uitzondering hierop is de … *tromgeroffel!* … root folder van je filesysteem

Extra oefening DOM-tree

Bekijk het bestand 'extra oefening dom tree.html' en teken een schematische weergave van de DOM-tree van die pagina, waarin je enkel de nodes voor de HTML-elementen toont.

Elementen uit de DOM-tree opvragen

Een javascript programma dat in een browseromgeving uitgevoerd wordt, kan toegang krijgen tot de DOM-tree van de huidige pagina via een (voorgedefinieerde) '**document**' globale variabele.

De onderdelen van de DOM-tree noemt men ook in deze context 'nodes'. Men gebruikt dezelfde terminologie die we reeds eerder zagen : parent nodes, child nodes, ancestor nodes, descendent nodes, etc.

Veel van die nodes komen overeen met HTML elementen, maar er zijn ook andere soorten nodes (bv. text nodes voor de teksten tussen begin- en eindtag van een element, of comment nodes voor stukjes commentaar uit het HTML document).

Een verwijzing naar een element node uit de DOM-tree kan je (onder andere) op de volgende drie manieren bekomen :

- **document.getElementById("**abc**");**

  doorzoekt de DOM-tree naar het element met id="abc" en retourneert een verwijzing naar dit element. Indien geen element wordt gevonden, retourneert de method de null waarde.

- **document.getElementsByClassName("**xyz**");** let op : getElement**S** meervoud!!

  doorzoekt de DOM-tree naar elementen met class="xyz" en retourneert een **lijst** met verwijzingen naar deze elementen. De volgorde van de elementen in die lijst is dezelfde als hun onderlinge volgorde in het document. De lijst is een NodeList object en is op dezelfde manier te gebruiken als een array - dus wat ons betreft is het verschil niet belangrijk. Indien geen enkel element wordt gevonden, retourneert de method een lege lijst (vgl. lege array).

- **document.getElementsByTagName("**img**");**

  doorzoekt de DOM-tree naar <img> elementen en retourneert een lijst met verwijzingen naar deze elementen. Je krijgt hetzelfde soort lijst als bij getElementsByClassName, maar de elementen werden gezocht op basis van hun tag i.p.v. hun CSS-classes.

In een volgend deel komen handigere methods aan bod (querySelector en querySelectorAll), voorlopig beperken we ons tot de bovenstaande (en gebruiken slechts zelden getElementsByTagName).

Je zult normaal gezien de return value van deze methods in een variabele opslaan, om verderop in je code iets interessants te kunnen doen met de gevonden elementen.

Bijvoorbeeld :

```
<a id="lnkVlaanderen" href="http://www.vlaanderen.be">Vlaanderen</a>
…
let link = document.getElementById("lnkVlaanderen");
console.log ( link.textContent );                    // toont "Vlaanderen"
```

Bijvoorbeeld :

```
<p class="para">blabla1</p>
<p class="para">blabla2</p>
<p class="para">blabla3</p>
…
let paras = document.getElementsByClassName("para");
paras[1].textContent = "boemboem2";                  // verandert blabla2 in boemboem2
```

--

De methods getElementsByClassName en getElementsByTagName leveren een NodeList op en doorgaans zul je alle element in die verzameling willen overlopen om er iets mee te doen.

Het NodeList object stelt een lijst van verwijzingen naar DOM-tree elementen voor, en kan op dezelfde manier gebruikt worden als een Javascript array (al is het technisch gezien geen array).

Bijvoorbeeld :

```
let paragrafen=document.getElementsByTagName("p");
for (let i=0;i<paragrafen.length;i++) {
  paragrafen[i].textContent="Hello world!";
}
```

We kunnen de tekst van een element vervangen door diens .textContent property aan te passen.

Dit stukje code vervangt dus de tekst van elke paragraaf op de pagina door "Hello world!".

Tip van Flip 1

Een veel voorkomende fout bij beginnende Javascript programmeurs is, het resultaat van bv. getElementsByClassName te gebruiken alsof het één enkel DOM-tree element was i.p.v. een verzameling van DOM-tree elementen.

```
<p class="para">blabla1</p>
<p class="para">blabla2</p>
<p class="para">blabla3</p>
…
let paras = document.getElementsByClassName("para");
paras.textContent = "boemboem2";                    // FOUT, maar geen foutmelding!
```

Naargelang wat je (fout) probeert te doen, krijg je soms zelfs geen foutmelding op de console.

Methods als getElementsByClassName en getElementsByTagName retourneren altijd een verzameling in de vorm van een NodeList object

- zelfs als er maar 1 element gevonden wordt
- zelfs als er geen enkel element gevonden wordt

Kortom, als je iets wil doen met de returnvalue van getElementsByClassName dan moet er hoogstwaarschijnlijk ergens een […] index notatie aan te pas komen 😉

Opdracht 02

Probeer elk van de bovenstaande manieren uit waarmee je elementen uit de DOM-tree kunt opvragen.

Doe dit op het console venster en gebruik hiervoor de pagina op

http://www.nieuwsblad.be

Je zult natuurlijk zelf een passende id en className moeten zoeken in de DOM-tree van die pagina, deze uit de voorbeelden zullen hoogstwaarschijnlijk geen resultaten opleveren omdat die namen niet gebruikt worden (maar probeer deze ook eens uit, zodat je ook eens een lege nodelist ziet).

Je kunt de opdrachten gewoon in het console venster intypen en de return value bekijken.

--

Altijd het window load event afwachten

Enkele bladzijden terug werd al aangehaald dat de browser de DOM-tree opbouwt terwijl hij het HTML-document overloopt en analyseert.

Als de browser daarbij een <script> element tegenkomt, zal hij dit meteen uitvoeren. Het kan dus gebeuren dat een stuk Javascript wordt uitgevoerd terwijl de DOM-tree nog niet afgewerkt is! Indien deze Javascript code iets met de DOM-tree wil doen, kan dit dus problemen geven.

Code die de DOM-tree uitleest of aanpast, mag pas uitgevoerd worden wanneer de DOM-tree volledig is opgebouwd door de browser!

Je moet onderstaand fragment gebruiken :

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

zodat de browser onze setup functie pas uitvoert als de DOM-tree klaar is voor consumptie. De setup functie vormt op die manier ook het échte startpunt van onze applicatie.

Tip van Flip 2

Zo goed als elk Javascript programma dat je dit semester zal schrijven, moet dit fragment bevatten!

Vraag : waarom was dit niet nodig voor de vingeroefening uit de vorige les?

DOM-element property .textContent

DOM-tree elementen hebben een **.textContent property** waarmee je de tekst in het element kunt opvragen en wijzigen. Vooraleer je dit kunt doen, zul je natuurlijk eerst een verwijzing naar het juiste element in de DOM-tree moeten bemachtigen, bv. via document.getElementById(…).

Je kunt de tekst van een element **opvragen** door diens .textContent property te lezen.

Bijvoorbeeld

```
<a id="lnkVlaanderen" href="http://www.vlaanderen.be">Vlaanderen</a>
…
let link = document.getElementById("lnkVlaanderen");
console.log ( link.textContent );                    // toont "Vlaanderen"
```

Dit zal de tekst van de hyperlink met id “lnkVlaanderen” op de console zetten.

Let op, als je de tekst van een element opvraagt, krijg je de tekst uit het element en al zijn afstammelingen in 1 grote string!

Bijvoorbeeld

```
<p id="txtDemo">Dit is een <a href="...">hyperlink</a> waarop je kunt klikken.</p>
...
let txtDemo = document.getElementById("txtDemo");
console.log( txtDemo.textContent );
```

(op de console verschijnt de tekst : “Dit is een hyperlink waarop je kunt klikken.”)

Je vraag doorgaans enkel de tekst op van elementen zonder kinderen. De .textContent van een element met kinderen is soms nogal complex samengesteld, vooral qua whitespace. Je zult deze niet zo vaak gebruiken, maar het kan wel eens van pas komen als je bv. wil weten of een zoektekst ergens voorkomt in een deel van de pagina.

Je kunt de waarde van de .textContent property natuurlijk ook **wijzigen** :

```
txtDemo.textContent = "Dit is een tekst zonder hyperlink";
```

Het instellen van de textContent doe je normaliter enkel bij een element zonder kinderen.

Meer uitleg en een voorbeeld vind je op https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent

DOM-element property .innerHTML

Je kan een element een andere inhoud geven door diens **.innerHTML** property een nieuwe waarde te geven. Dit representeert de volledige tekst tussen de begin- en eindtag van dit element en kan dus ook HTML-tags (van de descendants) bevatten!

De innerHTML property van een element opvragen is doorgaans niet zo interessant. De property instellen is dit echter wel, je kan hiermee immers nieuwe HTML-elementen toevoegen als kinderen!

Bijvoorbeeld

```
<p id="txtDemo">Hier staat een beetje tekst</p>
…
let txtDemo = document.getElementById("txtDemo");
txtDemo.innerHTML = 'Make <a href="http://www.vlaanderen.be">Flanders</a> great again.'
```

Dit zal de tekst van de paragraaf veranderen alsook een hyperlink toevoegen aan de DOM-tree. Let trouwens op de mix van ' en " in bovenstaande code.

Het veranderen van de .innerHTML property van een element, verwijdert altijd de kinderen van dit element en vervangt ze door andere!

Bijvoorbeeld

```
<ul id="lstBoodschappen">
    <li>brood</li>
    <li>melk</li>
</ul>
…
let lstBoodschappen = document.getElementById("lstBoodschappen");
lstBoodschappen.innerHTML += "<li>kaas</li>";
```

Dit lijkt enkel een derde list item “kaas” toe te voegen, maar in werkelijkheid zijn ook de “brood” en “melk” items vervangen (door nieuwe elementen met dezelfde tekstuele inhoud).

De .innerHTML property is dus een makkelijke manier om nieuwe elementen aan de DOM-tree toe te voegen. We zullen later een alternatief zien dat enerzijds moeilijker is maar meer mogelijkheden biedt.

Meer info vind je op

https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML

Het voornaamste verschil tussen .textContent en .innerHTML is dat de nieuwe waarde van **.textContent** niet speciaal verwerkt wordt door de browser, maar integraal wordt overgenomen. Dit betekent dat eventuele HTML tags niet zullen leiden tot bijkomende nodes in de DOM-tree!

In het voorbeeld hierboven,

txtDemo.**textContent** = "dit is een hyperlink";

geeft : dit is een hyperlink

txtDemo.**innerHTML** = "dit is een hyperlink";

geeft : dit is een hyperlink

Opdracht 03

Maak een nieuw project met de volgende inhoud in het HTML-bestand (in de body) :

```
<p id="txtOutput">Hello world!</p>
```

Koppel een nieuw Javascript bestand aan de HTML, zodat de volgende code (correct) wordt uitgevoerd :

```
let pElement=document.getElementById("txtOutput");
pElement.textContent="Welkom!";
```

Laad de pagina in en ga na dat de tekst op het scherm niet "Hello World!" is, maar "Welkom!"

Je kunt in de file 'demonstratie window load event goed fout.zip' twee programma's vinden : een goede oplossing (mét window load event) en een foute oplossing (zonder window load event).

Video : debuggen met de Chrome Developer Tools

Bekijk het filmpje op

> Debugging JavaScript - Chrome DevTools 101
> https://www.youtube.com/watch?v=H0XScE08hy8

Dit legt uit hoe je breakpoints kunt instellen in je Javascript code, erdoor kunt stappen met de debugger en de waarden van variabelen kunt inspecteren.

Opdracht 04

Neem de oplossing van de vorige opdracht.

Zet een breakpoint op de eerste regel van de setup functie en herlaad de pagina.

Vergewis je ervan dat er wel degelijk "Hello world!" op het scherm staat.

Kijk in de developer tools naar de DOM-tree en zoek de node van het <p> element, daarin zie je weerom "Hello world!".

Stap met de debugger doorheen de code totdat de regel uitgevoerd is waarin "Welkom!" wordt toegekend aan de innerHTML property van het <p> element.

Ga na dat er nu "Welkom!" op het scherm staat en dat dit ook in de DOM-tree veranderd is.

Opdracht 05

Verander het voorgaande programma zodanig dat de tekst pas gewijzigd wordt als je op een button met opschrift "Wijzig" drukt. Je kunt je hiervoor op het rekenmasjien voorbeeld baseren.

Stop de code die de wijziging doorvoert in een functie met naam ‘wijzig’.

Voeg in de HTML file een <input> element van type=”button” toe met een id gelijk aan “btnWijzig”.

Voeg onderaan de setup functie de volgende regel toe :

```
let btnWijzig = document.getElementById("btnWijzig");
btnWijzig.addEventListener("click", wijzig);
```

Dit zorgt ervoor dat de ‘wijzig’ functie wordt opgeroepen telkens je op de button klikt.

DOM-element property .value voor tekstvelden

Je kan de tekst in een tekstveld bemachtigen via de **.value** property van het corresponderende <input> element uit de DOM-tree.

Het precieze type van het <input> element doet er niet toe, de waarde is altijd een string.

Je zult weerom eerst een verwijzing naar het element uit de DOM-tree moeten bemachtigen, vooraleer je diens .value kunt opvragen.

Bijvoorbeeld

```
<input type="text" id="txtInput">
…
let txtInput = document.getElementById("txtInput");
console.log( txtInput.value );
```

Dit zal de tekst in het tekstveld op de console zetten.

Je kan met je Javascript code een string in een tekstveld stoppen, door de .value property te wijzigen.

Tip van Flip 3

De DOM-tree element properties .textContent, .innerHTML en .value hebben allen iets te maken met ‘de’ tekst van elementen in de DOM-tree. Beginnende programmeur verwarren ze vaak en gebruiken de verkeerde property.

Onthou :

- .value hoort enkel bij <input> elementen
- .textContent en .innerHTML horen nooit bij <input> elementen
    - Strikt genomen, “bijna nooit” bv. wel als je de tekst op een button wil veranderen

Demonstratie

In de .zip files 'demonstratie images.zip' en 'demonstratie lijsten.zip' vind je twee voorbeelden van het gebruik van de .innerHTML en .value properties.

In beide demonstraties wordt met javascript inhoud aan een pagina toegevoegd.

Opdracht 06

Maak een nieuw project met de volgende inhoud in het html bestand (in de body) :

```
<input id="txtInput" type="text" />
<input id="btnKopieer" type="button" value="Kopieer" />
```

Stop de volgende code in de setup functie van je Javascript bestand :

```
let btnKopieer = document.getElementById("btnKopieer");
btnKopieer.addEventListener("click", kopieer);
```

Definieer ook een 'kopieer' functie in je Javascript bestand :

```
const kopieer = () => {
    let txtInput = document.getElementById("txtInput");
    let tekst = txtInput.value;
    console.log(tekst);
}
```

Laad de pagina in, typ wat tekst in het tekstveld, klik op 'Kopieer' en ga na dat de ingetypte tekst op de console verschijnt.

Opdracht 'kopieer'

Voeg een <p> element toe aan de html code van het vorige programma :

```
<p id="txtOutput">geen output</p>
```

Breid het programma uit zodat de ingetypt tekst niet op de console getoond wordt maar in dit <p> element terechtkomt.