

Web development I – JS labo 3

studiegebied **HWB**

bachelor in het **toegepaste Informatica**

campus **Kortrijk**

academiejaar **2022-2023**

Javascript – deel 03

Deze les behandelt het window load event en toont hoe eenvoudige CSS properties kunnen ingesteld worden met javascript.

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 '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.

Event-based programming

In een programma met een grafische user interface zal de meeste functionaliteit pas in gang schieten op het moment dat de gebruiker een actie onderneemt (bv. op een knop klikt). Indien de gebruiker niks doet, wacht het programma gewoon.

Voor dit soort programma's is het nuttig om onze code te structureren volgens de gebeurtenissen (Engels : events) waarop het programma moet reageren, men noemt dit event-based programming.

In een browser-omgeving worden er vele soorten events gegenereerd waarop je als programmeur kunt inpikken. Een uitgebreid overzicht vind je op

<https://developer.mozilla.org/en-US/docs/Web/Events>

In wat volgt hebben we het specifiek over de events die de browser ons aanbiedt voor elementen in de DOM-tree. Je vindt deze in het bovenstaande overzicht terug in de categorie "DOM events" (linkerkolom).

Per element kunnen we onze code aan een specifiek soort event koppelen met de `addEventListener` method :

```
elementNode.addEventListener("anEvent", ourFunction);
```

De verschillende onderdelen in deze opdracht zijn

elementNode

dit is een verwijzing naar een element node uit de DOM-tree, die we bv. met `document.getElementById` hebben bemachtigd.

"anEvent"

dit is een String waarde met de naam van het event waarin we geïnteresseerd zijn, bv. "click"

ourFunction

dit is de naam van de function die moet opgeroepen worden wanneer `anEvent` zich voordoet op `elementNode`. Zo'n function noemen we een event listener.

Bijvoorbeeld, om een function **printHello** uit te laten voeren wanneer de gebruiker op een `<div>` met id **"abc"** klikt moeten we volgende code uitvoeren :

in de HTML file

```
<div id="abc">...</div>
```

in de javascript file

```
let elementNode=document.getElementById("abc");
elementNode.addEventListener("click", printHello)
```

Window load event

De browser biedt ons ook de mogelijkheid om code te laten uitvoeren zodra de DOM-tree klaar is voor gebruik en alle resources zijn ingeladen : het load event van het window object.

Het window object is bereikbaar via de gelijknamige globale variabele.

Naargelang waar we naar de javascript code verwijzen in ons HTML document, kan het voorkomen dat de javascript code begint uit te voeren nog vooraleer de browser klaar is met het opbouwen van de DOM-tree.

Het spreekt voor zich dat we zelden het gewenste resultaat zullen bekomen indien onze code de DOM-tree aanspreekt terwijl die nog in opbouw is. Denk bv. aan het koppelen van event listeners aan allerlei elementen van de pagina.

Het is dus wenselijk dat onze code wacht totdat de DOM-tree klaar is, we doen dit als volgt :

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

De setup function is een event listener die eigenlijk de initialisatie code bevat van onze applicatie. Typisch is dat hierin allerlei andere event listener functions aan hun resp. elementen worden gekoppeld.

Elementen van een NodeList overlopen

Vorige keer zagen we dat `document.getElementsByClassName` en `document.getElementsByTagName` een NodeList object opleveren. Anders gezegd, de return value van deze beide methods is telkens een verwijzing naar een NodeList object.

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

Bijvoorbeeld, om alle paragrafen in een pagina de tekst "Hello world!" te geven :

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

CSS properties instellen van een element

Om een CSS property van een element in te stellen in een javascript programma, moeten we de **.style** object property van een `elementNode` aanspreken.

Een `elementNode` is dus een soort object met verschillende properties (enigszins vergelijkbaar met datavelden in Java objecten), waarvan er eentje de naam "style" draagt en die de toegangspoort is tot de CSS properties van dat element.

Bijvoorbeeld, om de tekstkleur van een element aan te passen, moeten we de CSS color property instellen met

```
elementNode.style.color="red";
```

Merk op dat de namen van CSS properties in javascript enigszins anders moeten geschreven worden :

```
background-color wordt backgroundColor  
bv. elementNode.style.backgroundColor="red";
```

```
margin-left wordt marginLeft  
bv. elementNode.style.marginLeft="10px";
```

Merk ook op dat het rechtstreeks instellen van allerlei CSS properties in javascript code eigenlijk niet zo best is, dan staat immers een deel van de opmaak code in het javascript bestand i.p.v. het CSS bestand. Veel beter is om classes te definiëren met bijbehorende CSS-stijlregels, zodat je vanuit de javascript code eenvoudigweg classes aan elementen kunt toevoegen of verwijderen.

CSS classes instellen van een element

De classes van een HTML element kun je opvragen (en wijzigen) via diens `.className` property. De waarde van deze property is een string.

Bijvoorbeeld

```
<p id="txtOutput" class="abc ">blablabla</p>
...
let txtOutput = document.getElementById("txtOutput");
console.log( txtOutput.className );           // toont "abc"
txtOutput.className = "invalid";               // 1 class instellen
txtOutput.className = "";                      // alle classes verwijderen
```

De `.className` property bevat eigenlijk dezelfde tekst als je ook in de HTML code zou schrijven, en kan dus meerdere classes bevatten gescheiden door spaties.

Opdracht paragrafen

Schrijf een HTML pagina met daarin vier paragrafen (korte) lorem ipsum tekst. De tweede en de vierde paragraaf krijgen een class "belangrijk" waardoor ze via een bijbehorende CSS-regel dikgedrukt worden.

Voorzie ook een CSS-regel waardoor alle elementen met class "opvallend" in het rood komen te staan. Voorlopig heeft dit geen effect omdat er zulk geen elementen zijn.

Schrijf nu een javascript programma dat alle elementen uit de DOM-tree met class "belangrijk" opvraagt en ze ook de class "opvallend" geeft.

Probeer je programma uit en vergewis je ervan dat paragrafen twee en vier dikgedrukt én in het rood staan.

Zorg ervoor dat je code algemeen genoeg is en bv. ook werkt als we de HTML code wijzigen zodat paragrafen drie en vier "belangrijk" zijn. Anders gezegd, je mag niet hardcoderen dat het om de eerste en de vierde paragraaf gaat.

Opdracht 01

Stop de inhoud van 'demo slider.zip' in een Webstorm project en open de HTML pagina in de browser.

Open het console venster in de Chrome Developer Tools en kijk wat er gebeurt als je de slider verschuift.

Vergewis je ervan dat als je stopt met slepen, er op de console 2 meldingen komen voor de laatste waarde.

Opdracht 02

Bestudeer de code van de demonstratie op het Sources tabblad in de Chrome Developer Tools.

Beantwoord volgende vragen :

- Waar wordt de event listener gekoppeld aan de slider?
- Waarom moeten we die op twee soorten events koppelen?
- In de CSS file wordt nergens een rode kleur opgegeven, waar wordt dan wel de rode kleur van het blokje ingesteld?
- Waarom schrijven we telkens sliders[0] en colorDemos[0] en niet gewoon sliders of colorDemos?

In de javascript code wijzigen we een CSS property van een element. We zagen in een eerdere les dat je de relevante CSS-regels voor een element op het Elements tabblad in de Chrome Developer Tools kunt bekijken, nml. rechts op het "Styles" tabblad.

Waar vind je de CSS-properties die programmatorisch werden ingesteld, zoals bv. de rode achtergrondkleur van het blokje?

Opdracht 03

Ga naar het Sources tabblad van de Chrome Developer Tools en open daar het code.js javascript bestand.

Zet een breakpoint op de eerste regel van de setup function en herlaad de pagina. De uitvoering zal stoppen bij dit breakpoint.

Ga stap voor stap verder met het uitvoeren door op  te klikken en hou rechts (onder kopje "Local") de waarden van de lokale variabelen 'sliders' en 'colorDemos' in de gaten. Zodra beiden een waarde hebben stop je voorlopig met uitvoeren.

Wat voor soort waarde zit er in die variabelen?

Toen in de vorige les de werking van `getElementsByClassName` aan bod kwam, spraken we over een `NodeList` terwijl er nu iets anders staat. Vergewis je ervan dat dit op hetzelfde neerkomt

<http://stackoverflow.com/questions/15627385/what-is-the-difference-between-a-htmlcollection-and-a-nodelist-in-dom>

Hoeveel elementen zitten er in elke verzameling?

Kun je uit de HTML code afleiden waarom er dit precies zoveel zijn?

Klik op het driehoekje naast de sliders lokale variabele zodat je het eerste element kunt zien. Op welke index positie in de lijst staat dit element?

Klik op dit eerste element, je zult nu naar het bijbehorende element springen in de DOM-tree op het Elements tabblad van de Chrome Developer Tools.

Ga nu terug naar het Sources tabblad in de Chrome Developer Tools.


Heb je er trouwens op gelet dat het blokje (nog) niet rood is? Ga stap voor stap verder met de uitvoering totdat je de opdracht uitgevoerd hebt die het blokje rood kleurt.

Snap je nu waarom er een `[0]` achteraan `colorDemos[0]` staat? Zoniet, vraag je docent om uitleg.

Zet nu een breakpoint op de eerste regel van de update function, die een event listener is voor de input en change events van de slider.

Laat de uitvoering van het programma gewoon verder lopen door op  te klikken.

Versleep de slider en vergewis je ervan dat de uitvoering stopt in de update function.

Laat de uitvoering van het programma weer verder lopen door op  te klikken.

Opdracht Colorpicker

Maak de colorpicker opdracht zoals beschreven in het filmpje 'opdracht colorpicker.mp4'.

Je kan hiervoor de demonstratie gebruiken als startpunt.

Hints : je kan dezelfde update functie gebruiken voor alle sliders als je ze bij elke slider als event handler toevoegt. Die update functie leest dan de waarde uit van de drie sliders en bouwt zo een `rgb(...,...,...)` String op. Die String kun je dan gebruiken om de `backgroundColor` in te stellen van het gekleurde vakje.

DOM-element property `.style` versus `.className`

We hebben nu twee manieren gezien om de styling van een element te wijzigen via javascript code:

- CSS-property waarden instellen via de `.style` property
- CSS-classes veranderen via de `.className` property

Normaliter geef je er de voorkeur aan om een oplossing op basis van CSS classes te maken, zo blijven de presentatie details netjes in de CSS bestanden staan en wordt deze niet tussenin de javascript code geplaatst.

Soms zijn er echter teveel mogelijkheden en is het niet doenbaar om voor elk een aparte CSS-class te voorzien. Bijvoorbeeld

- in de colorpicker opdracht kun je onmogelijk voor elke RGB combinatie een aparte CSS-class voorzien dus daar stelden we de swatch kleur in via de `.style` property
- in een spel met bewegende figuren, worden deze verplaatst door de `left` en `top` CSS properties te manipuleren (dit veronderstelt natuurlijk absolute positionering). Ook hier is het ondoenbaar om voor elke mogelijke positie een aparte CSS-class te definiëren en zullen we deze properties wijzigen via `.style`

DOM-element property `.classList`

We zagen reeds eerder dat de classes van een HTML element kunnen opgevraagd (en gewijzigd) worden via diens `.className` property.

De waarde van deze property is een String en bevat eigenlijk dezelfde tekst die je ook in de html code zou schrijven, en kan dus meerdere classes bevatten gescheiden door spaties.

Bijvoorbeeld

```
<p id="txtOutput" class="important">blablabla</p>
...
let txtOutput = document.getElementById("txtOutput");
txtOutput.className += " invalid";           // class toevoegen
console.log( txtOutput.className );         // toont "important invalid"
```

(merk op dat er een spatie staat voor 'invalid' om de classes te scheiden!)

Als je slechts 1 class wil toevoegen (of verwijderen) en eventuele andere classes wil behouden en dubbels vermijden, zul je dus een hoop string bewerkingen moeten doen (knip- en plakwerk)!

Gelukkig bestaat er ook een handige `.classList` property waarmee dit eenvoudiger kan :

```
txtOutput.classList.add("invalid");
txtOutput.classList.remove("invalid");
```

Dit vermijdt het string knip- en plakwerk.

Meer uitleg over de mogelijkheden van `.classList` vind je op

<https://developer.mozilla.org/en/docs/Web/API/Element/classList>

DOM-element property .textContent

DOM-tree elementen hebben een .textContent property waarmee je de tekst in het element kunt opvragen en wijzigen.

Let op : je krijgt 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 kunt via de waarde van de .textContent property ook wijzigen natuurlijk :

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

Meer uitleg en een voorbeeld vind je op

<https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent>

Aandachtspunten

- Als het element kinderen heeft, is de waarde van de .textContent property dus een combinatie van de teksten van meerdere elementen. Dit maakt het soms ingewikkeld, vooral qua whitespace e.d.
 - beperk je dus best tot de .textContent van elementen die geen kinderen hebben.
- Als je de waarde van **.textContent** overschrijft, wordt deze niet speciaal verwerkt door de browser maar integraal overgenomen. Dit betekent dat eventuele HTML tags niet zullen leiden tot bijkomende nodes in de DOM-tree!
 - hiervoor kun je de eerder geziene **.innerHTML** property gebruiken

In het voorbeeld hierboven,

```
txtDemo.textContent = "dit is een <a href='http://www.example.com'>hyperlink</a>";
```

geeft : dit is een hyperlink

```
txtDemo.innerHTML = "dit is een <a href='http://www.example.com'>hyperlink</a>";
```

geeft : dit is een [hyperlink](#)

Number

Alle getallen worden in javascript voorgesteld door Number objecten 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 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 "rekenmachine" demonstratie zagen we reeds hoe we een Number kunnen bekomen uit een String m.b.v. **parseInt** :

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

(let op de meegegeven radix 10 zodat de functie weet in welk talstelsel de cijfers moeten geïnterpreteerd worden)

Naast de eerdere geziene parseInt bestaat er ook een parseFloat function, deze zal net als parseInt eveneens een Number produceren op basis van de meegegeven string, maar houdt ook rekening met cijfers na de komma.

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";
```

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

Bijvoorbeeld :

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

Merk op dat we voor de console.log parameter geen .toString() moeten doen, die functie kan dat zelf.

Opdracht Producten

Maak een webpagina met volgende inhoud :

producten	prijs	aantal	btw	subtotaal
product1	10.00 Eur	<input type="text" value="0"/>	6%	0.00 Eur
product2	15.00 Eur	<input type="text" value="0"/>	21%	0.00 Eur
product3	12.20 Eur	<input type="text" value="0"/>	21%	0.00 Eur
totaal				0.00 Eur

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	<input type="text" value="1"/>	6%	10.60 Eur
product2	15.00 Eur	<input type="text" value="1.5"/>	21%	27.22 Eur
product3	12.20 Eur	<input type="text" value="2"/>	21%	29.52 Eur
totaal				67.35 Eur

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 negeert de `parseFloat()` functie 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 cellen van eenzelfde soort dezelfde class (bv. 'prijs', 'aantal', 'btw' en 'subtotaal'). Dan kun je makkelijk alle cellen van een bepaalde soort opvragen via `getElementsByClassName`. Bovendien staan ze in dezelfde volgorde in elke lijst (op index 'i' vind je in elke lijst de corresponderende cel uit rij i)!