

## Javascript – deel 04

Deze les behandelt events en hoe je met Javascript de CSS-styling van elementen kunt wijzigen.

### 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 04' 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 inspikken. 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 in de DOM-tree 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`. Deze **callback functie** noemen we een **event listener** of ook wel **event handler**.

Kort gezegd, telkens "`anEvent`" zich voordoet bij `elementNode` zal de browser `ourFunction` uitvoeren.

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);
```

Merk op dat er geen haakjes ( ) staan achter de naam van de callback functie !!!!!



#### Tip van Flip 4

Zet in deze lessenreeks nooit haakjes ( ) achter de naam van de callback functie bij het registreren van een event handler. Doe je dit wel, dan zal de callback functie nooit worden opgeroepen (maar je krijgt helaas geen error op de console).

### Opdracht 01

Maak een webpagina waarin een <div> voorkomt en stop daarin 4 paragrafen (<p> elementen). In elke paragraaf kopieer je de eerste zin van de bekende [Lorem Ipsum](#) tekst ("Lorem ipsum dolor sit amet ...").

Voorzie een CSS bestand dat de <div> een lichtgrijze achtergrond geeft en 30px padding, de paragrafen krijgen allen een gele achtergrond.

Telkens de gebruiker (eender waar) in de <div> klikt, verschijnt er het woordje "klik" op de console.

Probeer je programma uit door zowel op het lichtgrijze gebied als op de gele gebieden te klikken!

## Opdracht 02

Pas de oplossing van opdracht 01 aan zodat er ook event listeners aan de <div> gekoppeld worden voor de onderstaande events ?

- Event “mouseenter” : de listener toont “enter” op de console
- Event “mousemove” : de listener toont “move” op de console
- Event “mouseleave” : de listener toont “leave” op de console

(let op de kleine letters in de event namen, bv. mouseenter en niet mouseEnter)

In totaal worden er aan het <div> element 4 verschillende event listeners gekoppeld, je moet hiervoor dus 4 functies voorzien.

## 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.

## 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 properties in C# 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** uit CSS wordt **backgroundColor** in Javascript  
bv. `elementNode.style.backgroundColor = "red";`

**margin-left** uit CSS wordt **marginLeft** in Javascript  
bv. `elementNode.style.marginLeft="10px";`

## Opdracht 03

Pas de oplossing van opdracht 1 aan zodat een klik op het <div> element diens achtergrondkleur rood maakt.

Doe dit door de CSS-style property 'backgroundColor' aan te passen.

Merk op dat direct ingestelde CSS-properties altijd voorrang krijgen op waarden die via een CSS regel worden ingesteld uit het CSS bestand. Je kunt dit makkelijk zelf nagaan op het Elements tabblad in Chrome Developer Tools.

Deze voorrangsregeling is precies dezelfde als bij "styling via het style attribuut in HTML" t.o.v. "styling via CSS-regels".

## DOM-element property `.className`

De CSS-classes van een HTML element kun je opvragen 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.

In principe zou je de string ook kunnen aanpassen om classes toe te voegen of te verwijderen, maar erg handig is dit niet omdat je dan in een string moet knippen en plakken.

Doorgaans is het veel makkelijker om de `.classList` property te gebruiken (zie hieronder) en de `.className` property gewoon te negeren.

## DOM-element property `.classList`

We zagen hierboven dat de classes van een HTML element kunnen opgevraagd worden via de `.className` property. De waarde hiervan is een string en bevat de classes gescheiden door spaties.

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");
if ( txtOutput.classList.contains("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>

## 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 in de pagina.

Schrijf nu een Javascript programma dat na het inladen van de pagina, 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 tweede en de vierde paragraaf gaat.

## Opdracht 04

Stop de inhoud van 'demonstratie 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 05

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 06

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.

Stopt de uitvoering in jouw browser nu nogmaals? Zoja, waarom eigenlijk?

## 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 die bij elk van de drie sliders 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.

## CSS-style properties instellen of CSS-classes manipuleren?

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 toevoegen/verwijderen via de `.classList` property

Normaliter geef je er de voorkeur aan om een oplossing op basis van CSS classes te maken, dan staan de presentatie details netjes in de CSS bestanden waar ze thuishoren (i.p.v. vermengd te worden met de Javascript code).

Soms zijn er echter teveel mogelijke waarden 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 stellen we de swatch achtergrondkleur 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, dus zullen we deze properties wijzigen via `.style`.

## Target van een event

We hebben al verschillende malen gebruik gemaakt van eventlisteners en bijna elke keer registreerden we per element een eigen eventlistener.

Wanneer bv. twee elementen totaal ander klikgedrag vertonen, dan is het logisch om per element een eigen eventlistener functie te voorzien. In de rekenmasjien demo is er per button 1 aparte callback functie.

De code :

```
let btnOptellen=document.getElementById("btnOptellen");
let btnAftrekken=document.getElementById("btnAftrekken");
let btnVermenigvuldigen=document.getElementById("btnVermenigvuldigen");
let btnDelen=document.getElementById("btnDelen");

btnOptellen.addEventListener("click", optellen);
btnAftrekken.addEventListener("click", aftrekken);
btnVermenigvuldigen.addEventListener("click", vermenigvuldigen);
btnDelen.addEventListener("click", delen);
```

stelt de volgende event handlers in :

- een click op #btnOptellen roept callback functie 'optellen' op
- een click op #btnAftrekken roept callback functie 'aftrekken' op
- een click op #btnVermenigvuldigen roept callback functie 'vermenigvuldigen' op
- een click op #btnDelen roept callback functie 'delen' op

De oplossing van de colorpicker opdracht gebruikt echter dezelfde event handler voor meerdere sliders.

De code :

```
let sliders = document.getElementsByClassName("slider");
for (let i = 0; i < sliders.length; i++) {
  sliders[i].addEventListener("change", update);
  sliders[i].addEventListener("input", update);
}
```

registreert dezelfde event handler bij elk van de drie sliders (en dan nog voor 2 verschillende soorten events, maar dat doet hier niet ter zake). Deze callback dient om de kleur van de swatch in te stellen en moest hiervoor sowieso de waarde van elk van de drie sliders opvragen. Het gedrag was dus identiek voor elke slider, vandaar dat we dezelfde functie konden gebruiken.

Maar wat als het gedrag grotendeels gelijkloopt en enkel in een klein detail verschilt?

Wat heel vaak gebeurt is dat een pagina een verzameling elementen bevat die eigenlijk allemaal op dezelfde manier reageren op een event, maar waarbij het gedrag afgestemd is op het eigenlijke element waar het event ontstond.

*Bijvoorbeeld, veronderstel een pagina met een aantal images die stuksgewijs verdwijnen als we erop klikken. De code in de click event listener zal voor elk element hetzelfde doen : een element verwijderen uit de DOM-tree. Het gedrag is echter niet identiek, want het te verwijderen element hangt af van het element waarop geklikt werd.*

*Dan rijst de vraag : “Hoe kunnen we achterhalen op welk element er precies geklikt werd?”*

Welnu, elke event listener functie heeft eigenlijk een parameter (die we meestal 'e' of 'event' noemen) met informatie over het precieze event dat zich voordeed. In de voorbeelden tot nu toe hebben we dit nooit gebruikt, simpelweg omdat we het niet nodig hadden.

Deze 'event' parameter is dus een event object met een aantal interessante properties, zoals de .target property die verwijst naar het DOM-tree element waar het event zich voordeed.

In een click event listener vertelt **event.target** ons dus op welk element geklikt werd!

Bijvoorbeeld :

Deze callback functie heeft een event parameter :

```
const maakRood = (event) => {
  let element = event.target;
  element.style.color = "red ";
}
```

Indien we deze callback functie bij 10 verschillende <p> elementen registreren als event listener voor het “click” event, dan zal enkel de paragraaf waarop geklikt werd een rode kleur krijgen.

### Opdracht Paragrafen met klik

Pas de oplossing van de ‘paragrafen’ opdracht aan zodanig dat een paragraaf pas rood kleurt als erop geklikt wordt i.p.v. bij het inladen van de pagina.

Sommige <p> elementen hebben dus een CSS-class ‘belangrijk’ en als erop geklikt wordt krijgen ze er een CSS-class ‘opvallend’ bij, waardoor ze rood worden. Andere paragrafen vertonen dit gedrag niet!