Informatietechnologie 2

JavaScript

DOM: attributen, styling, classes, events, timeouts en intervallen, willekeurige getallen en forms

Kristof Michiels

Belangrijkste bouwstenen tot nu toe:

- Werken met variabelen
- Beslissingsstructuren
- Loops
- Arrays
- Functies
- Objecten
- DOM: elementen vastnemen (1 of meerdere)
- DOM: elementen aanmaken, tekst aanmaken/wijzigen en plaatsen

Komen er deze week nog bij:

- DOM: werken met attributen
- DOM: elementen van styling voorzien
- DOM: werken met className en classList
- DOM: events
- Timeouts en intervallen
- Willekeurig getallen
- Werken met forms

Belangrijk: al deze bouwstenen zeer goed inoefenen en omzetten in vaardigheden!

Werken met de DOM: het vervolg

document.querySelector()

Deze document method is een modern alternatief voor .getElementById(). Ze geeft het eerste (!) element terug dat overeenkomt met een als argument meegegeven CSS selector. Je kunt hiervoor alle selectors gebruiken zoals je ze ook binnen een stylesheet gebruikt. Je schrijft ze op dezelfde manier: .mijnlink, #mijnnav, h1, ...

```
<h1>Dit is een voorbeeld</h1>
<nav id="mijnnav">
    <a href="#" class="mijnlink">Home</a>
</nav>
```

```
document.querySelector("h1").innerHTML = "Nieuwe tekst voor titel";
document.querySelector(".mijnlink").innerHTML = "Nieuwe tekst voor link";
```

document.querySelectorAll()

Deze document method is een modern alternatief zowel voor .getElementsByTagName() als .getElementsByClassName(). Ze geeft ALLE elementen terug die overeenkomen met een als argument meegegeven CSS selector. Ook hier weer kan je alle selectors gebruiken zoals je ze binnen een stylesheet gebruikt.

```
<p class="voorbeeld">Dit is een paragraaf</p>
<p class="voorbeeld">Dit is een andere paragraaf</p>
```

```
let mijnElementen = document.querySelectorAll(".voorbeeld");
let i = 0;
while (i < mijnElementen.length) {
    mijnElementen[i].innerHTML= "We veranderen de tekst binnen de paragraaf";
    i++;
}
```

.getAttribute() en .setAttribute()

Je herinnert je zonder twijfel nog dat HTML-elementen over attributen (kunnen) beschikken. Denk maar aan een a-element dat altijd een href-attribuut hoort te hebben. In het voorbeeld hieronder zie je een a-element met drie verschillende attributen: href, class en style. Met JavaScript kunnen we die attributen van (een) vastgenomen element(en) gaan opvragen, wijzigen of aanmaken. We doen dit met .getAttribute() en .setAttribute()

```
<nav>
    <a href="https://google.com" class="extern" style="color: red;">Google</a>
</nav>
```

.getAttribute()

Met de .getAttribute() method vraag je de inhoud op van een bestaand attribuut. Deze method neemt één argument, namelijk de naam van het attribuut, tussen aanhalingstekens.

```
<nav>
    <a href="https://google.com" class="extern" style="color: red;">Google</a>
</nav>
```

```
let mijnElement = document.querySelector("a");
console.log(mijnElement.getAttribute("href")); // https://google.com
console.log(mijnElement.getAttribute("class")); // extern
console.log(mijnElement.getAttribute("style")); // color: red;
```

.setAttribute()

Met de .setAttribute() method kan je de inhoud van een bestaand attribuut wijzigen of een nieuw attribuut aanmaken. Je geeft aan deze method 2 argumenten mee: de eerste verwijst naar de naam van het attribuut, de tweede naar de waarde ervan. Beiden horen tussen aanhalingstekens te staan. Geef je een attribuutnaam mee die nog niet aan het element is gekoppeld, dan wordt dit attribuut aangemaakt.

```
<a href="https://google.com" class="extern" style="color: red;">Google</a>
```

```
let mijnElement = document.querySelector("a");
mijnElement.setAttribute("href","https://docs.google.com");
mijnElement.setAttribute("style","color: blue;");
mijnElement.setAttribute("id","mijnlink");
```

HTML Data-* attributen

HTML biedt ons een speciaal soort attributen aan die we kunnen gebruiken om data op te slaan binnen een element. Je kan die data dan via JavaScript gaan gebruiken in je code. Elk element kan er zoveel bevatten als nodig. Data-* attributen bestaan uit 2 delen:

- De attribuut-naam: gebruik enkel kleine letters, en minstens 1 karakter lang na de prefix "data-"
- De attribuut-waarde: kan elke string bevatten (steeds string, ook bij getal!).

```
<ul>
     <li data-dier-type="vogel" data-aanwezig-in-zoo="true">Uil</li>
     <li data-dier-type="vis" data-aanwezig-in-zoo="false">Zalm</li>
     <li data-dier-type="spin" data-aanwezig-in-zoo="true">Tarantula</li>
</ul>
```

HTML Data-* attributen

Deze attributen zijn niet zichtbaar in je browser. Deze negeert deze data elementen volledig bij het weergeven van de pagina. Ze dienen enkel om data op te slaan die jij vanuit Javascript wenst te gebruiken. Je kan ze in HTML handmatig aan je elementen toevoegen of je kan ze meegeven vanuit JavaScript. Dat doe je met .setAttribute(). Het opvragen van de informatie in een data-* attribuut doe je met .getAttribute().

```
<div id="reisaanbieding" data-bestemming="Parijs"></div>
```

```
let mijnDiv = document.querySelector("div");
let bestemming = mijnDiv.getAttribute("data-bestemming");
console.log(bestemming); //Parijs
mijnDiv.setAttribute("data-periode","3 dagen");
mijnDiv.setAttribute("data-prijs","250 eur");
mijnDiv.setAttribute("data-vantot","3 april tot 7 april 2019");
```

Elementen van styling voorzien

HTML elementen van de nodige styling voorzien: het is iets dat we vaak gaan doen met behulp van JavaScript. We kunnen dit doen op 4 verschillende manieren:

- met het style attribuut
- met de style eigenschap
- met de className-eigenschap
- met de classList-eigenschap

Stylen met het style attribuut

Hiermee zorgen we voor inline styling. We zagen het enkele slides terug:

```
<nav>
    <a href="https://google.com" class="extern" style="color: red;">Google</a>
</nav>
```

```
let mijnElement = document.querySelector("a");
console.log(mijnElement.getAttribute("style")); // color: red;
mijnElement.setAttribute("style","color: blue;font-family: helvetica;");
```

Stylen met de style eigenschap

De style eigenschap biedt een andere mogelijkheid om een vastgenomen element te stylen. Let op: niet te verwarren met het style attribuut. Je doet het zoals te zien in onderstaand voorbeeld. Maar let op: de JavaScript syntax is lichtelijk anders dan bij CSS voor de style properties. Een voorbeeld: background-color wordt backgroundColor.

```
<h1>Dit is een hoofding</h1>
```

```
let mijnHoofding = document.querySelector("h1");
mijnHoofding.style.color = "red";
mijnHoofding.style.backgroundColor = "yellow";
mijnHoofding.style.borderBottom = "0.1rem solid black";
mijnHoofding.style.textAlign = "center";
console.log(mijnHoofding.style.backgroundColor);
```

Stylen met de style eigenschap

Ook hier wordt de styling inline meegegeven. Een voordeel tegenover het rechtstreeks gebruik van het style attribuut is dat hier niet telkens de volledige inline styling wordt overschreven. Handig, toch gaan wij indien mogelijk steeds gaan werken met de volgende methodes: gebruiken maken van styling via klasses. We gebruiken dan CSS waarvoor het bedoeld is: stylen. En we gebruiken JavaScript waarvoor het bedoeld is: interactie beschrijven.

```
h1.aangepast {
    color: red;
    background-color: yellow;
    border-bottom: 0.1rem solid black;
    text-align: center;
}
```

Werken met klasses

Elk HTML-element kan één of meerdere klasses bevatten. We kunnen deze klasses met behulp van JavaScript gaan opvragen en/of aanpassen. Er zijn twee eigenschappen die we hiervoor kunnen gebruiken: className en classList. Het laat ons toe de styling zoveel mogelijk via CSS classes te beschrijven. Nog even meegeven: je kan ook rechtstreeks via het class-attribute (dus met setAttribute) klasses manipuleren, maar deze methode laat ik hier even buiten beschouwing.

```
<h2>Een hoofding</h2>
<h2 class="hoofding">Een hoofding</h2>
<h2 class="hoofding aangepast">Een hoofding</h2>
```

De className eigenschap

Met deze eigenschap krijgen we toegang tot de klasses die aan een element zijn gekoppeld. Net zoals bij .innerHTML kunnen we zowel opvragen als aanpassen. Alle klasses komen terug als één enkele string. Je kan dus de klasses toevoegen maar weet dat je telkens de volledige string overschrijft. Was er al een klasse aanwezig, dan moet je die bij het aanpassen van de property mee toevoegen aan de string.

```
let mijnElement = document.querySelector("h2");
mijnElement.className = "mijnklasse"; // klassenaam toevoegen
mijnElement.className = ""; // klassena(a)m(en) verwijderen
if (mijnElement.className === "mijnklasse") {
    mijnElement.className = "";
} else {
    mijnElement.className = "mijnklasse"
}
mijnElement.className = "mijnklasse mijnandereklasse";
```

De className eigenschap

Nog een klein voorbeeld: een element onzichtbaar maken en tonen vanuit je code. Je ziet meteen ook dat je bij meerdere klasses moet opletten om geen bestaande klasses te overschrijven.

```
<div class="mijndiv">Tekst komt hier</div>
```

```
.onzichtbaar {
    display: none;
}
```

```
let mijnElement = document.querySelector(".mijndiv");
mijnElement.className = "mijndiv onzichtbaar";
mijnElement.className = "mijndiv";
```

De classList eigenschap

De classList eigenschap van een element is een interessant alternatief voor className. classList geeft alle klasses terug voor een bepaald element. classList.length geeft het aantal klasses die aan het element zijn gekoppeld. Bovendien beschikt classList over enkele bijzonder krachtige methods die je in de voorbeelden aan het werk kan zien:

- classList.add() : een klasse toevoegen
- classList.remove() : een klasse verwijderen
- classList.contains() : checken of een klasse aanwezig is
- classList.toggle() : een klasse toevoegen indien afwezig, verwijderen indien aanwezig

De classList eigenschap

```
<p class="klasse1 klasse2">Ik ben een paragraaf</p>
```

```
let elementKlasses = document.querySelector("p").classList;
console.log(elementKlasses.value); // klasse1 klasse2
console.log(elementKlasses[0]); // klasse1
console.log(elementKlasses.length); // 2
elementKlasses.add("klasse3"); // "klasse3" toegevoegd
elementKlasses.add("klasse4","klasse5"); // "klasse4" en "klasse5" toegevoegd
if (elementKlasses.contains("klasse3")) {
elementKlasses.remove("klasse3");
}
```

Events

Wat zijn events?

Op een webpagina vinden constant gebeurtenissen plaats. Een gebruiker beweegt met de muispijl over een bepaald gedeelte van een site, een pagina laadt in, een gebruiker klikt op een knop of op een link ... De lijst van events is lang. Kijk hier maar eens: https://www.w3schools.com/jsref/dom_obj_event.asp.

Sommige zijn algemeen (op pagina-niveau bvb), anderen kunnen op een specifiek element plaatsgrijpen. Voorbeelden: click, mouseover, mouseout, focus, blur, keyup, keydown, keypress...

Wat we met JavaScript gaan doen is met code vertellen dat we willen "luisteren" wanneer bepaalde van die events gebeuren. We kunnen er dan een functie aan koppelen die moet uitgevoerd worden telkens het event gebeurt.

De event property

Je neemt het element vast en koppelt de event handling code vast aan de gelijknamige property van het element.

```
<div id="mijndiv">Klik op mij als je kan :-)</div>
```

```
mijnDiv = document.querySelector("#mijndiv");
mijnDiv.onclick = function() {
    if (mijnDiv.style.backgroundColor === "yellow") {
        mijnDiv.style.backgroundColor = "orange";
    } else {
        mijnDiv.style.backgroundColor = "yellow";
    }
};
```

Event listeners

We kunnen ook werken met event listeners ("event luisteraars"). Je kan listeners toevoegen, verwijderen. Je kan een listener ook laten luisteren naar meerdere events. We gaan bij voorkeur gebruik maken van event listeners.

```
<div id="mijndiv">Klik op mij als je kan :-)</div>
```

```
mijnDiv = document.querySelector("#mijndiv");
function veranderVanKleur() {
  if (mijnDiv.style.backgroundColor === "yellow") {
        mijnDiv.style.backgroundColor = "orange";
    } else {
        mijnDiv.style.backgroundColor = "yellow";
    }
}
mijnDiv.addEventListener("click", veranderVanKleur);
```

Event listeners

Meerdere functies koppelen aan hetzelfde elementen:

```
mijnElement.addEventListener("click", mijnFunctie);
mijnElement.addEventListener("click", mijnTweedeFunctie);
mijnElement.addEventListener("click", function(){ console.log("Hallo Wereld!"); });
// laatste regel = een anonieme functie, kan ook
```

Event listeners

Verschillende events koppelen aan hetzelfde element:

```
mijnElement.addEventListener("mouseover", mijnFunctie);
mijnElement.addEventListener("click", mijnFunctie);
mijnElement.addEventListener("mouseout", mijnFunctie);
```

Event listeners

Events verwijderen van een element doe je op deze manier:

```
mijnElement.removeEventListener("mouseover", mijnFunctie);
mijnElement.removeEventListener("click", mijnFunctie);
mijnElement.removeEventListener("mouseout", mijnFunctie);
```

Event listeners

Events kunnen aan meerdere elementen tegelijk worden gekoppeld. Stel je een pagina voor met verschillende divs waaraan je een mouseover event en een mouseout event wil koppelen:

```
let mijnDivs = document.querySelectorAll("div");
let i = 0;
while (i < mijnDivs.length) {
    mijnElement[i].addEventListener("mouseover", function(){
        this.style.backgroundColor = "blue";
    });
    mijnElement[i].addEventListener("mouseout", function(){
        this.style.backgroundColor = "white";
    });
    i++;
}
```

Werken met "this"

We zagen het al in enkele van de voorgaande voorbeelden. Bij een functie die je aan een event koppelt verwijs je naar het element zelf met "this". We hebben dit al gezien bij object methods.

```
<button type="button">Klik op mij!</button>
```

```
let mijnKnop = document.querySelector("button");
function mijnFunctie() {
    this.classList.add("gekleurd");
}
mijnElement.addEventListener("click", mijnFunctie);
```

Timeouts en Intervallen

Timeouts en Intervallen

Soms willen we dat onze JavaScript code uitgesteld verloopt. We gebruiken dit bvb voor slideshows, games, visuele effecten ... We kunnen hiervoor beroep doen op timer methods. Het zijn geen events, maar voelen zo aan.

setTimeout()

setTimeout() gebeurt eenmaal en met aangegeven vertraging in milliseconden.

```
function simpeleBoodschap() {
     console.log("Ping ping!");
}

setTimeout(simpeleBoodschap,5000);
```

setInterval()

setInterval() voert een bepaalde functie constant uit, met een tussenpauze van een aangegeven interval in milliseconden.

```
function simpeleBoodschap() {
     console.log("Ping ping!");
}

setInterval(simpeleBoodschap,5000);
```

Timers neutraliseren

setTimeout en setInterval hebben allebei een neutraliserende tegenhanger: clearTimeout() en clearInterval().

```
let timeoutBehandeling = setTimeout(veranderAfb,50000);
mijnAfbeelding.addEventListener("click", function(){ clearTimeout(timeoutBehandeling); });

let intervalBehandeling = setInterval(veranderAfb,5000);
mijnAfbeelding.addEventListener("click", function(){ clearInterval(intervalBehandeling); });
```

Willekeurige getallen revisited

Math.random()

We zagen eerder dat deze method een willekeurig float-getal geeft tussen 0 en 1 (=> 0 maar <1). Onhandig om bvb. een dobbelsteenworp te simuleren, maar door gebruik te maken van enkele eenvoudige functies die je op volgende slides kan vinden wordt dit kinderspel.

```
let mijnWillekeurigGetal = Math.random();
console.log(mijnWillekeurigGetal); // 0.007036082356442597
mijnWillekeurigGetal = Math.random();
console.log(mijnWillekeurigGetal); // 0.21367095047572682
```

Een willekeurig getal tussen 2 waarden

Het volgende voorbeeld geeft een willekeurig getal terug tussen twee meegegeven waarden. Deze waarde is niet lager dan (wel mogelijk gelijk aan) min, en is kleiner dan (en niet gelijk aan) max.

```
function geefWillekeurigGetal(min, max) {
  return Math.random() * (max - min) + min;
}

console.log(geefWillekeurigGetal(1, 6)); // 1.198938711936496
console.log(geefWillekeurigGetal(1, 6)); // 5.399657152628693
console.log(geefWillekeurigGetal(1, 6)); // 4.781542674964008
console.log(geefWillekeurigGetal(1, 6)); // 2.4529292539948866
console.log(geefWillekeurigGetal(1, 6)); // 3.0686741921278715
```

Een willekeurig geheel getal tussen 2 waarden

Dit voorbeeld geeft een willekeurig geheel getal terug tussen twee meegegeven waarden. De waarde is niet lager dan min (of het volgende gehele getal groter dan min als min geen integer is), en het is kleiner dan (maar niet gelijk aan) max.

```
function geefWillekeurigGeheelGetal(min, max) {
  min = Math.ceil(min);
  max = Math.floor(max);
    return Math.floor(Math.random() * (max - min)) + min;
}

console.log(geefWillekeurigGeheelGetal(1, 6)); // 4
console.log(geefWillekeurigGeheelGetal(1, 42)); // 24
console.log(geefWillekeurigGeheelGetal(100, 100000)); // 16749
console.log(geefWillekeurigGeheelGetal(1, 12000000)); // 9668902
console.log(geefWillekeurigGeheelGetal(0.5, 2000.5)); // 1119
```

Willekeurig geheel getal tussen 2 waarden (inclusief)

Bij de vorige functie kon de max-waarde nooit teruggegeven worden. Met deze functie lossen we dit op. Deze is dus de veelzijdigste en we zullen ze heel vaak in onze oefeningen nodig hebben.

```
function geefWillekeurigGeheelGetalInclusief(min, max) {
  min = Math.ceil(min);
  max = Math.floor(max);
  return Math.floor(Math.random() * (max - min + 1)) + min;
}

console.log(geefWillekeurigGeheelGetalInclusief(1,6)); // 5
console.log(geefWillekeurigGeheelGetalInclusief(1,6)); // 6
console.log(geefWillekeurigGeheelGetalInclusief(1,6)); // 1
console.log(geefWillekeurigGeheelGetalInclusief(1,6)); // 1
console.log(geefWillekeurigGeheelGetalInclusief(1,6)); // 2
```

Forms en JavaScript

Toegang tot formuliervelden

De inhoud van een tekstveld kunnen we opvragen met de value eigenschap. Geldt ook voor checkboxes en radiobuttons.

```
<input type="text" id="mijnveld" naam="mijnveld">
```

```
let mijnVeld = document.querySelector("input");
let mijnTekst = mijnVeld.value;
```

Events voor formuliervelden

Ook form-elementen beschikken over events die je kan gebruiken. Voorbeelden hier zijn: focus (een gebruiker klikt op een veld), blur (een gebruiker gaat weg van een veld), change (de value van een veld verandert), keypress (een toets wordt ingedrukt bij een veld), ... We kunnen deze events op dezelfde manier als getoond bij de eerdere slides. Probeer het zelf eens!

Form events

Ook voor de form zelf bestaan er een aantal events. Hier zie je het submit event aan het werk. Het is een belangrijk event dat verhindert dat een klikken op submit de pagina opnieuw doet inladen.

```
document.querySelector("form").addEventListener("submit", function(){ return false; });
```

Je kan dit ook op element-niveau oplossen met event.preventDefault()

```
document.querySelector("#mijnlink").addEventListener("click", function(event){
  event.preventDefault()
});
```

Genoeg voor nu! We hebben vandaag gezien:

- DOM: werken met attributen
- DOM: elementen van styling voorzien
- DOM: werken met className en classList
- DOM: events
- Timeouts en intervallen
- Willekeurig getallen
- Werken met forms

IT2 - JavaScript: les 2

DOM: attributen, styling, classes, events, timeouts en intervallen, willekeurige getallen en forms

kristof.michiels01@ap.be