1 DOM Inleiding

Wanneer je een webpagina opent in een browser, haalt de browser de inhoud van de pagina op van de server en verwerkt deze. Tijdens het ophalen en verwerken van de inhoud van de pagina, bouwt de browser een model van de structuur van de webpagina. Dit model wordt gebruikt om de pagina op het scherm (of ander medium) weer te geven.

Dit model, het World Wide Web Consortium Document Object Model (W3C DOM) oftewel de voorstelling van de webpagina, kunnen we manipuleren in een script. Zo kunnen we gegevens uit het model ophalen in ons script en aanpassen.

Het model is een 'levende' datastructuur. Als we het model aanpassen, gaat de voorstelling van de webpagina op het scherm aangepast worden en onze aanpassingen zichtbaar zijn.

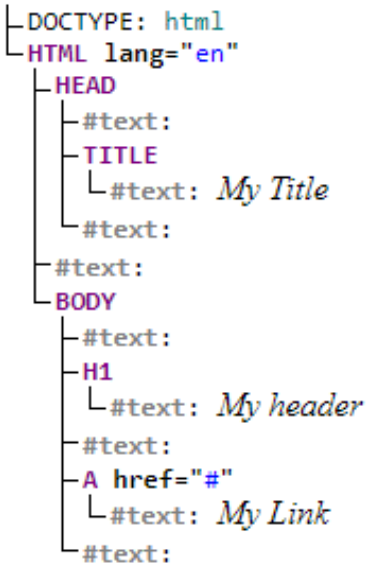
1.1 Het HTML DOM

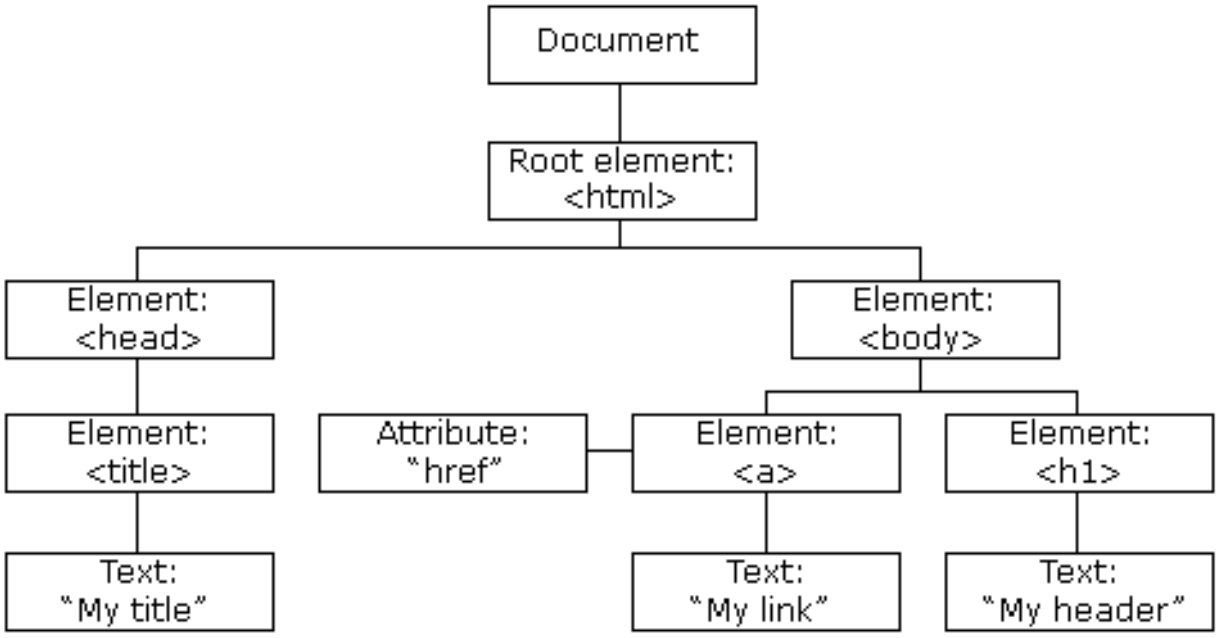
Het HTML DOM *is* het standaardmodel voor HTML-documenten en bevat:

- Alle HTML-elementen als objecten
- De attributen of eigenschappen van alle HTML-elementen
- De methoden om toegang te krijgen tot alle HTML-elementen
- De events voor alle HTML-elementen

HTML-code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>My Title</title>
</head>
<body>
    <h1>My header</h1>
    <a href="#">My Link</a>
</body>
</html>
```





Wil je graag zelf eens een HTML DOM view genereren van een HTML-pagina, surf dan naar de website van Ian Hickson (http://software.hixie.ch/utilities/js/live-dom-viewer/).

1.2 De relatie tussen het HTML DOM en Javascript

Met het HTML DOM, krijgt Javascript de kracht die het nodig heeft om een dynamische HTML te creëren. Het HTML DOM geef de mogelijkheid om:

- De HTML-elementen op een webpagina te benaderen
- De HTML-elementen op een webpagina aan te passen
- De HTML-attributen op een webpagina aan te passen
- De CSS-stijleigenschappen op een webpagina aan te passen
- Bestaande HTML-elementen te verwijderen
- Bestaande HTML-attributen te verwijderen
- Nieuwe HTML-elementen toe te voegen
- Nieuwe HTML-attributen toe te voegen
- Te reageren op bestaande HTML-events op een webpagina
- Nieuwe HTML-events te creëren in een webpagina
- …

We hebben, met Javascript, toegang tot het HTML DOM. In het HTML DOM zijn alle elementen gedefinieerd als objecten. De programmeerinterface bestaat uit de attributen en methoden van ieder object:

- De HTML DOM methoden zijn acties die we kunnen uitvoeren op de elementen.
- De HTML DOM eigenschappen of attributen van elementen zijn waarden die we kunnen instellen of veranderen.

Via de attributen en methoden van de objecten kunnen we bestaande HTML-elementen benaderen, aanpassen of verwijderen en nieuwe HTML-elementen toevoegen. Dit noemen we ook wel ‘DOM manipulatie’.

1.3 Begrippen van het HTML DOM

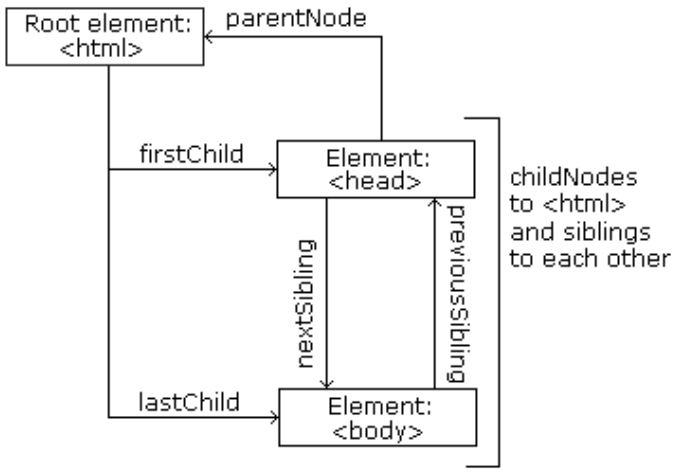
Vooraleer we aan de slag gaan met het HTML DOM, moeten we eerst enkele belangrijke begrippen onder de knie krijgen. Deze begrippen komen meermaals terug in dit cursusdeel.

Alles in een HTML document noemen we een 'node' en alle nodes kunnen we aanspreken en gebruiken in Javascript. We kunnen nieuwe nodes creëren en bestaande nodes aanpassen of verwijderen.

Document node	Het volledige document.
Element node	Een HTML-element
Text node	De tekst in een HTML-element
Attribute node (deprecated)	Het attribuut van een HTML-element
Comment node	Een stukje commentaar

Alle nodes in de DOM hebben een hiërarchische relatie tegenover elkaar.

Root node	De top node, de hoogste node in een document.
Parent node	Elke node heeft exact één parent node, met uitzondering van de rootnode.
Child nodes	Een parent node kan meerdere child nodes hebben.
Siblings	Child nodes met dezelfde parent node zijn siblings.



2 DOM Manipulatie – Deel 1

2.1 Benaderen van Element nodes

Er bestaan verschillende HTML DOM methoden om bestaande HTML-elementen te zoeken en te benaderen in het DOM. Volgende methoden komen in deze cursus aan bod:

- Benaderen via id: getElementById()
- Benaderen via tag-naam: getElementsByTagName()
- Benaderen via class-naam: getElementsByClassName()
- Benaderen via naam: getElementsByName()
- Benaderen via CSS-selector: querySelector() of querySelectorAll()

Het resultaat van deze methoden is ofwel één object, ofwel een verzameling objecten.

Opgelet! Bij sommige methoden plaatsen we een 's' achter het woord 'Element'. Dit doen we omdat bij deze methoden de kans bestaat dat er meerdere elementen terug gegeven kunnen worden als verzameling objecten.

2.1.1 Benaderen via id

We kunnen via de methode getElementById() één uniek HTML-element benaderen in het DOM door gebruik te maken van de id van dat element. Deze methode hebben we in de voorgaande hoofdstukken al regelmatig gebruikt en is de eenvoudigste en meest gebruikte manier om elementen te benaderen in het DOM.

HTML-code

```
<body>
    <p id="demo">Dit is een paragraaf</p>
</body>
```

JS-code

```
// Geeft het element als één object terug.
let object = document.getElementById("demo");

// Geeft null terug, want het element met id 'test' werd niet gevonden.
let object = document.getElementById("test");
```

2.1.2 Benaderen via tag-naam

We kunnen via de methode getElementsByTagName() alle HTML-elementen van eenzelfde type benaderen in het DOM door gebruik te maken van de tag-naam.

HTML-code

```
<div id=”tekst”>
    <p>Dit is een paragraaf</p>
    <p>Dit is een tweede paragraaf</p>
</div>
```

JS-code

```
// Geeft alle p-elementen als een object collectie terug.
let object = document.getElement<b>s</b>ByTagName(“p”);

// Geeft het eerste p-element als een object terug.
let object = document.getElement<b>s</b>ByTagName(“p”)[0];

// Geeft het tweede p-element als een object terug.
let object = document.getElement<b>s</b>ByTagName(“p”)[1];

// Geeft het tweede p-element binnen het element met de id ‘tekst’ als
   een object terug.
let x = document.getElementById(“tekst”);
let y = x.getElement<b>s</b>ByTagName(“p”)[1];
```

2.1.3 Benaderen via class-naam

We kunnen via de methode getElementsByClassName() meerdere HTML-elementen met dezelfde class benaderen in het DOM.

HTML-code

```
<body>
    <h1 class=”demo”>Dit is een titel</h1>
    <p class=”demo”>Dit is een paragraaf</p>
</body>
```

JS-code

```
// Geeft alle elementen met de class 'demo' als een object collectie
   terug.
let object = document.getElements**s**ByClassName("demo");

// Geeft null terug, want een element met class 'test' bestaat niet.
let object = document.getElements**s**ByClassName("test");
```

2.1.4 Benaderen via name

We kunnen via de methode getElementsByName() meerdere HTML-elementen met dezelfde name benaderen in het DOM.

HTML-code

```
<form action="mailto:kw@pxl.be" method="post" enctype="text/plain">
    <label for="naam">Naam</label>
    <input type="text" name="volledigenaam" id="naam" />
    ...
</form>
```

JS-code

```
// Geeft het element met de name 'volledigenaam' als een object terug.
let object = document.getElements**s**ByName("volledigenaam");

// Geeft null terug, want een element met name 'test' bestaat niet.
let object = document.getElements**s**ByName("test");
```

2.1.5 Benaderen via CSS selector

We kunnen via de methode querySelectorAll() meerdere HTML-elementen die met een specifieke CSS-selector (id, class, tag, geneste selector, …) matchen, benaderen in het DOM.

HTML-code

```
<body>
    <h1 class="demo">Dit is een titel</h1>
    <p class="demo">Dit is een paragraaf</p>
    <p>Dit is een tweede paragraaf</p>
    ...
</body>
```

CSS-code

```
.demo {
    Font-weight: 400;
    font-style: italic;
}
p.demo {
    color: #f00;
    font-size: 16px;
}
```

JS-code

```
// Geeft het element met de CSS-selector "p.demo" als een object terug.
let object = document.querySelectorAll("p.demo");

// Geeft alle elementen met de CSS-selector ".demo" als object collectie
   terug.
let object = document.querySelectorAll(".demo");
```

Naast de methode querySelectorAll() kunnen we ook de methode querySelector() gebruiken. Het gebruik is afhankelijk van onze wensen:

- De methode querySelectorAll() benadert alle elementen die met de opgegeven CSS-selector matcht. Matcht er maar één element, wordt deze in een object collectie teruggegeven.
- De methode querySelector() benadert slechts één element die met de CSS-selector matcht. Matchen er meerdere elementen, wordt het eerste element als object teruggegeven.

3 DOM Manipulatie - Deel 2

3.1 Aanpassen van bestaande nodes

Via de HTML DOM attributen en eigenschappen kunnen we bestaande HTML DOM nodes aanpassen.

3.1.1 Aanpassen van inhoud van bestaande HTML-elementen

De makkelijkste manier om de inhoud van een html-element aan te passen is de eigenschap 'innerHTML'. Deze eigenschap hebben we al meermaals doorheen deze cursus gebruikt.

JS-code

```
// De paragraaf met id 'uitkomst' krijgt een nieuw stukje tekst.
document.getElementById("uitkomst").innerHTML = "Nieuwe tekst!";
```

3.1.2 Aanpassen van de attribuut-waarden

Via de attribuutnaam kunnen we de waarden van een HTML-attribuut aanpassen.

JS-code

```
// De afbeelding met id 'logo' krijgt een nieuwe image source.
document.getElementById("uitkomst").src = "assets/logo2.png";
```

3.1.3 Aanpassen van de CSS-stijleigenschappen

Via de stijleigenschap kunnen we de opmaak van specifieke HTML-elementen aanpassen.

JS-code

```
// De paragraaf met id 'uitkomst' krijgt de tekstkleur rood.
document.getElementById("uitkomst").style.color = "#f00";
```

3.2 Toevoegen van nieuwe nodes

We kunnen via Javascript 'runtime' nieuwe HTML-elementen met inhoud aanmaken. Dit doen we met de volgende twee functies:

- Met 'createElement()' maken we het nieuwe HTML-element aan.
- Met 'createTextNode()' maken we de platte tekst die als inhoud in het HTML-element moet komen te staan.

Naast het aanmaken van de nieuwe HTML-elementen met inhoud, moeten we ook aangeven waar deze HTML-elementen mogen worden toegevoegd aan het DOM. Dit doen we met één van de volgende functies:

- Met 'appendChild()' kan je een node toevoegen als laatste node aan een geselecteerd element.
- Met 'insertBefore()' kan je een node toevoegen aan een geselecteerd element en aangeven op welke locatie binnen dat geselecteerde element.

Voorbeeld met appendChild()

HTML-code

```
// Header in de HTML-code met een hoofding niveau 1.
<header id="startheader">
   <h1>Welkom op deze website!</h1>
</header>
```

JS-code

```
// Maak een nieuwe paragraaf aan.
let nieuweParagraaf = document.createElement("p");

// Geef de paragraaf optioneel een id (bijvoorbeeld 'test').
nieuweParagraaf.id = "test";

// Maak een tekstnode aan.
let tekstParagraaf = document.createTextNode("Dit is een stukje tekst");

// Voeg de tekstnode toe aan de nieuwe paragraaf.
nieuweParagraaf.appendChild(tekstParagraaf);

// Voeg de paragraaf toe aan de header als laatste child-element.
document.getElementById("startheader").appendChild(nieuweParagraaf);
```

```
▼<header id="startheader">
    <h1>Welkom op deze website!</h1>
    <p id="test">Dit is een stukje tekst</p>
  </header>
```

Voorbeeld met insertBefore()

HTML-code

```
// Header in de HTML-code met een hoofding niveau 1.
<header id="startheader">
   <h1 id="titel">Welkom op deze website!</h1>
</header>
```

JS-code

```
// Maak een nieuwe paragraaf aan.
let nieuweParagraaf = document.createElement("p");

// Geef de paragraaf optioneel een id.
nieuweParagraaf.id = "test";

// Maak een tekstnode aan.
let tekstParagraaf = document.createTextNode("Dit is een stukje tekst");

// Voeg de tekstnode toe aan de nieuwe paragraaf.
nieuweParagraaf.appendChild(tekstParagraaf);

// Voeg de paragraaf toe aan de header als eerste child-element.
document.getElementById("startheader").insertBefore(nieuweParagraaf,
document.getElementById("titel"));
```

```
▼<header id="startheader">
    <p id="test">Dit is een stukje tekst</p>
    <h1 id="titel">Welkom op deze website!</h1>
  </header>
```

Bij de functie 'insertBefore()' worden altijd twee parameters meegeven, namelijk:

- het element dat ingevoegd moet worden;
- het element waarvoor het ingevoegd moet worden.

Je kan voor de tweede parameter ook via de eigenschappen 'firstChild' en 'lastChild' van de parent werken.

JS-code

```
// Voeg de paragraaf toe aan de header als eerste child-element.
document.getElementById("startheader").insertBefore(nieuweParagraaf,
document.getElementById("startheader").firstChild);
```

3.3 Verwijderen van bestaande nodes

Met 'removechild()' kunnen we een element uit het DOM verwijden.

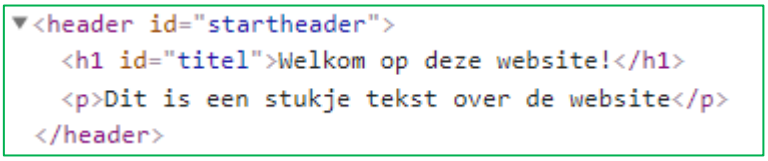
HTML-code

```
// Header in de HTML-code met een hoofding niveau 1 en twee paragrafen.
<header id="startheader">
   <h1 id="titel">Welkom op deze website!</h1>
   <p>Dit is een stukje tekst over de website</p>
   <p>Dit is een random quote – Random Persoon</p>
</header>
```

JS-code

```
// Verwijder de laatste paragraaf in de header uit het dom.
// Om de laatste paragraaf te krijgen gebruiken we een CSS-psuedoklasse!
let header = document.getElementById("startheader");
let paragraaf = header.querySelectorAll("p:last-child")[0];

header.removeChild(paragraaf);
console.log("De laatste paragraaf is nu weg!");
```



Bovenstaande functie roep je nooit aan voor het desbetreffende element, maar voor de parent ervan. Het te verwijderen element geef je in de functie als parameter mee.

3.4 Overige DOM-methodes en eigenschappen

Javascript heeft nog veel meer functies en eigenschappen om met het DOM aan de slag te gaan. In onderstaande lijst vind je de meest gebruikte. *Uit de naamgeving is vaak af te leiden wat deze functies en eigenschappen doen/zijn, maar meer opzoekwerk is aangeraden!*

Functies:

- cloneNode()
- getAttribute()
- removeAttribute()
- setAttribute()
- …

Eigenschappen:

- nodeName
- nodeType
- parentNode
- childNodes[]
- firstChild
- lastChild
- previousSibling
- nextSibling
- …

Meer informatie?

https://www.w3schools.com/js/js_htmldom_nodes.asp
https://www.w3schools.com/js/js_htmldom_navigation.asp