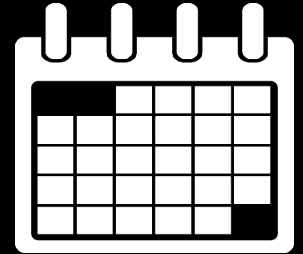


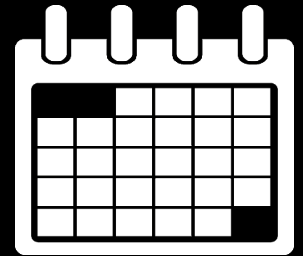
# User Interfaces 1

# JavaScript

DOM



1. `<script>` tag
2. window object
3. document object
4. DOM API
  1. Element selecteren
  2. Elementen doorlopen
  3. Elementen aanpassen
  4. Attributen lezen en schrijven



1. `<script>` tag
2. window object
3. document object
4. DOM API
  1. Element selecteren
  2. Elementen doorlopen
  3. Elementen aanpassen
  4. Attributen lezen en schrijven

## <script> tag

---

- Om JavaScript in je webpagina te krijgen maak je gebruik van het <script> tag

```
<script src="js/script.js" type="text/javascript"></script>
```

- Attributen:

src

- Url naar een extern script-bestand

type

- Geeft de script-taal aan, a.d.h.v. media-type
- Standaard 'text/javascript' -> JavaScript



**Géén inline code!**

Je kan rechtstreeks in de script-tag code schrijven, maar dit doen we niet.

## <script> tag: verwerking

---

- Een `<script>` tag wordt **direct verwerkt/uitgevoerd op het moment dat de parser het tegenkomt** bij het lezen van het html-document!
- Plaatsing in het html-document:
  - In het **<head> element**
    - Wat met interactie met het html-document, want deze is nog niet gelezen/verwerkt door parser?  
=> zie '**defer**', '**document object**' en '**Events**'
    - Meerdere script-tags: let op volgorde!



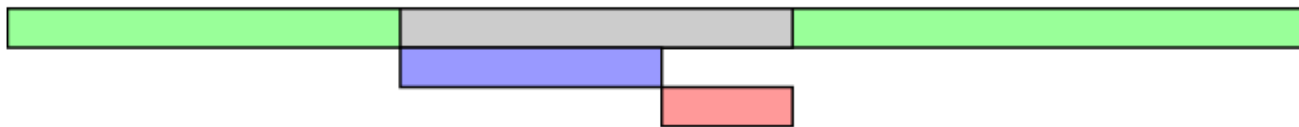
Oudere technieken (**bad-practices, passen we niet toe!**):

- Ergens in het `<body>` element
- Op het einde van het html-document: juist voor of na de sluitingstag `</body>`

# <script> tag: verwerking defer

## <script>

Let's start by defining what `<script>` without any attributes does. The HTML file will be parsed until the script file is hit, at that point parsing will stop and a request will be made to fetch the file (if it's external). The script will then be executed before parsing is resumed.

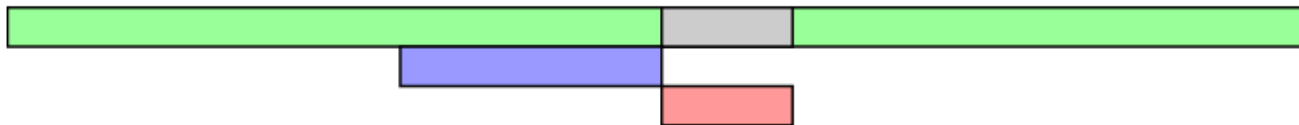


## Legend

- HTML parsing
- HTML parsing paused
- Script download
- Script execution

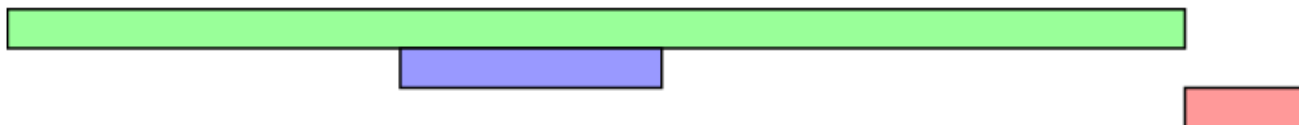
## <script async>

`async` downloads the file during HTML parsing and will pause the HTML parser to execute it when it has finished downloading.

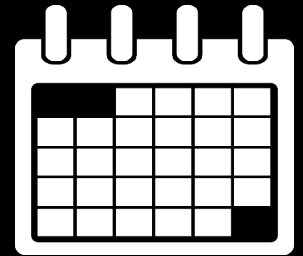


## <script defer>

`defer` downloads the file during HTML parsing and will only execute it after the parser has completed. `defer` scripts are also guaranteed to execute in the order that they appear in the document.



**Voorkeur!**



1. `<script>` tag
2. `window` object
3. `document` object
4. **DOM API**
  1. Element selecteren
  2. Elementen doorlopen
  3. Elementen aanpassen
  4. Attributen lezen en schrijven

# Het window object

---

```
let url = window.prompt("Geef een URL met http://");  
window.alert("Redirecting to new URL!");  
window.location = url;
```

- Een script dat uitgevoerd wordt binnen een browser heeft automatisch de beschikking over het `window` object (dit stelt het browservenster voor)
- In het voorbeeld gebruiken we de methoden `alert()` en `prompt()`, en de eigenschap `location` van het `window` object
- Het `window` object is het "global object" en mag weggelaten worden  
bv. `location()` ipv `window.location()`



# Het window object

---

<https://developer.mozilla.org/en/DOM/window>

geeft een overzicht van alle eigenschappen en methodes

Belangrijkste attributen:

- **document**, location, name, history, screen, ...

Belangrijkste methodes:

- **addEventListener()**, alert(), prompt(),  
confirm(), close(), print(), back(), home(), ...

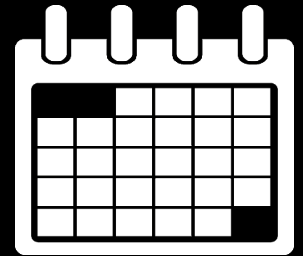
# Het window object

---

```
console.log(window.screen.width);  
console.log(window.screen.availWidth);  
console.log(window.screen.height);  
console.log(window.screen.availHeight);  
console.log(window.navigator.userAgent);  
console.log(window.location.href);
```

```
1920  
1920  
1080  
1040  
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.149 Safari/537.36  
http://localhost:63342/WindowObject/WindowObject.html
```

```
1920  
1920  
1080  
1080  
Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:74.0) Gecko/20100101 Firefox/74.0  
http://localhost:63342/WindowObject/WindowObject.html
```



1. `<script>` tag
2. window object
3. document object
4. DOM API
  1. Element selecteren
  2. Elementen doorlopen
  3. Elementen aanpassen
  4. Attributen lezen en schrijven

# HTML = boomstructuur

---

## An HTML Document

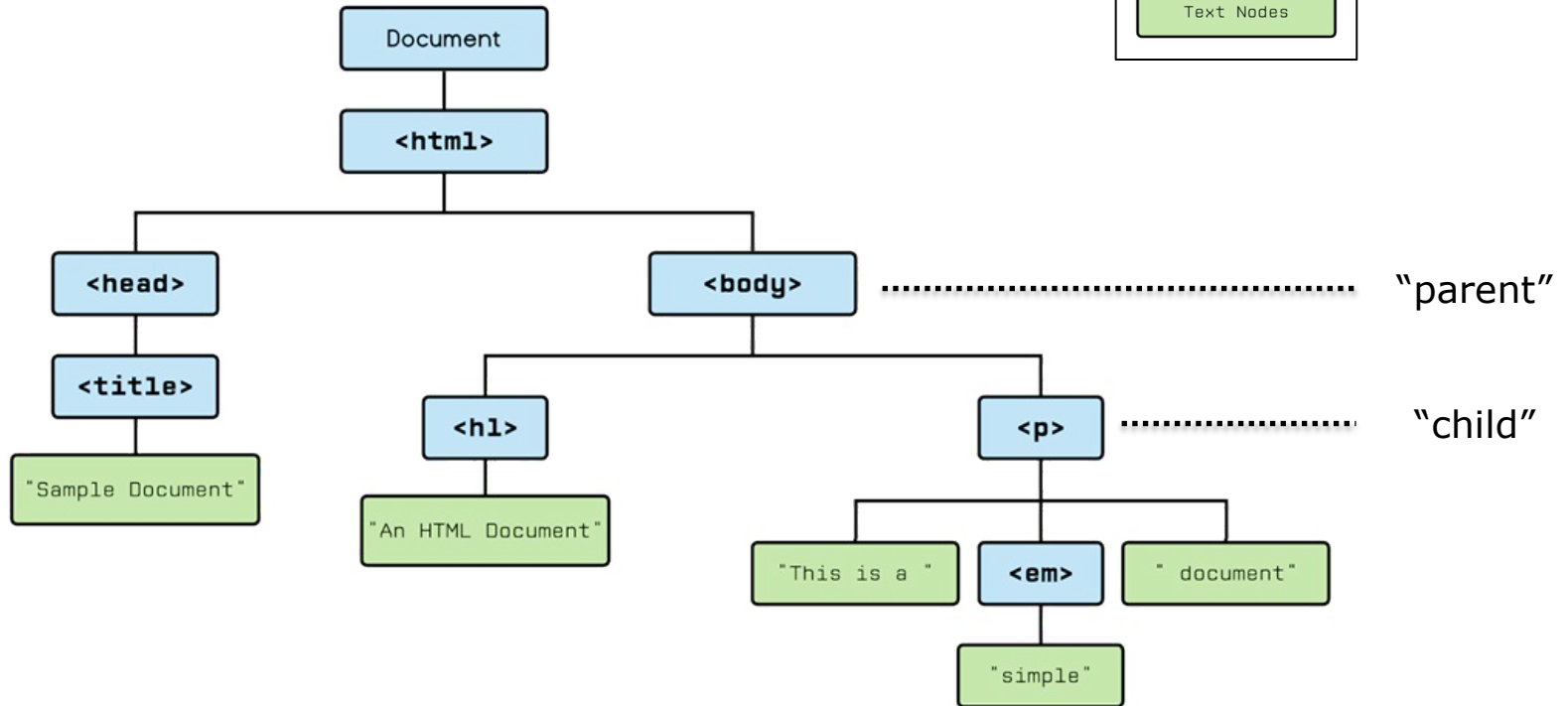
*This is a simple document.*

```
<html>
<head>
  <title>Sample Document</title>
</head>
<body>
  <h1>An HTML Document</h1>
  <p>This is a <em>simple</em> document.</p>
</body>
</html>
```

Elk HTML document kan je beschouwen als een boomstructuur

# HTML = boomstructuur

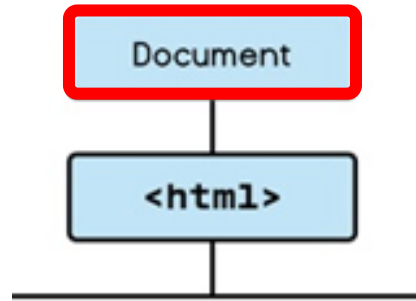
```
<html>
<head>
  <title>Sample Document</title>
</head>
<body>
  <h1>An HTML Document</h1>
  <p>This is a <em>simple</em> document.</p>
</body>
</html>
```



Elk HTML document kan je beschouwen als een boomstructuur

# document **object**

---



- Stelt de "root" van de boomstructuur voor.
- Is een attribuut van het `window` object.
- Heeft zelf attributen en methodes om de overige *nodes* van de boom op te zoeken, te inspecteren en te wijzigen  
=> DOM (document object model).

# DOM: Document Object Model

---

- Set van API's om vanuit JavaScript het HTML document te kunnen uitlezen en aanpassen. Start vanuit het `document` object.
  - W3C standaard, door elke browser "*op zijn manier*" geïmplementeerd.
  - Zeer uitgebreid: zie <https://developer.mozilla.org/en/DOM>
- *We bespreken uitsluitend de belangrijkste objecten en methodes*

# Probleem bij laden DOM script

index.html:

```
<html>
<head>
  <meta charset="utf-8" />
  <title>DOM Probleem</title>
  <script src="scripts/vraag.js"></script>
</head>
<body>
  <p>
    Welkom <span id="user"></span>
  </p>
</body>
</html>
```

scripts/vraag.js:

```
const naam = prompt("Geef je naam:");
let element =
  document.getElementById("user");
element.innerHTML = naam;
```

- We vragen de gebruiker zijn naam
- Het element met id 'user' wordt opgehaald
- De inhoud van dat element wordt opgevuld met de naam van de gebruiker

✖ Uncaught TypeError: Cannot set property 'innerHTML' of null




→ Als we dit testen blijkt het toch niet te werken. Waarom?  
→ Op moment van uitvoering is het HTML document nog niet in de browser ingelezen!  
bewijs: voer enkel het geel gekleurd deel (kopieer) van de code uit  
in de browser-console (F12)...




# Oplossing 1 laadprobleem

index.html:

```
<html>
<head>
  <meta charset="utf-8" />
  <title>DOM Oplossing</title>
  <script src="scripts/vraag.js"></script>
</head>
<body>
  <p>
    Welkom <span id="user"></span>
  </p>
</body>
</html>
```



scripts/vraag.js:



```
window.addEventListener("load", init);

function init() {
  const naam = prompt("Geef je naam:");
  let element =
    document.getElementById("user");
  element.innerHTML = naam;
}
```

We stellen het uitvoeren van de code uit tot het document in de browser geladen is.

Met deze `addEventListener` functie zeggen we:

Als het `load` event optreedt  
(de browser-pagina is volledig ingelezen),  
roep dan de `init` functie aan...

Zie onderdeel 'Events'!

# Oplossing 2 laadprobleem

index.html:

```
<html>
<head>
  <meta charset="utf-8" />
  <title>DOM Probleem</title>
  <script src="scripts/vraag.js" defer>
  </script>
</head>
<body>
  <p>
    Welkom <span id="user"></span>
  </p>
</body>
</html>
```

scripts/vraag.js:

```
const naam = prompt("Geef je naam:");
let element =
    document.getElementById("user");
element.innerHTML = naam;
```

Het script wordt direct geladen, maar het uitvoeren van de code wordt automatisch uitgesteld tot het document in de browser geladen is.

Zie onderdeel '<script> tag'!

# Oplossing voor laden DOM scripts

---



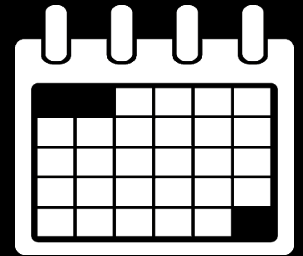
Vanaf nu gebruiken we altijd deze techniek:

- zorgt ervoor dat de code onder functie '*functienaam*' pas opgestart wordt nadat het html-document volledig geladen is a.d.h.v. `load` event van het `window` object

```
window.addEventListener("load", functienaam);
```

- al dan niet gecombineerd met het `defer` attribuut op de `<script>` tag om de uitvoering van het script-bestand uit te stellen tot na het parsen van het html-document

```
<script src="js/script.js" defer><script>
```



1. `<script>` tag
2. window object
3. document object
4. **DOM API**
  1. Element selecteren
  2. Elementen doorlopen
  3. Elementen aanpassen
  4. Attributen lezen en schrijven

# DOM API

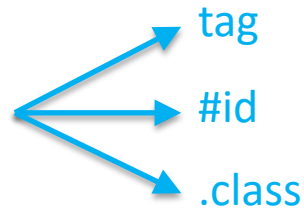
---

- Elementen selecteren

```
{  
- getElementById()  
- getElementsByTagName()  
- getElementsByClassName()  
}
```

```
- querySelector()  
- querySelectorAll()
```

css syntax !



+ combinaties zoals in css rules

- Elementen doorlopen

```
- parentNode  
- childNodes
```

- Elementen maken

```
- createElement
```

- Elementen aanpassen

```
- insertAdjacentElement / insertAdjacentHTML  
- innerHTML  
- nodeValue
```

- Attributen lezen en schrijven

```
- get- / setAttribute()  
- Standaard attributen: via eigenschappen van het element
```

# document.getElementById

---

```
<body>
  <p>Regel 1 <span></span></p>
  <p>Regel 2 <span id="uniek"></span></p>
</body>
```

```
addEventListener("load", uniek);

function uniek() {
  let element = document.getElementById("uniek");
  element.innerHTML = "Je bent uniek!";
}
```

Regel 1

Regel 2 Je bent uniek!

Haalt het (eerste) element met de gegeven **id** uit het HTML document op

(in principe kan er maar één zo'n id in een zelfde webpagina bestaan)

*Het document moet vooraf geladen zijn!*

# document.getElementsByTagName

```
<body>
  <p>Welkom <span></span></p>
  <p>Welkom <span></span></p>
</body>
```

Geeft een **array** van alle elementen met de opgegeven **tag name** terug

```
addEventListener("load", welkom);
```

De volgorde in de array is de volgorde waarin ze voorkomen in het HTML document

```
function welkom() {
  let element = document.getElementsByTagName("span")[0];
  element.innerHTML = "Jos";

  document.getElementsByTagName("span")[1].innerHTML = "Jennifer";
}
```

Welkom Jos

Welkom Jennifer

```
addEventListener("load", welkom);
```

```
function welkom() {
  let element = document.getElementsByTagName("span");
  element.innerHTML = "Jos";
  console.log(element);

  document.getElementsByTagName("span")[1].innerHTML = "Jennifer";
}
```

index 'vergeten'...

▼ HTMLCollection(2)  
length: 2  
▶ 0: span  
▶ 1: span  
innerHTML: "Jos"

het attribuut wordt ipv aan het element aan de lijst toegekend!  
Dit heeft echter geen enkele zinvolle betekenis voor onze webpagina...

# document.getElementsByClassName

Geeft een **array** met alle elementen van die opgegeven **class** terug

De volgorde in de array is de volgorde waarin ze voorkomen in het HTML document

```
addEventListener("load", groet);
```

```
function groet() {  
  let groeten = document.getElementsByClassName("groet");  
  console.log(groeten);  
  groeten[0].innerHTML = "Jos";  
  groeten[1].innerHTML = "Jennifer";  
}
```

▼ HTMLCollection(2)  
length: 2  
▶ 0: span.groet  
▶ 1: p.groet

```
<body>  
  <p>Demo</p>  
  <p>Hallo <span class="groet"></span></p>  
  <p class="groet">Welkom</p>  
</body>
```

Demo

Hallo

Welkom

```
<body>  
  <p>Demo</p>  
  <p>Hallo <span class="groet">Jos</span></p>  
  <p class="groet">Jennifer</p>  
</body>
```

Demo

Hallo Jos

Jennifer



# document.querySelector

---

Geeft het **eerste** element dat voldoet aan de beschrijving in de CSS selector

```
addEventListener("load", query);

function query() {
  document.querySelector("p.example span#user").innerHTML = "USER";
  document.querySelector("p.example span").innerHTML = "Jos";
}
```

```
<body>
  <p>Demo</p>
  <p class="example">
    Welkom <span></span>
  </p>
  <p class="example">
    Welkom <span id="user"></span>
  </p>
</body>
```

Demo  
Welkom  
Welkom

```
<body>
  <p>Demo</p>
  <p class="example">
    Welkom <span>Jos</span>
  </p>
  <p class="example">
    Welkom <span id="user">USER</span>
  </p>
</body>
```

Demo  
Welkom Jos  
Welkom USER

# document.querySelectorAll

Geeft **alle** elementen die voldoet aan de beschrijving in de CSS selector

```
addEventListener("load", selector);

function selector() {
  let list = document.querySelectorAll("h2, p.wel");
  console.log(list);
  for (item of list) {
    item.style.color = "red";
  }
}
```

▼ NodeList(3)  
length: 3  
▶ 0: h2  
▶ 1: h2.ha  
▶ 2: p.wel

```
<body>
  <h1>Demo</h1>
  <h2>Ha</h2>
  <h2 class="ha">ha</h2>
  <p>No</p>
  <p class="wel">Yes</p>
</body>
```

Demo

Ha

ha

No

Yes

Demo

Ha

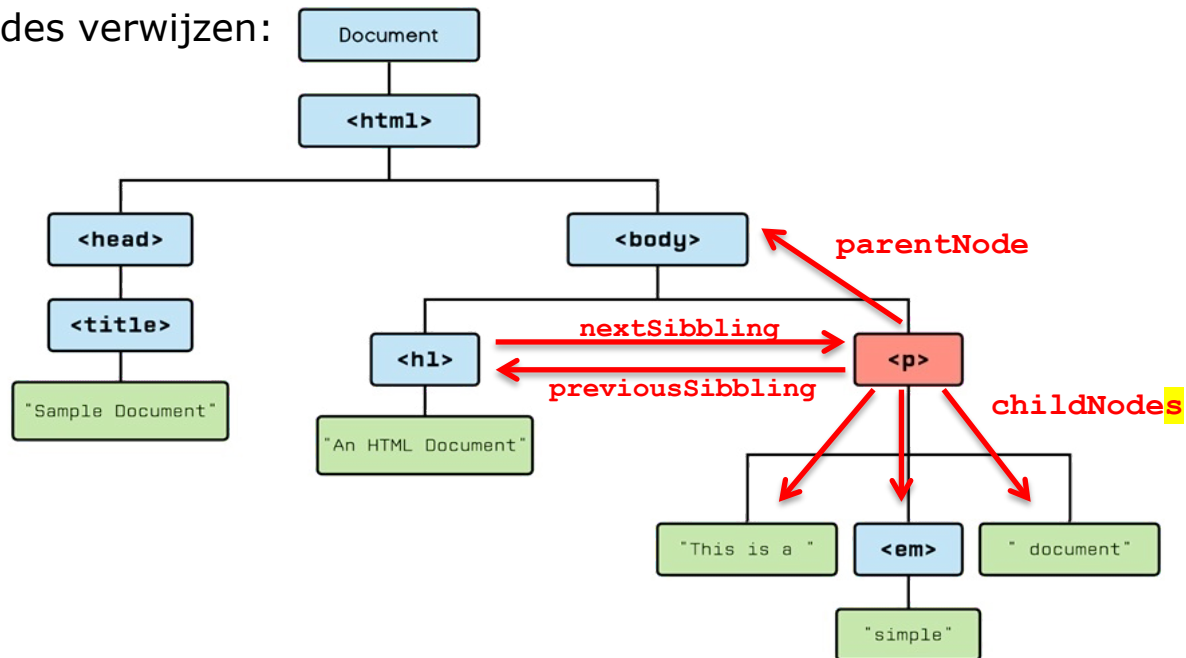
ha

No

Yes

# Nodes/Elementen doorlopen

De DOM boomstructuur bestaat uit nodes, waarbij elke node attributen heeft die naar gerelateerde nodes verwijzen:



**Nodes die een HTML-element voorstellen zijn van het type 'Element'** (in het voorbeeld voorgesteld als `<tag>`) en hebben extra eigenschappen en methoden om gerelateerde elementen aan te spreken!

Opgelet: een Text-node (groen) zijn dus geen Element-node en is dus niet bereikbaar via de specifieke eigenschappen en methoden van een Element-node!



**Opgelet:**

Je webpagina dynamisch wijzigen, wijzigt ook de structuur!

# Node vs Element

---

## Node

### Eigenschaften

```
parentNode  
childNodes  
first- / lastChild  
previous- / nextSibling  
nodeType / nodeValue  
...
```

### Methoden

```
getRootNode()  
hasChildNodes()  
contains()  
insertBefore()  
removeChild()  
replaceChild()  
appendChild()  
...
```

## Element<sub>-node</sub>

### Extra eigenschappen

```
children  
first- / lastElementChild  
previous- / nextElementSibling  
attributes  
id  
className / classList  
style  
innerHTML  
...
```

### Extra methoden

```
getElementsBy...()  
querySelector...()  
insertAdjacentElement() / -HTML()  
before() / after()  
append()  
hasAttribute()  
get- / setAttribute()  
removeAttribute()  
...
```

# parentNode



```
<body>
  <p>Demo <span></span></p>
  <p>Welkom <span></span></p>
</body>
```

Demo

Welkom

```
<body>
  <p>Demo <span>parentNode</span></p>
  <p>Verdwijn <span>Jos</span></p>
</body>
```

Demo parentNode

Verdwijn Jos

```
addEventListener("load", parent);

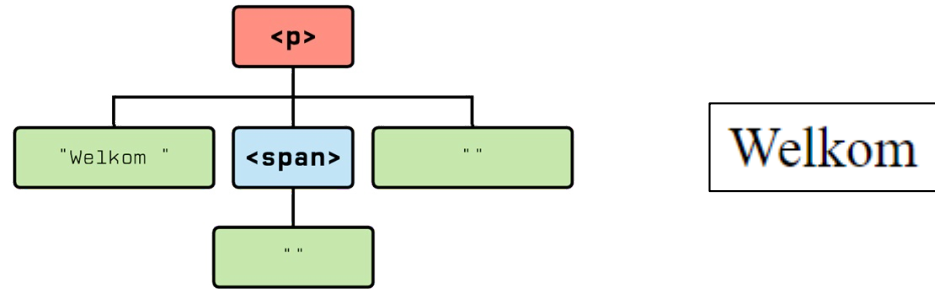
function parent() {
  let tweedeSpan = document.querySelectorAll("span")[1];
  tweedeSpan.innerHTML = "Jos";
  let parent = tweedeSpan.parentNode;
  parent.childNodes[0].nodeValue = "Verdwijn ";
  document.querySelectorAll("span")[0].innerHTML = " parentNode";
}
```

**parentNode** bevat het unieke parent element...

# childNodes

**childNodes** is een **array** van alle nodes (elements én textnodes) die onder het element zitten...

```
<body>
  <p class="parent">
    Welkom <span id="user"></span>
  </p>
</body>
```



```
addEventListener("load", child);

function child() {
  let parent = document.querySelector(".parent");
  console.log(parent.childNodes);
  spanElement = parent.childNodes[1];
  parent.childNodes[0].nodeValue = "voor ";
  parent.childNodes[2].nodeValue = " achter";
  spanElement.innerHTML = "Jos";
}
```

▼ NodeList(3) ⓘ  
length: 3  
▶ 0: text  
▶ 1: span#user  
▶ 2: text

# childNodes

```
addEventListener("load", child);

function child() {
  let parent = document.querySelector(".parent");
  console.log(parent.childNodes);
  let spanElement = parent.childNodes[1];
  parent.childNodes[0].nodeValue = "voor ";
  parent.childNodes[2].nodeValue = " achter";
  spanElement.innerHTML = "Jos";
}
```

<element>

**innerHTML** = tussen openings- en sluitingstag  
→ heeft dus altijd een tag nodig...

"Text"

**nodeValue** = inhoud van de string

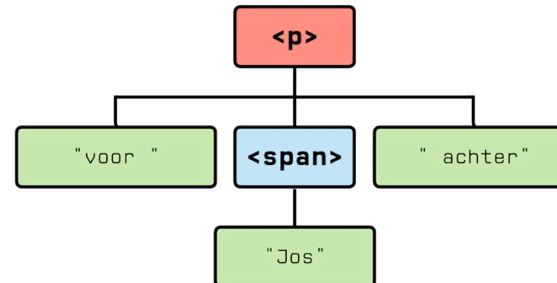
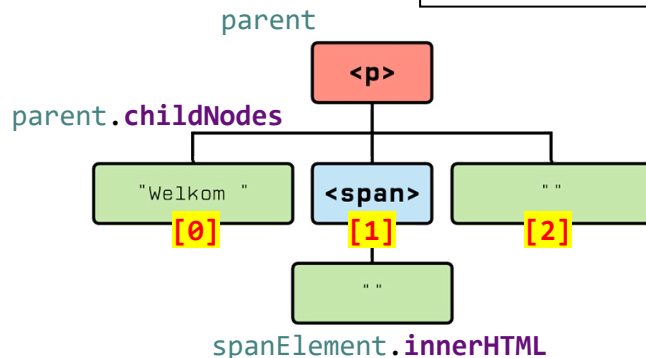
het **innerHTML** attribuut van elementen en  
het **nodeValue** attribuut van textnodes

<p class="parent">Welkom <span id="user"></span></p>

Welkom

<p class="parent">voor <span id="user">Jos</span> achter</p>

voor Jos achter



## Node.childNodes vs Element.children

de eigenschap 'children' is van een Element-node, geeft enkel de kinderen van het type 'Element' terug en dus niet de Text-nodes!

# Elementen maken

---

Dit kan op 2 manieren:

- `document.createElement()`
  - element moet nadien nog a.d.h.v. DOM API's verder uitgewerkt worden (attributen, inhoud...)

```
let newParagraphe = document.createElement("p");
```

- (html-)string literal
  - string literal moet 'geldige' html-syntax zijn!

```
let newParagrapheString = "<p> ... </p>";
```

=> Deze nieuwe inhoud moet dan nog in de DOM-structuur verwerkt worden !!



# Elementen aanpassen

---

- Inhoud
  - `insertAdjacentElement()` / `-HTML()`
    - `append()` / `appendChild()`
    - `before()` / `after()`
  - `innerHTML`
  - `nodeValue`
- Attributen
  - `get-` / `setAttribute()`
  - `removeAttribute()`
  - Als eigenschap van Element
- Opmaak (CSS)
  - `style` attribute
  - `className` / `classList`

# Inhoud: innerHTML

---

innerHTML bevat de HTML die in het element zit en kan aangepast worden...

```
<body>
  <p class="groet">Welkom <span>Jos</span></p>
  <p>Inhoud: <span></span></p>
</body>
```

opgelet met 'empty' elementen  
vb. innerHTML van <img>

```
addEventListener("load", inner);

function inner() {
  let element = document.querySelector(".groet"); // eerste class=="groet"
  let span = document.querySelectorAll("span")[1]; // tweede span
  span.innerHTML = element.innerHTML;
  console.log(element.innerHTML);
}
```

Welkom Jos

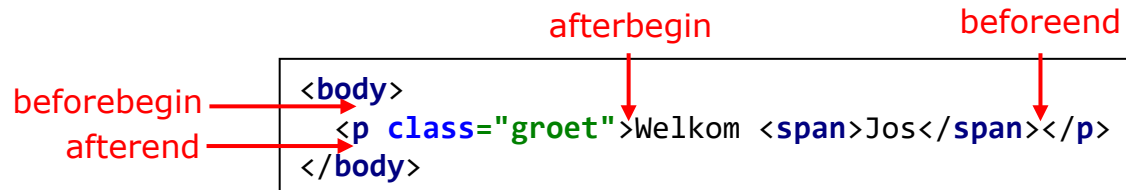
Inhoud:

Welkom Jos

Inhoud: Welkom Jos

# Inhoud: insertAdjacentElement() / -HTML()

insertAdjacent...() kan inhoud toevoegen t.o.v. het geselecteerde element



Alternatieve methoden:

<code>before()</code>	~ "beforebegin"
<code>after()</code>	~ "afterend"
<code>append()</code>	~ "beforeend"

```
addEventListener("load", inner);

function inner() {
  let element = document.querySelector(".groet"); // eerste class=="groet"

  /* insertAdjacentElement() */
  let newSpan = document.createElement("span"); // nieuwe span
  element.insertAdjacentElement("afterbegin", newSpan);

  /* insertAdjacentHTML() */
  element.insertAdjacentHTML("beforeend", "<span>...</span>");
}
```

# Attributen: `setAttribute()` / `getAttribute()`

```
<body>
  <p></p>
  <h3>attribuut src: <span></span></h3>
</body>
```

```
addEventListener("load", vervangAutoDoorFiets);

function vervangAutoDoorFiets() {
  let image = document.getElementsByTagName("img")[0];
  image.setAttribute("src", "images/fiets.jpg");
  image.setAttribute("alt", "fiets");

  let space = document.querySelector("span");
  space.innerHTML = image.getAttribute("src");
}
```



attribuut src: images/fiets.jpg

# Attributen: removeAttribute()

```
<body>
  <p></p>
  <h2>attribuut height: <span></span></h2>
</body>
```

```
addEventListener("load", verwijderAfmetingen);

function verwijderAfmetingen() {
  let image = document.querySelector("img");
  image.removeAttribute("height");
  image.removeAttribute("width");

  let attribuut = document.getElementsByTagName("span")[0];
  attribuut.innerHTML(image.getAttribute("height"));
}
```



**attribuut height:**

De attributen height en width zijn niet meer aanwezig!

# Attributen: als eigenschap van Element

```
<body>
  <p>
    
  </p>
</body>
```

```
addEventListener("load", vervangAutoDoorFiets);

function vervangAutoDoorFiets() {
  let image = document.querySelector("img");
  image.src = "images/fiets.jpg";
  image.alt = "fiets";
}
```

```
<body>
  <p>
    
  </p>
</body>
```



# Opmaak (CSS): style eigenschap

---

De eigenschap `style` is een object dat de 'inline' opmaak weerspiegelt van het element

Je kan via de eigenschappen van dit object dus css-eigenschappen voor het element aanspreken/wijzigen

Naamgeving: camelCase ipv kebab-case

```
<body>
  <p class="groet">Welkom <span>Jos</span></p>
</body>
```

```
addEventListener("load", inner);

function inner() {
  let element = document.querySelector(".groet"); // eerste class=="groet"

  /* className */
  element.style.fontSize = "1.2em";           // css: font-size
  element.style.backgroundColor = "yellow";    // css: background-color
}
```

# Opmaak (CSS): className / classList

---

**className** komt overeen met het attribuut 'class' en is dus een string met door spaties gescheiden opsomming van css-klassen

**classList** is een soort Array van de css-klassen! (-> voorkeur!)

```
<body>
  <p class="groet">Welkom <span>Jos</span></p>
</body>
```

```
addEventListener("load", inner);

function inner() {
  let element = document.querySelector(".groet"); // eerste class=="groet"

  /* className */
  element.className += " bold";

  /* classList */
  element.classList.add("bold");
}
```