# Samenvatting V

## 1. In HTML

```
<!-- for CSS -->
<link rel="stylesheet" href="style.css">

 <!-- for JS -->
<script src="code.js" type="text/javascript"></script>
```

## 2. JS Setup

```
const setup = () => //this arrow used to be the keyword  'function' in the
past!
{
    // ...
}
window.addEventListener("load", setup);
```

## 3. IMPORTANT!

### *Elementen uit de DOM-tree opvragen (enkel voor eindtests)*

Om elementen uit de DOM-tree te bemachtigen geldt specifiek voor de eindtests :

- gebruik WEL `.querySelector` of `.querySelectorAll`
- gebruik NIET `.getElementById`, `.getElementsByClassName`, `.getElementsByTagName`, ...

### *Elementen aan de DOM-tree toevoegen (enkel voor de eindtests)*

Om elementen aan de DOM-tree toe te voegen geldt specifiek voor de eindtests :

- gebruik WEL `.insertAdjacentHTML` of `.createElement` + `.appendChild`
- gebruik NIET `.innerHTML`
    - Uitzondering : je mag wel `parent.innerHTML = ""` gebruiken (om makkelijk alle kinderen van een parent element te verwijderen)

## Class toevoegen of verwijderen

```
txtOutput.classList.add("invalid"); // toevoegen
txtOutput.classList.remove("invalid"); // verwijderen
```

## Elementen uit DOM-tree opvragen

- zie `Deel_04` > `Opdracht_Colorpicker` / `Opdracht_Paragrafen`

```
document.getElementById(id)
document.getElementsByClassName(className)
document.getElementsByTagName(tagName)

document.querySelector(selector)
// retourneert een verwijzing naar het eerste element dat door de CSS-selector geselecteerd wordt

document.querySelectorAll(selector)
// geeft een verzameling terug met verwijzingen naar alle elementen die door de selector geselecteerd worden. Deze verzameling is een NodeList, je kunt er op dezelfde manier mee omgaan als met het resultaat van bv. getElementsByClassName.

document.getElementsByName(name)
```

## Elementen te zoeken, die kinderen zijn van een element dat we reeds eerder te pakken kregen

```
let frmContact = document.getElementById("frmContact");
// ...
 let errorMessages = frmContact.getElementsByClassName("errorMessage");

 // Je ziet dat de zoektocht begint bij het frmContact object en niet bij document zoals gewoonlijk.

 // ----- SHORTER ALTERNATIVE ------

 let errorMessages=document.querySelectorAll("#frmContact .errorMessage");
```

## Content toevoegen op bepaalde plaatsen

- zie `Deel_06` > `Opdracht_01`

```
list.innerHTML = "<li>item 1</li><li>item 2</li>";
list.innerHTML += "<li>new item</li>"; // replaces entire list

// better -> insertAdjacentHTML
list.insertAdjacentHTML("beforeend", "<li>new item</li>");
```

## Vergelijkingsoperatoren

```
x = = y // we are used to this
x = = = y // we rather use this in JS
```

## Sorteren / Vergelijken

- zie `Deel_07` > `Opdracht_Gemeente`

```
// strings sorteren
let string1 = "peer";
let string2 = "appel";
let resultaat = string1.localeCompare(string2);

// array van strings sorteren
const compare = (a, b) =>
{
    return a.localeCompare(b);
}
let array = ["zebra", "aap", "giraf", "ezel"];
array.sort(compare);

// array van getallen sorteren
const compare = (a,b) =>
{
    return a - b;
}
let array = [34, 67, 12, 5, 23];
array.sort(compare);
```

- **Merk op!** : de **vergelijkingsfunctie** is weliswaar **optioneel**, maar het standaardgedrag van `sort()` zonder vergelijkingsfunctie levert vaak een **zinloze volgorde** op. In de praktijk zul je dan ook bijna altijd een vergelijkingsfunctie voorzien.

## Arrays

Hierboven zagen we al hoe we zaken in een array sorteren.

```
let p1 = {}; // een nieuw leeg object
p1.naam = "Jan Janssens";
p1.gemeente = "Melle";

let p2 = {}; // een nieuw leeg object
```

```
p2.naam = "Mieke Mickelson";
p2.gemeente = "Bruhhe";

let personen = []; // een leeg array
personen.push( p1 ); // voeg p1 achteraan toe aan array
personen.push( p2 ); // voeg p2 achteraan toe aan array
```

## Loops

Om een stuk code te herhalen gebruik je

- `for-loop` met teller
- `while loop` of `do-while loop`

Om een array of lijst te overlopen kun je een `for-of loop` gebruiken:

```
let items = document.querySelectorAll(".item");
for (let item of items)
{
    console.log( item.textContent );
}
```

Je gebruikt geen `for-in loop` (die dient namelijk om de properties van een object te overlopen).

## Waarden in een form opvragen

- zie `javascript_deel_07.pdf` voor meer uitleg
- zie `Deel_07` > `Opdracht_Formwaarden`

We kunnen de `.value` **property** opvragen van het corresponderende DOM-tree element. Let er wel op dat dit steeds een **string** zal zijn, ook al heeft het element bv. een `type="number"` attribuut (je zult dit dus nog moeten **parsen**)!

For an **input field**:

```
// HTML
<input type="number" id="txtAmount">

// JavaScript
let txtAmount = document.querySelector("#txtAmount");
let amountAsText = txtAmount.value;
let amount = Number.parseInt(amountAsText, 10);
```

For a checkbox/radio button:

```
// HTML
<input type="checkbox" id="chkPriority">

// JavaScript
let chkPriority = document.querySelector("#chkPriority");
console.log(chkPriority.checked);
```

Het DOM-element van een heeft volgende nuttige properties :

- `.selected` een boolean die aangeeft of de option geselecteerd is
- `.value` de waarde van het value attribuut van de option
- `.text` de tekst van de option (tussen begin- en eindtag)

## Node vs. Element

| | Node | Element |
|---|---|---|
| What is it? | Any single "point" in the tree | A specific "tag" (like `<body>`) |

- DOM-tree Node relationship properties:
  <- geven ons *alle* soorten Nodes | stellen enkel HTML-elementen voor ->

```
n.parentNode
// geeft parent van n, of null indien er geen is

n.childNodes | n.children
// geeft lijst die alle kinderen van n bevat


n.firstChild / n.lastChild | n.firstElementChild / n.lastElementChild
//geeft het eerste/laatste kind van n, of null indien er geen is

n.nextSibling / n.previousSibling | n.nextElementSibling /
n.previousElementSibling
// geeft volgende/vorige sibling van n, of null indien er geen is
```

- DOM-tree Node type properties:

```
n.nodeName
// geeft het soort element Node (ex. img, h1,...) als string
```

```
n.nodeType
// geeft een getal naargelang welk soort node n is (ex. 1 = element nod, 3 =
text node,...)

n.nodeValue
// enkel voor text nodes -> geeft tekst van node als string, of null als het
NIET om een text node gaat
```

We speak of **text node**, but what is a text node?
=> extra nodes in de DOM-tree die de teksten bevatten die tussen de **element tags** staan

## Rechtstreeks met Node objecten werken

Dit is krachtiger dan `.innerHTML` of `insertAdjacentHTML()`, maar de code wordt al snel onoverzichtelijk.
Indien mogelijk geef je beter de voorkeur aan `.insertAdjacentHTML()` met `firstElementChild` / `lastElementChild` of met slimme CSS-selectoren.

**Node aanmaken**:

```
document.createElement(s)
//returns a new element node
// 's' = ex. "img", "h1",...
    // example:
    let elementNode = document.createElement("a");

document.createTextNode(t)
// returns a new text node
// 't' = string with text in node
    // example:
    let textNode = document.createTextNode("Hello World!");
```

**Node toevoegen**:

```
p.appendChild(c)
// node 'c' kan aan een parent 'p' worden toegevoegd als laatste kind
```

**Node verwijderen**:

```
p.removeChild(c)
// node 'c' kan uit een parent 'p' worden verwijderd
```

***Example***:

```
//HTML
<div class="gallery">
    <img src="images/a.png">
    <img src="images/b.png">
    <img src="images/c.png">
    <img src="images/d.png">
</div>

// JavaScript
let gallery = document.querySelector(".gallery");

let image = document.createElement("img");
image.setAttribute("src", "images/e.png");

// image toevoegen na laatste kind
gallery.appendChild(image);

//  met src="images/d.png"
let someImage = gallery.children[3];

// d.png wordt verwijderd uit de gallery
gallery.removeChild(someImage);
```

## Objecten in Javascript

- zie `javascript_deel_09` voor meer uitleg

In onze cursus beperken we ons tot hele **simpele objecten**, die **enkel properties** bevatten en **geen methodes**. (Enkel data, geen gedrag). => "dataholders"

***Zelf simpele objecten maken***:

```
let student = {};
student.voornaam = "Aerith";
student.familienaam = "Gainsborough";

let locatie = {};
locatie.straat = "Flower Street 7";
locatie.postcode = "2222";
locatie.gemeente = "Midgar";

locatie.postcode = "1985"; // wijzig waarde van 'postcode' property

student.adres = locatie; // voeg 'adres' property toe
```

```
console.log(student.voornaam + " woont in " + student.adres.postcode);
//output
```

## JSON

Bij een webapplicatie is het soms nodig om wat data uit een Javascript programma als tekst voor te stellen.

Je kan makkelijk een object omzetten naar een JSON-tekst met `JSON.stringify()` :

```
let persoon = {
voornaam : "Jan",
familienaam : "Janssens",
aantalKinderen : 0 };

let tekst = JSON.stringify( persoon );

console.log( tekst );
// output:
// {"voornaam":"Jan","familienaam":"Janssens","aantalKinderen":0}
```

Met `JSON.parse()` kunnen we een Javascript object reconstrueren op basis van een JSON-tekst :

```
let tekst = '{"voornaam":"Mieke", "familienaam":"Mickelson", "leeftijd":32}';
// let op de " en ' afwisseling!

let p = JSON.parse( tekst );

console.log( p.leeftijd );
// output: 32
```

JSON is een soort **data opslagformaat**. Een alternatief voor JSON is XML.
XML is minder vlot te lezen en gebruikt veel meer tekst om informatie voor te stellen, maar het is wel beter geschikt voor zeer complexe data. (Je gaat bv. bij sommige games XML files zien staan in de game folders)