

Hoofdstuk 1: Javascript + Vite + Bootstrap projecttemplate.....	9
Hoofdstuk 2: Functions.....	12
Doel van dit hoofdstuk.....	12
Concept & uitleg.....	12
Voorbeeld in jouw template.....	13
Mini-opdracht.....	14
Begeleide oplossing.....	15
Hoofdstuk 3: Function Scoping.....	16
Concept & uitleg.....	16
Denkbeeld.....	17
Oefening: Function Scope Demo.....	17
Mini-opdracht.....	18
Begeleide oplossing.....	19
Wat je nu beheerst.....	19
Hoofdstuk 4: Named Functions.....	20
Theorie: Named Functions.....	20
Demo: Kleur tonen met named functions.....	20
Wat zagen we?.....	22
Mini-opdracht.....	22
Begeleide oplossing mini-opdracht.....	23
Samenvatting.....	24
Hoofdstuk 5: Functions & Strings in ES6 (Template Literals).....	24
Theorie: Template Literals .....	24
Real-life analogie.....	25
Demo: Welkomstbericht generator .....	25
Mini-opdracht.....	27
Mini-opdracht begeleide oplossing.....	27
Samenvatting.....	28
Hoofdstuk 6: ES6 Arrow Functions (=>).....	29
Theorie: Arrow Functions.....	29
Vergelijking.....	29
Let op.....	30
Oefening: Bereken + Toon (arrow style).....	30
Wat zag je?.....	31
Mini-opdracht.....	31
Oplossing mini-opdracht.....	32
Samenvatting.....	32

Hoofdstuk 7: ES6 Arrays.....	33
Theorie: Arrays.....	33
Vergelijking in real life.....	33
Demo: gastenlijst (array + UI).....	34
Mini-opdracht.....	35
Mini-opdracht oplossing.....	36
Samenvatting.....	37
Hoofdstuk 8: DOM Nodes & Root Nodes.....	37
Theorie .....	37
Denkbeeld.....	38
Demo: tekst zetten via root nodes.....	38
Wat gebeurde hier?.....	40
Mini-opdracht.....	40
Mini-opdracht oplossing.....	40
Samenvatting.....	41
Volgend hoofdstuk.....	41
Hoofdstuk 9: JavaScript Output Methodes.....	42
Theorie: Output Methodes.....	42
Demo oefening: 3 outputmethoden.....	43
Mini-opdracht.....	44
Mini-opdracht oplossing.....	45
Samenvatting.....	45
Hoofdstuk 10: document.write() en document.writeln().....	46
Theorie .....	46
Demo: Waarom je het niet gebruikt.....	47
Mini-opdracht.....	48
Mini-opdracht oplossing.....	48
Samenvatting.....	49
Hoofdstuk 11: innerHTML vs textContent vs innerText.....	49
Theorie .....	49
Veiligheid tip.....	50
Demo: drie output methoden.....	50
Mini-opdracht.....	52
Oplossing mini-opdracht.....	52
Samenvatting.....	53
Hoofdstuk 12: DOM Selectors.....	53
Theorie: DOM Selectors.....	53

Denkbeeld .....	54
Demo: lijst filteren met querySelectorAll .....	54
Mini-opdracht .....	55
Oplossing mini-opdracht .....	55
Samenvatting .....	56
Hoofdstuk 13: Andere DOM-element properties & acties .....	56
Theorie .....	56
Metafoor .....	57
Demo: focus highlight + child counter .....	57
Mini-opdracht .....	59
Mini-opdracht oplossing .....	59
Samenvatting .....	60
Hoofdstuk 14: Klassen manipuleren (classList) .....	61
Theorie .....	61
Demo: Highlight toggle + knop states .....	61
Mini-opdracht .....	63
Mini-opdracht oplossing .....	63
Samenvatting .....	64
Hoofdstuk 15: Events & Formulieren .....	64
Theorie: Wat is een Event? .....	64
Metafoor .....	65
Demo: Live naam weergave + form submit .....	65
Mini-opdracht .....	67
Mini-opdracht oplossing .....	67
Samenvatting .....	68
Hoofdstuk 16: Event Types & preventDefault() (Dieper) + Mouse Events .....	69
Theorie .....	69
Demo: Link + Box hover effect + Submit block .....	70
Mini-opdracht .....	71
Oplossing mini-opdracht .....	72
Samenvatting .....	72
Hoofdstuk 17: Mini-Project – Todo App (Events + Arrays + DOM) .....	73
Uitleg .....	73
UI Design .....	73
index.html .....	74
main.js .....	74
Demo features die studenten leren .....	75

Mini-Opdracht.....	76
Samenvatting.....	76
Hoofdstuk 18: Objects & Date() in JavaScript.....	77
Theorie: Wat is een object?.....	77
Theorie: Date() object.....	77
Use case: Leeftijd berekenen.....	78
Demo: Leeftijd-calculator.....	78
Mini-opdracht.....	79
Mini-opdracht oplossing.....	80
Samenvatting.....	81
Hoofdstuk 19: Object Constructors & Inheritance (Intro).....	81
Theorie.....	82
Demo: User maken en tonen.....	82
Mini-opdracht.....	84
Oplossing mini-opdracht.....	84
Inheritance Light.....	85
Samenvatting.....	85
Hoofdstuk 20: ES6 Classes + extends + super.....	86
Theorie: Wat is een Class?.....	86
Inheritance met extends.....	87
Demo: User + Admin UI.....	87
Mini-opdracht.....	89
Samenvatting.....	89
Hoofdstuk 21: Mini Project – Product Catalog (Webshop Logica).....	90
Concept.....	90
Class Model.....	90
HTML UI.....	90
Styles (optioneel animatie).....	91
JavaScript.....	91
Test checklist.....	92
Mini-opdracht (uitbreiding).....	93
Samenvatting.....	93
Hoofdstuk 22: Browser Object Model (BOM).....	94
Theorie.....	94
Veiligheid: popups & redirects.....	95
Demo: BOM Info Display.....	95
Mini-opdracht.....	96

Mini-opdracht oplossing.....	96
Samenvatting.....	97
Hoofdstuk 23: Cookies & LocalStorage.....	98
Demo: Naam onthouden.....	99
Demo 2: Array opslaan (favorietenlijst) .....	100
Mini-opdracht.....	101
Samenvatting.....	102
Hoofdstuk 24: AJAX & API's Intro.....	103
Theorie.....	103
Voorbeeld van async denken.....	104
Demo: Data ophalen met fetch().....	104
Wat leerde je eigenlijk?.....	105
Mini-opdracht.....	105
Samenvatting.....	106
Hoofdstuk 25: JSON File Load + Local API Simulator.....	106
Wat is JSON?.....	106
JSON in Vite project plaatsen.....	107
UI: Lokaal JSON bestand laden.....	107
UI resultaat.....	108
Mini-opdracht.....	109
Mini-opdracht oplossing (docentversie).....	109
Samenvatting.....	110
Hoofdstuk 26: POST met Fetch (Fake API) & CRUD Intro.....	111
Theorie.....	111
Denk in stappen.....	111
Mini Fake API: Create Product.....	111
index.html.....	112
main.js.....	112
Wat werkt hier?.....	113
Mini-Opdracht.....	113
Mini-Opdracht Oplossing (docentversie).....	114
Samenvatting.....	114
Hoofdstuk 27: PUT & DELETE (Fake CRUD).....	115
Theorie.....	115
Mini Fake Admin Panel.....	115
index.html (blok toevoegen).....	115
main.js.....	116

Mini-opdracht.....	118
Samenvatting.....	118
Hoofdstuk 28: Echte HTTP Requests (fetch POST/PUT/DELETE) + API Testing Tools.....	119
Theorie.....	119
API Tools uitleg.....	119
Belangrijk.....	120
Demo 1: GET request (opfrisser) .....	120
Demo 2: POST request (echt).....	120
Demo 3: DELETE request (echt).....	121
Wat studenten nu snappen.....	122
Mini-opdracht.....	122
Samenvatting.....	122
Hoofdstuk 29: Async/Await Deep Dive + UX voor Loading & Errors.....	123
Theory: Async/Await.....	123
UX Rules (professioneel).....	123
Demo: Fake API + Loading UI.....	124
Bonus: Retry button.....	125
Advanced Concepts (kort).....	125
Mini-opdracht.....	126
Mini-opdracht Oplossing (docentversie, samenvatting).....	126
Samenvatting.....	126
Hoofdstuk 30: Fetch Timeout, AbortController & Race Conditions.....	127
Theorie.....	127
UX Patterns.....	127
Demo: Cancel & Timeout.....	128
Wat werkt hier?.....	129
Race Condition Uitleg.....	129
Mini-opdracht.....	130
Docent-oplossing (verkort).....	130
Samenvatting.....	130
Hoofdstuk 31: Multiple Requests + Promise.all + Loading Skeletons.....	131
Theorie.....	131
Skeleton UI.....	131
Skeleton CSS.....	131
Demo: Parallel API Load (users + posts).....	132
UX Resultaat.....	133
Mini-opdracht.....	134

Samenvatting.....	134
Hoofdstuk 32: Modules, Imports & Professionele Projectstructuur.....	135
Waarom modules?.....	135
Nieuwe Project structuur.....	136
We bouwen: User Card Component + API Service.....	136
js/components/userCard.js.....	136
js/utils/format.js.....	136
js/main.js.....	137
index.html voor test.....	137
Wat je nu hebt gebouwd.....	137
Extra inzichten.....	137
Mini-opdracht.....	138
Docent-oplossing hints.....	138
Samenvatting.....	139
Hoofdstuk 33: Eindproject — Mini Shopping Dashboard.....	139
Project Concept.....	139
Structuur.....	140
data/products.json.....	140
Utilities.....	140
Components.....	141
Services.....	141
HTML sectie.....	142
js/main.js.....	142
Wat dit project demonstreert.....	143
Mini-opdracht (student) .....	144
Wat studenten nu kunnen.....	144
Hoofdstuk 34: Developer Tools & Debugging Mastery.....	145
Theorie (kort & scherp).....	145
Demo 1 — Console power: table, group, fouten lezen.....	145
Demo 2 — Breakpoints, Watch & Call Stack.....	146
Demo 3 — Network: Throttling & Offline, Errors lezen.....	147
Mini-opdracht.....	149
Begeleide oplossing (met inline comments).....	149
Samenvatting.....	150
Hoofdstuk 35: ES6+ Advanced Patterns & Array Mastery.....	151
Theorie.....	151
Demo 1 — Destructuring + Spread + Rest.....	152

Demo 2 — map, filter, reduce.....	153
Immutability (zoals React).....	154
Mini-opdracht.....	154
Samenvatting.....	155
Hoofdstuk 36: Modern Forms + Validatie + UX Patterns.....	155
Theorie.....	155
Demo Form: Login + Live Feedback.....	156
Mini-Opdracht.....	157
Oplossing (docentversie).....	158
Samenvatting.....	158

# Hoofdstuk 1: Javascript + Vite + Bootstrap projecttemplate

## Doele van dit hoofdstuk

In dit hoofdstuk maken we een basisproject dat we **bij elke oefening** opnieuw gebruiken. Je leert:

- een Vite projectstructuur begrijpen
- Bootstrap via npm + SCSS integreren
- een testcomponent bouwen om te controleren of alles werkt
- een vaste oefen-template inzetten voor alle volgende hoofdstukken

Na dit hoofdstuk heb je een solide startpunt voor alle volgende JavaScript-oefeningen.

## Concept en uitleg

### *Waarom Vite?*

Vite is een moderne bundler met:

- bliksemsnelle development server
- native ES modules (super logisch voor modern JS)
- automatische refresh bij codewijzigingen
- simpele build-flow voor productie

### *Waarom Bootstrap via npm?*

We gebruiken npm zodat we:

- Bootstrap als onderdeel van onze build hebben
- SCSS kunnen overschrijven en uitbreiden
- géén CDN nodig hebben
- professioneel en future-proof werken

### *Hoe gaan we oefenen?*

Elke oefening start van dezelfde template:

- `index.html` voor markup
- `main.js` voor logica
- `styles.scss` voor styling
- Bootstrap layout voor oefen-UI

We verververs dus enkel de inhoud van de `main.js` en eventueel HTML-blokken. Dit geeft structuur en voorkomt chaos.

## Template voorbeeld (Smoke Test)

Dit is het stukje dat **bewijst dat je setup werkt**. Als de knop werkt, ben je klaar voor de rest van de cursus. Je krijgt eerst de template van de docent.

Bestand: `src/index.html`

```
<div class="container py-4">
  <h1 class="h4 mb-3">Vite + Bootstrap werkt </h1>

  <div class="card shadow-sm">
    <div class="card-header bg-primary text-white">
      Test
    </div>
    <div class="card-body">
      <p class="mb-3">Klik op de knop om te testen of alles goed geladen
      is.</p>

      <button id="btnTest" class="btn btn-primary">
        Test template
      </button>

      <div id="out" class="alert alert-secondary mt-3 mb-0">
        Output verschijnt hier
      </div>
    </div>
  </div>
</div>
```

Bestand: `src/js/main.js`

```
import '../scss/styles.scss'
import * as bootstrap from 'bootstrap'

document.addEventListener('DOMContentLoaded', () => {
  const btn = document.getElementById('btnTest')
  const out = document.getElementById('out')

  btn.addEventListener('click', () => {
    out.className = 'alert alert-success mt-3 mb-0'
    out.textContent = 'Alles werkt. Klaar voor de oefeningen!'
  })
})
```

Bestand: `src/scss/styles.scss`

```
@import "bootstrap/scss/bootstrap";

body {
  background-color: #f8f9fa;
}
```

Als je nu `npm run start` doet en klikt op de knop, zie je een groene alert:

Alles werkt. Klaar voor de oefeningen!

Dan is je omgeving 100% klaar.

## Mini-opdracht

**Voeg onder de testcard een tweede card toe met:**

- een tekstinput
- een knop "Toon naam"
- een alert die "Welkom, [naam]" toont

Gebruik Bootstrap classes.

## Begeleide oplossing

Toevoeging in index.html (onder de testcard)

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-dark text-white">Naam test</div>
  <div class="card-body">
    <input id="inpName" class="form-control mb-2" placeholder="Geef je naam in" />
    <button id="btnName" class="btn btn-dark w-100">Toon naam</button>

    <div id="nameOut" class="alert alert-secondary mt-3 mb-0">Nog niets ingegeven</div>
  </div>
</div>
```

Toevoeging in main.js

```
const btnName = document.getElementById('btnName')
const inpName = document.getElementById('inpName')
const nameOut = document.getElementById('nameOut')

btnName.addEventListener('click', () => {
  const naam = inpName.value.trim()

  if (!naam) {
    nameOut.className = 'alert alert-warning mt-3 mb-0'
    nameOut.textContent = '⚠ Vul je naam in aub'
    return
  }

  nameOut.className = 'alert alert-success mt-3 mb-0'
  nameOut.textContent = `Welkom, ${naam}!`
})
```

## Resultaat

Je hebt nu:

- een werkende Vite + Bootstrap basis
- een vaste oefentemplate
- UI-interactie met DOM
- duidelijke output feedback

We gaan vanaf nu **elke oefening in deze template schrijven.**

## Hoofdstuk 2: Functions

### Doel van dit hoofdstuk

Na dit hoofdstuk kan je:

- uitleggen wat een functie is
- een functie definiëren en aanroepen
- werken met parameters & return
- pure functies (berekening) scheiden van impure functies (UI)
- input lezen en output tonen met JS
- Bootstrap UI koppelen aan functies

### Concept & uitleg

#### Wat is een functie?

Een functie is **herbruikbare code** die iets doet en (meestal) een resultaat teruggeeft.

#### Waarom?

- Nooit dubbel werk (**DRY** principe)
- Logica netjes gescheiden
- Helder, leesbaar, uitbreidbaar
- In UI: berekening apart, tonen apart

#### Structuur

```
function naam(parameter) {  
    return resultaat;  
}  
  
naam(waarde)
```

#### Belangrijk

Term	Betekenis
Parameter	naam in functie definitie
Argument	waarde die je doorgeeft
Return	wat de functie teruggeeft

## Pure vs Impure

Type	Doel
Pure	berekenen, niets aanpassen buiten functie
⚠️ Impure	DOM, alert, console.log, data wijzigen

We gebruiken **pure functies** voor **logica**, en **impure functies** voor **UI**.

### Voorbeeld in jouw template

We voegen **1 oefenblok** toe met unieke ID's.

src/index.html → onder de vorige testcard voeg je toe:

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-primary text-white">Hoofdstuk 2: Functions</div>
  <div class="card-body">

    <label class="form-label">Wat is je naam?</label>
    <input id="fn_nameInput" class="form-control mb-2" placeholder="vb. Tom">

    <button id="fn_btnGreet" class="btn btn-primary w-100">Groet</button>

    <div id="fn_output" class="alert alert-secondary mt-3 mb-0">
      Resultaat komt hier...
    </div>
  </div>
</div>
```

Opgelet: **fn\_ prefix** = “functions chapter”. Zo vermijden we ID-conflicten door heel de cursus.

src/js/main.js → voeg onderaan toe:

```
// -----
// Hoofdstuk 2: Functions
// -----


// Pure function (berekening)
function maakGroet(naam) {
  return `Hallo ${naam}!`;
}

// Impure function (UI)
function handleGroetClick() {
  const naamInput = document.getElementById('fn_nameInput');
  const output = document.getElementById('fn_output');

  const naam = naamInput.value.trim();

  if (!naam) {
    output.className = "alert alert-warning mt-3 mb-0";
    output.textContent = "⚠️ Geef een naam in!";
  }
}
```

```

    return;
}

const boodschap = maakGroet(naam);

output.className = "alert alert-success mt-3 mb-0";
output.textContent = boodschap;
}

// Event koppelen
document.addEventListener("DOMContentLoaded", () => {
  document.getElementById('fn_btnGreet')
    ?.addEventListener("click", handleGroetClick);
});

```

#### Expected behaviour

- Lege input → gele waarschuwing
- Geldige naam → groene groet

Dit lijkt simpel, maar vormt het fundament voor **ELK volgend hoofdstuk**.

#### Mini-opdracht

Maak functie berekenVierkant(getal):

Input	Output
5	" $5^2 = 25$ "
10	" $10^2 = 100$ "

UI:

- input veld (nummer)
- knop “Bereken vierkant”
- Bootstrap alert onderaan

Zelfde ID-stijl:

```

sq_input
sq_btn
sq_output

```

## Begeleide oplossing

HTML (onder vorige card)

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-dark text-white">Mini-opdracht: Vierkant berekenen</div>
  <div class="card-body">

    <input id="sq_input" type="number" class="form-control mb-2" placeholder="vb. 5">
    <button id="sq_btn" class="btn btn-dark w-100">Bereken vierkant</button>

    <div id="sq_output" class="alert alert-secondary mt-3 mb-0">
      Resultaat komt hier...
    </div>
  </div>
</div>
```

JS

```
// Pure berekening
function berekenVierkant(getal) {
  return getal * getal;
}

// UI handler
function handleVierkant() {
  const inp = document.getElementById('sq_input');
  const out = document.getElementById('sq_output');

  const waarde = Number(inp.value);

  if (!waarde) {
    out.className = "alert alert-warning mt-3 mb-0";
    out.textContent = "⚠ Geef een getal in!";
    return;
  }

  const resultaat = berekenVierkant(waarde);

  out.className = "alert alert-success mt-3 mb-0";
  out.textContent = `${waarde}² = ${resultaat}`;
}

// Event koppelen
document.addEventListener("DOMContentLoaded", () => {
  document.getElementById('sq_btn')
    .addEventListener("click", handleVierkant);
});
```

# Hoofdstuk 3: Function Scoping

## Doele van dit hoofdstuk

Na dit hoofdstuk kan je:

- uitleggen wat **scope** betekent
- het verschil begrijpen tussen **function scope** en **block scope**
- lokale vs globale variabelen herkennen
- nested functions gebruiken
- scoping-fouten herkennen zoals `ReferenceError: x is not defined`

## Concept & uitleg

### Wat is scope?

Scope bepaalt **waar** een variabele toegankelijk is.

Scope type	Betekenis
Global scope	overal zichtbaar
Function scope	enkel binnen een functie zichtbaar
Block scope ( <code>let</code> , <code>const</code> )	enkel binnen <code>{ }</code> zichtbaar
Lexical scope	inner functions kunnen outer data gebruiken

### ES6 sleutelregel

Gebruik `let` en `const` X gebruik geen `var` meer ondanks dat dit altijd zal blijven werken (oude function-scoped variabelen)

### Klassieke bug

```
{  
  let x = 5;  
}  
console.log(x); // X ReferenceError
```

### Nested functions

```
function outer() {  
  const naam = "Tom";  
  
  function inner() {  
    console.log(naam); // werkt  
  }  
  
  inner();  
}
```

Binnenste ziet buitenste. Buitenste ziet binnenste **niet**.

## Denkbeeld

Stel je voor:

- Je code is een huis.
- Een functie is een kamer.
- Wat in de kamer ligt, kan je daar gebruiken.
- Maar je kan niet door de muur kijken naar binnen.

Inner functions = een klein kamertje in de kamer.

## Oefening: Function Scope Demo

src/index.html → voeg dit oefenblok toe:

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-info text-white">Hoofdstuk 3: Function
Scoping</div>
  <div class="card-body">

    <button id="sc_btn" class="btn btn-info w-100 mb-2">Voer scope demo
uit</button>

    <div id="sc_output" class="alert alert-secondary mb-0">
      Klik op de knop om scope te testen...
    </div>
  </div>
</div>
```

src/js/main.js → voeg onderaan toe:

```
// -----
// Hoofdstuk 3: Function Scoping
// -----


function toonBericht() {
  const binnen = "Ik leef binnen de functie ☺";
  function inner() {
    return `Inner ziet: ${binnen}`;
  }
  return inner();
}

function handleScope() {
  const output = document.getElementById("sc_output");
  // inner sees outer
  const bericht = toonBericht();

  output.className = "alert alert-success mb-0";
  output.textContent = bericht;
```

```

try {
  // X buiten de functie proberen de variabele te gebruiken
  console.log(binnen);
} catch (err) {
  console.warn("binnen is niet zichtbaar buiten de functie");
}

document.addEventListener("DOMContentLoaded", () => {
  document.getElementById("sc_btn")
    ?.addEventListener("click", handleScope);
});

```

### Expected behaviour

Klik op de knop → groene alert:

Inner ziet: Ik leef binnen de functie ☺

Console:

binnen is niet zichtbaar buiten de functie

### Mini-opdracht

Maak:

- functie geheimBericht()
- lokale variabele secret = "Code unlocked ☺"
- nested function leesSecret()
- return dat inner result
- UI met knop + output

Gebruik ID's:

sc2\_btn  
sc2\_output

Hint: zelfde patroon als boven, andere tekst.

## Begeleide oplossing

HTML

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-warning">Mini-opdracht: Geheim bericht</div>
  <div class="card-body">
    <button id="sc2_btn" class="btn btn-warning w-100 mb-2">Toon
    geheim</button>
    <div id="sc2_output" class="alert alert-secondary mb-0">
      Wacht op geheim...
    </div>
  </div>
</div>
```

JS

```
function geheimBericht() {
  const secret = "Code unlocked 🔑";

  function leesSecret() {
    return secret;
  }

  return leesSecret();
}

function handleSecret() {
  const out = document.getElementById("sc2_output");
  out.className = "alert alert-success mb-0";
  out.textContent = geheimBericht();
}

document.addEventListener("DOMContentLoaded", () => {
  document.getElementById("sc2_btn")
    ?.addEventListener("click", handleSecret);
});
```

### Wat je nu beheerst

- Function scope
- Block scope
- Lexical scope
- Inner vs outer toegang
- Variable accessibility errors herkennen

Dit is superbelangrijk voor:

- Callbacks
- Modules
- DOM events
- Promises / async
- API calls later

## Hoofdstuk 4: Named Functions

### Doele van dit hoofdstuk

Na deze module kan je:

- uitleggen wat een **named function** is
- function declarations schrijven
- named functions koppelen aan knoppen
- het verschil begrijpen tussen **pure** en **impure** functies
- foutmeldingen beter lezen dankzij functienamen
- input → verwerking → output met duidelijke code

### Theorie: Named Functions

#### Wat is een named function?

Een functie met een **duidelijke naam**:

```
function zegHallo() {  
    return "Hallo!";  
}
```

#### Waarom belangrijk?

Reden	Waarom
Lesbaarheid	Je ziet in één oogopslag wat code doet
Debugging	Foutmeldingen tonen functienaam
Structuur	Bouwblokken die makkelijk herbruikbaar zijn
Teamwerk	Iedereen begrijpt het sneller

#### Named vs Anonymous

#### Slecht (moeilijk debuggen)

```
button.addEventListener("click", function() {  
    ...  
});
```

#### Goed (duidelijk & traceerbaar)

```
button.addEventListener("click", toonKleur);
```

#### Demo: Kleur tonen met named functions

We bouwen:

- inputveld
- knop
- tekstvak
- gekleurde box (visuele feedback)

```

src/index.html (+ blok onder vorige card)
<div class="card shadow-sm mt-4">
  <div class="card-header bg-secondary text-white">
    Hoofdstuk 4: Named Functions
  </div>
  <div class="card-body">

    <label class="form-label">Typ een primaire kleur (rood of blauw):</label>
    <input id="nf_input" class="form-control mb-2" placeholder="bv. rood">

    <button id="nf_btn" class="btn btn-secondary w-100 mb-2">Toon
kleur</button>

    <div id="nf_text" class="alert alert-secondary mb-2">
      Wacht op input...
    </div>

    <div id="nf_box"
      class="d-flex justify-content-center align-items-center fw-bold"
      style="height: 70px; border-radius: 6px; border: 1px solid #ccc;
background:#f8f9fa;">
      Geen kleur
    </div>

  </div>
</div>

```

```

src/js/main.js (+ onderaan toevoegen)
// -----
// Hoofdstuk 4: Named Functions
// -----


// Pure function
function bepaalKleurInfo(kleur) {
  const input = kleur.toLowerCase();

  if (input === "rood") return { text: "Je koos rood", color: "red" };
  if (input === "blauw") return { text: "Je koos blauw", color: "blue" };

  return { text: "Onbekende kleur", color: null };
}

// Impure function (DOM)
function toonKleur() {
  const inp = document.getElementById("nf_input");
  const out = document.getElementById("nf_text");
  const box = document.getElementById("nf_box");

  const waarde = inp.value.trim();

  if (!waarde) {
    out.className = "alert alert-warning mb-2";
    out.textContent = "⚠ Geef een kleur in";
  }
}

```

```

        box.style.background = "#f8f9fa";
        box.textContent = "Geen kleur";
        return;
    }

    const resultaat = bepaalKleurInfo(waarde);

    if (!resultaat.color) {
        out.className = "alert alert-danger mb-2";
        out.textContent = resultaat.text;
        box.style.background = "#f8f9fa";
        box.textContent = "Onbekend";
        return;
    }

    out.className = "alert alert-success mb-2";
    out.textContent = resultaat.text;
    box.style.background = resultaat.color;
    box.textContent = resultaat.color.toUpperCase();
}

// Event koppeling
document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("nf_btn")
        ?.addEventListener("click", toonKleur);
});

```

### Wat zagen we?

- Functie **met naam** is beter traceerbaar
- Pure functie berekent
- Impure functie werkt met UI
- Input → verwerking → output + visuele feedback

### Mini-opdracht

Maak een **fruit-kiezer** met emoji.

#### Vereisten

Input	Tekst	Emoji
appel	"Je koos appel"	🍎
banaan	"Je koos banaan"	🍌
anders	waarschuwing	?

#### UI-elementen

- Input veld
- Knop
- Tekstvak (alert)
- Blok met emoji

## Verplichte ID's

fr\_input  
fr\_btn  
fr\_text  
fr\_box

## Tip

Pure function retourneert:

```
{ text: "Je koos appel", emoji: "🍎" }
```

## Begeleide oplossing mini-opdracht

### HTML

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-warning">Mini-opdracht: Fruit kiezer</div>
  <div class="card-body">

    <input id="fr_input" class="form-control mb-2" placeholder="bv. appel">
    <button id="fr_btn" class="btn btn-warning w-100 mb-2">Kies
    fruit</button>

    <div id="fr_text" class="alert alert-secondary mb-2">Wacht op
    input...</div>

    <div id="fr_box"
      class="d-flex justify-content-center align-items-center fw-bold"
      style="height:70px; border:1px solid #ccc; border-radius:6px;
    background:#f8f9fa;">
      ?
    </div>
  </div>
```

### JS

```
function bepaalFruit(naam) {
  const f = naam.toLowerCase();

  if (f === "appel") return { text: "Je koos appel", emoji: "🍎" };
  if (f === "banaan") return { text: "Je koos banaan", emoji: "🍌" };

  return { text: "Onbekend fruit", emoji: "?" };
}

function toonFruit() {
  const inp = document.getElementById("fr_input");
  const txt = document.getElementById("fr_text");
  const box = document.getElementById("fr_box");

  const waarde = inp.value.trim();
```

```

if (!waarde) {
    txt.className = "alert alert-warning mb-2";
    txt.textContent = "⚠️ Vul een fruitsoort in!";
    box.textContent = "?";
    return;
}

const res = bepaalFruit(waarde);

txt.className = "alert alert-success mb-2";
txt.textContent = res.text;
box.textContent = res.emoji;
}

document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("fr_btn")
        ?.addEventListener("click", toonFruit);
});

```

### Samenvatting

Concept	Belangrijk
Named functions	leesbaar, debugbaar
Pure functions	berekenen, géén DOM
Impure	DOM manipulatie
Mapping	user input naar UI resultaat
UX	visuele feedback is cruciaal

## Hoofdstuk 5: Functions & Strings in ES6 (Template Literals)

### Leerdoelen

Na dit hoofdstuk kan je:

- ES6 **template literals** gebruiken (```)
- variabelen en functies verwerken in strings
- multiline strings schrijven
- dynamische tekst genereren in UI
- string-interpolatie gebruiken in JS functies

### Theorie: Template Literals

#### Oude manier (slecht leesbaar)

```

const naam = "Tom";
const zin = "Hallo, " + naam + "! Welkom.";

```

#### Moderne ES6 manier

```

const naam = "Tom";
const zin = `Hallo, ${naam}! Welkom.`;

```

## Voordelen

Voordeel	Waarom
Leesbaar	natuurlijke zinnen
Variabelen in string	<code> \${variabele}</code>
Multiline strings	geen \n nodig
Perfect voor HTML	dynamische UI content

## Multiline voorbeeld

```
const blokje = `  
Dit is een titel  
En dit is een tweede lijn  
Het leest heerlijk  
`;
```

## Real-life analogie

Template literals zijn **post-its met invulvelden**:

“Welkom, **{naam}**! Je score is **{score}**.”

In JS:

```
`${naam} ${score}`
```

## Demo: Welkomstbericht generator

We bouwen:

- input veld voor naam
- input veld voor leeftijd
- knop
- output met template literal

```
index.html (plaats onder vorige card)  
<div class="card shadow-sm mt-4">  
  <div class="card-header bg-info text-white">  
    Hoofdstuk 5: Template Literals  
  </div>  
  <div class="card-body">  
  
    <label class="form-label">Naam:</label>  
    <input id="tl_name" class="form-control mb-2" placeholder="bv. Tom">  
  
    <label class="form-label">Leeftijd:</label>  
    <input id="tl_age" type="number" class="form-control mb-2" placeholder="bv. 25">  
  
    <button id="tl_btn" class="btn btn-info w-100 mb-2">Maak  
    welkomstbericht</button>  
  
    <div id="tl_output" class="alert alert-secondary mb-0">  
      Wacht op input...
```

```

</div>

</div>
</div>

main.js (onderaan toevoegen)
// -----
// Hoofdstuk 5: Template Literals
// -----


// Pure function
function maakWelkomstZin(naam, leeftijd) {
    return `Welkom ${naam}! Jij bent ${leeftijd} jaar oud. `;
}

// UI handler
function toonWelkomstZin() {
    const naam = document.getElementById("tl_name").value.trim();
    const leeftijd = document.getElementById("tl_age").value.trim();
    const out = document.getElementById("tl_output");

    if (!naam || !leeftijd) {
        out.className = "alert alert-warning mb-0";
        out.textContent = `⚠ Vul naam en leeftijd in`;
        return;
    }

    const tekst = maakWelkomstZin(naam, leeftijd);

    out.className = "alert alert-success mb-0";
    out.textContent = tekst;
}
}

// Event
document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("tl_btn")
        .addEventListener("click", toonWelkomstZin);
});

```

Output type:

*Welkom Tom! Jij bent 25 jaar oud.*

## Mini-opdracht

Maak een **profielkaart generator**.

### Inputvelden

- naam
- stad
- hobby

### Output (Template Literal)

Voorbeeld:

Hey, ik ben Sara uit Gent.  
Mijn hobby is schilderen.  
Leuk je te ontmoeten!

Gebruik:

- 3 inputvelden
- 1 knop
- 1 output <div>
- Template literal met minimaal **twee regels tekst**
- Validatie (geen lege waarden)

### Verplichte ID's

pf\_name  
pf\_city  
pf\_hobby  
pf\_btn  
pf\_out

## Mini-opdracht begeleide oplossing

### HTML

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-dark text-white">Mini-opdracht:
    Profielkaart</div>
  <div class="card-body">

    <input id="pf_name" class="form-control mb-2" placeholder="Naam">
    <input id="pf_city" class="form-control mb-2" placeholder="Stad">
    <input id="pf_hobby" class="form-control mb-2" placeholder="Hobby">

    <button id="pf_btn" class="btn btn-dark w-100 mb-2">Maak profiel</button>

    <div id="pf_out" class="alert alert-secondary mb-0">
      Wacht op input...
    </div>

  </div>
</div>
```

```

JS
function maakProfielZin(naam, stad, hobby) {
    return `
Hey, ik ben ${naam} uit ${stad}.
Mijn hobby is ${hobby}.
Leuk je te ontmoeten!
`;
}

function toonProfiel() {
    const naam = document.getElementById("pf_name").value.trim();
    const stad = document.getElementById("pf_city").value.trim();
    const hobby = document.getElementById("pf_hobby").value.trim();
    const out = document.getElementById("pf_out");

    if (!naam || !stad || !hobby) {
        out.className = "alert alert-warning mb-0";
        out.textContent = `⚠️ Vul alle velden in`;
        return;
    }

    const tekst = maakProfielZin(naam, stad, hobby);

    out.className = "alert alert-success mb-0";
    out.textContent = tekst;
}

document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("pf_btn")
        ?.addEventListener("click", toonProfiel);
});

```

## Samenvatting

Concept	Uitleg
Template literals	`Hallo \${naam}` ``
String interpolation	variabelen in tekst
Multiline strings	tekstblokken zonder
Pure fn	maakt tekst
UI fn	toont tekst in alert

# Hoofdstuk 6: ES6 Arrow Functions (=>)

## Leerdoelen

Na dit hoofdstuk kan je:

- een klassieke functie omschrijven naar een **arrow function**
- arrow function syntax herkennen en toepassen
- **implicit return** gebruiken
- parameters correct verwerken
- arrow vs gewone functies begrijpen (basis)
- arrow functions in UI handlers toepassen

We focussen eerst op **syntaxis & gebruik**. Het verschil in `this` leren we later bij OOP.

## Theorie: Arrow Functions

Arrow functions zijn **kortere ES6 functies**:

### Klassiek

```
function optellen(a, b) {  
    return a + b;  
}
```

### Arrow

```
const optellen = (a, b) => {  
    return a + b;  
};
```

**Arrow met implicit return**  (**één regel!!**)

```
const optellen = (a, b) => a + b;
```

### Enkele parameters (haakjes optioneel)

```
const vierkant = x => x * x;
```

### Geen parameters

```
const hallo = () => "Hallo!";
```

### Waarom gebruiken?

Voordeel	Waarom
Kort	minder typwerk
Duidelijk	super voor kleine functies
Modern	hoort bij ES6 mindset
Perfect voor callbacks	events, array methods straks

### Vergelijking

Klassiek	Arrow
<code>function(x){ return x+1 }</code>	<code>x =&gt; x + 1</code>
<code>function(){}</code>	<code>() =&gt; {}</code>
<code>function naam(){}</code>	<code>const naam = () =&gt; {}</code>

## Let op

Arrow functions hebben **geen eigen this**. Daarover later bij OOP.

## Oefening: Bereken + Toon (arrow style)

We bouwen:

- 2 inputvelden
- optellen met arrow function
- resultaat tonen met Bootstrap alert

```
index.html blok
<div class="card shadow-sm mt-4">
  <div class="card-header bg-success text-white">
    Hoofdstuk 6: Arrow Functions
  </div>
  <div class="card-body">

    <label class="form-label">Getal 1:</label>
    <input id="af_num1" type="number" class="form-control mb-2"
placeholder="vb. 5">

    <label class="form-label">Getal 2:</label>
    <input id="af_num2" type="number" class="form-control mb-2"
placeholder="vb. 7">

    <button id="af_btn" class="btn btn-success w-100 mb-2">Tel op</button>

    <div id="af_out" class="alert alert-secondary mb-0">
      Wacht op input...
    </div>
  </div>
</div>

main.js blok
// -----
// Hoofdstuk 6: Arrow Functions
// -----


// Pure arrow function
const telOp = (a, b) => a + b;

// UI handler (arrow + block body)
const verwerkSom = () => {
  const n1 = Number(document.getElementById("af_num1").value);
  const n2 = Number(document.getElementById("af_num2").value);
  const out = document.getElementById("af_out");

  if (!n1 || !n2) {
    out.className = "alert alert-warning mb-0";
    out.textContent = `⚠️ Vul twee getallen in`;
  }
}
```

```

        return;
    }

const resultaat = telOp(n1, n2);

out.className = "alert alert-success mb-0";
out.textContent = `Resultaat: ${n1} + ${n2} = ${resultaat}`;
};

// Event
document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("af_btn")
        .addEventListener("click", verwerkSom);
});

```

### Wat zag je?

- pure arrow function: const telOp = (a, b) => a + b
- arrow function als event handler
- implicit return
- input → berekening → output

### Mini-opdracht

Maak een arrow function die checkt of een getal even is.

#### Verwacht gedrag

Input	Output
4	"4 is even"
7	✗ "7 is niet even"

#### IDs te gebruiken

ev\_input  
ev\_btn  
ev\_out

#### hints

Arrow function:

```
const isEven = num => num % 2 === 0;
```

## Oplossing mini-opdracht

### HTML

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-dark text-white">Mini-opdracht: Even
  checker</div>
  <div class="card-body">

    <input id="ev_input" type="number" class="form-control mb-2"
placeholder="vb. 6">
    <button id="ev_btn" class="btn btn-dark w-100 mb-2">Controleer</button>

    <div id="ev_out" class="alert alert-secondary mb-0">
      Wacht op input...
    </div>

  </div>
</div>
```

### JS

```
const isEven = num => num % 2 === 0;

const checkEven = () => {
  const inp = Number(document.getElementById("ev_input").value);
  const out = document.getElementById("ev_out");

  if (!inp) {
    out.className = "alert alert-warning mb-0";
    out.textContent = `⚠️ Vul een getal in`;
    return;
  }

  const resultaat = isEven(inp)
    ? `${inp} is EVEN`
    : `${inp} is ONEVEN ✗`;

  out.className = "alert alert-success mb-0";
  out.textContent = resultaat;
};

document.addEventListener("DOMContentLoaded", () => {
  document.getElementById("ev_btn")
    ?.addEventListener("click", checkEven);
});
```

### Samenvatting

Je kent nu	Wat het betekent
Arrow syntax	const fn = () => {...}
Implicit return	x => x * 2
Event handler met arrow	btn.addEventListener("click", ()=>{})
Separatie van logica en UI	clean & schaalbaar

# Hoofdstuk 7: ES6 Arrays

## Leerdoelen

Na dit hoofdstuk kan je:

- een array definiëren in ES6
- items toevoegen, uitlezen en tellen
- .push() en .length gebruiken
- input opslaan in een array
- een lijst tonen in de UI via JavaScript
- basis denken richting datastructuren

## Theorie: Arrays

### Wat is een array?

Een **lijst** met meerdere waarden in één variabele.

```
const namen = ["Tom", "Sara", "Liam"];
```

### Waarom arrays?

Voordeel	Waarom
Bewaart meerdere waarden	“groepeert data”
Behoud volgorde	belangrijk voor logica
Makkelijk uitbreiden	.push(...)
Basis voor echte databanken	denken in collecties

### Basis handelingen

Doel	Code	Uitleg
Item ophalen	namen[0]	eerste item
Aantal items	namen.length	lengte van lijst
Item toevoegen	namen.push("Jan")	voeg achteraan toe

### Vergelijking in real life

Een array is een **rij stoelen** in een klas. Iedere stoel = een plek voor data.

Stoel 0 = eerste Stoel 1 = tweede Stoel 2 = derde ...

Programmeren is vaak “data + acties op data”. Arrays zijn je rugzak.

## Demo: gastenlijst (array + UI)

We bouwen:

- input veld "naam"
- knop "voeg toe"
- array bewaart namen
- lijstje toont alle namen live

```
index.html blok
<div class="card shadow-sm mt-4">
  <div class="card-header bg-primary text-white">
    Hoofdstuk 7: Arrays
  </div>
  <div class="card-body">

    <label class="form-label">Voeg een naam toe:</label>
    <input id="arr_name" class="form-control mb-2" placeholder="bv. Lisa">

    <button id="arr_btn" class="btn btn-primary w-100 mb-2">Voeg toe</button>

    <div class="alert alert-info mb-2">
      Totaal namen: <span id="arr_count">0</span>
    </div>

    <ul id="arr_list" class="list-group"></ul>

  </div>
</div>

main.js blok
// -----
// Hoofdstuk 7: Arrays
// -----


// Lege array om namen op te slaan
const namen = [];

// Pure functie -> maakt lijst HTML
function maakLijstHTML(items) {
  return items.map(item => `<li class="list-group-item">${item}</li>`).join("");
}

// UI handler
function voegNaamToe() {
  const inp = document.getElementById("arr_name");
  const lijst = document.getElementById("arr_list");
  const count = document.getElementById("arr_count");

  const naam = inp.value.trim();

  if (!naam) {
```

```

    alert("⚠ Geef een naam in!");
    return;
}

// voeg toe aan array
namen.push(naam);

// UI updaten
lijst.innerHTML = maakLijstHTML(namen);
count.textContent = namen.length;

// veld leegmaken
inp.value = "";
}

// Event Listener
document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("arr_btn")
        ?.addEventListener("click", voegNaamToe);
});

```

Je hebt nu een **dynamische gastenlijst**. Dit is al een mini-database.

## Mini-opdracht

Maak een **takenlijst** (todo list).

### Vereisten

Inputveld: taak Knop: “Voeg taak toe” Lijst toont taken Tel taken in badge

### Verplichte ID's

todo\_input  
todo\_btn  
todo\_list  
todo\_count

### Extra uitdaging

Voeg aan elk item toe (alleen tekst, nog geen echte delete-knop).

## Mini-opdracht oplossing

### HTML

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-dark text-white">Mini-opdracht: Takenlijst</div>
  <div class="card-body">

    <input id="todo_input" class="form-control mb-2" placeholder="Nieuwe taak">
    <button id="todo_btn" class="btn btn-dark w-100 mb-2">Voeg toe</button>

    <div class="alert alert-secondary mb-2">
      Taken: <span id="todo_count">0</span>
    </div>

    <ul id="todo_list" class="list-group"></ul>

  </div>
</div>
```

### JS

```
const taken = [];

function maakTodoHTML(items) {
  return items.map(item => `<li class="list-group-item">
${item}</li>`).join("");
}

function voegTaakToe() {
  const inp = document.getElementById("todo_input");
  const lijst = document.getElementById("todo_list");
  const count = document.getElementById("todo_count");

  const taak = inp.value.trim();

  if (!taak) {
    alert("⚠️ Vul een taak in!");
    return;
  }

  taken.push(taak);
  lijst.innerHTML = maakTodoHTML(taken);
  count.textContent = taken.length;
  inp.value = "";
}

document.addEventListener("DOMContentLoaded", () => {
  document.getElementById("todo_btn")
    ?.addEventListener("click", voegTaakToe);
});
```

## Samenvatting

Je hebt geleerd	Betekenis
[ ]	array maken
.push()	waarde toevoegen
.length	aantal items ophalen
map()	array omzetten naar HTML
State + UI	basis van moderne apps

Dit is de fundering voor:

- todo apps
- databanken
- API's & JSON
- frameworks zoals Vue & React

## Hoofdstuk 8: DOM Nodes & Root Nodes

### Leerdoelen

Na dit hoofdstuk kan je:

- uitleggen wat een **DOM-node** is
- **root nodes** herkennen (`document`, `document.documentElement`, `document.body`)
- HTML als **boomstructuur** zien
- nodes inspecteren in de browser
- root nodes aanspreken in JavaScript
- tekst in de pagina zetten via root-node selectie

### Theorie

#### Wat is de DOM?

#### DOM = Document Object Model

De DOM is een **boom van nodes** die jouw HTML in JavaScript voorstelt.

Browser vertaalt HTML → boomstructuur:

```
document
  └── html
    ├── head
    └── body
```

## Wat is een Node?

Een **bouwblok** van de DOM.

Voorbeelden:

HTML	Node type
<html>	Element node
<body>	Element node
Tekst tussen tags	Text node
Commentaar	Comment node

## Root nodes (basis van alles)

Node	Betekenis
document	startpunt van de DOM
document.documentElement	<html> element
document.body	<body> element

## Visualisatie in devtools

Rechtsklik → Inspect → Elements

Hier zie je de DOM live. Dit is hoe JS je website **ziet**.

## Denkbeeld

HTML = huisplan DOM = het echte gebouw in 3D waarin je rondwandelt Nodes = kamers, muren, deuren

## Demo: tekst zetten via root nodes

We maken een card die:

- tekst schrijft naar <body>
- achtergrond verandert
- toont welke root node we selecteren

```
index.html blok
<div class="card shadow-sm mt-4">
  <div class="card-header bg-info text-white">
    Hoofdstuk 8: DOM Root Nodes
  </div>
  <div class="card-body">

    <button id="rn_btnMessage" class="btn btn-info w-100 mb-2">Zet bericht in
    body</button>
    <button id="rn_btnColor" class="btn btn-dark w-100 mb-2">Verander
    achtergrond</button>

    <div id="rn_output" class="alert alert-secondary mb-0">
      Wacht op actie...
    </div>
```

```

</div>
</div>

main.js blok
// -----
// Hoofdstuk 8: Root Nodes
// -----


function plaatsBerichtInBody() {
    // selecteer de body
    const bodyNode = document.body;

    // maak een nieuw element
    const p = document.createElement("p");
    p.textContent = "Bericht toegevoegd via root node! ";
    p.className = "text-center mt-2 text-success fw-bold";

    // voeg toe aan body
    bodyNode.appendChild(p);
}

function veranderAchtergrond() {
    const htmlNode = document.documentElement; // <html>

    // lichte highlight
    htmlNode.style.background = "#e6f3ff";
}

function toonFeedback(tekst) {
    const out = document.getElementById("rn_output");
    out.className = "alert alert-success mb-0";
    out.textContent = tekst;
}

// Event listeners
document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("rn_btnMessage")
        ?.addEventListener("click", () => {
            plaatsBerichtInBody();
            toonFeedback("Bericht toegevoegd aan body ");
        });

    document.getElementById("rn_btnColor")
        ?.addEventListener("click", () => {
            veranderAchtergrond();
            toonFeedback("Achtergrond aangepast via <html> node 🎉");
        });
});

```

## Wat gebeurde hier?

- We selecteerden root nodes
- We creëerden een nieuw element via JS
- We voegden dat toe aan <body>
- We veranderden stijl via <html>

Jij kunt nu de **DOM-boom manipuleren vanaf de wortel**.

## Mini-opdracht

Maak een knop die:

- <body> een donker thema geeft
- tekst toont: "Dark mode geactiveerd 🌙"
- tweede knop maakt het weer licht

### Verplichte ID's:

dm\_on  
dm\_off  
dm\_status

### Effecten

Actie	Effect
Dark mode	document.body.style.background = "#222", text wit
Light mode	reset naar wit

## Mini-opdracht oplossing

### HTML

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-dark text-white">Mini-opdracht: Dark Mode</div>
  <div class="card-body">

    <button id="dm_on" class="btn btn-dark w-100 mb-2">Dark mode 🌙</button>
    <button id="dm_off" class="btn btn-light w-100 mb-2">Light mode
    ○ </button>

    <div id="dm_status" class="alert alert-secondary mb-0">Wacht op
    actie...</div>

  </div>
</div>
```

### JS

```
function activeerDarkMode() {
  document.body.style.background = "#222";
  document.body.style.color = "#fff";
}

function activeerLightMode() {
  document.body.style.background = "#fff";
```

```

        document.body.style.color = "#000";
    }

    function toonStatus(msg) {
        const out = document.getElementById("dm_status");
        out.className = "alert alert-success mb-0";
        out.textContent = msg;
    }

document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("dm_on")
        ?.addEventListener("click", () => {
            activeerDarkMode();
            toonStatus("Dark mode actief 🌙");
        });

    document.getElementById("dm_off")
        ?.addEventListener("click", () => {
            activeerLightMode();
            toonStatus("Light mode actief ☀️");
        });
});

```

## Samenvatting

Concept	Wat je nu weet
DOM	Boomstructuur van pagina
Node	elk element in boom
Root nodes	document, html, body
Manipulatie	DOM aanpassen via JS
Append	elementen dynamisch toevoegen

Jij kunt nu **vanaf de wortel de pagina beïnvloeden**. Dat is key voor alles wat volgt.

## Volgend hoofdstuk

### Hoofdstuk 9: JavaScript Output Methods

We gaan:

- verschillende manieren ontdekken om output te tonen
- DOM vs console vs.alert
- waar je wat gebruikt in echte projecten

Als je klaar bent:

**doorgaan** Geweldig. We bouwen verder aan jouw JS-fundament.

# Hoofdstuk 9: JavaScript Output Methodes

## Leerdoelen

Na dit hoofdstuk kan je:

- verschillende manieren gebruiken om informatie te tonen in JavaScript
- begrijpen wanneer je welke output-methode gebruikt
- console debuggen, alert tonen, HTML manipuleren
- Bootstrap alerts gebruiken voor nette UI-meldingen
- DOM output methoden vergelijken (**innerText vs textContent vs innerHTML** volgt in apart hoofdstuk)

## Theorie: Output Methodes

### 1) Console

Voor developers, debugging en testen

```
console.log("Dit is een log");
console.warn("Dit is een waarschuwing");
console.error("Dit is een fout");
```

Waarom gebruiken? Debug info Fouten lezen Data bekijken (arrays, objects)

### 2) Alert

Popup waarschuwing voor gebruiker (zeldzaam in echte apps)

```
alert("Hallo gebruiker!");
```

Waarom voorzichtig mee zijn? ↗ Blokkeert pagina ↗ Niet mooi → Enkel voor snelle testjes of dringende meldingen

### 3) Tekst op pagina

Mooiste manier voor eindgebruikers

```
document.getElementById("id").textContent = "Hallo!";
```

Gebruiken voor: Meldingen UI feedback Output van berekeningen

We gebruiken **Bootstrap alerts** voor professioneel resultaat.

## Demo oefening: 3 outputmethoden

### Wat we bouwen

- 3 knoppen: Console, Alert, UI
- Output naar console, popup en HTML card
- Zichtbaar leerresultaat

```
index.html
<div class="card shadow-sm mt-4">
  <div class="card-header bg-primary text-white">
    Hoofdstuk 9: Output Methodes
  </div>
  <div class="card-body">

    <button id="om_console" class="btn btn-outline-primary w-100 mb-2">
      Console Log
    </button>

    <button id="om_alert" class="btn btn-outline-warning w-100 mb-2">
      Alert Meldingsvenster
    </button>

    <button id="om_ui" class="btn btn-outline-success w-100 mb-2">
      Toon in pagina
    </button>

    <div id="om_output" class="alert alert-secondary mb-0">
      Output wordt hier zichtbaar...
    </div>
  </div>
</div>

main.js
// -----
// Hoofdstuk 9: Output Methodes
// -----


// Pure tekstfunctie
function maakBericht() {
  return "Hello output wereld! 🌎";
}

// Impure UI output
function toonInUI(msg) {
  const out = document.getElementById("om_output");
  out.className = "alert alert-success mb-0";
  out.textContent = msg;
}

// Event handlers
function logNaarConsole() {
  console.log(maakBericht());
```

```

}

function toonAlert() {
    alert(maakBericht());
}

function toonOutputUI() {
    toonInUI(maakBericht());
}

// Event listeners
document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("om_console")?.addEventListener("click",
logNaarConsole);
    document.getElementById("om_alert")?.addEventListener("click", toonAlert);
    document.getElementById("om_ui")?.addEventListener("click", toonOutputUI);
});

```

Console Alert UI weergave Functie-ontkoppeling (clean code)

## Mini-opdracht

Maak een blok dat:

- 2 inputvelden heeft: voornaam en achternaam
- knop: “Toon resultaten”
- output toont op 3 manieren:

Methode	Wat
console	“Voornaam: Tom, Achternaam: Vanhoutte”
alert	“Hallo Tom Vanhoutte!”
pagina	Bootstrap alert met volledige naam

### Verplichte ID's

on\_first  
on\_last  
on\_btn  
on\_out

### Hint

Gebruik template literals:

```
`Hallo ${voornaam} ${achternaam}!`
```

## Mini-opdracht oplossing

### HTML

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-dark text-white">Mini-opdracht: Output All</div>
  <div class="card-body">

    <input id="on_first" class="form-control mb-2" placeholder="Voornaam">
    <input id="on_last" class="form-control mb-2" placeholder="Achternaam">

    <button id="on_btn" class="btn btn-dark w-100 mb-2">Toon
    resultaten</button>

    <div id="on_out" class="alert alert-secondary mb-0">Wacht op input...</div>

  </div>
</div>
```

### JS

```
function toonAllOutputs() {
  const first = document.getElementById("on_first").value.trim();
  const last = document.getElementById("on_last").value.trim();
  const out = document.getElementById("on_out");

  if (!first || !last) {
    out.className = "alert alert-warning mb-0";
    out.textContent = "⚠ Vul naam en achternaam in";
    return;
  }

  const line = `Voornaam: ${first}, Achternaam: ${last}`;
  const hello = `Hallo ${first} ${last}!`;

  console.log(line);
  alert(hello);

  out.className = "alert alert-success mb-0";
  out.textContent = hello;
}

document.addEventListener("DOMContentLoaded", () => {
  document.getElementById("on_btn")
    ?.addEventListener("click", toonAllOutputs);
});
```

### Samenvatting

Methode	Gebruik
console.log	Debugging / leren
alert()	Snel testen
DOM output	Echte user feedback

Je kunt nu **verschillende output kanalen** gebruiken en weet **waarom**.

## Hoofdstuk 10: document.write() en document.writeln()

### Leerdoelen

Na dit hoofdstuk kan je:

- uitleggen wat `document.write()` doet
- het verschil kennen tussen `.write()` en `.writeln()`
- begrijpen waarom deze functies **niet** meer gebruikt worden in moderne webapps
- de moderne alternatieven toepassen

### Theorie

#### Wat doet `document.write()`?

Schrijft **rechtstreeks** tekst/HTML naar de pagina **tijdens het laden**.

```
document.write("Hallo wereld");
```

#### Wat doet `document.writeln()`?

Hetzelfde, maar voegt een **lijnbreak** toe.

```
document.writeln("Hallo");
document.writeln("wereld");
```

#### Waarom gebruiken we dit bijna nooit meer?

Probleem	Uitleg
Overschrijft pagina	Alles kan verdwijnen als je het na load gebruikt
Geen controle	Plakt HTML direct op document-stream
Niet modern	Geen componenten, geen dynamiek
Niet toegankelijk	Breekt makkelijk layouts

#### Waarvoor bestaat het nog?

Zeer oude tutorials Legacy systemen leren begrijpen Debuggen in *heel oude* scripts

### Moderne alternatieven

Gebruik:

- `textContent`
- `innerHTML`
- DOM methods (`appendChild`, `createElement`)
- Bootstrap alerts

We gebruiken **document.write één keer**, puur educatief.

## Demo: Waarom je het niet gebruikt

```
index.html blok
<div class="card shadow-sm mt-4">
  <div class="card-header bg-danger text-white">
    Hoofdstuk 10: document.write() (niet meer gebruiken)
  </div>
  <div class="card-body">

    <button id="dw_btn" class="btn btn-danger w-100 mb-2">
      document.write uitvoeren
    </button>

    <div id="dw_msg" class="alert alert-secondary mb-0">
      Druk op de knop en let op de pagina...
    </div>

  </div>
</div>

main.js blok
// -----
// Hoofdstuk 10: document.write()
// -----


function demonstreerWrite() {
  // Waarschuwing tonen
  alert("Let op: de pagina zal overschreven worden!");

  // Dit overschrijft alle HTML
  document.write("<h1>De hele pagina is vervangen 😳</h1>");
  document.write("<p>Daarom gebruiken we dit niet meer.</p>");
  document.writeln("<p>Gebruik DOM methods en Bootstrap.</p>");
}

// Event
document.addEventListener("DOMContentLoaded", () => {
  document.getElementById("dw_btn")
    ?.addEventListener("click", demonstreerWrite);
});
```

Dit is een **live lesson**:

- Click = hele document weg
- Student snapt meteen waarom dit verouderd is

## Mini-opdracht

Toon aan dat **moderne DOM output beter is.**

Maak een vergelijkingskaart met 2 knoppen:

Knop	Actie
"Write old way"	<code>document.write("ಠ_ಠ Oeps")</code>
"Write modern way"	update een <div> met mooie Bootstrap alert

### ID's

`ow_old`  
`ow_new`  
`ow_out`

### Doele

Output 1 (slecht):

ಠ\_ಠ pagina stuk

Output 2 (goed):

Bootstrap success alert:

Moderne DOM output is beter!

## Mini-opdracht oplossing

### HTML

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-warning text-dark">Mini-opdracht: oud vs
nieuw</div>
  <div class="card-body">

    <button id="ow_old" class="btn btn-outline-danger w-100 mb-2">Old way
X</button>
    <button id="ow_new" class="btn btn-outline-success w-100 mb-2">Modern
</button>

    <div id="ow_out" class="alert alert-secondary mb-0">
      Wacht op een keuze...
    </div>

  </div>
</div>
```

### JS

```
function oudOutput() {
  alert("Nu breekt de pagina ಠ_ಠ");
  document.write("ಠ_ಠ Oeps, document.write overschreef alles!");
}

function nieuwOutput() {
```

```

const box = document.getElementById("ow_out");
box.className = "alert alert-success mb-0";
box.textContent = " Moderne DOM output is beter!";
}

document.addEventListener("DOMContentLoaded", () => {
  document.getElementById("ow_old")?.addEventListener("click", oudOutput);
  document.getElementById("ow_new")?.addEventListener("click", nieuwOutput);
});

```

## Samenvatting

Methode	Gebruik	Status
document.write()	pagina direct overschrijven	✗ legacy
textContent, innerHTML	tekst/HTML tonen	modern
DOM methods	UI bouwen	aanbevolen

**Document.write = no-go** in moderne webapps.

## Hoofdstuk 11: innerHTML VS textContent VS innerText

### Doel van dit hoofdstuk

Na deze module kan je:

- het verschil uitleggen tussen innerHTML, textContent en innerText
- veilig tekst tonen in de UI
- HTML injecteren wanneer het WEL mag
- beschermen tegen gebruikersinput die HTML bevat
- een realistische interface bouwen met visuele feedback

Dit hoofdstuk is waar studenten vaak denken “Ah ok... nu snap ik het.”

### Theorie

#### textContent

**Toont ruwe tekst.** Negeert HTML. Veilig.

```
element.textContent = "<b>Hallo</b>";
```

Resultaat op pagina:

```
<b>Hallo</b>
```

Gebruik wanneer: Je content is tekst Je wilt HTML niet uitvoeren Security is belangrijk

#### innerHTML

**Voert HTML uit.** Kan styling en tags toevoegen.

```
element.innerHTML = "<b>Hallo</b>";
```

Resultaat op pagina:

## Hallo

Gebruik wanneer: Je wilt HTML inzetten ⚡ Alleen met veilige content

### innerText

Toont tekst zoals zichtbaar in de browser. Houdt rekening met:

- CSS
- zichtbaarheid
- line breaks

Wordt minder gebruikt dan de andere twee.

### Kort overzicht

Methode	Voert HTML uit?	Beste gebruik
textContent	Nee	Veilig tekst tonen
innerHTML	Ja	Dynamische HTML, icons, kleur
innerText	Nee, maar format volgt UI	Tekst zoals gebruiker ziet

### Veiligheid tip

Gebruik NOoit direct innerHTML met onbekende user-input.

Dit is hoe hackers scripts injecteren.

We tonen dat zo meteen.

### Demo: drie output methoden

```
index.html
<div class="card shadow-sm mt-4">
  <div class="card-header bg-info text-white">
    Hoofdstuk 11: innerHTML vs textContent vs innerText
  </div>

  <div class="card-body">
    <input id="ht_input" class="form-control mb-2" placeholder="Typ iets met
HTML">

    <button id="ht_btn_show" class="btn btn-info w-100 mb-3">
      Toon resultaten
    </button>

    <div class="alert alert-secondary mb-2">textContent:</div>
    <div id="ht_textContent" class="border p-2"></div>

    <div class="alert alert-secondary mt-3 mb-2">innerText:</div>
    <div id="ht_innerText" class="border p-2"></div>

    <div class="alert alert-secondary mt-3 mb-2">innerHTML:</div>
    <div id="ht_innerHTML" class="border p-2"></div>
```

```

</div>
</div>

main.js
// -----
// Hoofdstuk 11: innerHTML vs textContent vs innerText
// -----


function toonOutputTypes() {
    const input = document.getElementById("ht_input").value;

    document.getElementById("ht_textContent").textContent = input;
    document.getElementById("ht_innerText").innerText = input;
    document.getElementById("ht_innerHTML").innerHTML = input;
}

document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("ht_btn_show")
        ?.addEventListener("click", toonOutputTypes);
});

```

Test met:

<b>Hallo</b>

Je zal meteen zien:

- textContent toont letterlijk <b>Hallo</b>
- innerHTML maakt het vet
- innerText werkt bijna hetzelfde maar behoudt browserstijl logica

## Mini-opdracht

Maak een inputveld waarin studenten HTML proberen te typen, maar jij toont het altijd **veilig**.

### UI eisen

- Input veld
- Knop: "Veilig tonen"
- Output card
- Geen HTML mag uitgevoerd worden

### ID's

safe\_input  
safe\_btn  
safe\_output

### Hint

```
safe_output.textContent = userInput;
```

## Oplossing mini-opdracht

### HTML

```
<div class="card shadow-sm mt-4">
  <div class="card-header bg-dark text-white">Mini-opdracht: Veilige
  output</div>
  <div class="card-body">

    <input id="safe_input" class="form-control mb-2" placeholder="Typ HTML
    code">

    <button id="safe_btn" class="btn btn-dark w-100 mb-2">Veilig
    tonen</button>

    <div id="safe_output" class="alert alert-secondary mb-0">
      Wacht op input...
    </div>
  </div>
</div>
```

### JS

```
function toonVeilig() {
  const val = document.getElementById("safe_input").value;
  const out = document.getElementById("safe_output");

  out.className = "alert alert-success mb-0";
  out.textContent = val;
}

document.addEventListener("DOMContentLoaded", () => {
  document.getElementById("safe_btn")
    ?.addEventListener("click", toonVeilig);
});
```

## Samenvatting

Methode	Gebruik
textContent	Veilig tekst tonen, voorkom XSS
innerHTML	HTML uitvoeren, styling, icons
innerText	Tekst tonen zoals zichtbaar op pagina

Belangrijkste les:

Gebruik `textContent` voor user input. Gebruik `innerHTML` alleen voor **jouw eigen veilige strings**.

## Hoofdstuk 12: DOM Selectors

### Leerdoelen

Na dit hoofdstuk kan je:

- elementen selecteren met verschillende DOM-methodes
- het verschil begrijpen tussen **single** en **multiple** selectors
- `getElementById`, `getElementsByName`, `querySelector`, `querySelectorAll` gebruiken
- elementen dynamisch manipuleren in UI
- lijstitems filteren op basis van input

### Theorie: DOM Selectors

Wanneer je de DOM wil gebruiken, moet je **iets kunnen aanwijzen**. Dit zijn je tools:

Methode	Teruggeven	Gebruik
<code>getElementById()</code>	1 element	snel & meest gebruikt
<code>getElementsByName()</code>	HTMLCollection	meerdere elementen met een class
<code>querySelector()</code>	1 element	CSS-selector stijl
<code>querySelectorAll()</code>	NodeList	meerdere elementen met CSS-selectors

### Voorbeelden

```
getElementById  
const title = document.getElementById("titel");  
  
getElementsByName  
const items = document.getElementsByName("item");
```

geeft een `HTMLCollection` → lijkt op array maar is geen echte array

```
querySelector  
const btn = document.querySelector(".btn-primary");  
  
querySelectorAll  
const links = document.querySelectorAll("a.link");
```

geeft een NodeList → kan je **forEach** op gebruiken

## Denkbeeld

Stel de DOM voor als een speelterrein. Deze functions zijn **zoeklichten**:

- `getElementById` → “zoek dat ene ding daar”
- `getElementsByName` → “pak alle elementen in deze rij”
- `querySelector` → “haal het eerste voorbeeld van dit patroon”
- `querySelectorAll` → “haal alles dat past op dit patroon”

## Demo: lijst filteren met `querySelectorAll`

We maken:

- inputveld
- lijst van namen
- live filter op basis van wat je typt

```
index.html
<div class="card shadow-sm mt-4">
  <div class="card-header bg-primary text-white">
    Hoofdstuk 12: DOM Selectors
  </div>
  <div class="card-body">

    <label class="form-label">Zoek een naam:</label>
    <input id="sel_input" class="form-control mb-3" placeholder="bv. Tom">

    <ul id="sel_list" class="list-group">
      <li class="list-group-item naam-item">Tom</li>
      <li class="list-group-item naam-item">Sara</li>
      <li class="list-group-item naam-item">Liam</li>
      <li class="list-group-item naam-item">Emma</li>
      <li class="list-group-item naam-item">Noah</li>
    </ul>

  </div>
</div>

main.js
// -----
// Hoofdstuk 12: DOM Selectors
// -----


function filterNamen() {
  const value = document.getElementById("sel_input").value.toLowerCase();
  const items = document.querySelectorAll(".naam-item"); // NodeList

  items.forEach(item => {
    const text = item.textContent.toLowerCase();
    item.style.display = text.includes(value) ? "block" : "none";
  });
}
```

```

document.addEventListener("DOMContentLoaded", () => {
  document.getElementById("sel_input")
    ?.addEventListener("keyup", filterNamen);
});

```

Live filtering Simpele UX Klassieke querySelectorAll praktijk

## Mini-opdracht

Maak een **kleurenswitcher**:

### UI

- Drie knoppen: **rood, blauw, groen**
- Alle `.kleur-box` elementen veranderen naar die kleur

### IDs & Classes

```

ks_red
ks_blue
ks_green
kleur-box

```

### Hints

```
const boxes = document.querySelectorAll(".kleur-box");
```

Loop door de boxes en verander de achtergrondkleur.

## Oplossing mini-opdracht

### HTML

```

<div class="card shadow-sm mt-4">
  <div class="card-header bg-dark text-white">Mini-opdracht:</div>
  Kleurenswitcher</div>
  <div class="card-body">

    <button id="ks_red" class="btn btn-danger w-100 mb-2">Rood</button>
    <button id="ks_blue" class="btn btn-primary w-100 mb-2">Blauw</button>
    <button id="ks_green" class="btn btn-success w-100 mb-3">Groen</button>

    <div class="d-flex gap-2">
      <div class="kleur-box p-3 border flex-fill text-center">Box 1</div>
      <div class="kleur-box p-3 border flex-fill text-center">Box 2</div>
      <div class="kleur-box p-3 border flex-fill text-center">Box 3</div>
    </div>

  </div>
</div>

```

```

JS
function setColor(color) {
  const boxes = document.querySelectorAll(".kleur-box");
  boxes.forEach(box => box.style.background = color);
}

document.addEventListener("DOMContentLoaded", () => {
  document.getElementById("ks_red")?.addEventListener("click", () =>
  setColor("red"));
  document.getElementById("ks_blue")?.addEventListener("click", () =>
  setColor("blue"));
  document.getElementById("ks_green")?.addEventListener("click", () =>
  setColor("green"));
});

```

### Samenvatting

Selector	Gebruik
getElementById	1 element, snel
getElementsByClassName	collectie, geen echte array
querySelector	1 element, CSS selectors
querySelectorAll	meerdere, NodeList met forEach

Jij kan nu de DOM **gericht doorzoeken en manipuleren**. Dat is web-toverkunst.

## Hoofdstuk 13: Andere DOM-element properties & acties

### Leerdoelen

Na deze module kan je:

- `.focus()` gebruiken om een veld actief te zetten
- `.blur()` gebruiken om focus weg te halen
- `.children` en `.childNodes` herkennen
- `.children.length` gebruiken om het aantal child-elementen te tellen
- interactieve UI-feedback bouwen

### Theorie

#### `.focus()`

Zet het cursor-focus in een element.

```
document.getElementById("veld").focus();
```

#### `.blur()`

Haalt de focus weg.

```
document.getElementById("veld").blur();
```

### Gebruiks momenten:

Waarvoor?
Automatisch cursor zetten in input
Focus verplaatsen na actie
Velden laten "oplichten" bij focus

### .children

Alle **element**-kinderen van een node.

container.children

### .childNodes

**Alles:** whitespace, tekst, comments, elementen.

container.childNodes

### .children.length

Aantal echte HTML-kinderen.

### Vergelijking

Property	Wat zit erin?	Gebruik
.children	Alleen elementen	UI manipulatie
.childNodes	Tekst, spaties, comments, elementen	Debugging, zeldzaam nodig

### Metafoor

Stel je DOM voor als een huis.

- .children = echte kamers
- .childNodes = kamers + stof in de hoek + post-it op deur + lucht

We willen meestal kamers, niet stof.

### Demo: focus highlight + child counter

We maken:

- 2 inputvelden
- knop om focus te zetten
- info-vak dat telt hoeveel children een container heeft

```
index.html
<div class="card shadow-sm mt-4">
  <div class="card-header bg-warning text-dark">
    Hoofdstuk 13: DOM Focus & Children
  </div>
  <div class="card-body">

    <label class="form-label">Naam:</label>
    <input id="d13_name" class="form-control mb-2" placeholder="bv. Tom">

    <label class="form-label">Email:</label>
```

```

<input id="d13_email" class="form-control mb-3" placeholder="bv.
tom@mail.be">

    <button id="d13_focusName" class="btn btn-warning w-100 mb-2">Focus op
Naam</button>
    <button id="d13_focusEmail" class="btn btn-dark w-100 mb-2">Focus op
Email</button>

    <div class="mt-3 alert alert-secondary">
        Totaal elementen in card-body: <span id="d13_count"></span>
    </div>
</div>

main.js
// -----
// Hoofdstuk 13: focus(), blur(), children
// -----


function updateChildCount() {
    const container = document.querySelector(".card-body");
    const count = container.children.length;
    document.getElementById("d13_count").textContent = count;
}

// Event binding
document.addEventListener("DOMContentLoaded", () => {
    updateChildCount();

    document.getElementById("d13_focusName")
        ?.addEventListener("click", () => {
            document.getElementById("d13_name").focus();
        });
    document.getElementById("d13_focusEmail")
        ?.addEventListener("click", () => {
            document.getElementById("d13_email").focus();
        });
});

```

## Mini-opdracht

Maak een **focus training spelletje**:

### Doel

Als de student op "Start oefening" klikt:

- input krijgt focus
- border wordt blauw bij focus
- border wordt grijs bij blur
- een teller toont hoeveel velden de container heeft

### UI

- Input
- Knop: "Start oefening"
- Alert met child count

### Verplichte ID's

ft\_input  
ft\_btn  
ft\_count

### Hints

```
input.addEventListener("focus", ...)  
input.addEventListener("blur", ...)
```

Border kleur tip:

```
input.style.border = "2px solid blue";
```

## Mini-opdracht oplossing

### HTML

```
<div class="card shadow-sm mt-4">  
  <div class="card-header bg-info text-white">Mini-opdracht: Focus Game</div>  
  <div class="card-body" id="ft_body">  
  
    <input id="ft_input" class="form-control mb-2" placeholder="Klik start...>">  
    <button id="ft_btn" class="btn btn-info w-100 mb-2">Start oefening</button>  
  
    <div class="alert alert-secondary mb-0">  
      Kind elementen: <span id="ft_count">0</span>  
    </div>  
  </div>
```

```

JS
function updateFTCount() {
  const body = document.getElementById("ft_body");
  document.getElementById("ft_count").textContent = body.children.length;
}

document.addEventListener("DOMContentLoaded", () => {
  const inp = document.getElementById("ft_input");

  updateFTCount();

  inp.addEventListener("focus", () => {
    inp.style.border = "2px solid blue";
  });

  inp.addEventListener("blur", () => {
    inp.style.border = "1px solid #ccc";
  });

  document.getElementById("ft_btn")?.addEventListener("click", () => {
    inp.focus();
  });
});

```

### Samenvatting

Concept	Wat je nu weet
.focus()	Zet cursor in veld
.blur()	Haalt focus weg
.children	Alleen echte elementen
.childNodes	Alles (tekst, comments, whitespace)
.children.length	Tel elementen

Jij stuurt nu de DOM als een chauffeur die de weg kent.

## Hoofdstuk 14: Klassen manipuleren (`classList`)

### Leerdoelen

Na dit hoofdstuk kan je:

- CSS-klassen toevoegen, verwijderen en togglen via JS
- `.classList.add()`, `.remove()`, `.toggle()` gebruiken
- UI interactief dynamisch maken
- actieve knoppen en dark-mode gedrag bouwen

### Theorie

#### `element.classList`

Geeft toegang tot de lijst met klassen van een element.

#### Belangrijkste methodes

Methode	Uitleg
<code>.add("naam")</code>	klasse toevoegen
<code>.remove("naam")</code>	klasse verwijderen
<code>.toggle("naam")</code>	toevoegen <b>of</b> verwijderen afhankelijk van status

#### Voorbeelden

```
box.classList.add("active");
box.classList.remove("hidden");
box.classList.toggle("highlight");
```

#### Waarom belangrijk?

Interactieve UI Dark mode Actieve menu's Accordion gedrag Animaties triggeren

Alles modern aan UI komt van **klassen schakelen**.

#### Demo: Highlight toggle + knop states

We bouwen:

- 3 boxen
- 1 knop: "Highlight wisselen"
- Klik = highlight CSS aan/uit

```
index.html
<div class="card shadow-sm mt-4">
  <div class="card-header bg-success text-white">
    Hoofdstuk 14: Klassen manipuleren
  </div>
  <div class="card-body">

    <button id="cl_toggle" class="btn btn-success w-100 mb-3">
      Toggle Highlight
    </button>
```

```

<div class="d-flex gap-2">
    <div class="cl-box p-3 border flex-fill text-center bg-light">Box
1</div>
    <div class="cl-box p-3 border flex-fill text-center bg-light">Box
2</div>
    <div class="cl-box p-3 border flex-fill text-center bg-light">Box
3</div>
</div>

</div>
</div>

CSS (voeg toe aan styles.scss onderaan)
.highlight {
    background: gold;
    font-weight: bold;
    transition: 0.2s ease;
}

main.js
// -----
// Hoofdstuk 14: classList
// -----


function toggleHighlight() {
    const boxes = document.querySelectorAll(".cl-box");
    boxes.forEach(box => box.classList.toggle("highlight"));
}

document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("cl_toggle")
        ?.addEventListener("click", toggleHighlight);
});

```

Je hebt nu **elementen die reageren als echte UI widgets**

## Mini-opdracht

Maak een **Dark / Light Theme toggle**.

### UI

- 1 knop: "Dark mode"
- Wanneer actief:
  - body krijgt klasse dark-mode
  - knoptekst verandert naar "Light mode"
  - alert toont status

### Verplichte ID's

dm2\_btn  
dm2\_status

### CSS toevoegen

```
.dark-mode {  
    background: #111 !important;  
    color: #fff !important;  
}
```

## Mini-opdracht oplossing

### HTML

```
<div class="card shadow-sm mt-4">  
    <div class="card-header bg-dark text-white">Mini-opdracht: Dark Mode  
    Toggle</div>  
    <div class="card-body">  
  
        <button id="dm2_btn" class="btn btn-dark w-100 mb-2">Dark mode  
        ☽</button>  
  
        <div id="dm2_status" class="alert alert-secondary mb-0">Light mode actief  
        ☀</div>  
  
    </div>  
</div>
```

### JS

```
function toggleDarkMode() {  
    const body = document.body;  
    const btn = document.getElementById("dm2_btn");  
    const status = document.getElementById("dm2_status");  
  
    const isDark = body.classList.toggle("dark-mode");  
  
    if (isDark) {  
        btn.textContent = "Light mode ☀";  
        status.className = "alert alert-success mb-0";  
        status.textContent = "Dark mode actief ☽";  
    } else {
```

```

        btn.textContent = "Dark mode 🌙";
        status.className = "alert alert-secondary mb-0";
        status.textContent = "Light mode actief ☀️";
    }
}

document.addEventListener("DOMContentLoaded", () => {
    document.getElementById("dm2_btn")
        ?.addEventListener("click", toggleDarkMode);
});

```

Klasse toggle Dynamische tekst UI state management Realtime styling

### Samenvatting

Methode	Action
.classList.add()	zet CSS klasse
.classList.remove()	verwijderd CSS klasse
.classList.toggle()	schakel aan/uit

Jij kan nu **state, styling en interactie combineren**. Dit is hoe je webapps laat voelen als apps, niet documenten.

## Hoofdstuk 15: Events & Formulieren

### Leerdoelen

Na dit hoofdstuk kan je:

- addEventListener correct gebruiken
- click-, input- en submit-events verwerken
- formulieren uitlezen en valideren
- foutmeldingen tonen met Bootstrap alerts
- dynamische feedback geven tijdens typen
- HTML formuliergedrag begrijpen

### Theorie: Wat is een Event?

Een **event** is iets dat gebeurt in de browser:

Gebeurtenis	Voorbeeld
Click	gebruiker klikt knop
Input	gebruiker typt in veld
Submit	formulier wordt verstuurd
Mouseover	muis beweegt over element
Focus / Blur	veld wordt actief/inactief

JS luistert naar die gebeurtenissen. Wanneer iets gebeurt, **voeren we een functie uit**.