

Blok 8 - OOP

- [Prompt Engineering 4](#)
- [Voorbeeld advanced Prompting](#)
- [PHP 3 - Debuggen](#)

Prompt Engineering 4

ToDo advanced oefeningen

Dynamic Prompting

Let AI create prompts

Let AI ask you questions

Voorbeeld advanced Prompting

De Tree of Thought (ToT)-methode

(eenvoudige versie)

Ik wil eenvoudig MVC PHP framework maken. Als voorbeeld maak je een controller, een view en een model. De database die je kan gebruiken is bijgevoegd.

Stap 1: Genereer ten minste drie logische denkstappen die mogelijke richtingen verkennen.

stap 2: Evalueer die denkstappen.

Er worden drie manieren van een aanpak van dit probleem bedacht en elke manier wordt beoordeeld op sterke- en zwakke punten.

<deel van het antwoord weggehaald>

Richting	Sterke punten	Zwakke punten	Beoordeling
A: Kernstructuur	Zorgt voor een goed georganiseerde basis en scheidt verantwoordelijkheden netjes.	Mogelijk over-engineered voor een eerste voorbeeld; vertraagt de “werkende demo”.	Hoog – goed als je het framework later wilt uitbreiden.
B: Voorbeeldgericht	Levert snel zichtbare resultaten; handig voor leren of demonstratie.	Kan leiden tot spaghetti-architectuur als je niet oplet; framework groeit ad hoc.	Midden – goed voor snelle start, minder voor schaalbaarheid.
C: Databasegericht	Helpt bij correcte datamodellering; logisch als data centraal staat.	Makkelijk om te verzanden in datadetails voordat de flow duidelijk is.	Laag-Midden – bruikbaar als datamodellering complex is, anders omslachtig.

PHP 3 - Debuggen

1 Foutafhandeling en Basis Debugging in PHP

Leerdoelen

- Je weet wat de verschillende typen PHP-fouten zijn (`Errors`, `Warnings`, `Notices`).
- Je kunt foutmeldingen in PHP interpreteren.
- Je kunt foutweergave instellen met `error_reporting()` en `ini_set()`.
- Je kunt basisprincipes van foutafhandeling toepassen (zoals `isset()`, `empty()`, `file_exists()`).
- Je gebruikt debuggingtechnieken zoals `echo`, `print_r()` en `var_dump()`.

Uitleg

Soorten fouten in PHP

- **Parse error:** fout in de code (bijvoorbeeld vergeten ; of haakje).
- **Fatal error:** code probeert iets wat niet kan, bijvoorbeeld een functie die niet bestaat.
- **Warning:** foutmelding, maar de code gaat door.
- **Notice:** melding van iets wat waarschijnlijk fout is (bijv. niet bestaande variabele).

⚙️ Foutmeldingen tonen of verbergen

```
<?php
// Toon alle fouten
ini_set("display_errors", 1);
error_reporting(E_ALL);
```

```
?>
```

⚠ Op een live website wil je foutmeldingen verbergen voor gebruikers. Gebruik dan:

```
ini_set("display_errors", 0);
```

▣ Veelgebruikte foutafhandlingstechnieken

- `isset($variabele)` – controleert of een variabele bestaat.
- `empty($variabele)` – controleert of een variabele leeg is.
- `file_exists("bestand.txt")` – controleert of een bestand bestaat.
- `die("Foutmelding")` of `exit()` – stopt het script bij een ernstige fout.

▣ Debuggen zonder debugger

- `echo` – handig om waardes snel te tonen.
- `print_r($array)` – toont de inhoud van een array of object.
- `var_dump($variabele)` – toont type + waarde (ook handig bij fouten met getallen).

▣ Opdracht 1 – Debugging oefenen

1. Maak een PHP-bestand genaamd `korting.php` met deze functie:

```
<?php
function berekenKorting($prijs, $korting) {
    return $prijs - $prijs * $korting / 100;
}
echo berekenKorting(100); // Fout! Tweede parameter ontbreekt
?>
```

2. Voeg foutmeldingen toe via `error_reporting()` zodat je de waarschuwing ziet.
3. Gebruik `isset()` en `empty()` om de fout af te vangen.
4. Test wat er gebeurt als je een variabele gebruikt die niet bestaat.
5. Gebruik `var_dump()` en `echo` om te zien wat je fout doet.

▣ Reflectie

- Wat is het verschil tussen een warning en een fatal error?
- Waarom is het handig om fouten wel te tonen in de ontwikkelfase, maar niet op een live website?
- Welke debuggingtechniek vond je het meest bruikbaar?

☐☐ Inleveren

- Lever het bestand `korting.php` in.
- Lever een korte uitleg in (.txt of .pdf) waarin je aangeeft:
 - Welke fout(en) je hebt gevonden
 - Welke techniek je gebruikte om het op te lossen
 - Wat je ervan geleerd hebt

2 Bestanden en Loggen

☐☐ Leerdoelen

- Je kunt bestanden aanmaken, lezen, schrijven en toevoegen met PHP.
- Je weet wanneer het handig is om gegevens in bestanden op te slaan in plaats van een database te gebruiken.
- Je begrijpt wat logbestanden zijn en waarom ze nuttig zijn.
- Je kunt een eenvoudig logsysteem implementeren dat fouten of gebeurtenissen opslaat.

☐☐ Uitleg

☐☐ Waarom bestanden gebruiken?

Bestanden kunnen handig zijn voor eenvoudige opslag zoals instellingen, bezoekerslogs of tijdelijke data. In kleine projecten is dit vaak eenvoudiger dan een database.

☐☐ Bestandsfuncties in PHP

- `file_put_contents("bestand.txt", "tekst")` – schrijft tekst naar bestand (overschrijft).
- `file_get_contents("bestand.txt")` – leest hele bestand in één keer.
- `fopen()` + `fwrite()` – uitgebreidere controle (bijv. toevoegen).
- `file_exists("bestand.txt")` – controleert of bestand bestaat.
- `fclose()` – sluit een bestand (nodig bij `fopen()`).

☐ Loggen: fouten en gebeurtenissen bijhouden

Een logbestand houdt bij wat er gebeurt in een script. Bijvoorbeeld foutmeldingen of bezoekersactiviteit.

Voorbeeld – logregel opslaan:

```
<?php
$melding = "[" . date("Y-m-d H:i:s") . "] Foutmelding\n";
file_put_contents("error.log", $melding, FILE_APPEND);
?>
```

`FILE_APPEND` zorgt dat de nieuwe regel onderaan toegevoegd wordt in plaats van het bestand te overschrijven.

☐ Opdracht 1 – bezoekersteller.php

1. Maak een nieuw bestand `bezoekersteller.php`.
2. Laat het script een teller bijhouden in `teller.txt`:
 - Bestaat het bestand nog niet? Begin bij 1.
 - Zo niet? Lees het getal in, verhoog met 1, en schrijf het terug.
3. Toon het aantal bezoeken op het scherm.

☐ Opdracht 2 – gastenboek.php (uitbreiding)

1. Maak een script waarin een gebruiker een bericht kan achterlaten via een formulier.
2. Sla elk bericht op in `gastenboek.txt` met datum/tijd.

3. Toon de laatste 5 berichten boven het formulier.
4. Gebruik `htmlspecialchars()` om invoer veilig weer te geven.

☐☐ Reflectie

- Wat zijn de voor- en nadelen van gegevens opslaan in een bestand ten opzichte van een database?
- Wat gebeurt er als je `file_put_contents()` gebruikt zonder `FILE_APPEND`?
- Waarom is loggen belangrijk, zelfs in kleine projecten?

☐☐ Inleveren

- Lever je bestanden `bezoekersteller.php` en/of `gastenboek.php` in.
- Lever een reflectieverslag in (.txt of .pdf).

3 Geavanceerde Functies, Abstractie en Modulaire Code

☐☐ Doelstellingen

- Je begrijpt waarom functies handig zijn (herbruikbaarheid, leesbaarheid, structuur).
- Je kunt functies gebruiken om complexe taken te verbergen (abstractie).
- Je snapt het verschil tussen globale en lokale variabelen (scope).
- Je kunt functies opslaan in aparte bestanden en deze inladen met `include()` of `require()`.

☐☐ Uitleg

☐☐ Waarom functies gebruiken?

- Je hoeft een stuk code maar één keer te schrijven.

- Je kunt de code op meerdere plekken gebruiken.
- Je maakt je code overzichtelijker en makkelijker te begrijpen.

☐ Abstractie: de details verbergen

Een functie kan iets ingewikkelds doen, zonder dat je telkens opnieuw de details hoeft te schrijven.

```
function berekenKorting($bedrag, $percentage) {  
    return $bedrag - ($bedrag * $percentage / 100);  
}  
  
echo berekenKorting(100, 10); // → 90
```

Je hoeft niet telkens opnieuw de hele berekening te typen. De functie “verbergt” die logica.

☐ Variable scope (bereik van een variabele)

- Variabelen binnen een functie zijn **lokaal**: ze bestaan alleen daarbinnen.
- Variabelen buiten functies zijn **globaal**.
- Je kunt met `global` een globale variabele binnen een functie gebruiken, maar dat is niet altijd wenselijk.

```
$x = 5;  
  
function testScope() {  
    $x = 10;  
    echo $x; // Toont 10, NIET 5  
}  
  
testScope();  
echo $x; // Toont nog steeds 5
```

☐ Modulaire code met `include()`

Je kunt functies in een apart bestand zetten, bijvoorbeeld `utils.php`, en die inladen in andere bestanden.

```
// Bestand: utils.php  
function toonWelkomstbericht($naam) {  
    echo "<p>Welkom, $naam!</p>";  
}
```

```
// Bestand: index.php
include("utils.php");
toonWelkomstbericht("Ali");
```

Je kunt ook meerdere functies in één bestand zetten, zoals:

- `validation.php`: functies voor formuliercontrole
- `format.php`: functies voor tekst- of getalopmaak

Opdracht – utils.php gebruiken

1. Maak een nieuw bestand `utils.php` en schrijf daarin minstens 3 functies:
 - `formatteerBedrag($bedrag)` → toont bijv. "€ 12,50"
 - `valideerInvoer($waarde)` → controleert of de waarde niet leeg is
 - `toonWelkomstbericht($naam)` → toont HTML met de naam
2. Maak een ander bestand `test_utils.php` waarin je `include("utils.php")` gebruikt en de functies test.
3. Gebruik `echo`, `print_r()` en `var_dump()` om het resultaat van de functies te tonen.

Reflectie

- Wat zijn de voordelen van functies in aparte bestanden bewaren?
- Welke functie vond je het handigst om te schrijven en waarom?
- Wat heb je geleerd over het verschil tussen globale en lokale variabelen?

Inleveren

- Lever de bestanden `utils.php` en `test_utils.php` in.
- Lever een reflectiedocument in (.txt of .pdf) waarin je uitlegt wat je hebt gedaan en geleerd.

4 Form Validatie en Beveiliging

Leerdoelen

- Je kunt formulierinvoer controleren op geldigheid.
- Je kent de risico's van onveilige invoer (zoals XSS).
- Je kunt invoer ontsnappen met `htmlspecialchars()`.
- Je gebruikt functies als `isset()`, `empty()` en `trim()` om invoer te valideren.

Uitleg

Waarom valideren?

Gebruikers maken fouten. Hackers doen het expres. Daarom controleer je altijd of een formulier goed is ingevuld voordat je ermee werkt.

Veelvoorkomende gevaren

- **XSS (Cross-Site Scripting):** iemand vult HTML of JavaScript in een formulier in, dat op jouw pagina wordt getoond.
- **Lege velden:** de gebruiker vergeet iets in te vullen.
- **Verkeerde types:** een tekst waar een getal moet staan.

Handige functies voor validatie

- `isset($_POST["naam"])` – is het veld verzonden?
- `empty($_POST["naam"])` – is het veld leeg?
- `trim()` – verwijdert spaties voor en na de invoer.
- `htmlspecialchars()` – maakt HTML onschadelijk.

```
<?php
if ($_SERVER["REQUEST_METHOD"] === "POST") {
    $naam = trim($_POST["naam"]);
    if (empty($naam)) {
        echo "Vul je naam in.";
    } else {
        echo "Welkom, " . htmlspecialchars($naam);
    }
}
```

```
}  
?>
```

☐☐ Simpel formulier

```
<form method="post">  
  <input type="text" name="naam" placeholder="Je naam">  
  <input type="submit" value="Verstuur">  
</form>
```

☐☐ Opdracht 1 – formulier_validatie.php

1. Maak een PHP-bestand met een formulier waarin je de gebruiker vraagt om:
 - Naam
 - Leeftijd
 - Bericht
2. Controleer of alle velden zijn ingevuld. Toon een foutmelding als iets ontbreekt.
3. Gebruik `htmlspecialchars()` om het bericht veilig weer te geven.
4. Gebruik `is_numeric()` om te controleren of de leeftijd een getal is.

Extra (optioneel)

- Maak je foutmeldingen visueel opvallend met HTML (bijv. ``).

☐☐ Reflectie

- Waarom is het belangrijk om input van gebruikers altijd te controleren?
- Wat zou er kunnen gebeuren als je `htmlspecialchars()` niet gebruikt?
- Welke fouten kwamen er tijdens het testen naar boven?

☐☐ Inleveren

- Lever je bestand `formulier_validatie.php` in (.php).

- Lever een reflectiedocument in (.txt of .pdf) met jouw antwoorden.

5 Superglobals en Associatieve Arrays

Leerdoelen

- Je begrijpt wat superglobals zijn in PHP (`$_GET`, `$_POST`, `$_SESSION`, `$_FILES`).
- Je weet wat een associatieve array is en hoe je ermee werkt.
- Je kunt formulierdata opslaan en verwerken met behulp van superglobals.

Uitleg

Wat zijn superglobals?

Superglobals zijn ingebouwde variabelen in PHP die overal beschikbaar zijn. Je gebruikt ze bijvoorbeeld om gegevens uit een formulier op te halen.

- `$_POST` – bevat formulierdata die via POST is verzonden
- `$_GET` – bevat data uit de URL (bijv. `?pagina=contact`)
- `$_SESSION` – bevat gegevens die je wilt onthouden tussen pagina's
- `$_FILES` – bevat geüploade bestanden

Associatieve arrays

Een associatieve array is een array met 'sleutels' in plaats van indexnummers:

```
$persoon = [  
    "naam" => "Ali",  
    "leeftijd" => 19,  
    "email" => "ali@example.com"  
];  
  
echo $persoon["naam"]; // Toont: Ali
```

Formulieren leveren ook associatieve arrays op via `$_POST`:

```
echo $_POST["naam"];
```

☐☐ Opdracht 1 – formulier_superglobals.php

1. Maak een formulier met de volgende velden:

- Naam
- E-mail
- Bericht

2. Verwerk de invoer met `$_POST` en sla de gegevens op in een associatieve array:

```
$data = [  
    "naam" => $_POST["naam"],  
    "email" => $_POST["email"],  
    "bericht" => $_POST["bericht"]  
];
```

3. Toon de waarden netjes op het scherm met `htmlspecialchars()`.

4. Gebruik eventueel `print_r($data)` om de hele array te tonen voor debugging.

☐☐ Reflectie

- Wat is het verschil tussen een normale array en een associatieve array?
- Waarom is `$_POST` eigenlijk een associatieve array?
- Wanneer zou je liever `$_GET` gebruiken in plaats van `$_POST`?

☐☐ Inleveren

- Lever je bestand `formulier_superglobals.php` in (.php).
- Lever een reflectiedocument in (.txt of .pdf) met je antwoorden.

6 Sessies en Cookies

Leerdoelen

- Je begrijpt wat sessies en cookies zijn.
- Je kunt sessies gebruiken om gegevens tijdelijk op te slaan voor een gebruiker.
- Je kunt cookies aanmaken en uitlezen met PHP.
- Je kunt de juiste techniek kiezen om gebruikersgegevens te bewaren.

Uitleg

Wat is een sessie?

Een sessie is een manier om informatie te onthouden zolang een gebruiker actief is op de website. Bijvoorbeeld: je logt in en blijft ingelogd op alle pagina's.

Een sessie start je met:

```
<?php
session_start();
$_SESSION["naam"] = "Ali";
echo $_SESSION["naam"];
?>
```

Je kunt sessiegegevens gebruiken zolang de browser open is (of tot je ze verwijdert met `session_destroy()`).

Wat is een cookie?

Een cookie wordt opgeslagen in de browser van de gebruiker, meestal voor een langere tijd. Handig om voorkeuren of gebruikersgegevens te onthouden tussen bezoeken.

```
// Cookie instellen
setcookie("taal", "NL", time() + 3600); // 1 uur geldig

// Cookie uitlezen
echo $_COOKIE["taal"];
```

☐ Verschillen tussen sessies en cookies

Kenmerk	Sessie	Cookie
Opslaglocatie	Server	Browser van gebruiker
Levensduur	Tijdelijk (tot de browser sluit of je het wist)	In te stellen (bijv. 1 uur, 30 dagen)
Toepassing	Ingelogde gebruiker, winkelmandje	Voorkeuren, laatst bezochte pagina

☐ Opdracht 1 – sessie_en_cookie.php

1. Maak een pagina waar de gebruiker zijn naam kan invoeren via een formulier.
2. Na verzenden:
 - Sla de naam op in een **sessie** én in een **cookie** (1 uur).
 - Toon op de pagina: “Welkom terug, [naam]!” als de gebruiker opnieuw langskomt.
3. Laat ook zien wat er gebeurt als de gebruiker het formulier overslaat.
4. Gebruik `session_start()` bovenaan het script.

Extra (optioneel)

- Voeg een link toe die de sessie wist (`session_destroy()`).

☐ Reflectie

- Wanneer kies je voor een sessie? En wanneer voor een cookie?
- Wat gebeurt er als je `session_start()` vergeet?
- Waar moet je op letten bij het gebruik van cookies met privacy?

☐ Inleveren

- Lever het bestand `sessie_en_cookie.php` in (.php).
- Lever een reflectiedocument in (.txt of .pdf) met je antwoorden.

7 State en Bestandgebaseerde 'Database'

Leerdoelen

- Je begrijpt het concept van 'state' in webapplicaties.
- Je kunt gebruikersgegevens opslaan in een tekstbestand.
- Je kunt meerdere records opslaan als gestructureerde regels (bijv. JSON of CSV).
- Je kunt gegevens uit zo'n bestand inlezen en tonen aan de gebruiker.

Uitleg

Wat is 'state'?

Een webpagina 'vergeet' wat er net gebeurd is zodra je hem ververs. Daarom moet je zelf bijhouden wat de toestand (state) is van je applicatie.

Je kunt dat doen met sessies, cookies of door gegevens op te slaan in een bestand of database.

Bestanden als 'mini-database'

In plaats van een echte database zoals MySQL, kun je voor simpele toepassingen gegevens bewaren in een tekstbestand.

Bijvoorbeeld: als iemand een bericht achterlaat in een formulier, voeg je dat toe aan `data.txt`.

Structuur

- Elk bericht komt op een eigen regel
- Of je gebruikt een JSON-array, met meerdere objecten erin
- Of je gebruikt CSV (waardes gescheiden met komma's)

```
// Voeg toe aan tekstbestand
$bericht = htmlspecialchars($_POST["bericht"]);
```

```
file_put_contents("data.txt", $bericht . "\n", FILE_APPEND);

// Lees het bestand
$inhoud = file("data.txt");
foreach ($inhoud as $regel) {
    echo "<p>" . trim($regel) . "</p>";
}
```

Opdracht 1 – gastenboek_met_bestand.php

1. Maak een formulier waarin iemand een naam en een bericht kan achterlaten.
2. Als de gebruiker op "Verstuur" klikt:
 - Sla het bericht op in een bestand `gastenboek.txt`.
 - Voeg ook datum/tijd toe met `date()`.
3. Toon de laatste 5 berichten onder het formulier.
4. Zorg dat de HTML veilig blijft via `htmlspecialchars()`.

Extra (optioneel)

- Laat de berichten in omgekeerde volgorde zien (nieuwste bovenaan).
- Sla de data op in JSON-formaat i.p.v. tekstregels.

Reflectie

- Wat zijn de voordelen van een tekstbestand boven een database?
- Wanneer loop je tegen de beperkingen aan?
- Hoe zou je dit uitbreiden zodat iemand ook berichten kan verwijderen?

Inleveren

- Lever je bestand `gastenboek_met_bestand.php` in (.php).

- Lever het bestand `gastenboek.txt` mee met minimaal 3 testberichten.
- Lever een reflectie in (.txt of .pdf).

8 Inleiding tot PHP Include-logica en Templatebestanden

Leerdoelen

- Je begrijpt het nut van het opdelen van HTML/PHP-bestanden in herbruikbare componenten.
- Je kunt `include` en `require` gebruiken om logica en opmaak te splitsen.
- Je kunt een eenvoudige template-structuur bouwen (header, content, footer).

Uitleg

Waarom opdelen in componenten?

Als je meerdere pagina's hebt met dezelfde header of footer, is het onhandig om die steeds te kopiëren. Daarom gebruik je `include()` of `require()` om ze in te laden.

```
<?php include 'header.php'; ?>

<h1>Welkom op de homepage</h1>

<?php include 'footer.php'; ?>
```

Verschil tussen include en require

- `include()`: Laadt het bestand in. Als het bestand niet bestaat, gaat de rest van de pagina gewoon door.
- `require()`: Laadt ook het bestand in, maar als het niet bestaat, stopt de pagina met een foutmelding.

Voorbeeldstructuur

- `header.php` – HTML <head>, navigatie
- `footer.php` – Copyright info, afsluitende HTML
- `index.php` – Hoofdpagina die alles samenvoegt

☐☐ Opdracht 1 – templatestructuur

1. Maak drie bestanden:
 - `header.php`: bevat een <header> met de titel van je site en een eenvoudige navigatie (bijv. naar home en contact).
 - `footer.php`: bevat een <footer> met een copyrightregel.
 - `index.php`: bevat de pagina-inhoud en gebruikt `include()` om de header en footer in te laden.
2. Zorg dat alles er netjes uitziet (je mag CSS toevoegen).
3. Test wat er gebeurt als je `include("footer.php")` verandert in `require("footer.php")` en het bestand bestaat niet.

☐☐ Reflectie

- Waarom is het handig om componenten zoals header/footer apart te houden?
- Wat is het risico als je alles in één bestand zou houden?
- Wanneer zou je `require()` verkiezen boven `include()`?

☐☐ Inleveren

- Lever `index.php`, `header.php` en `footer.php` in.
- Lever een reflectiedocument in (.txt of .pdf) met je antwoorden.

9 Validatie en Veiligheid bij Formulieren

Leerdoelen

- Je begrijpt waarom inputvalidatie belangrijk is.
- Je kunt formulierinvoer controleren met PHP (server-side).
- Je kent de risico's van onveilige invoer (bijv. XSS, SQL-injectie).
- Je kunt input veilig maken met functies zoals `htmlspecialchars()` en `filter_var()`.

Uitleg

Waarom valideren?

Gebruikers kunnen fouten maken (of expres verkeerde dingen invullen). Als je invoer niet controleert, kan dat leiden tot bugs of zelfs beveiligingsproblemen.

Soorten validatie

- **Client-side:** Met HTML of JavaScript (bijv. `required`, `type="email"`).
- **Server-side:** Met PHP (altijd nodig, want je kunt client-side validatie omzeilen).

Veiligheid: wat kan er misgaan?

- **XSS (Cross-Site Scripting):** Als je niet ontsnapte HTML toont, kan iemand scripts injecteren.
- **SQL-injectie:** Als je invoer rechtstreeks in een query zet (komt later aan bod).

Voorbeeld inputcontrole

```
<?php
$naam = "";
$fout = "";

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    if (empty($_POST["naam"])) {
        $fout = "Naam is verplicht!";
    } else {
        $naam = htmlspecialchars($_POST["naam"]);
    }
}
```

```
}  
}  
?>  
  
<form method="post">  
  Naam: <input type="text" name="naam">  
  <input type="submit" value="Verstuur">  
</form>  
  
<?php  
if ($fout) {  
  echo "<p style='color:red;'>$fout</p>";  
} else if ($naam) {  
  echo "<p>Welkom $naam</p>";  
}  
?>
```

☐☐ Opdracht 1 – formuliercontrole

1. Maak een formulier met de volgende velden:
 - Naam (verplicht)
 - Email (verplicht, geldig e-mailadres)
 - Bericht (optioneel, max. 200 tekens)
2. Controleer de invoer met PHP:
 - Laat foutmeldingen zien als iets ontbreekt of fout is.
 - Gebruik `filter_var()` om het e-mailadres te controleren.
 - Gebruik `htmlspecialchars()` om veilige output te tonen.
3. Toon een nette samenvatting van de ingevulde gegevens als alles goed is.

Extra (optioneel)

- Voeg visuele validatie toe met CSS-klassen (bijv. rode rand bij fout).
- Voeg een resetknop toe aan het formulier.

☐☐ Reflectie

- Wat gebeurt er als je geen server-side validatie gebruikt?
- Wat is het verschil tussen validatie en het veilig maken van input?
- Hoe zou je dit formulier uitbreiden voor een login- of registratiepagina?

Inleveren

- Lever je formulierpagina in als `formulier_validatie.php`.
- Lever een reflectiedocument in (.txt of .pdf).