

OOP

Auto generated, moet in zijn geheel worden gecheckt

1 Wat is OOP en waarom gebruiken we het?

□□ Leerdoelen

- Je begrijpt het verschil tussen procedureel programmeren en OOP.
- Je weet wat een klasse, object en methode zijn.
- Je kunt herkennen wanneer OOP handig is in grotere projecten.

□□ Uitleg

Wat is OOP?

OOP staat voor **Objectgeoriënteerd programmeren**. In plaats van losse functies en variabelen, werk je met objecten die gedrag en gegevens combineren.

Waarom OOP gebruiken?

- Je kunt code beter organiseren en hergebruiken.
- Je maakt je programma makkelijker uitbreidbaar.
- Je kunt complexe systemen opdelen in logische blokken (objecten).

Voorbeeld: procedureel vs OOP

Procedureel:

```
<?php
$naam = "Tessa";
$leeftijd = 19;

function begroet($naam) {
    echo "Hallo, ik ben " . $naam;
}

begroet($naam);
?>
```

Objectgeoriënteerd:

```
<?php
class Persoon {
    public $naam;
    public $leeftijd;

    public function begroet() {
        echo "Hallo, ik ben " . $this->naam;
    }
}

$tessa = new Persoon();
$tessa->naam = "Tessa";
$tessa->leeftijd = 19;
$tessa->begroet();
?>
```

Opdracht – Van procedureel naar objectgeoriënteerd

Je gaat een simpel PHP-script omzetten naar OOP.

1. Maak een bestand `mijn-procedureel.php` en schrijf de volgende code:

```
<?php
$autoMerk = "Toyota";
```

```
$bouwjaar = 2018;

function toonAuto($merk, $jaar) {
    echo "Dit is een $merk uit $jaar";
}

toonAuto($autoMerk, $bouwjaar);

?>
```

2. Maak nu een tweede bestand: `mijn-oop.php` en herschrijf dit script in OOP-stijl:

```
<?php
class Auto {
    public $merk;
    public $bouwjaar;

    public function toon() {
        echo "Dit is een $this->merk uit $this->bouwjaar";
    }
}

$auto = new Auto();
$auto->merk = "Toyota";
$auto->bouwjaar = 2018;
$auto->toon();

?>
```

☐☐ Reflectie

- Wat vond je makkelijker: de procedurele of de OOP-versie?
- Waarom denk je dat OOP belangrijk is bij grotere projecten?
- Kun je zelf een ander voorbeeld bedenken waarbij OOP handig zou zijn?

☐☐ Inleveren

- Lever beide bestanden in: `mijn-procedureel.php` en `mijn-oop.php`

- Maak een kort reflectiebestand: `reflectie-les1-<jouw-naam>.txt`
- Let op: je code moet uitvoerbaar zijn in de browser (via XAMPP of localhost).

2 Klassen en objecten

□□ Leerdoelen

- Je weet wat een klasse en een object zijn.
- Je kunt een eenvoudige klasse maken in PHP.
- Je kunt een object aanmaken op basis van die klasse en ermee werken.

□□ Uitleg

Wat is een klasse?

Een klasse is een soort blauwdruk of bouwtekening. Je beschrijft hiermee hoe een bepaald type object eruitziet en wat het kan doen.

Wat is een object?

Een object is een concreet exemplaar van een klasse. Je maakt een object aan met `new`. Een klasse kun je meerdere keren gebruiken om verschillende objecten te maken.

Class, Object, Properties en Methods worden nog een keer duidelijk gemaakt in:

<https://slides.com/jamescandan/oop-php-for-beginners-1-2/fullscreen#/5>

<https://www.youtube.com/embed/tZJcsFp9ttU>

Dus een class is een sjabloon van een object en met het commando `new <class_naam>` maken we een nieuw object van de sjabloon. In een class heten de variabelen **properrties** en de fucnties heten **methods**.

Voorbeeld:

```
<?php
class Hond {
    public $naam;
    public $leeftijd;

    public function blaf() {
        echo "Woef! Ik ben " . $this->naam;
    }
}

$mijnHond = new Hond();
$mijnHond->naam = "Rex";
$mijnHond->leeftijd = 3;
$mijnHond->blaf();
?>
```

Opdracht – Maak je eigen klasse

1. Maak een nieuw bestand: `mijn-klas.php`
2. Maak hierin een klasse `Student` met de eigenschappen `voornaam`, `achternaam` en `opleiding`.
3. Maak een methode `stelVoor()` die zegt: "Hallo, ik ben [voornaam] [achternaam] en ik doe de opleiding [opleiding]."
4. Maak een object van de klasse en test het resultaat in je browser.

Reflectie

- Wat viel je op bij het maken van een object?
- Wat betekent `$this` precies in de methode?
- Wat gebeurt er als je een eigenschap niet vult voor je `stelVoor()` gebruikt?

Inleveren

- Bestandsnaam: `mijn-klas.php`

- Reflectievragen in een apart txt- of PDF-bestand: `reflectie-les2-<jouw-naam>.txt`

3 Constructors en eigenschappen

□□ Leerdoelen

- Je weet wat een constructor is in een klasse.
- Je kunt automatisch waarden toekennen bij het maken van een object.
- Je begrijpt het verschil tussen eigenschappen (properties) en methoden.

□□ Uitleg

Wat is een constructor?

Een constructor is een speciale methode die automatisch wordt uitgevoerd als je een object maakt. In PHP heet deze methode `__construct()`.

Waarom handig?

Met een constructor hoef je niet handmatig eigenschappen toe te wijzen nadat je een object hebt aangemaakt.

Voorbeeld zonder constructor:

```
<?php
class Auto {
    public $merk;
    public $bouwjaar;
}

$auto = new Auto();
$auto->merk = "Toyota";
$auto->bouwjaar = 2018;
?>
```

Voorbeeld met constructor:

```
<?php
class Auto {
    public $merk;
    public $bouwjaar;

    public function __construct($merk, $bouwjaar) {
        $this->merk = $merk;
        $this->bouwjaar = $bouwjaar;
    }
}

$auto = new Auto("Toyota", 2018);
?>
```

Opdracht – Constructor gebruiken

1. Maak een nieuw bestand: `les3-student.php`
2. Maak opnieuw een klasse `Student` met eigenschappen: `voornaam`, `achternaam` en `opleiding`.
3. Voeg een constructor toe waarin deze waarden worden ingesteld.
4. Maak een methode `stelVoor()` die de gegevens netjes toont.
5. Maak minstens twee objecten aan en laat de `stelVoor()`-methode zien in de browser.

Reflectie

- Wat is het voordeel van een constructor ten opzichte van handmatige toekenning?
- Wat gebeurt er als je het aantal parameters niet goed doorgeeft?
- Wat betekent `$this->...` in de constructor?

Inleveren

- Bestandsnaam: `les3-student.php`
- Reflectievragen in een apart bestand: `reflectie-les3-<jouw-naam>.txt`

4 Methoden met parameters en returnwaarden

□□ Leerdoelen

- Je begrijpt hoe je een methode maakt die parameters accepteert.
- Je kunt een methode schrijven die een waarde teruggeeft met `return`.
- Je ziet hoe objectmethoden kunnen samenwerken met parameters uit de code.

□□ Uitleg

Wat is een parameter?

Een parameter is een waarde die je doorgeeft aan een methode, zodat deze methode die waarde kan gebruiken in de berekening of actie.

Wat is een returnwaarde?

Met `return` geef je een resultaat terug uit een methode, zodat je dat resultaat ergens anders kunt gebruiken.

Voorbeeld:

```
<?php
class Rekening {
    public $saldo = 0;

    public function stort($bedrag) {
        $this->saldo += $bedrag;
    }

    public function toonSaldo() {
        return $this->saldo;
    }
}
```



```
$mijnRekening = new Rekening();  
$mijnRekening->stort(50);  
echo "Je saldo is: ". $mijnRekening->toonSaldo();  
?>
```

Opdracht – Werk met parameters en return

1. Maak een nieuw bestand: `les4-rekening.php`
2. Maak een klasse `Rekening` met een startsaldo van 0.
3. Voeg een methode `stort($bedrag)` toe die het bedrag bij het saldo optelt.
4. Voeg een methode `opname($bedrag)` toe die het bedrag van het saldo afhaalt.
5. Voeg een methode `toonSaldo()` toe die het saldo teruggeeft met `return`.
6. Test de klasse met een paar stortingen en opnames. Toon telkens het saldo in de browser.

Reflectie

- Wat gebeurt er als je een opname doet die groter is dan je saldo?
- Wat zijn de voordelen van werken met `return`?
- Kun je meerdere parameters doorgeven aan een methode? Probeer dit eens uit.

Inleveren

- Bestandsnaam: `les4-rekening.php`
- Reflectievragen in een apart bestand: `reflectie-les4-<jouw-naam>.txt`

5 Encapsulation en private properties

Leerdoelen

- Je weet wat encapsulation is.
- Je begrijpt het verschil tussen `public` en `private`.
- Je kunt `getters` en `setters` gebruiken om toegang tot eigenschappen te regelen.

Uitleg

Encapsulation betekent dat je data (eigenschappen) beschermt tegen direct aanpassen van buitenaf. In plaats daarvan gebruik je methoden om de data op een gecontroleerde manier te bekijken of aan te passen.

Waarom gebruiken?

Zo voorkom je fouten doordat andere delen van je code zomaar eigenschappen aanpassen zonder controle.

Voorbeeld:

```
<?php
class Bankrekening {
    private $saldo = 0;

    public function stort($bedrag) {
        if ($bedrag > 0) {
            $this->saldo += $bedrag;
        }
    }

    public function getSaldo() {
        return $this->saldo;
    }
}
```

```
}  
  
$rekening = new Bankrekening();  
$rekening->stort(100);  
echo "Saldo: ".$rekening->getSaldo();  
?>
```

Opdracht – Bescherm je data

1. Maak een nieuw bestand: `les5-bankrekening.php`
2. Maak een klasse `Bankrekening` met een `private` eigenschap `$saldo`.
3. Maak een `stort()`-methode die alleen positief bedrag accepteert.
4. Maak een `getSaldo()`-methode die het saldo teruggeeft.
5. Test de klasse met verschillende stortingen en probeer ook een negatieve storting (die moet worden geweigerd).

Reflectie

- Wat is het verschil tussen `public` en `private`?
- Waarom zou je eigenschappen `private` maken?
- Wat gebeurt er als je probeert `$saldo` direct aan te passen?

Inleveren

- Bestandsnaam: `les5-bankrekening.php`
- Reflectievragen in een apart bestand: `reflectie-les5-<jouw-naam>.txt`

6 Objecten in objecten (Samenstelling)

Leerdoelen

- Je leert hoe objecten eigenschappen van andere objecten kunnen bevatten.
- Je begrijpt hoe objecten kunnen samenwerken.
- Je kunt objecten combineren in een programma.

Uitleg

In OOP kun je een object gebruiken als eigenschap van een ander object. Dat noemen we "compositie" of "samenstelling". Zo kun je complexe structuren maken.

Voorbeeld:

```
<?php
class Adres {
    public $straat;
    public $huisnummer;

    public function __construct($straat, $huisnummer) {
        $this->straat = $straat;
        $this->huisnummer = $huisnummer;
    }
}

class Persoon {
    public $naam;
    public $adres;

    public function __construct($naam, $adres) {
        $this->naam = $naam;
        $this->adres = $adres;
    }

    public function toonInfo() {
        echo $this->naam . " woont op " . $this->adres->straat . " " . $this->adres->huisnummer;
    }
}
```

```
$mijnAdres = new Adres("Lindelaan", 12);  
$persoon = new Persoon("Ali", $mijnAdres);  
$persoon->toonInfo();  
?>
```

Opdracht – Objecten combineren

1. Maak een bestand `les6-persoonadres.php`.
2. Maak een klasse `Adres` met de eigenschappen `straat` en `huisnummer`.
3. Maak een klasse `Persoon` met `naam` en een eigenschap `adres` (die een `Adres`-object bevat).
4. Maak van beide klassen een object aan in je script, en toon de combinatie van gegevens.

Reflectie

- Wat zijn voordelen van het werken met samengestelde objecten?
- Hoe helpt dit om je code overzichtelijk te houden?

Inleveren

- Bestandsnaam: `les6-persoonadres.php`
- Reflectievragen in een apart bestand: `reflectie-les6-<jouw-naam>.txt`

7 Constructors en automatische initialisatie

Leerdoelen

- Je begrijpt wat een constructor is.
- Je kunt een constructor gebruiken om eigenschappen meteen in te stellen.
- Je kunt de constructor van samengestelde objecten gebruiken.

Uitleg

Een constructor is een speciale functie in een klasse die automatisch wordt uitgevoerd bij het aanmaken van een object. Hiermee kun je meteen eigenschappen instellen.

Voorbeeld:

```
<?php
class Student {
    public $naam;
    public $leeftijd;

    public function __construct($naam, $leeftijd) {
        $this->naam = $naam;
        $this->leeftijd = $leeftijd;
    }

    public function toon() {
        echo \"$this->naam is $this->leeftijd jaar oud.\";
    }
}

$student = new Student(\"Fatima\", 17);
$student->toon();
?>
```

Opdracht – Maak je eigen constructor

1. Maak een nieuw bestand `les7-student.php`.
2. Maak een klasse `Student` met de eigenschappen `naam` en `leeftijd`.
3. Voeg een constructor toe die deze eigenschappen invult.
4. Voeg een methode toe die de naam en leeftijd toont.

5. Maak een object aan met jouw gegevens en test het script.

☐☐ Reflectie

- Waarom is het handig dat de constructor automatisch wordt uitgevoerd?
- Hoe verschilt dit van handmatig eigenschappen instellen?

☐☐ Inleveren

- Bestandsnaam: `les7-student.php`
- Reflectievragen in een apart bestand: `reflectie-les7-<jouw-naam>.txt`

8 Klassen opdelen in bestanden (modulariteit)

☐☐ Leerdoelen

- Je leert waarom het handig is om klassen in aparte bestanden te zetten.
- Je leert hoe je bestanden met `require_once` of `include_once` laadt.
- Je maakt je eerste modulaire project.

☐☐ Uitleg

Als je meerdere klassen maakt in je project, wordt het onoverzichtelijk als je alles in één bestand zet. Daarom splits je de klassen op in losse bestanden. Je laadt die bestanden dan met `require_once` of `include_once` in je hoofdsript.

Voorbeeldstructuur:

```
projectmap/  
|- persoon.php  
|- adres.php
```

|- main.php

persoon.php

```
<?php
class Persoon {
    public $naam;
    public function __construct($naam) {
        $this->naam = $naam;
    }
}
```

adres.php

```
<?php
class Adres {
    public $straat;
    public function __construct($straat) {
        $this->straat = $straat;
    }
}
```

main.php

```
<?php
require_once 'persoon.php';
require_once 'adres.php';

$persoon = new Persoon("Ali");
$adres = new Adres("Lindelaan");

echo $persoon->naam . " woont op " . $adres->straat;
```

Opdracht – Bestanden scheiden

1. Maak een map aan genaamd `les8`.
2. Maak hierin drie bestanden: `main.php`, `auto.php` en `eigenaar.php`.
3. In `auto.php`: klasse `Auto` met merk en bouwjaar.

4. In `eigenaar.php`: klasse `Eigenaar` met naam.
5. In `main.php`: laad de twee andere bestanden, maak een object van elk, toon de info.

☐☐ Reflectie

- Wat zijn de voordelen van het opdelen van code?
- Wat is het verschil tussen `require_once` en `include_once`?

☐☐ Inleveren

- Zip de hele map `les8` met daarin alle bestanden.
- Reflectievragen in een apart tekstbestand: `reflectie-les8-<jouw-naam>.txt`

9 Overerving – Een klasse uitbreiden

☐☐ Leerdoelen

- Je begrijpt wat overerving is.
- Je kunt een subklasse maken die eigenschappen en methoden van een bovenliggende klasse overneemt.
- Je kunt eigen extra eigenschappen/methoden toevoegen aan een subklasse.

☐☐ Uitleg

Met overerving kun je een klasse maken die gebaseerd is op een andere klasse. Je gebruikt hiervoor het sleutelwoord `extends`.

Voorbeeld:

```

<?php
class Dier {
    public $soort;
    public function geluid() {
        echo "Dit dier maakt een geluid.";
    }
}

class Hond extends Dier {
    public function geluid() {
        echo "Woef!";
    }
}

$beest = new Hond();
$beest->geluid(); // toont: Woef!

?>

```

De klasse `Hond` heeft automatisch ook toegang tot alles van `Dier`, tenzij het wordt overschreven zoals hierboven bij `geluid()`.

Opdracht – Subklassen maken

1. Maak een bestand `voertuig.php` waarin een klasse `Voertuig` zit met de eigenschappen `merk` en `topsnelheid`, en een methode `toon()`.
2. Maak een bestand `auto.php` met een klasse `Auto` die `Voertuig` uitbreidt. Voeg een extra eigenschap `aantalDeuren` toe en pas de `toon()` methode aan.
3. Maak een bestand `main.php` waarin je beide bestanden laadt en een `Auto` object toont.

Reflectie

- Waarom is overerving handig?
- Wat gebeurt er als je een methode overschrijft in een subklasse?

Inleveren

- Zip de drie bestanden en lever in als `les9-<jouw-naam>.zip`
- Reflectievragen in een tekstbestand: `reflectie-les9-<jouw-naam>.txt`

10 Autoloading en namespaces

□□ Leerdoelen

- Je begrijpt het nut van autoloading.
- Je kunt een eenvoudige autoloader schrijven met `spl_autoload_register`.
- Je weet wat een namespace is en waarom je het gebruikt.

□□ Uitleg

Bij grotere projecten wil je niet elk PHP-bestand handmatig includen. Een autoloader zorgt ervoor dat klassen automatisch geladen worden zodra ze nodig zijn.

Voorbeeld autoloader:

```
<?php
spl_autoload_register(function ($class) {
    require_once $class . '.php';
});
```

Als je dan een object maakt van een klasse `Auto`, zoekt PHP automatisch naar `Auto.php`.

Namespaces

Bij grote projecten kunnen klassennamen botsen. Daarom gebruik je namespaces, zodat je klassen groepeert en naamconflicten voorkomt.

```
<?php
namespace MijnProject\Modellen;

class Gebruiker {
    // ...
}
```

In andere bestanden gebruik je de volledige naam of `use`:

```
use MijnProject\\Modellen\\Gebruiker;
```

Opdracht – Autoloader en namespaces

1. Maak een map `les10` met een submap `klassen`.
2. Maak in `klassen` een bestand `Gebruiker.php` met een klasse `Gebruiker` in de namespace `App\\Modellen`.
3. Maak in de hoofdmap een bestand `index.php` waarin je een autoloader schrijft.
4. Maak een object van `App\\Modellen\\Gebruiker` en laat iets zien via `echo`.

Reflectie

- Waarom zou je een autoloader gebruiken in plaats van `require_once`?
- Wat is het nut van een namespace?

Inleveren

- Zip je hele map `les10` en lever in als `les10-<jouw-naam>.zip`
- Voeg reflectie toe in `reflectie-les10-<jouw-naam>.txt`

11 Eindopdracht – Mini Project

Doel

Je laat zien dat je de belangrijkste concepten van OOP beheerst in PHP door een kleine, maar volledige objectgeoriënteerde toepassing te bouwen.

De opdracht

Maak een kleine toepassing waarin voertuigen beheerd worden. De toepassing werkt met drie klassen:

- **Voertuig** (basisklasse)
- **Auto** (subklasse van Voertuig)
- **Motor** (subklasse van Voertuig)

Je maakt een overzichtspagina waarop de voertuigen worden weergegeven. Elk voertuig toont informatie en gebruikt zijn eigen `toonInfo()`-methode. Gebruik namespaces en een autoloader.

☐☐ Benodigheden

1. Map `eindopdracht` met submap `classes`.
2. Klasse `Voertuig` met eigenschappen:
 - `merk`
 - `topsnelheid`

En methode `toonInfo()` die basisinfo toont.

3. Klasse `Auto` (extends `Voertuig`) met extra eigenschap `aantalDeuren`. Overschrijft `toonInfo()`.
4. Klasse `Motor` (extends `Voertuig`) met extra eigenschap `typeHelm`. Overschrijft `toonInfo()`.
5. Autoloader in `index.php`. Maak in dit bestand enkele objecten aan van `Auto` en `Motor`, en roep `toonInfo()` aan.
6. Gebruik namespaces in al je klassen.

☐☐ Reflectie

- Wat ging goed, wat vond je lastig?
- Wat zou je in een groter project nog verder willen leren?

☐☐ Inleveren

- Lever je map in als: `eindopdracht-oop-<jouw-naam>.zip`
 - Lever je reflectie in als: `reflectie-eindopdracht.txt`
-

Revision #6

Created 18 June 2025 14:33:53 by Max

Updated 22 June 2025 09:39:28 by Max