



Versie 2016-2017
Bouke Loman

Inhoud

1.	Variabelen.....	3
2.	\$_POST & \$_GET (formulieren)	4
2.1	\$_POST of \$_GET?	5
3.	IF-statement (beslissing).....	6
3.1	Operatoren	6
4.	Switch-statement.....	8
5.	Loops (herhalingen)	9
5.1	For-loop	9
5.2	While-loop	10
6.	Arrays.....	11
6.1	Standaard Array.....	11
6.2	Associatief Array.....	12
6.3	Multidimensionale Arrays.....	12
7.	Require & Include	14
8.	Functies.....	16
8.1	Functie parameters.....	17
8.2	Optionele parameters.....	17
8.3	Lokale vs globale variabelen	18
9.	Sessions	19
9.1	Sessions starten	19
9.2	Sessions vullen en uitlezen	20
9.3	Session verwijderen.....	20
9.4	Header refresh.....	20
10.	Extra informatie over time() en date().....	21
11.	Database connecties in PHP.....	22
11.1	Verbinding maken met PDO	22
11.2	Try-catch	23
11.3	Een Query?	23
11.4	Een query uitvoeren in PHP (Prepared statements)	24
11.5	Iets doen met het resultaat van een SELECT query	25
11.6	Gegevens in een database toevoegen	25
11.7	Gegevens uit de database wijzigen.....	26
11.8	Gegevens uit de database verwijderen	27
12.	PSD (programma structuur diagram)	28
12.1	Uitbreiden PSD's	29
12.2	Structorizer	30



1. Variabelen

Wanneer je gaat programmeren zal je veel te maken krijgen met variabelen. Wanneer je een waarde, bijvoorbeeld een getal of een stuk tekst, wil onthouden of manipuleren, dan doe je dit in een programmeertaal middels een variabele. Om dit makkelijk te kunnen onthouden heb ik hieronder een afbeelding wat dit illustreert.



```
//Bijbehorende PHP code  
$Getal = 15;
```

In principe kun je een variabele dus zien als een doosje met daarin een waarde. Die waarde kan van alles zijn en omdat het doosje open te maken is kan hetgeen er in zit worden veranderd. Dit is ook de reden dat het een variabele heet. Dat wat in het doosje zit kun je namelijk tijdens het programma veranderen.

Voorbeeld:

```
$Getal = 15;  
echo $Getal;
```

in dit geval komt het getal 15 op het scherm te staan.

3

Voorbeeld:

```
$Getal = 15;
```

```
//regels code ertussen
```

```
$Getal = 20; //Het doosje wordt hier geopend en de inhoud wordt gewisseld naar 20  
echo $Getal;
```

in dit geval komt het getal 20 op het scherm te staan. Dit omdat dit de laatste waarde is die we in de variabele (doosje) met de naam "Getal" hebben gestopt.



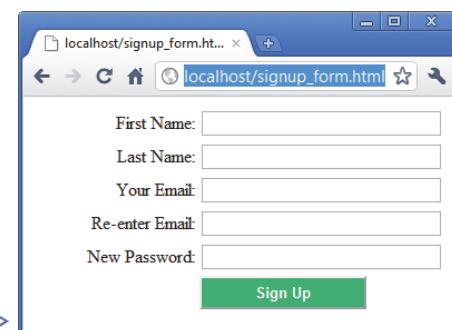
2. \$_POST & \$_GET (formulieren)

Eerder dit jaar hebben we in de cursus HTML geleerd hoe we formulieren kunnen maken binnen onze website. HTML is echter een opmaaktaal en we kwamen er dus al snel achter dat we niet veel met de gegevens uit de formulieren konden doen. PHP biedt echter uitkomst en kan op alle mogelijke manieren gegevens uit formulieren verwerken.

Om de door de gebruiker ingevulde gegevens op te vragen in PHP maakt PHP één van de twee SUPERGLOBAL variabelen `$_POST` of `$_GET` aan wanneer een formulier gesubmit wordt. Zowel de `$_POST` als `$_GET` variabele is van het datatype Array, waarbij de key-values de namen zijn van de velden uit het HTML formulier. Hieronder een schema om dit te verduidelijken:

Html + resultaat:

```
<label for="firstname">First Name:</label>
<input type="text" id="firstname" name="firstname" />
<label for="lastname">Last Name:</label>
<input type="text" id="lastname" name="lastname" />
<label for="email">Your Email:</label>
<input type="text" id="email" name="email" />
<label for="re-email">Re-enter Email:</label>
<input type="text" id="re-email" name="re-email" />
<label for="password">New Password:</label>
<input type="text" id="password" name="password" />
<input type="submit" name="verzenden" value="Sign Up" />
```



PHP code (inclusief controle):

```
//haalt de voornaam op
if(isset($_POST['firstname']))
    $firstname = $_POST['firstname'];
else
    $firstname = null;

//haalt de achternaam op
if(isset($_POST['lastname']))
    $lastname = $_POST['lastname'];
else
    $lastname = null;

//haalt het email adres op
if(isset($_POST['email']))
    $email = $_POST['email'];
else
    $email = null;

//haalt het Re-type Email adres op
if(isset($_POST['re-email']))
    $reEmail = $_POST['re-email'];
else
    $reEmail = null;

//haalt het password op
if(isset($_POST['password']))
    $password = $_POST['password'];
else
    $password = null;
```

4

Zoals je ziet corresponderen de namen van de HTML velden met de key-values in het `$_POST` Array.



Let op! Dit is net als alle andere zaken in PHP hoofdlettergevoelig. Het is dus belangrijk om secuur te werken anders krijg je gegarandeerd foutmeldingen. Voor de leesbaarheid van de code is het netjes om de waardes die je verkrijgt uit de `$_POST` of `$_GET` te stoppen in eigen variabelen, zoals ook in bovenstaand voorbeeld is gedaan. Deze variabelen kun je dan naar hartenlust binnen je code gebruiken.

2.1 `$_POST` of `$_GET`?

Wij hebben het steeds over 2 SUPERGLOBALS, `$_POST` en `$_GET` en we zien in het voorbeeld alleen `$_POST`. Wat is dan het verschil?

In de form tag van je HTML geef je achter de parameter “method=” aan of je POST of GET wilt gebruiken.

```
<form id="MijnForm" action="" method="post">
```

De meeste formulieren zullen via de POST methode worden verzonden omdat dit onzichtbaar is voor de gebruiker. Je ziet er niets van binnen de browser. Wanneer een site HTTPS gebruikt is het zelfs niet zichtbaar voor mogelijke hackers. Dit is gewenst omdat formulieren vaak privacy gevoelige informatie bevatten. Een nadeel hiervan is dat wanneer je je webpagina verstuurt de browser een pop-up toont om te vragen of de gegevens nogmaals verstuurd dienen te worden. In sommige situaties wil je dit niet omdat dit erg gebruiksonvriendelijk is. Goede voorbeelden hiervan zijn zoekvelden.

Wanneer een gebruiker een zoekterm invult en op zoeken drukt dan zal je merken dat er iets met de URL van de pagina is gebeurt. Wanneer je namelijk voor de GET methode kiest dan zullen de gegevens van je velden in de URL balk terugkomen. Dit is dus wel zichtbaar in je browser.

5

Achter de URL komt een “?” waarna de veldnamen en hun waardes worden opgesomd in het formaat:

?Veldnaam1=veldwaarde1&veldnaam2=veldwaarde2

Zie deze URL:



Wanneer je nu de pagina verstuurt zal deze meteen opnieuw herladen omdat hij de benodigde gegevens uit de URL haalt. Je kunt op deze manier ook links maken met daarin variabelen zonder dat hiervoor een formulier hoeft te worden ingevuld. Zie het voorbeeld hieronder:

```
<a href="index.php?pagina=10&gebruiker=henk">Voorbeeld</a>
```



3. IF-statement (beslissing)

Het leven zit vol beslissingen. Iedere dag krijgen we ermee te maken, de ene belangrijker dan de andere. Veelal maak je je beslissingen aan de hand van 1 of meerdere voorwaarden.

Voorbeeld:

Remco heeft een nieuwe televisie gezien van € 1400. Op een bepaalde dag zal hij beslissen of hij deze gaat aanschaffen. Echter is er 1 belangrijke voorwaarde, namelijk dat hij genoeg geld op de bank heeft staan.

Binnen programmeertalen zal je regelmatig beslissingen gaan maken aan de hand van de gestelde voorwaarden. Men gebruikt hiervoor het IF-statement, welke in vrijwel iedere programmeertaal op deze manier bruikbaar is (de eventuele afwijkende syntax daargelaten). Ons voorbeeld van Remco kunnen we vrij gemakkelijk omschrijven naar een IF-statement op de volgende manier:

Als het banksaldo **groter is dan** € 1400

```
if($BankSaldo > 1400)
{
    echo "Remco gaat de tv aanschaffen.";
}
else
{
    echo "Remco moet nog even doorsparen.";
}
```

Dan gaat Remco de tv aanschaffen

Anders moet hij nog doorsparen.

Na het woord IF komt dus tussen de haakjes de voorwaarde(n) te staan.

Wanneer aan de voorwaarde wordt voldaan (voorwaarde = true), dan wordt het stukje code tussen de { ... } uitgevoerd.

Wanneer niet aan de voorwaarden wordt voldaan (voorwaarde = false), dan wordt de code in het ELSE blok tussen de { ... } uitgevoerd.

6

3.1 Operatoren

Wanneer we even beter gaan kijken naar de voorwaarde

`($BankSaldo > 1400)`

dan zien we dat het `>` teken wordt gebruikt. Een dergelijk teken noemen we een operator. PHP kent een groot aantal operatoren waarvan de volgende de meest gebruikte zijn:

<code>></code>	groter dan
<code>>=</code>	groter dan of gelijk aan
<code><</code>	kleiner dan
<code><=</code>	kleiner dan of gelijk aan
<code>==</code>	gelijk aan
<code>!=</code>	niet gelijk aan

Met deze operatoren wordt vrijwel iedere voorwaarde opgebouwd. Daarnaast heb je nog 2 aparte operatoren, namelijk:

<code>&&</code>	AND (en)
<code> </code>	OR (of)



Met deze operatoren kun je meerdere voorwaarden tegelijk controleren, bijvoorbeeld:

```
if($BankSaldo > 1400 && $TVopVoorraad == 'ja')
```

Het banksaldo moet groter zijn dan € 1400 **EN** de tv moet op voorraad zijn.

```
if($BankSaldo > 1400 || $GevondenInOudeSok > 1400)
```

Het banksaldo moet groter zijn dan € 1400 **OF** het geld in een oude sok is meer dan € 1400.
Natuurlijk kun je met alle operatoren naar hartenlust combineren.

Tot slot kan het ook voorkomen dat je nog een voorwaarde wil controleren nadat aan de vorige voorwaarden niet voldaan is. Dit klinkt lastig, maar dat is het niet. Om in de tv branche te blijven hebben we hieronder een lijst met 5 tv's en hun bijbehorende prijzen.

Panasonic	€ 1599
Sony	€ 1299
Philips	€ 1099
LG	€ 799
Samsung	€ 599

We gaan een programma maken dat controleert welke tv's je kunt kopen wanneer je een bepaald bedrag op de bank hebt staan. Dit bedrag wordt telkens met € 500 verhoogd.

```
if($BankSaldo > 1500)
{
    echo "U kunt de Panasonic kopen.";
}
elseif($BankSaldo > 1000 && $BankSaldo <= 1500)
{
    echo "U kunt de Sony of Philips kopen.";
}
elseif($BankSaldo > 500 && $BankSaldo <= 1000)
{
    echo "U kunt de Lg of Samsung kopen.";
}
else
{
    echo "U moet nog even doorsparen.";
}
```

7

In principe kan dit oneindig zo door blijven gaan. Later leren we een betere manier wanneer we met veel ELSEIF stukken te maken krijgen.



4. Switch-statement

In het hoofdstuk over If-statements hebben we geleerd dat we middels een If, Elseif structuur een variabele of meerdere variabelen kunnen controleren aan de hand van een voorwaarde. Wanneer er aan deze voorwaarde wordt voldaan wordt het stukje bijbehorende code uitgevoerd. Nu kan het voorkomen dat je een enkele variabele moet controleren op een verschillende inhoud, dan is er een meer overzichtelijker manier van opschrijven door gebruik te maken van een Switch-statement. We zullen dit toelichten aan de hand van een voorbeeld.

Gegeven is de variabele kleur:

```
$Kleur = "groen";
```

In ons voorbeeld heeft deze variabele de waarde "groen". In principe kan de variabele echter iedere waarde bevatten die een kleur voorstelt. Graag willen we bij een bepaalde kleur een actie uitvoeren in de vorm van een stukje code. Laten we bijvoorbeeld een tekst uitvoeren waarbij aan de hand van de kleur een bijbehorend stuk fruit wordt getoond. Dit kan op de volgende manier worden opgelost:

```
if($Kleur == "rood")
{
    echo "De kleur komt overeen met een Appel";
}
elseif($Kleur == "groen")
{
    echo "De kleur komt overeen met een Kiwi";
}
elseif($Kleur == "geel")
{
    echo "De kleur komt overeen met een Banaan";
}
elseif($Kleur == "oranje")
{
    echo "De kleur komt overeen met een Sinaasappel";
}
elseif($Kleur == "paars")
{
    echo "De kleur komt overeen met een Druif";
}
else
{
    echo "We kunnen geen stukje fruit vinden";
}
```

8

Maar het is netter om het met een Switch-statement op te lossen:

```
switch($Kleur)
{
    case "rood":    echo "De kleur komt overeen met een Appel";
                     break;
    case "groen":   echo "De kleur komt overeen met een Kiwi";
                     break;
    case "geel":    echo "De kleur komt overeen met een Banaan";
                     break;
    case "oranje":  echo "De kleur komt overeen met een Sinaasappel";
                     break;
    case "paars":   echo "De kleur komt overeen met een Druif";
                     break;
    default:        echo "We kunnen geen stukje fruit vinden";
                     break;
}
```



Zoals je ziet levert dit compactere en beter leesbare code op. Na de switch geef je de variabele mee die je wilt controleren en achter de case komt dan de voorwaarde te staan waaraan deze moet voldoen. Daarnaast is het altijd te adviseren om een “default” actie in te stellen. Dit betekend eigenlijk dat wanneer er aan geen één voorwaarde wordt voldaan de code achter het default stukje wordt uitgevoerd. Naast bovengenoemde voordelen levert een Switch-statement in de meeste gevallen ook een betere performance op, ook al is dit zo klein dat het over het algemeen te verwaarlozen is.

5. Loops (herhalingen)

Binnen geautomatiseerde systemen komt het regelmatig voor dat bepaalde taken meer dan één keer dienen te worden uitgevoerd. Tijdens het programmeren in PHP zal je dan ook regelmatig in een situatie belanden waarbij je bepaalde stukjes code veelvuldig wil laten uitvoeren.

PHP biedt deze mogelijkheid in de vorm van loops. Hieronder zullen we aan de hand van voorbeelden de twee meest gebruikte loop methodes bespreken, de for-loop en de while-loop.

5.1 For-loop

Wanneer je van tevoren weet hoe vaak je iets wil herhalen, dan maak je gebruik van een for-loop. Bij een for-loop geef je namelijk van tevoren aan hoe vaak er herhaald dient te worden. Dit gaat via de volgende opzet:

```
for($i=0;$i<10;$i++)
{
    // code die herhaalt moet worden (in dit geval 10x)
}
```

Zoals je ziet wordt er gebruik gemaakt van de variabele \$i. Dit is een veelgebruikte benaming voor de teller waarmee men een for-loop opbouwt. Natuurlijk kun je hem ook iedere gewenste naam geven zoals \$steller.

9

Een andere ongeschreven regel is het feit dat \$i bijna altijd op 0 wordt geïnitialiseerd. Kijk maar eens naar hetgeen tussen de haakjes. Als eerste argument staat er \$i=0.

Waarom nou 0 en geen 1? Dit heeft er mee te maken dat bijvoorbeeld Arrays beginnen met de key-value 0. Om deze uit te lezen is het dus handig om bij 0 te beginnen. Wat een Array is leren we in één van de volgende lessen.

Als tweede parameter geef je het bereik aan van een for-loop. Wil ik bijvoorbeeld een stukje code 10 keer herhalen, dan geef ik de waarde zoals aangegeven \$i<10. Oftewel 0-9=10 herhalingen.

Tot slot geef je als derde parameter aan hoeveel je \$i wil verhogen. Over het algemeen is dit in 90% van de gevallen met 1. Dit kan op 2 manieren:

\$i=\$i+1

Een verkorte notatie hiervoor is:

\$i++



5.2 While-loop

De while-loop wordt in tegenstelling tot de for-loop voornamelijk gebruikt om stukken code te herhalen totdat er aan een bepaalde voorwaarde wordt voldaan. Het makkelijkste is om dit aan de hand van een voorbeeld uit te leggen.

Voorbeeld:

In het hoofdstuk over if-statements kwamen we Remco tegen die een tv wil gaan kopen van 1400,- Euro. Met het if-statement hebben we destijds gecontroleerd of Remco de tv kon gaan kopen of dat hij nog even moest doorsparen. De while-loop biedt ons de mogelijkheid om dit te blijven controleren totdat Remco 1400,- Euro heeft gespaard.

Beginsituatie:

- Remco heeft 305,- Euro op zijn bankrekening;
- Hij gaat elke maand 50,- Euro sparen;
- Daarnaast krijgt hij van de bank iedere maand 8% rente.

Met deze gegevens en gebruikmakend van een while-loop kunnen we gaan kijken hoeveel maanden het nog duurt voordat Remco voldoende geld heeft gespaard om de tv te kopen.

Zolang banksaldo kleiner is dan 1400,- Euro

```
$BankSaldo = 305; //beginsituatie
$aalntMaanden = 0;
while($BankSaldo < 1400)
{
    $BankSaldo = $BankSaldo + 50; //50 euro sparen
    $BankSaldo = $BankSaldo * 1.08; //8% rente
    $aalntMaanden++; //aalntMaanden + 1
}
echo "Na " . $aalntMaanden . " heeft Remco genoeg geld";
```

Dan spaart Remco 50,- Euro per maand
En krijgt hij 8% rente van de bank.

Zoals je ziet is de voorwaarde van een while-loop hetzelfde qua opbouw als die van een if-statement. Het verschil zit in het feit dat de code tussen de {} meerdere malen wordt doorlopen (loop) terwijl dit bij een if-statement maar één keer is. Daarom wordt de uiteindelijke uitkomst van het aantal maanden ook buiten de loop afgedrukt op het scherm. Zou je dit in de loop doen dan krijg je een x aantal regels op je scherm, waarbij x het aantal keren is waarin door de loop is gegaan.

De voorwaarde van while-loop kan overigens alle operatoren bevatten die we geleerd hebben uit het if-statement hoofdstuk. Daarnaast kun je zowel het if-statement als de for- en while-loop naar hartenlust combineren.



6. Arrays

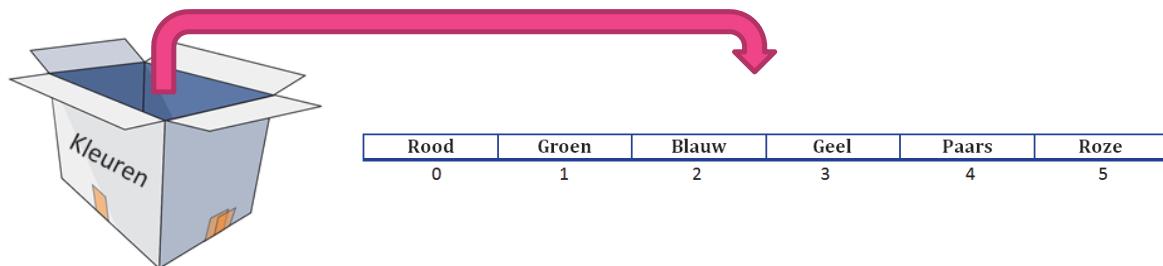
Wanneer je dit hoofdstuk gaat lezen gaan we ervan uit dat je kennis hebt over variabelen en hoe je deze kunt gebruiken. Zoals we geleerd hebben kun je een variabele vergelijken met een doosje en zijn waarde met hetgeen wat in het doosje zit. Het vervelende van een variabele is dat je maar één waarde in dat doosje kan stoppen. Wil je bijvoorbeeld meerdere waarden onthouden dan moet je hiervoor meerdere variabelen (doosjes) aanmaken. Dit is natuurlijk niet erg handig.



Gelukkig heeft PHP hiervoor een speciaal type variabele, namelijk het Array. Arrays zijn er in verschillende vormen en maten. Hierdoor zijn ze complexer en moeilijker te begrijpen dan normale variabelen, echter essentieel wanneer je echt wil gaan programmeren. Om het makkelijker te maken zullen we ook in dit hoofdstuk proberen middels visuele voorbeelden het verhaal te vertellen.

6.1 Standaard Array

Laten we beginnen met een standaard array. Stel je hetzelfde doosje voor als bij een variabele, echter nu hebben we binnenin het doosje een aantal vakjes gemaakt. Hierdoor kunnen we in ieder vakje een waarde bewaren.



11

```
$Kleuren = array("Rood", "Groen", "Blauw", "Geel", "Paars", "Roze");
```

Binnenin de doos zitten 6 vakjes. In ieder vakje kunnen we nu een waarde stoppen. In het voorbeeld is er gekozen voor een Array met de naam "Kleuren" waarin we dus waardes van verschillende kleuren stoppen. Omdat we nu vakjes hebben binnenin de doos kunnen we met 1 variabele meerdere waardes bewaren. Overigens kan een Array vele malen meer waarden (vakjes) bevatten als nu in ons voorbeeld.

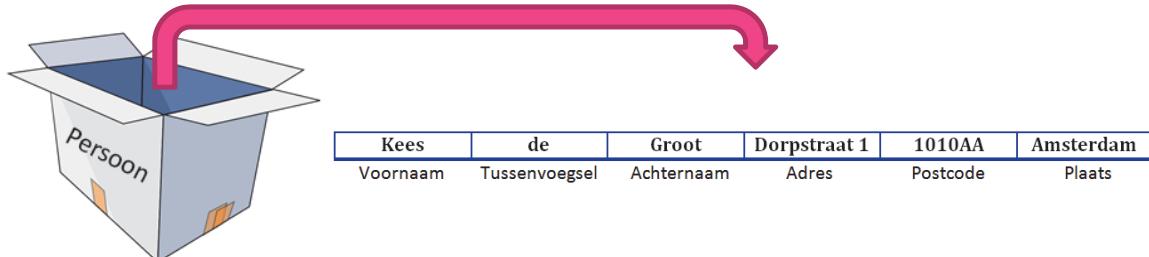
Normaliter gebruiken we het "echo" commando om de inhoud van een variabele op het scherm te tonen. In principe werkt dit bij een Array ook op deze manier, echter moet je nu verplicht aangeven om welk vakje het gaat. Dit omdat PHP anders niet weet welke van de, in ons geval 6, waarden hij moet laten zien. Als hulp heeft PHP voor ons ieder vakje van een nummer voorzien. Zoals je wellicht al is opgevallen begint het tellen **niet** bij 1 maar bij 0! Hou hier dus rekening mee! Hieronder een voorbeeld:

```
echo $Kleuren[0]; // toont Rood  
echo $Kleuren[1]; // toont Groen  
echo $Kleuren[5]; // toont Roze  
echo $Kleuren[6]; // Geeft een foutmelding aangezien alleen vakje 0-5 bestaat  
echo $Kleuren; // Geeft een foutmelding omdat je het vakje niet hebt aangegeven  
var_dump($Kleuren) // toont een schematische weergave van het Array en zijn Inhoud
```



6.2 Associatief Array

Naast het standaard Array is er ook de mogelijkheid tot een Associatief Array. Eigenlijk wil dit niets anders zeggen dan dat je namen geeft aan je vakjes in plaats van de nummers die PHP er automatisch aan toekent. Dit is vooral handig wanneer je gegevens verzameld die als een groep bij elkaar horen, in plaats van allemaal losse gegevens. Een goed voorbeeld hiervan zijn je NAW gegevens zoals hieronder is gedemonstreerd:



```
$Persoon = array("Voornaam" => "Kees",
                  "Tussenvoegsel" => "de",
                  "Achternaam" => "Groot",
                  "Adres" => "Dorpstraat 1",
                  "Postcode" => "1010AA",
                  "Plaats" => "Amsterdam");
```

Zoals je ziet in de code zal je zelf moeten aangeven hoe de vakjes gaan heten. Daarna komt er een soort van pijltje met daarachter de waarde die in het vakje hoort te zitten. Om de waarde op het scherm te kunnen tonen zal je wederom moeten aangeven uit welke vakje PHP de waarde moet tonen. In dit geval kun je dat op de volgende manier doen:

```
echo $Persoon["Voornaam"]; // toont Kees
echo $Persoon["Adres"]; // toont Dorpstraat 1
echo $Persoon[Adres]; // Geeft een foutmelding, aanhalingsstekens vergeten
echo $Persoon[1]; // Geeft een foutmelding, een Associatief Array kent geen nr's
echo $Persoon; // Geeft een foutmelding omdat je het vakje niet hebt aangegeven
var_dump($Persoon) // toont een schematische weergave van het Array en zijn Inhoud
```

12

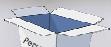
6.3 Multidimensionale Arrays

In sommige gevallen kan het voorkomen dat de gegevens die je wilt bewaren niet weg te zetten zijn in een standaard Array of een Associatief Array. Ook hier kent men binnen PHP een oplossing voor, namelijk multidimensionale Arrays. Eigenlijk is een multidimensionaal Array niets anders dan 2 of meerdere Arrays in elkaar. Dit klinkt ingewikkeld en dat is het misschien in het begin ook wel, maar wanneer je zowel de logica achter een standaard Array en een associatief Array begrijpt, dan valt de complexiteit wel mee.

Laten we tot een voorbeeld komen. Net als bij het voorbeeld van een associatief Array gaan we NAW gegevens opslaan, echter nu niet van 1 persoon maar van meerdere personen. Technisch gezien hebben we dus meerdere associative Arrays nodig om dit te bewerkstelligen. Anders gezegd, voor ieder setje NAW gegevens hebben we een doosje nodig met daarin de vakjes waarin we de waardes stoppen (het Array Persoon uit het vorige voorbeeld). We zijn nog niet klaar, want we willen al deze losse doosjes bundelen tot één geheel. Daarom kopen we een kast en zetten we op iedere plank een doosje neer. De kast kun je zien als het overkoepelende Array, met daarin de Arrays (de doosjes) met daarin de NAW gegevens van iedere persoon. Dit overkoepelende Array krijgt de naam "Personen". Om dit duidelijk te krijgen zullen we ook dit hieronder visualiseren.



0



1



2



3



Personen					
Kees	de	Groot	Dorpstraat 1	1010AA	Amsterdam
Voornaam	Tussenvoegsel	Achternaam	Adres	Postcode	Plaats
Henk		Jansen	Duckstraat 4	2020BB	Eindhoven
Voornaam	Tussenvoegsel	Achternaam	Adres	Postcode	Plaats
Julia	van	Ess	Hooglaan 12	3030CC	Utrecht
Voornaam	Tussenvoegsel	Achternaam	Adres	Postcode	Plaats
Merel		Swinkels	Munt 125	4040DD	Rotterdam
Voornaam	Tussenvoegsel	Achternaam	Adres	Postcode	Plaats

```
$Personen = array(array("Voornaam" => "Kees",
    "Tussenvoegsel" => "de",
    "Achternaam" => "Groot",
    "Adres" => "Dorpstraat 1",
    "Postcode" => "1010AA",
    "Plaats" => "Amsterdam"),
array("Voornaam" => "Henk",
    "Tussenvoegsel" => "",
    "Achternaam" => "Jansen",
    "Adres" => "Duckstraat 4",
    "Postcode" => "2020BB",
    "Plaats" => "Eindhoven"),
array("Voornaam" => "Julia",
    "Tussenvoegsel" => "van",
    "Achternaam" => "Ess",
    "Adres" => "Hooglaan 12",
    "Postcode" => "3030CC",
    "Plaats" => "Utrecht"),
array("Voornaam" => "Merel",
    "Tussenvoegsel" => "",
    "Achternaam" => "Swinkels",
    "Adres" => "Munt 125",
    "Postcode" => "4040DD",
    "Plaats" => "Rotterdam")
);
```

13

Zoals je ziet kan een multidimensionaal Array worden opgebouwd uit een combinatie van standaard en associatieve Arrays. Dit is geheel aan de gebruiker, wat hij of zij wenst. In ons voorbeeld zien we dat iedere plank binnen de kast een nummer heeft gekregen. Het overkoepelende Array "Personen" is dus een standaard Array. Daarbinnen hebben we op iedere plank een doos gezet met NAW gegevens. Dit zijn dus ieder op zijn beurt een associatief Array. In dit geval zal je om de juiste waarde te tonen aan PHP kenbaar moeten maken op welke plank hij moet kijken en vervolgens welk vakje hij uit de doos die op de plank staat moet kiezen. Hieronder enkele voorbeelden.

```
echo $Personen[0]["Voornaam"]; // toont Kees
echo $Personen[2]["Voornaam"]; // toont Julia
echo $Personen[3]["Adres"]; // toont Munt 125
echo $Personen[3][Adres]; // Geeft een foutmelding, aanhalingsstekens vergeten
echo $Personen[1]; // Geeft een foutmelding omdat je 1 vakje niet hebt aangegeven
echo $Personen; // Geeft een foutmelding omdat je 2 vakjes niet hebt aangegeven
var_dump($Personen) // toont een schematische weergave van het Array en zijn Inhoud
```



7. Require & Include

Wanneer je een website gaat maken of een groter web-based systeem, is het belangrijk om dit in een bepaalde structuur te doen. Door structuur toe te voegen aan je project blijft het overzichtelijker en makkelijker om te beheren. Hoe groter een project/systeem des te meer profijt je hebt van een goede structuur. Dit geldt zeker wanneer je werkt met meerdere ontwikkelaars aan één en hetzelfde project.

Voorbeeld van een mapstructuur van een website:

Files	11-6-2014 11:54	File folder
Images	11-6-2014 11:54	File folder
Jquery	11-6-2014 11:57	File folder
Modules	11-6-2014 11:54	File folder
Configuratie.php	10-6-2014 13:59	PHP File
Functies.php	10-6-2014 13:59	PHP File
index.php	10-6-2014 13:59	PHP File
Style.css	10-6-2014 13:59	CSS File

Dit is een voorbeeld. Denk zelf na over een structuur die het beste bij jou en je systeem/website past.

Wanneer je je systeem/website opdeelt middels een structuur, dan zal je op de één of andere manier de bestanden moeten laten samenwerken. PHP heeft hiervoor de functies `Require` en `Include` standaard in het pakket zitten. Via deze functies kun je op elke gewenste plek in je PHP code een ander PHP bestand invoegen. De inhoud van het ingevoegde bestand wordt letterlijk op die plek geplakt en uitgevoerd.

Hieronder zien we de code die nodig is om een bestand toe te voegen:

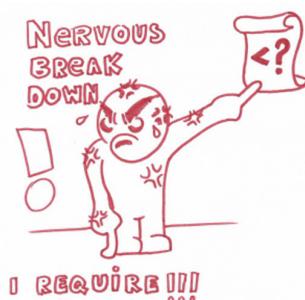
14

```
include ('./modules/bestand.php');  
of  
require ('./modules/bestand.php');
```

Beide functies doen in principe hetzelfde tenzij het bestand dat je wilt invoegen niet gevonden kan worden. De `require()` functie genereert een fatale fout en stopt het script, terwijl de `include()` functie alleen een waarschuwing geeft en de rest van de code gewoon verder uitvoert.
Welke functie moet je dan gebruiken, vraag je je misschien af?



Dat ligt dus aan het bestand dat je toevoegt. Kan de rest van het systeem niet goed functioneren zonder dat bestand dan dien je `Require` te gebruiken. Een goed voorbeeld hiervan zijn bestanden met configuratie gegevens, database connecties of belangrijke functies. Wanneer je bijvoorbeeld een systeem hebt waaraan modules kunnen worden toegevoegd, dan kan het handig zijn dat wanneer het PHP bestand van die specifieke module ontbreekt, het systeem zelf wel doordraait. Gebruik in dit soort gevallen wel `include()`. Wanneer je er niet uitkomt, dan kun je in bijna alle gevallen het beste `require()` gebruiken.



Hieronder een voorbeeld van een modulaire site:

```
<?php
//invoegen ondersteunende bestanden
require('./Configuratie.php');
require('./Functies.php');

//haalt het pagina nr op
if(isset($_GET['pagina']))
    $pagina = $_GET['pagina'];
else
    $pagina = null;
?>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <link rel="stylesheet" type="text/css" href="Style.css" />
    <title>Modulaire site</title>
</head>
<body>
<div id="Wrapper">
    <div id="Header">
        <?php
            require('./modules/header.php');
        ?>
    </div>
    <div id="Menu">
        <?php
            require('./modules/menu.php');
        ?>
    </div>
    <div id="Content">
        <?php
            switch($pagina)
            {
                case 1: require('./modules/Pagina1.php');
                break;
                case 2: require('./modules/Pagina2.php');
                break;
                case 2: require('./modules/Pagina2.php');
                break;
                case 3: require('./modules/Pagina3.php');
                break;
                case 4: require('./modules/Pagina4.php');
                break;
                default:require('./modules/Home.php');
                break;
            }
        ?>
    </div>
</div>
</body>
</html>
```



8. Functies

In de praktijk komt het vaak voor dat bepaalde code regelmatig moet worden uitgevoerd of dat bepaalde code door meerdere bestanden wordt gebruikt. Het is dan inefficiënt om telkens weer dat stukje code opnieuw te schrijven (of te kopiëren) op de plek waar je het nodig hebt. Ten eerste kost het teveel werk en ten tweede is het totaal niet onderhoudsvriendelijk. Stel je voor dat je op een later tijdstip wat functionaliteit in dat desbetreffende stukje code moet wijzigen, dan moet je dat doen op alle plaatsen waar je ooit die code had gebruikt. Gelukkig kent PHP een makkelijkere manier om dit op te lossen in de vorm van functies.

PHP kent standaard al een groot aantal meegeleverde functies. Voorbeelden hiervan zijn:

- empty()
- isset()
- round()
- require()
- include()
- count()

Daarnaast heb je de mogelijkheid om zelf functies te schrijven. Dit gaan we hieronder verder behandelen.

Een functie is eigenlijk een stukje code wat staat te wachten om uitgevoerd te worden op het moment dat de functie wordt aangeroepen. Dit betekent dat de code ook daadwerkelijk alleen uitgevoerd wordt op het moment dat de functie wordt aangeroepen. Gebeurt dit niet dan slaat PHP alle code binnen de functie over. Vaak worden veelgebruikte functies verzameld in een bestand of in een module zodat deze middels een require of include binnen het gehele programma te gebruiken zijn.

16

Laten we eens naar een voorbeeld kijken:

```
function ToonGegevens()  
{  
    echo 'Kees de Groot <br />';  
    echo 'Dorpstraat 1 <br />';  
    echo '1010AA Amsterdam';  
}
```

Bovenstaande functie toont de gegevens van Kees de Groot, wanneer de functie wordt aangeroepen. Zoals je ziet begin je de functie met het sleutelwoord "function". Direct daarachter zet je de naam van de functie neer. Kies altijd een naam die past bij hetgeen de functie uitvoert. Dit zorgt ervoor dat je code altijd goed te lezen blijft. Na de naam komen altijd 2 haakjes met daartussen optioneel een aantal parameters (dit wordt nog behandeld). Om de functie te activeren kun je hem aanroepen bij zijn naam zoals hieronder is weergegeven:

```
ToonGegevens(); //roept de functie aan. De code binnen de functie wordt nu uitgevoerd
```



8.1 Functie parameters

Bij een functie heb je de mogelijkheid om gegevens mee te geven aan de functie door (een) variabele(n) mee te geven bij het aanroepen van de functie. Het criterium hierbij is echter wel dat de functie zo is gedefinieerd dat hij een aantal parameters heeft. Deze parameters staan direct tussen de haakjes achter de naam van de functie. Om dit te verduidelijken gaan we een functie maken die 2 getallen bij elkaar optelt.

```
function Optellen($Getal1,$Getal2)
{
    $Totaal = $Getal1 + $Getal2;

    return $Totaal;
}
```

Zoals je ziet hebben we hierboven een functie met de naam "Optellen". Het verschil met de vorige functie is dat er nu 2 variabelen, namelijk Getal1 en Getal2, tussen de haakjes zijn gedefinieerd. Dit noemen ze de parameters van een functie. Wanneer je de functie aanroeft ben je dan ook verplicht om 2 parameters mee te geven in de aanroep. Doe je dit niet dan krijg je een foutmelding. Overigens kun je parameters wel optioneel maken, maar daarover later meer. Zoals de naam van de functie beaamt gaan deze beide getallen bij elkaar optellen. Graag wil je feedback van de functie, in dit geval de totaal waarde. Iets teruggeven uit de functie kun je doen door het sleutelwoord "return" te gebruiken. In ons voorbeeld zie je dat de variabele "Totaal" wordt teruggegeven. Dat wil zeggen, je geeft **de inhoud** van de variabele "Totaal" terug, niet de variabele zelf. Hieronder een voorbeeld.

```
$Getal1 = 10;
$Getal2 = 5;
$Uitkomst = Optellen($Getal1,$Getal2);
echo $Uitkomst; // de waarde 15 komt op het scherm te staan.
```

17

De volgende aanroepen leveren overigens een foutmelding op omdat de functie Optellen verwacht dat je 2 parameters meegeeft:

```
$Uitkomst = Optellen(); //Geeft een foutmelding. geen parameters meegegeven
$Uitkomst = Optellen(10); //Geeft een foutmelding. Maar 1 vd 2 parameters meegegeven
```

8.2 Optionele parameters

In het vorige stukje hebben we gezien dat we verplicht waren het aantal parameters mee te geven, welke in de functie header zijn gedefinieerd. In sommige gevallen wil je echter de mogelijkheid hebben om een functie verschillende dingen te laten doen op basis van de parameters die wel of niet zijn meegegeven. Voor onderstaand voorbeeld gaan we even terug naar de functie "ToonGegevens". Deze functie gaf de NAW gegevens weer van Kees de Groot. We willen deze functie uitbreiden door optioneel ook zijn e-mail en telefoonnummer te tonen. Zie hieronder de aangepaste functie:

```
function ToonGegevens($Extra = NULL)
{
    echo 'Kees de Groot <br />';
    echo 'Dorpstraat 1 <br />';
    echo '1010AA Amsterdam <br />';

    if(!empty($Extra))
    {
        echo 'Kees.de.groot@hotmail.com <br />';
        echo '06-12345678';
    }
}
```



Wat gelijk opvalt is dat de functie nu een parameter heeft met de naam "Extra". Achter deze parameter staat echter het volgende:

= **NULL**

Dit geeft aan dat de parameter optioneel is. Je bent dus niet verplicht om bij de aanroep van de functie een parameter mee te geven. Letterlijk maak je een lege variabele \$Extra aan welke alleen gevuld wordt op het moment dat er bij de aanroep van de functie een parameter wordt meegegeven. Binnen de code van de functie controleren we middels !empty() of de variabele \$Extra een waarde bevat. Alleen wanneer dit het geval is tonen we de extra gegevens. Hieronder de aanroep in werking:

```
ToonGegevens(); // laat alleen de basis gegevens zien  
ToonGegevens('iets'); // later zowel de basis als de extra gegevens zien
```

8.3 Lokale vs globale variabelen

Voor veel beginnende programmeurs is het volgende soms lastig te begrijpen. Zoals gezegd wordt alle code binnen een functie-blok alleen uitgevoerd wanneer de functie wordt aangeroepen. Binnen de functie kun je gebruik maken van variabelen. Echter deze variabelen zijn alleen zichtbaar en te gebruiken binnen het functie-blok. We noemen deze variabelen dan ook lokale variabelen.

Alle variabelen die niet binnen een functie-blok zijn gedefinieerd en dus beschikbaar zijn in het hoofdprogramma, noemen we globale variabelen. Het kan dus voorkomen dat er zowel binnen de functie als in het hoofdprogramma variabelen bestaan met dezelfde naam. Toch zullen ze elkaar niet overschrijven omdat ze, zoals ze dat noemen, in een andere Scope zitten. Hieronder is middels een voorbeeld geprobeerd dit te verduidelijken. Uiteindelijk zal je hiermee veel moeten oefenen.

```
<?php  
function Optellen($Getal1,$Getal2)  
{  
    $Totaal = $Getal1 + $Getal2;  
  
    echo $Getal1; // de waarde 10 komt op het scherm te staan.  
    echo $Getal2; // de waarde 20 komt op het scherm te staan.  
    echo $Totaal; // de waarde 30 komt op het scherm te staan.  
    echo $Uitkomst; //Geeft een foutmelding. $Uitkomst bestaat niet binnen de  
    functie  
  
    return $Totaal; //Geeft de inhoud van Variabele Totaal terug  
}  
  
$Totaal = 5;  
  
//Functie wordt aangeroepen, hetgeen terugkomt stoppen we in de variabele Uitkomst  
$Uitkomst = Optellen(10,20);  
  
echo $Getal1; //Geeft een foutmelding. $Getal1 bestaat alleen binnen de functie  
echo $Getal2; //Geeft een foutmelding. $Getal2 bestaat alleen binnen de functie  
echo $Totaal; // de waarde 5 komt op het scherm  
echo $Uitkomst; // de waarde 30 komt op het scherm te staan.  
?>
```



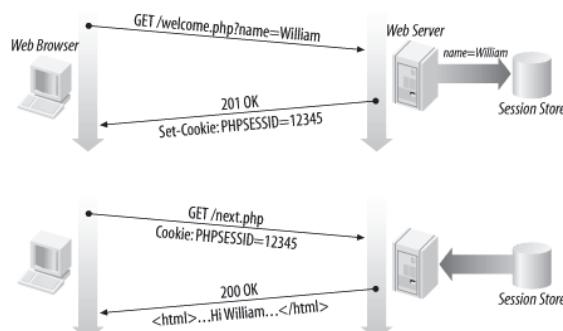
9. Sessions

Zoals we inmiddels geleerd hebben in het hoofdstuk over cookies, is het vaak handig of noodzakelijk om gegevens behorende bij een gebruiker te kunnen opslaan. Denk bijvoorbeeld aan een winkelwagentje die door de gebruiker gevuld wordt door producten.

In tegenstelling tot cookies, worden sessions vooral gebruikt om kortstondig gegevens op te slaan die na het verlaten van de browser ook weer weg zijn. Wanneer gebruik je dan wat? In principe geldt dat je zoveel mogelijk gebruik maakt van sessions en pas in het uiterste geval gebruik maakt van cookies. Alle eventuele privacy gevoelige informatie blijft bij een session namelijk op de server terwijl bij een cookie dit via internet verstuurd wordt om als bestand op de computer van de gebruiker te worden geplaatst. Dit betekent dan automatisch dat je nooit, maar dan ook nooit, paswoord gegevens opslaat in een cookie.

Een andere reden om zo weinig mogelijk cookies te gebruiken is dat ze steeds minder geliefd worden wegens het commerciële misbruik ervan. De Nederlandse wet verplicht je voortaan aan de gebruiker te vragen of je überhaupt cookies mag plaatsen.

Oke, terug naar het verhaal over sessions. Om dit te begrijpen kijken we naar het volgende schema waarin de communicatie tussen de client (bezoeker van je website) en de server wordt weergegeven, wanneer je een session aanmaakt en uitleest.



19

Op het moment dat je een session start in je code, maakt de PHP server een session variabele aan op de server waaraan hij een sessionId koppelt. Dit ID wordt middels een tijdelijke cookie naar de gebruiker gestuurd zodat de gegevens op de server aan de gebruiker worden gekoppeld. Elke keer wanneer de gebruiker een andere pagina binnen de website aanklikt, zal het sessionId in de cookie worden uitgelezen om zo de bijbehorende gegevens op de server terug te vinden.

Hoe doen we dit in PHP met een stukje code? PHP kent naast de al eerder besproken SUPERGLOBALS variabelen `$_POST` en `$_GET`, ook de SUPERGLOBAL `$_SESSION`. Middels deze variabele en een aantal session functies kunnen we gegevens gaan opslaan!

9.1 Sessions starten

Om met sessions te kunnen werken zal je eerst de functie `session_start()` moeten aanroepen. Dit kan je het beste doen als allereerste regel in je script. Mocht je meerdere PHP bestanden middels een `include()` of `require()` toevoegen aan je hoofdbestand, dan is het nog steeds alleen noodzakelijk om de session te starten in het hoofdbestand. Probeer je de session meer dan één keer te starten, dan zal dit resulteren in een foutmelding.

```
<?php  
session_start(); //maak een nieuwe sessie aan, aan het begin van je PHP bestand
```



9.2 Sessions vullen en uitlezen

Wanneer de session gestart is kun je ieder gewenst gegeven in de `$_SESSION` variabele stoppen zodat deze op de server wordt opgeslagen. Net als `$_POST` en `$_GET` is `$_SESSION` van het datatype Array. Hieronder een aantal voorbeelden:

```
$_SESSION['string'] = "PHP is leuk";
$_SESSION['int'] = 35;
$_SESSION['variabele'] = $variabele;
```

Wil je nu na het herladen van de pagina, binnen hetzelfde bestand of binnen een ander PHP bestand in hetzelfde domein, de gegevens weer uitlezen, dan gaat dat op de volgende manier:

```
echo $_SESSION['string']; //geeft PHP is leuk op het beeldscherm weer
$Getal = $_SESSION['int']; // $Getal = 35
$variabele = $_SESSION['variabele']; //zet de inhoud terug in $variabele
```

Zoals je misschien wel opgevallen is is dit het tegenovergestelde van een session vullen.

9.3 Session verwijderen

Standaard worden de gegevens in een sessie 30 minuten op de server opgeslagen en zullen dan verdwijnen. Elke keer wanneer de gebruiker een andere pagina van je website bezoekt gaan er opnieuw 30 nieuwe minuten in. Dit is ook de reden waarom je automatisch uitlogt wanneer er een tijd lang geen activiteit is geweest.

Soms is het echter handig, bijvoorbeeld bij een inlogssysteem, dat je tussendoor de sessie kan verwijderen. Dit bijvoorbeeld als de gebruiker wil uitloggen. Hiervoor gebruiken we het volgende stukje code:

```
session_destroy();
```

20

9.4 Header refresh

Vaak als je met sessions aan de gang gaat moet je nog eenmaal de pagina verversen of een link aanklikken op de site alvorens de gegevens ook echt verwerkt zijn. Een goed voorbeeld hiervan zie je vaak terug wanneer je inlogt op een forum. Je krijgt dan een tussenscherm met de melding: "Binnen 5 seconden wordt u doorgelijdt naar de hoofdpagina". Om automatisch een verversing van de pagina te doen en dit probleem te omzeilen kunnen we gebruik maken van een header refresh.

```
header("location:http://www.website.nl");
```

De pagina wordt op deze manier meteen ververst. Dit is vrijwel onzichtbaar voor de gebruiker.

```
header("Refresh: 5; URL=http://www.website.nl");
```

De pagina wordt na 5 seconden ververst, zoals in het voorbeeld van ons forum.



10. Extra informatie over time() en date()

In het boek hebben we het één en ander geleerd over de functies time en date. We weten inmiddels dat we met de functie time() de UNIX tijd kunnen opvragen. Dit is dus het aantal seconden vanaf 1 januari 1970. Met de functie date() hebben we geleerd dat we deze UNIX tijd kunnen omzetten naar een leesbaar formaat zoals een datum.

In de praktijk komen we echter nog een aantal zaken tegen die we hier willen bespreken. Zo verplicht PHP vanaf versie 5.3 dat je de tijdszone aangeeft voordat je met de functie date() aan de slag gaat. Wanneer je dit niet doet zal je een foutmelding krijgen op je scherm. Om de tijdszone te zetten, gebruiken we de functie date_default_timezone_set(), zoals hieronder weergegeven:

```
date_default_timezone_set('Europe/Amsterdam');
```

De tijdszone die we in nederland invullen is: Europe/Amsterdam.

Een ander probleem wat we in de praktijk veel tegen komen is dat een datum veld in de database op de Amerikaanse manier wordt opgeslagen in het formaat YYYY-MM-DD. Wanneer we dit uitlezen en op ons scherm tonen is dit niet prettig leesbaar voor Nederlandse gebruikers die gewend zijn aan het DD-MM-YYYY formaat. Je kunt dit middels speciale Sql commando's omzetten, maar gelukkig biedt PHP een eenvoudigere oplossing via de functie strtotime(). Deze functie kan de uit de database opgevraagde datum terug converteren naar een UNIX time. Dat is erg handig want dan kunnen wij deze gebruiken om via de date() functie dit weer om te zetten naar het Nederlandse formaat.

Hieronder het voorbeeld uitgewerkt:

```
$DatumDatabase = '2015-01-03'; //3 Januari 2015 Amerikaanse Stijl  
$NLDatum = date('d-m-Y', strtotime($DatumDatabase)); //Geeft 03-01-2015
```



11. Database connecties in PHP

Om dynamische websites te maken, waarin men gegevens kan ophalen of opslaan, zal er in 99% van de gevallen gebruik worden gemaakt van een database. Wanneer men in PHP een website gaat ontwikkelen wordt er veelal gekozen voor een MySQL database. PHP biedt de gebruiker een aantal keuzes wanneer het aankomt op het maken van een database connectie met de MySQL database server.

Vroeger werd veel gebruik gemaakt van de ingebouwde MySQL extensie (inclusief bijbehorende functies) binnen PHP. Deze wordt echter vanaf PHP versie 6 niet meer ondersteund. Er is namelijk een opvolger in de vorm van de MySQLi extensie, waarbij de i staat voor improved (verbeterd). Deze extensie bevat een aantal extra functies/voordelen zoals een object georiënteerde interface en de mogelijkheid tot prepared statements. Hierop komen we later nog terug. MySQLi is dan ook een zeer geschikte extensie om te gebruiken bij de koppeling tussen PHP en de MySQL database.

Er is echter inmiddels een nieuwe extensie, PDO (PHP Data Objects), welke steeds meer de voorkeur geniet over de MySQLi extensie. Deze relatief nieuwe PHP extensie ondersteunt op het moment al meer dan 18 verschillende databases, in tegenstelling tot MySQLi, die alleen de MySQL database ondersteund. Je kunt je voorstellen dat dit voor programmeurs vele voordelen oplevert. Je kunt namelijk middels het aanleren van een methode vrijwel elke gewenste database aanspreken. Enkele voorbeelden van bekende databases zijn:

- Microsoft SQL Server
- PostgreSQL
- Oracle
- SQLite

22

Vanaf nu gaan we dus aan de slag met PDO!

11.1 Verbinding maken met PDO

Om gegevens op te halen of weg te schrijven naar een database zal er eerst een connectie gemaakt moeten worden tussen PHP en de database server. Goed om te weten is dat PDO compleet object georiënteerd is. Een nieuwe database connectie maken is eigenlijk niets meer dan het aanmaken van een nieuwe instantie van de PDO class. Dit klinkt wellicht heel ingewikkeld, echter wanneer je de code ziet valt dit reuze mee. Overigens gaan we verderop in de opleiding nog uitgebreid terugkomen op het object georiënteerde programmeren.

Een MySql verbinding maken:

```
try
{
    $pdo = new PDO("mysql:host=localhost;dbname=mobieljes", 'root', '');
}
catch(PDOException $e)
{
    echo $e->getMessage();
}
```

Ter illustratie staan hieronder 2 voorbeelden van connecties naar andere databases.



Oracle:

```
try
{
    $pdo = new PDO("OCI:dbname=mobieljes;charset=UTF-8", 'root', '');
}
catch(PDOException $e)
{
    echo $e->getMessage();
}
```

ODBC:

```
try
{
    $pdo = new PDO("odbc:Driver={Microsoft Access Driver (*.mdb)};Dbq=C:\accounts.mdb;Uid=root");
}
catch(PDOException $e)
{
    echo $e->getMessage();
}
```

Zoals je kunt zien werkt dit vrijwel op dezelfde manier. Vanaf nu zullen we ons echter alleen richten op de verbinding met een MySql database.

11.2 Try-catch

Misschien is het je al opgevallen dat er om de connectie regel een try-catch blok staat.

Try-catch blokken komen in heel veel programmeertalen voor en zijn bedoeld om fout afhandeling te regelen. Het komt er eigenlijk op neer dat elke regel code in het try gedeelte wordt uitgeprobeerd, vandaar de naam try. Wanneer er ergens in die code iets misgaat, wordt er een fout gegenereerd.

In een programmeertaal spreken ze er vaak over dat de foutmelding omhoog wordt gegooid.

In het catch gedeelte, verderop in de code, wordt de foutmelding dan weer opgevangen, vandaar de naam catch. Binnen het catch blok kun je dan als programmeur bepalen hoe je de fout wil afhandelen. Veelal wordt er een foutmelding op het scherm getoond.

23

Bij PDO is het erg belangrijk voor de veiligheid dat je tenminste de connectieregel in een try-catch blok zet omdat anders wellicht gevoelige informatie door de PHP error-handler op het scherm wordt gezet. Deze informatie kan dan bijvoorbeeld door hackers gebruikt worden om gegevens uit je database te halen of te verwijderen.

11.3 Een Query?

Nu we weten hoe we een connectie kunnen maken met de database, willen we natuurlijk iets gaan doen met die database. Veruit het meest zal je gegevens uit een database willen opvragen. Dit opvragen werkt via een SELECT statement. Tijd voor een voorbeeld.



We hebben de volgende database tabel:

Tabel Mobieltjes

MobielCode	Type	Kleur	Simlock	Prijs
1	Iphone4	Zwart	JA	499
2	Iphone5	wit	NEE	300
3	Iphone5	wit	NEE	200
4	Galaxy3	zwart	NEE	299
5	lumia920	rood	JA	499
6	Lumia920	geel	JA	499
7	JamesBond	Gold	JA	999
8	Lumia620	zwart	NEE	299

Wanneer we alle gegevens uit de tabel willen opvragen zal de query (een commando dat je kunt uitvoeren op de database) er als volgt uitzien:

SELECT * FROM mobieltjes;

Vaak willen we echter specifieke gegevens uit een tabel, bijvoorbeeld alleen de mobieltjes welke geen simlock hebben. De Query ziet er dan als volgt uit:

SELECT * FROM mobieltjes WHERE Simlock = 'NEE';

Voorlopig zullen we ons in eerste instantie op deze manier van query's richten. In leerjaar 2 gaan we dieper in op de mogelijkheden.

24

11.4 Een query uitvoeren in PHP (Prepared statements)

Om een query uit te voeren binnen PHP zal je als programmeur enkele regels code moeten schrijven. Binnen PDO kun je een query uitvoeren op 2 manieren, via de basismethode en via prepared statements. Prepared statements genieten onze voorkeur, omdat deze 2 voordelen hebben t.o.v. de basismethode, namelijk betere performance en bescherming tegen Sql injection. Middels Sql injection kunnen hackers namelijk je database uitlezen of vernietigen, met alle gevolgen van dien. Genoeg gelezen, hier een stukje voorbeeldcode waarin we uitgaan van een SELECT query waarin we alle niet gesimlockte telefoons willen opvragen.

```
$parameters = array(':simlock'=>'nee');
$stmt = $pdo->prepare('select * from mobieltjes where simlock = :simlock');

$stmt->execute($parameters);
```

We kijken eerst even naar de 2^e regel. We zien hier de Sql query zoals eerder is besproken, met het verschil dat er achter simlock = een named parameter :simlock staat. De bedoeling is dat deze :simlock straks vervangen wordt door de waarde waarop we willen zoeken (in geval van ons voorbeeld de waarde "Nee").

We moeten er dus voor zorgen dat we het systeem vertellen welke waarde op de plek van :simlock moet komen te staan. Dit doen we door een nieuwe variabele aan te maken van het type array, die we de naam \$parameters meegeven (zie regel 1). Als eerst geef je het array op de plek van de key value de naam van de named parameter mee, dus in ons geval :simlock.



Als value geef je vervolgens de waarde mee die op de plek van de named parameter moet komen binnen de query (in ons geval "nee").

Overigens kun je op die plek ook een zelfgemaakte variabele meegeven of een `$_POST` of `$_GET`.

```
$parameters = array(':simlock'=>$MijnVariabele);  
of  
$parameters = array(':simlock'=>$_POST['variabele']);
```

in principe hebben we nu al het voorbereidende werk gedaan. Het enige wat ons rest is de query echt laten uitvoeren op de database. Dit gebeurt op regel 3 middels de execute functie. Het array `$parameters` wordt hier meegegeven en de database voert na het koppelen de query daadwerkelijk uit.

11.5 Iets doen met het resultaat van een SELECT query

In het vorige blokje hebben we geleerd hoe we een query op de database kunnen uitvoeren middels een prepared statement. Wanneer de query is gelukt, en daar gaan we even vanuit, dan zal dit een resultaat opleveren. Dit resultaat bevat de gegevens uit de database die we bijvoorbeeld willen weergeven op het scherm. We hebben hiervoor het volgende stukje code nodig:

```
while ($row = $sth->fetch())  
{  
    echo $row['Type']. '<br />';  
}
```

De `fetch()` methode vertaalt 1 rij uit de database naar een gewenst datatype in PHP. Standaard is dit via de `fetch_both` modus welke een array aanmaakt die als key value de kolomnamen bevat. In eerste instantie gaan we alleen deze modus gebruiken. In ons voorbeeld wordt het array `$row` aangemaakt. Om de waarde uit de kolom **Type** van de tabel **Mobieltjes** weer te geven krijg je dus de volgende code:

```
echo $row['Type']. '<br />';
```

25

Omdat de `Fetch` methode maar 1 rij omzet, zal je om alle rijen uit het resultaat weer te geven gebruik moeten maken van een `while`-loop. In feite zeg je dus: zolang als er nog rijen zijn dan `fetch` (verwerk) je deze rij. We zijn dus nu in staat om gegevens uit de database op te vragen en deze weer te geven op het scherm.

11.6 Gegevens in een database toevoegen

Naast het opvragen van gegevens kan het handig zijn om nieuwe gegevens aan de database toe te voegen. Er is bijvoorbeeld een nieuwe mobiele telefoon uitgekomen en deze willen we toevoegen aan onze tabel `mobieltjes`. Hoe doen we dat dan? Het gaat middels een `INSERT` query zoals hieronder is aangegeven.

```
INSERT INTO mobieltjes (MobielCode, Type, Kleur, Simlock, Prijs)  
VALUES (9, 'Iphone6', 'wit', 'NEE', 699);
```

Je ziet dus dat je de kolomnamen aangeeft en vervolgens aangeeft welke waarden je daar wilt invoegen. Vaak werken tabellen met een zogenaamde `auto_increment`. Dit is een optie waarbij de database zelf nummers optelt, waardoor je nooit een dubbele primaire sleutel kan krijgen. In ons voorbeeld zou `MobielCode` een `auto_increment` kolom kunnen zijn. In dat geval hoeven we de waarde 9 niet meer mee te geven.



Om dit via PDO in een stukje PHP code uit te voeren doen we bijna hetzelfde als we deden bij het opvragen van gegevens. Laten we kijken naar het stukje code wat we nodig hebben nadat we een connectie met de database hebben opgezet.

```
$parameters = array(':mobielcode'=>9,
                     ':type'=>'Iphone6',
                     ':kleur'=>'wit',
                     ':simlock'=>'NEE',
                     ':prijs'=>699);

$stmt = $pdo->prepare('INSERT INTO mobieljes (MobielCode, Type, Kleur,
Simlock, Prijs) VALUES (:mobielcode, :type, :kleur, :simlock, :prijs)');

$stmt->execute($parameters);
```

zoals je ziet is het vrijwel identiek aan datgene wat we doen met de SELECT queries.

Mocht je werken met een auto_increment kolom dan wil je natuurlijk graag weten welk nummer de database heeft gegenereerd. Voeg daarvoor nog 1 regel toe achteraan het vorige stukje code.

```
echo $pdo->lastInsertId();
```

11.7 Gegevens uit de database wijzigen

We weten inmiddels hoe we gegevens uit de database kunnen opvragen en hoe we gegevens aan de database kunnen toevoegen. Het komt echter ook regelmatig voor dat bestaande gegevens in een database gewijzigd dienen te worden. Denk bijvoorbeeld aan een klant die gaat verhuizen, waardoor zijn of haar adresgegevens aangepast dienen te worden. Het aanpassen van een bestaande rij (record) in de database gaat middels een UPDATE query zoals deze hieronder is aangegeven.

26

UPDATE mobieljes SET Prijs = 399 WHERE MobielCode = 1;

Met bovenstaande query gaan we weer terug naar onze voorbeeldtabel met mobieljes, waarin we in dit geval de prijs van de telefoon met **MobielCode 1** willen veranderen in 399,-. Dit bijvoorbeeld omdat deze mobiele telefoon op dit moment in de aanbieding is.

Ook in dit geval lijkt het stukje PHP code om dit uit te voeren sprekend op de code die we gebruikten bij een SELECT en INSERT query. Hieronder wordt het benodigde stukje code weergegeven, in ons achterhoofd houdend dat we reeds een connectie met de database hebben opgezet.

```
$parameters = array(':prijs'=>299,
                     ':mobielcode'=>1);

$stmt = $pdo->prepare('UPDATE mobieljes SET Prijs = :prijs
WHERE MobielCode = :mobielcode;');

$stmt->execute($parameters);
```

Net zoals bij de INSERT kun je bij een UPDATE query additionele gegevens opvragen. Het kan bijvoorbeeld handig zijn om te weten hoeveel rijen zijn ge-updated na het uitvoeren van de query. Dit gaat met de volgende kleine aanpassing:

```
echo "aantal rijen: ".$stmt->rowCount();
```



11.8 Gegevens uit de database verwijderen

Tot slot, om het verhaal compleet te maken, kijken we hoe we gegevens uit een database kunnen verwijderen. Binnen Sql doen we dit met een DELETE query. Let op dat je erg nauwkeurig te werk gaat want voor je het weet verwijder je meer gegevens dan je had gewild. De DELETE query ziet er als volgt uit:

DELETE FROM mobieltjes WHERE 1;

Deze query verwijdert alle gegevens uit de tabel Mobieltjes.

DELETE FROM mobieltjes WHERE MobielCode = 1;

Deze query verwijdert alleen het mobieltje met MobielCode 1.

Om dit uit te voeren via PDO in PHP gebruiken we het volgende stukje code:

```
$parameters = array(':mobilcode'=>1);

$sth = $pdo->prepare('DELETE FROM mobieltjes WHERE MobielCode = :mobilcode');

$sth->execute($parameters);
```

Ook hier kun je net als bij de UPDATE query opvragen hoeveel rijen er zijn verwijderd door de volgende code toe te voegen:

```
echo " aantal rijen: ". $sth->RowCount();
```

In het voorbeeld is dit maar 1 rij omdat MobielCode een primary key veld is en de gegevens erin dus altijd maar één keer mogen voorkomen.



12. PSD (programma structuur diagram)

In de praktijk zal het regelmatig voorkomen dat je complexe stukken programmacode moet gaan programmeren. Het is dan van belang om eerst na te denken over een mogelijke oplossing alvorens je aan de slag gaat. Een goede methode om hier over na te denken is door het maken van een PSD. Een PSD geeft je een goed overzicht van wat een bepaald stuk code moet gaan doen zonder dat daarvoor een regel wordt geprogrammeerd. Dit soort PSD's worden ook vaak gebruikt in grotere teams en in technische ontwerpen, zodat iedere programmeur in zo'n team weet wat er geïmplementeerd moet worden.

Hoe ziet zo'n PSD er dan uit?

Hieronder een voorbeeld van een basis PSD met daarnaast de bijbehorende PHP code.



```
$GetalA = $_POST['GetalA'];
$GetalB = $_POST['GetalB'];
$Totaal = $GetalA + $GetalB;
echo "Het totaal is " . $Totaal;
```

Met het commando 'lees' bedoelen we dat er gebruikersinput wordt opgevraagd. In PHP lezen we in dit voorbeeld een formulier uit middels de `$_POST` methode. Het formulier heeft dus 2 vakjes, namelijk `GetalA` en `GetalB`.

Het commando 'Schrijf' doet exact het tegenovergestelde, het laat namelijk gegevens aan de gebruiker zien. In PHP gebruiken we hiervoor het commando 'echo' zoals in het voorbeeld ook zichtbaar is.

28

Alle handelingen die onzichtbaar zijn voor de gebruiker hebben geen extra commando in een PSD. We tellen in dit voorbeeld `GetalA` en `GetalB` op om een totaal te krijgen. Dit is dus wel iets wat in je programma gebeurt, maar de gebruiker merkt hier niets van.

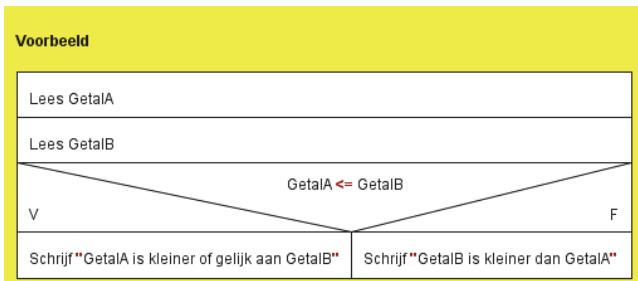
Zoals je kunt zien komt er in een PSD geen enkel stukje PHP terug. Dit komt omdat PSD's taalonafhankelijk zijn. Je kunt dus ook op basis van dit PSD bijvoorbeeld een stukje C# of Java code maken.



12.1 Uitbreiden PSD's

Omdat we in onze programma's ook gebruik maken van beslissingen (if-statement) en loops (While en For), heeft men binnen een PSD ook de mogelijkheid dit aan te geven. Hieronder staan 5 voorbeelden hoe je dit kunt weergeven.

Beslissing:

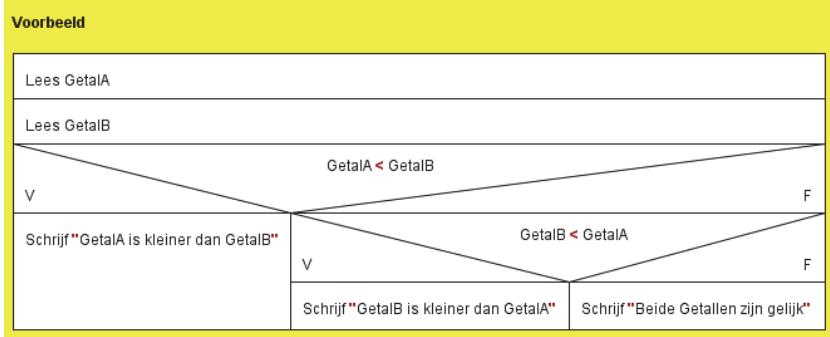


```
$GetalA = $_POST['GetalA'];
$GetalB = $_POST['GetalB'];

if($GetalA <= $GetalB)
{
    echo "GetalA is kleiner of gelijk aan GetalB";
}
else
{
    echo "GetalB is kleiner dan GetalA";
}
```

29

Dubbele beslissing:



```
$GetalA = $_POST['GetalA'];
$GetalB = $_POST['GetalB'];

if($GetalA < $GetalB)
{
    echo "GetalA is kleiner dan GetalB";
}
elseif($GetalB < $GetalA)
{
    echo "GetalB is kleiner dan GetalA";
}
else
{
    echo "Beide getallen zijn gelijk";
}
```



Herhaling (For-loop)

Voorbeeld

```
Lees GetalA  
Teller = 0  
Teller < 10  
    Totaal = Teller * GetalA  
    Schrijf Teller " x " GetalA " = " Totaal  
    Teller = Teller + 1
```

```
$GetalA = $_POST['GetalA'];  
  
for($Teller = 0;$Teller < 10;$Teller++)  
{  
    $Totaal = $Teller * $GetalA;  
    echo $Teller . " x " . $GetalA . " = " . $Totaal;  
}
```

Herhaling (While-loop)

Voorbeeld

```
GetalA = 10  
GetalB = 16  
Teller = 0  
Zolang GetalA < GetalB  
    GetalA = GetalA + 1  
    Teller = Teller + 1  
  
Schrijf "GetalA is " . $Teller . " kleiner dan GetalB"
```

30

```
$GetalA = 10;  
$GetalB = 16;  
$Teller = 0;  
  
While($GetalA < $GetalB)  
{  
    $GetalA++; //verkorte versie $GetalA = $GetalA + 1;  
    $Teller++; //verkorte versie $Teller = $Teller + 1;  
}  
  
echo "GetalA is " . $Teller . " kleiner dan GetalB";
```

12.2 Structorizer

Om PSD's te kunnen maken gebruiken we het programma Structorizer. Je kunt een link naar de website vinden in de map bestanden op ItsLearning. Zorg ervoor dat je ook Java hebt geïnstalleerd, anders werkt het programma niet.

