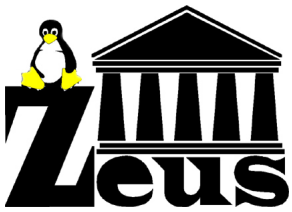

Zeus Werkgroep Informatica

Basiscursus PHP

Bernard Grymonpon (bernard@openminds.be)
Zeus Werkgroep Informatica, http://www.zeus.rug.ac.be/
Openminds b.v.b.a., http://www.openminds.be/

Inhoudsopgave

Voorwoord

Deze cursus is geen afzonderlijke leercursus, maar een begeleidende tekst die hoort bij de lessenreeks rond PHP die door Zeus Werkgroep Informatica `http://www.zeus.rug.ac.be` gegeven wordt. Deze lessenreeks wordt verzorgd door Bernard Grymonpon, zaakvoerder van Openminds bvba `http://www.openminds.be`.

De voorbeelden die hierbij horen zijn grotendeels overgenomen van voorbeelden uit de PHP handleiding.

Wanneer u delen van deze cursus wenst te hergebruiken, kan u altijd contact opnemen met de auteur op het mailadres *bernard@openminds.be*.

Hoofdstuk 1

PHP

All will be right at the end...

1.1 Wat is PHP

We beginnen met een klein stukje uit de handleiding van PHP 3.0. PHP versie 4 verschilt inzake doelstelling weinig tot niets.

> PHP (...) is an HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl with a couple of unique PHP-specific features thrown in. The goal of the language is to allow web developers to write dynamically generated pages quickly.

We kunnen PHP samenvatten als een scriptingtaal die qua syntax gebaseerd is op C, Java en Perl en met het doel programmeurs een taal te geven om snel dynamische webpagina's te ontwikkelen. De mogelijkheden van PHP waren oorspronkelijk vrij beperkt maar door de grote populariteit van PHP is daar vrij snel verandering in gekomen. PHP is gegroeid tot een van de grotere scriptingtalen die er bestaan en is een van de concurrenten van andere scriptingtalen. Verder is PHP een open source project, wat wil zeggen dat iedereen die het wil er toevoegingen en aanpassingen aan kan maken om aan zijn eisen te voldoen.
De naam PHP staat voor "PHP Hypertext Preprocessor". PHP is de taal, hypertext slaat op de HTML waarin het ingebakken kan worden en "preprocessor" slaat op het feit dat de php-code uitgevoerd wordt voor hij doorgestuurd wordt naar de browser.

Een klein voorbeeld is het alomgekende 'hello world'-programma (zie 1.1). Aangezien dit het eerste programma is dat iedereen moet krijgen wanneer een nieuwe taal aangeleerd worden, kunnen we u dit programma niet onthouden. Dit zou er bijvoorbeeld zo uit kunnen zien...

```
<?php
echo "Hello world";
?>
```

Figuur 1.1: 'Hello world', ons eerste php programma

1.2 De geschiedenis

Op het einde van 1994 heeft Rasmus Lerdorf een programma in Perl geschreven die enkele basisfuncties uitvoerde die hij veel nodig had. Op dat moment was er enkel sprake van een perl-parser als cgi-module. Gedurende een drietal jaar werd de parser uitgebreid en uiteindelijk kregen we PHP/FI 2.0... Op dat moment waren er al vele gebruikers van PHP. Wanneer echter Zeev Suraski en Andi Gutmans een volledige nieuwe parser uitbrachten in 1997 begon PHP aan zijn opgang. De nieuwe parser werd PHP 3.0 gedoopt en werd gepatched waar nodig. Er waren grote veranderingen tussen versie 2 en versie 3, zeker qua syntax. De nieuwe syntaxregels werden overgedragen naar versie 4, die ongeveer begin 2000 stabiel was. De grote verandering van versie 3 naar 4 was de parser die volledig herschreven werd en "Zend" gedoopt werd. Deze nieuwe parser is veel sneller en stabieler in vergelijking met versie 3. Qua functionaliteit kwam er echter weinig nieuws bij. Slechte enkele commando's werden uitgebreid en twee commando's werden verwijderd. Enkele belangrijke vernieuwingen waren het betere session-management en de mogelijkheid om gemakkelijker uitbereidingen te schrijven voor PHP.

1.3 Enkele eigenschappen van PHP

Onder de vele eigenschappen van PHP kunnen we toch wel enkele opmerkelijke vinden die PHP zijn eigen karakter geeft.

1.3.1 Weinig eisen

PHP-pagina's stellen bijzonder weinig eisen aan de browser van de mensen die onze pagina's gaan bezoeken. Aangezien er normaal geen interactie van de browser verwacht wordt (opstarten van java-programma's, javascript uitvoeren,...) kan zelfs een heel simpele tekstmode-browser gebruikt worden voor het ontwikkelen en testen van de gemaakte scripts. We kunnen natuurlijk Javascript meesturen, die dan uiteindelijk toch door de browser moet uitgevoerd worden, maar PHP heeft hier eigenlijk niets mee te maken, het is eerder een uitbreiding van de programmeur. Deze methode wordt veel gebruikt voor de controle van invoervelden van formulieren, maar later wordt hier verder op ingegaan.

Ook aan de serverkant is de belasting eerder laag. De PHP-parser is volledig herschreven voor versie 4 en er werd rekening gehouden met de belasting van het onderliggende systeem. Wanneer we een kleine server nodig hebben die slechts af en toe enkele pagina's moet verwerken, dan hebben we genoeg aan een lichte machine. Wanneer we een drukbezochte server hebben die per minuut verschillende pagina's moet parsen en server, dan moeten we kiezen voor een

machine met veel RAM. De processorkracht is hier minder van belang.

1.3.2 Gemakkelijk aan te leren

PHP heeft een syntax die vergelijkbaar is met verschillende programmeertalen. Wanneer we gewoon zijn in C,C++ of java te werken zullen we heel snel de overgang kunnen maken aangezien de verschillen miniem zijn. De syntax is eigenlijk een samenvoeging van de syntax van PERL (denk dat PHP oorspronkelijk "a quick perl script" was...) en C. Declaraties van variabelen zijn niet nodig, net als in Perl. De structuur van classes, syntax en groeperingen zijn overgenomen van de syntax van C en aanverwante talen.
Voor de beginnende programmeur is het wegvallen van de declaraties al een hele opluchting. De C/Perl-syntax is een heel leesbare syntax, zodat kleine programma's niet moeilijk te verstaan zijn.

```
<html>
  <head>
    <title>Example</title>
<!-- Changed by: Berreke, 11-Apr-2002 -->
  </head>
  <body>

    <h1>Hallo!</h1>

<?php
echo "Vandaag is het " . strftime ("%x") . ".";
?>

  We heten u welkom op deze PHP lessenreeks, en ...<br>
  ...<br>
  </body>
</html>
```

Figuur 1.2: Een tweede voorbeeld

1.3.3 Functionaliteit

PHP heeft door de jaren heen heel veel uitbereidingen gekend. Veel programeerbibliotheken werden toegevoegd, en de API die geleverd werd door de library werd gewoon –nou ja, bijna– doorgegeven aan de gebruiker. Zo kunnen we heel gemakkelijk gebruik maken van bijvoorbeeld XML, databases en grafische files, doordat de verschillende bibliotheken verwerkt werden in PHP. Wanneer we iets specifieks willen, kunnen we ook vrij gemakkelijk een nieuwe library toevoegen aan PHP, zodat we onze eigen functies kunnen oproepen in een dynamische webpagina. Ondertussen zijn er veel verschillende toevoegingen gebeurd aan PHP, zodat we kunnen spreken van een "vrij complete" taal inzake web-programmatie.

Door de jaren heen zijn er ook heel veel gebruikersgroepen ontstaan op het internet. Veel van deze gebruikersgroepen hebben ook extra aanvullingen gemaakt voor PHP en deze vrijgegeven

voor publiek gebruik. Deze extra stukjes code zijn soms nutteloos, maar af en toe vinden we toch een pareltje die we zelf kunnen gebruiken...

1.3.4 Snelheid

De parser is voor PHP4 volledig herschreven met snelheid als hoofddoel. De Zend-engine is het resultaat van vele uren programmeerwerk door de makers van PHP, maar het mag er zeker zijn. Zend is een van de snelste parsers die er momenteel bestaat. Wanneer we PHP volledig als module gaan compileren in Apache en hem niet als een CGI-script gaan gebruiken wordt de parser razendsnel.

1.3.5 Veiligheid

De scripts worden welliswaar op de server uitgevoerd, maar mits een correcte configuratie kunnen deze scripts uitgevoerd worden als de eigenaar van de file, en niet als de eigenaar van de webserver. Wanneer u dit niets zegt hebt u hier waarschijnlijk weinig aan, maar voor systeembeheerders is dit een hele zorg minder, aangezien de programma's geen extra mogelijkheden krijgen aangezien ze als een gewone gebruiker worden uitgevoerd. De scripts kunnen bij een goede webserver per directory aan of afgezet worden, zodat enkel gebruikers die we vertrouwen gebruik kunnen maken van PHP-script. Het grootste gevaar schuilt in het feit dat PHP-bestanden, wanneer we niet geïnterpreteerd worden gewoon als tekstbestanden verschijnen in de browser van de gebruiker. Hier kan een goede configuratie van de server echter een oplossing bieden.

1.4 Waarom PHP?

Eerst en vooral kunnen we hier opmerken dat PHP volledig gratis te gebruiken is. Wanneer we een server willen opzetten op het net, kunnen we gemakkelijk PHP gaan gebruiken voor onze webpagina's. Verder is het platform waarop de webserver draait die de PHP-pagina's interpreteerd eigenlijk van geen belang. PHP wordt vooral samen gebruikt met de Apache webserver. Deze webserver is beschikbaar voor *NIX platformen en sinds kort ook voor het Windows-platform. De beschikbaarheid van PHP is er alleen maar door uitgebreid en steeds meer mensen vinden hun weg naar PHP. Wanneer we kiezen voor Linux of *BSD als operating system is onze server volledig uitgerust met gratis software en hebben we toch een van de veiligste en stabielste systemen die mogelijk zijn.

1.4.1 Enkele verschillen met Java en C

Er bestaan 2 grote verschillen tussen Java en PHP.

Een eerste is de plaats waar het programma uitgevoerd wordt. Het Java-programma (of applet) wordt uitgevoerd op de machine van de persoon die aan het surfen is en niet de server waar de pagina vandaan komt. Dit houdt in dat speciale protocollen en programma's moeten gebruikt worden om aan extra gegevens te kunnen komen op de server (stel een databank).

Wanneer we een databank willen benaderen met een Java-applet moet de databank-server een extra beveiliging laten vallen, namelijk het feit dat enkel de locale machine moet kunnen verbinden. We moeten de verbinding voor iedereen openzetten, aangezien we willen dat iedere machine op het internet (die de applet draait) aan de databank kan. Aangezien PHP op de lokale machine wordt uitgevoerd, is hier geen probleem, enkel de lokale machine moet de databank kunnen benaderen.

Een tweede (kleiner) probleem is dat een beginner veel meer problemen zal hebben om een applet te schrijven die iets kleins doet, dan een PHP-script dat hetzelfde doet. Dit is in de eerste plaats te danken aan de gemakkelijke syntax van PHP en het feit dat er geen rekening moet gehouden worden met het feit dat we eigenlijk een programma aan het schrijven zijn voor een webpagina.

1.4.2 Verschillen met Javascript

Javascript is eerder beperkt en wordt uitgevoerd door de browser van de surfer. Deze browserafhankelijkheid geeft het probleem dat iedere browser op de markt de specifieke commando's moet kunnen. Dit zou echter de ideale wereld zijn, maar daar leven we niet in. De ondersteunde commando's verschillen van browser tot browser.
Javascript is verder ook vrij beperkt qua uitbreidingen. Connecties naar databanken en dergelijke worden niet ondersteund en zullen waarschijnlijk ook nooit ondersteund worden. Javascript wordt meer gebruikt om kleine controlles uit te voeren op invoer in forms en daarvoor is het ideaal geschikt. Verder wordt het ook veel gebruikt om de opmaak van pagina's iets extras te geven.

1.4.3 Verschillen met ASP en .NET

De dichtse benadering voor PHP is wel ASP (of .NET). Het is ook een scriptingtaal dat aan de serverside uitgevoerd wordt. De syntax was oorspronkelijk gebaseerd op die van visual basic, ondertussen kan er gebruik gemaakt worden van JScript, Visual Basic of C. Er zijn ook vele verschillende uitbreidingen mogelijk, waardoor de inzetbaarheid ook groot is. Een groot nadeel ten opzichte van PHP is het feit dat ASP enkel maar goed ondersteund worden op Microsoft Windows platformen. De uitbreidingen zijn allemaal geschreven naar Microsoftproducten toe. Aangezien de Microsoft-producten eerder slechte resultaten geven als het gaat om stabiliteit en performance op een gewone machine, is ASP een beetje benadeeld door niet platform-onafhankelijk te zijn. Er zijn echter al ASP-servers voor Linux en scripts die ASP vertalen naar PHP.

ASP heeft echter wel het voordeel dat het de gigantische Microsoft-promotie-machine achter zich heeft. Daardoor is ASP heel snel opgekomen als een van de grotere scripting-talen voor webpagina's.

ASP heeft ongeveer dezelfde eigenschappen zoals PHP. Wanneer we echter platformonafhankelijk werken moeten we kiezen voor PHP. De oplossingen voor ASP onder Unix zijn nog te beperkt of te traag.

1.5 De versie van PHP bepalen

Via het eenvoudig scriptje van figuur 1.3 kan je altijd de versie en alle extra onderdelen van PHP vinden die op de server geïnstalleerd zijn waar het script uitgevoerd wordt.

```
<?php

phpinfo();

?>
```

Figuur 1.3: PHPinfo

1.6 Nog enkele opmerkingen

Je hebt het misschien al gemerkt maar de term voor een PHP-programma is een beetje vaag. Soms wordt er verwezen met de naam "PHP-script", soms met de naam "PHP-programma" of gewoon met de vermelding "webpagina". Normaal zou de naam een mengeling van de 3 moeten zijn want uiteindelijk is het een webpagina (de surfer merkt het verschil niet, enkel de extensie van de pagina is anders), die door de webserver vanuit een script is gemaakt. De naam programma kunnen we ook gebruiken aangezien de scripts heel uitgebreid kunnen worden en volwaardige programma's kunnen worden... Doorheen deze cursus zullen de namen los door elkaar gebruikt worden, terwijl voor puur technische referenties de term "PHP-script" nog steeds het best de werkelijkheid benaderd.

Hoofdstuk 2

Variabelen gebruiken en program-flow

Variabelen hebben altijd al bestaan in programmeertalen. We gebruiken ze om gegevens doorheen het programma bij te houden, te vergelijken of door te geven aan andere delen van het programma. Bij PHP is dit ook zo, vandaar dit hoofdstuk over het hoe en waarom van variabelen.

De andere bouwstenen van programma's zijn lussen en controllestructuren. Door deze twee te combineren gaan we een programma schrijven, en kunnen we reeds enkele kleine programma's maken.

2.1 Variabelen

We zien dat een variabele gewoon wordt aangeduid door er een $ teken voor te zetten. Deze schrijfwijze is ontleend aan Perl, de taal waar de eerste PHP-parser in geschreven was. De variabelen werden gewoon verder gebruikt in de parser, dus was de keuze voor de Perl-stijl vrij duidelijk. De parser is ondertussen een C-programma geworden, maar de syntax is gelijk gebleven. Iedere variabele bestaat dus uit een 'dollar'-teken en de naam van de variabele.

Een variabele moet ook altijd een unieke naam hebben, om hem te onderscheiden van de andere variabelen die in het programma gebruikt worden. Wanneer we echter een variabele met dezelfde naam gebruiken wordt hij (ongeacht het type) overschreven met de nieuwe inhoud.

PHP kent 6 verschillende types van variabelen. Deze zijn integer (gehele getallen), double (kommagetallen), bool (waar of vals, enkel beschikbaar vanaf PHP4), array (waar we later heel uitgebreid op terugkomen), string (tekenreeksen) en object (waar we ook later op terugkomen). Een groot verschil tussen PHP en andere programmeertalen is het gebruik van deze types. Aangezien er geen declaratie nodig is, moet de gebruiker niet weten hoe PHP zijn variabele opslaat, of welk type hij eraan geeft. Dat gebeurt allemaal automatisch.

Een groot verschil ten opzichte van C en Java is het feit dat we geen declaraties moeten geven

van de variabelen. Wanneer we een variabele nodig hebben kunnen we die gewoon gebruiken op de plaats waar we die voor het eerst willen gebruiken.

Deze constructie zal velen aanzetten tot vreugdekreten, maar toch moeten we hier heel voorzichtig mee zijn. Wanneer we enkele voor- en nadelen op een rijtje zetten zal veel duidelijk worden...

De voordelen zijn vrij eenvoudig op te noemen. We kunnen waar we willen, wanneer we willen nieuwe variabelen in het leven roepen. We moeten er geen rekening mee houden wat er nu in de variabele zal komen, en als deze variabele veel of weinig geheugen zal gebruiken. Dit draagt ertoe bij dat we heel snel kunnen programmeren aangezien we geen duizend keer terug naar boven moeten om de variabele te declareren of om de juiste commando's te tikken om geheugen te voorzien voor de variabele.

De nadelen zijn in het eerste opzicht niet te vinden, maar ze zijn er toch. Wanneer we een variabele gebruiken die nog niet ingevuld is zullen we hierover geen melding krijgen. PHP interpreteert dit als een lege declaratie, en de bewuste variabele wordt gewoon vervangen door... niets. Normale programmeertalen zouden melding geven dat we gebruik maken van een 'niet-gedeclareerde' variabele. Schrijffouten leiden dikwijls tot deze (moeilijk opspoorbare) fouten.

Het stukje code in figuur 2.1 kan veel duidelijk maken.

Let op, de namen van variabelen zijn **niet** hoofdlettergevoelig, in tegenstelling tot java, c en andere hogere programmeertalen!

2.1.1 Enkele voorbeelden

```
<?php

$een_variabele="Hello world";
$a = 3;
$b = 4.5;
$PI = 3.1415;
$c = $a + $b * $PI;

echo $a;
echo " - ";
echo $b;
echo " - ";
echo $c;
echo " - ";
echo $PI;
echo " - ";
echo $een_variabele;

?>
```

Figuur 2.1: Variabelen gebruiken

In het eerste voorbeeld (2.1) zien we hoe we een string en enkele gehele en reële getallen kunnen toekennen aan een variabele. Met behulp van het echo commando zien we hoe we die kunnen uitschrijven naar de browser.

Let erop dat op een magische wijze telkens beslist wordt welk type er gebruikt wordt. De *casting* gebeurt automatisch, of zo lijkt het toch.

```
<?php

$a = 3;
echo $a . " - ";

$a = "Iets anders";
echo $a . " - ";

$a = 15.234356567;
echo $a . " - ";

?>
```

Figuur 2.2: Variabelen gebruiken

Het stukje code in 2.2 toont aan dat we gemakkelijk variabelen kunnen hergebruiken. In C of Java was dit onmogelijk geweest, aangezien je een variabele onmogelijk een geheel getal, een string en een reëel tegelijk kan laten zijn. In PHP is dit echter heel gemakkelijk te doen.

2.2 Operators

Merk op dat we in het laatste voorbeeld de dubbele "echo" commando's hebben vervangen door enkele door de tekst met een punt(.) aan elkaar te hangen. De punt-operator wordt dan ook de "concatenatie"-operator genoemd.

Laten we ook eens enkele operators bekijken die we kunnen gebruiken...

2.2.1 rekenkundige operatoren

Dit behoeft weinig of geen uitleg... De rekenkundige regels worden toegepast zoals die gedefineerd worden in de wiskunde.

operator	voorbeeld	verklaring
-	-$a	Convertie van teken.
+	7 + 2	Telt 7 en 2 op.
-	7 - 2	Trekt 2 van 7 af.
*	7 * 2	Vermenigvuldigt 7 en 2.
/	7 / 2	Deling van 7 door 2.
%	7 % 2	Berekent de rest bij een gehele deling.

2.2.2 vergelijkingsoperatoren

Zoals de naam al doet vermoeden dienen deze operatoren om waarden met elkaar te vergelijken. Onderstaande tabel toont de operatoren waarmee PHP bekend mee is.

operator	betekenis
==	is gelijk aan
<	is kleiner dan
>	is groter dan
<=	is kleiner dan of gelijk aan
>=	is groter dan of gelijk aan
!=	verschilt van
<>	verschilt van

We vestigen nog eens de aandacht op het verschil tussen de operatoren == en =. De eerste vergelijkt 2 variabelen, terwijl de tweede een waarde aan een variabele toekent! Voor C en Java mensen zal dit geen probleem zijn, maar Pascal en Visual-basic mensen zullen hier echter wel een probleem mee hebben.

```
<?php
$i = 4;
if ($i = 7) echo ("seven");
//"seven" is printed every time!
?>
```

Figuur 2.3: Het verschil tussen == en =

De code in figuur 2.3 is syntactisch juist, waardoor we geen foutmelding zullen krijgen. Om dergelijke onoplettendheden te voorkomen kunnen we in ons if statement eerst de waarde stellen en rechts van de operator de variabele. Indien we dan dezelfde fout maken krijgen we wel een foutmelding, want dan proberen we de waarde van de variabele toe te kennen aan het nummer 7, en dat gaat natuurlijk niet. Deze manier van programmeren vergt wel enige discipline van de programmeur.

2.2.3 logische operatoren

De logische operatoren combineren twee of meerdere statements. In onderstaande tabel zie je hoe ze genoteerd worden in php.

operator	operatornaam
`&&`	logische “*and*”
`and`	logische “*and*”
`||`	logische “*or*”
`or`	logische “*or*”
`xor`	“*exclusive or*”
`!`	“*not*”

2.2.4 Strings aan elkaar koppelen

We hebben al voorbeelden gezien waarin de punt-operator werd opgenomen. De functie van deze operator is het “*aan elkaar kleven*” van strings. Verwar deze operator niet met het decimale punt. Je onderscheid ze van elkaar door er spaties omheen te zetten, zoals in figuur 2.4.

```
<?php

echo ("4" . "5" . " "); // prints 45
echo (4 . 5 . " "); // prints 45
echo (4.5); // prints 4.5 ("." is interpreted as decimal point)

?>
```

Figuur 2.4: Gebruik van de punt-operator

2.3 Commentaar

Net als in elke andere programmeertaal kan ook in PHP commentaar worden opgenomen. Een hash (#) of twee *forward* slashes zorgen ervoor dat de rest van een lijn als commentaar wordt beschouwd. Om volledige blokken in commentaar om te zetten[1] gebruiken we `/*` en `*/`.

Figuur 2.5 is een klein voorbeeldje om dit alles duidelijk te maken...

We zien duidelijk de 3 verschillende types van commentaar. Denk eraan dat het onmogelijk is om (net zoals in andere programmeertalen) blokken commentaar te nesten.

2.4 Program-flow

Onder dit vreemde engelse woord verstaan we alles die het programma laat reageren op bepaalde omstandigheden. Hiermee bedoelen we in de eerste plaats de controlestructuren zoals iedereen ze kent uit de andere programmeertalen. Er zal hier weinig aandacht aan besteed

[1] handig bij het debuggen

```
<?php
/* After a stupid bet in the pub, I decide to start learning PHP.
   This all happened 26th of june. But hey, it's too late to complain
   now. Let's get on it... */

//First say hello.
echo "Hello, and welcome to my homepage!";  #too drunk to continue
# TODO: Complete this page when sober.
?>
```

Figuur 2.5: Commentaar invoegen

worden, aangezien we doorheen de cursus veel voorbeelden zullen zien van deze structuren. Ze staan hieronder allemaal opgesomd met een kort voorbeeldje waardoor de werking duidelijk wordt.

2.4.1 If-lussen

Iedereen die al geprogrammeer heeft zal wel de if-structuur kennen. Deze zorgt ervoor dat er een stukje code kan uitgevoerd worden wanneer er slechts aan bepaalde condities is voldaan. Om deze condities te vinden gebruiken we de logische operatoren.

```
<?php
if ($a == 0) {
  $title = "zero";
} else {
  $title = "not zero";
}
echo $title;
?>
```

Figuur 2.6: Een kleine if-lus

De code in figuur 2.6 geeft duidelijk weer wat er gaat gebeuren. Wanneer we aan de voorwaarde voldoen dat $a nul is, dan schrijven we de tekst “zero” uit. In het andere geval schrijven we “not zero” uit.

2.4.2 Verkorte if-notatie

Wie bekend is met C/C++, java of nog enkele andere programmeertalen, kennen de verkorte if-notatie vast wel. Ook in php kan je ze gebruiken. Voor de mensen die nog niet zo veel programmeer ervaring hebben leggen we ze nog eens kort uit.

De verkorte if-notatie ziet er als volgt uit: *condition ? true-statement : false-statement*. Eerst wordt de conditie gevalueerd en als de waarde true (respectievelijk false) oplevert, wordt de

waarde van het *true-statement* (respectievelijk *false-statement*) teruggegeven. Een voorbeeld wordt gegeven in 2.7, die dezelfde uitvoer geeft als 2.6.

```
<?php
$title = $a == 0 ? "zero" : "not zero";
echo $title;
?>
```

Figuur 2.7: De verkorte if-notatie

2.4.3 For-lussen

Soms willen we een bepaald aantal maal iets herhalen. Wanneer we vooraf altijd weten hoeveel keer het herhaald moet worden moeten we de for-lus gebruiken. Voorbeeld 2.8 print de tabellen van vermenigvuldiging uit

```
<?php

for($i=0 ; $i<=10; $i++){
  for($j=0; $j<=10; $j++)
echo $i * $j . "  ";
  echo "<br>";
}

?>
```

Figuur 2.8: De for lus

We zien hier ook een eerste gebruik van blokken in PHP code. Blokken kunnen we gebruiken om bepaalde commando's te groeperen. Een blok wordt gemaakt zoals we in C of Java gewoon zijn. Dit betekend dat we het blok openen met `{` en het sluiten met `}`.

Blokken kunnen we gebruiken op elke plaats waar een commando kan voorkomen want de blok wordt beschouwd als 1 commando.

Wanneer we hier een professionele oplossing zouden moeten maken kunnen we hier tabellen gebruiken om de opmaak wat beter te verzorgen.

2.4.4 While-lussen

Wanneer we echter vooraf niet weten hoeveel maal we iets willen herhalen kunnen we de while-lus gebruiken. Deze is handig om resultaten op te halen.

Het voorbeeld in figuur 2.9 toont de nummers 1 tot 10 in de browser met behulp van een while-lus.

```
<?php
$i = 1;
while ( $i <= 10 ) {
  echo $i;
  $i++;
}
?>
```

Figuur 2.9: De while lus

While-lussen worden veel gebruikt bij het onderzoeken van array's en database-resultaten, omdat we niet weten hoeveel resultaten er nu precies teruggegeven werden. De while-lus heeft 2 verschillende vormen, namelijk de `while(conditie){commando's}` en `do{commando's}while(conditie)`. De laatste constructie verschilt in de eerste doordat de commando's altijd minstens 1 maal uitgevoerd zullen worden.

2.4.5 Switch

Wanneer we een variabele met heel veel mogelijke waarden willen vergelijken kunnen we ofwel veel if-blokken gebruiken, ofwel gebruiken we het switch-statement. In andere programmeertalen wordt dit ook soms aangeduid als "case". Het voorbeeld in figuur 2.10 gebruikt een switch-statement.

2.4.6 Een overzichtje

Het voorbeeld in figuur 2.11 toont nog eens het gebruik van de 3 bovenstaande lusstructuren. Dit voorbeeld heb ik schaamteloos overgenomen van de tutorial van PHP zelf. We zien duidelijk de twee verschillende vormen van de while-lus.

2.4.7 break en continue

Soms hebben we situaties waar we de uitvoer van een bepaalde lus willen stopzetten (bv bij een fout), of wanneer we onmiddelijk aan een nieuwe iteratie van de lus willen beginnen. Hier komen break en continue van pas. Wanneer we ergens in onze lus een *break* staat, dan zal de lus onmiddelijk gestopt worden, en gaat de uitvoer van het script verder op de eerste regel na de lus.

Wanneer we echter *continue* gebruiken wordt ook de uitvoer van de huidige lus gestopt, maar gaan we verder met de lus met de volgende iteratiewaarde.

Met beide functies kunnen we een parameter meegeven hoeveel lussen we willen break-en of continue-en. Deze parameter moet echter niet tussen haakjes staan zoals we zullen zien in het voorbeeld...

Het voorbeeld in 2.12 kan meer duidelijk maken.

```
<?php
if ($i == 0) {
  print "i equals 0";
}
if ($i == 1) {
  print "i equals 1";
}
if ($i == 2) {
  print "i equals 2";
}

switch ($i) {
 case 0:
   print "i equals 0";
   break;
 case 1:
   print "i equals 1";
   break;
 case 2:
   print "i equals 2";
   break;
}
?>
```

Figuur 2.10: De switch samen met de if-versie

We zien dat de *continue* de lus verderzet, maar de *break* zorgt ervoor dat er meteen naar de laatste lijn gesprongen wordt.

We merken wel op dat door sommige mensen de *break* en *continue* constructies als slechte constructies bestempelen. Toch kan het soms zijn dat door het gebruik van een eenvoudige *break* of *continue* het programma een stuk mooier ineen zit. We raden dan ook aan om deze twee commando's enkel te gebruiken als er geen directe oplossing is voor het probleem.

Er bestaan nog enkele andere structuren, die vooral slaan op array's. We gaan hier dieper op in in het hoofdstuk van array's.

2.5 Je project opsplitsen

Grotere programmeerprojecten worden als snel onoverzichtelijk. Wanneer we programmastukken die regelmatig nodig zijn in functies gaan plaatsen kunnen we al een beter overzicht houden. Veelgebruikte functies (bv database-toegang) kunnen we dan nog eens onderbrengen in een aparte file, zodat we die gemakkelijk kunnen hergebruiken in andere projecten.

```
<?php

// Conditionele sprong

if ($a) {
  print "a is waar<BR>\n";
} elseif ($b) {
  print "b is waar<BR>\n";
} else {
  print "a en b zijn beiden niet waar<BR>\n";
}

// Lussen

do {
  $c = test_something();
} while ($c);

while ($d) {
  print "ok<BR>\n";
  $d = test_something();
}

for ($i = 0; $i < 10; $i++) {
  print "i=$i<BR>\n";
}

?>
```

Figuur 2.11: De verschillende mogelijkheden op een rij (zonder switch)

```
<?php
echo "Voorbeeld van break en continue... <br>";

$i = 0;

while($i++ < 100){
  // Enkel de oneven getallen
  if ($i % 2 == 0)
continue;

  // Stoppen als we aan 90 zijn
  if ($i > 89)
break;

  echo $i . "<br>";
}
echo "Einde!"

?>
```

Figuur 2.12: Break en continue

2.5.1 Functies

Wanneer we een stukje programma meermaals nodig hebben (neem bv een machtberekening) dan kunnen we dit in een aparte functie gaan gieten. Deze functie kunnen we dan net als andere ingebouwde php-functies gaan gebruiken doorheen ons project.

Met functies kunnen we heel veel uitvoeren, maar hier geven we enkel een basis. Een voorbeeld is gegeven in figuur 2.13.

In dit fragment zien we 2 functies. De eerste functie “schrijfuit” neemt een parameter, en gaat het commando “echo” uitvoeren met enkele waarden die aan elkaar ge-concat worden. De tweede functie is een machtfunctie. De macht wordt in de functie berekend in de variabele $result, en die wordt uiteindelijk met “return” teruggegeven aan het oproepende programma. Met die return kan je dus iets teruggeven dat dan verder gebruikt wordt.

De laatste lijn is iets speciaals. We gebruiken de schrijfuit-functie hier om een tekst uit te schrijven. In PHP kan dit perfect, want we hebben in onze functie-header niet moeten opgeven welk type de variabele $getal zou zijn! In C of Java zou dit niet mogelijk zijn, enkel overloading zou hier een oplossing kunnen bieden.

We moeten echter aanraden niet te veel op dit gedrag voort te gaan, want het is een slechte programmeerstijl. Een functie moet een welbepaalde gedefinieerde taak uitvoeren. Hoe strikter die taak omschreven kan worden, hoe beter de functie. Soms is het wel handig om van deze “feature” gebruik te maken.

Doorheen de cursus komen nog vele voorbeelden van functies aan bod, en daarom zullen we er hier niet meer zo diep op ingaan.

```
<?php

function schrijfuit($getal){
  // deze functie schrijft een getal uit met een stukje tekst ervoor
  // en een <br> om een nieuwe regel te nemen
  echo "Het getal is: " . $getal . "<br>";
}


function macht($getal, $macht){
  // deze functie berekent (op een brute manier) de
  // macht van een getal en geeft deze terug...
  $result = 1;

  for($i = 0; $i < $macht; $i++)
$result = $result * $getal;

  // teruggeven van een waarde
  return $result;
}


$iets = 3;
schrijfuit($iets);

// 2 tot de 3de berekenen met onze eigen functie
// en het resultaat opslaan in de variabele $uitkomst
$uitkomst = macht(2,$iets);
schrijfuit($uitkomst);

// vreemd, of juist niet?
schrijfuit("dit is geen getal, en toch werkt dit...");

?>
```

Figuur 2.13: Een machtfunctie

Oplettende C en Java programmeurs zullen opmerken dat er nergens een functie “main” is (of een equivalent daarvan). De main-functie bevat normaal het hoofdprogramma, en daar begint de uitvoer van het programma. In PHP begint de uitvoer echter aan het begin van de file. Dit is een logisch gevolg van het feit dat we PHP-code gewoon tussen HTML code kunnen invoegen. Wanneer we ergens een main-functie zouden moeten hebben, moeten we daar alle HTML plaatsen en verliezen we de functionaliteit van PHP.

2.5.2 require() en include()

Wanneer we ons project opsplitsen in verschillende files kunnen we de functies “require(filenaam)” en “include(filenaam)” gebruiken. Met deze functies kunnen we een file inplakken op de plaats waar we het commando plaatsen.

Het verschil tussen require en include is vrij simpel. Require zal altijd vervangen worden door de inhoud van de file. Wanneer een PHP-script uitgevoerd wordt gaan PHP eerst op zoek naar alle require-statements, en gaat die onmiddelijk gaan vervangen door de file die erin vermeld staat. Include daarintegen wordt niet onmiddelijk vervangen, dat gebeurt enkel wanneer de “include” uitgevoerd wordt. Wanneer deze dus door een if-constructie wordt voorafgegaan, en de conditie wordt niet voldaan, dan gaat deze file niet gelezen en ingevoegd worden.

Include kan je gebruiken om een bepaalde module in te voegen wanneer die aanwezig is, of om een bepaalde file meerdere keer te include-en in een for-lus. De filenaam kan dan bijvoorbeeld genummerd worden.

Er zijn echter enkele opmerking die je indachtig moet zijn. Wanneer we een file met include invoegen verlaat de parser PHP mode in het begin van de file. We moeten dus terug de PHP mode starten met de opentag `<?php` en deze afsluiten met `?>`. Een file wordt ook aanzien als een volledig blok commando’s. Wanneer we die dus na een if-constructie zetten, dan moeten we (ondanks het feit dat de include slechts 1 commando is) toch gebruik maken { en }.

En opnieuw, een voorbeeld in 2.14.

Uit een include-file kunnen we zelf een return-code genereren. Deze kunnen we dan gebruiken om te kijken als de code in de file correct is uitgevoerd (met correct bedoel ik niet de uitvoer van de code, maar interne returns van de functies in de code).

Het in te voegen bestand weergegeven in figuur 2.15 en het hoofdprogramma in figuur 2.16

Hier zien we een demonstratie van het gebruik van de “return”, en hoe we telkens in de file die ingevoegd wordt, we toch opnieuw de php-mode moeten openen en sluiten.

We merken ook nog op dat er naast de twee commando’s “require()” en “include()” ook nog twee andere bestaan: “require_once()” en “include_once()”. Het enige verschil is het feit dat door de laatste constructie te gebruiken men er zeker van is, mocht het commando meermaals uitgevoerd worden met dezelfde filename, het bestand toch slechts één keer ingevoegd wordt. Dit voorkomt problemen met dubbele defineringen van bv functies of objecten. Het is dus aan te raden deze laatste constructies te gebruiken, tenzij je heel zeker weet dat een bestand geen meerdere malen ingevoegd wordt.

```
<?php
/* This is WRONG and will not work as desired. */

if ($condition)
 include($file);
 else
 include($other);

/* This is CORRECT. */

if ($condition) {
  include($file);
} else {
  include($other);
}
?>
```

Figuur 2.14: include altijd in een blok plaatsen

```
<?php

echo "Before the return <br>\n";
if (1) {
  return 27;
}
echo "After the return <br>\n";

?>
```

Figuur 2.15: Het bestand includetest.php

```
<?php
$retval = include ('includetest.php');
echo "File returned: '$retval'<br>\n";
?>
```

Figuur 2.16: Het bestand includemain.php

2.6 Nog meer over variabelen

Wanneer we een variabele declareren in het hoofdprogramma, en daarna in een functie ervan gebruik willen maken moeten we aan de functie meedelen dat we de variabele bedoelen die globaal bestaat (ook buiten de functie). Wanneer we dit niet doen gaat de functie hiervoor een nieuwe variabele aanmaken. Het voorbeeld in figuur 2.17 geeft weer wat we bedoelen.

```
<?php

$a = 3;

function doeiets(){
  echo "In de functie doeiets(): ". $a . "<br>";
}

function doeietsbeter(){
  global $a;
  echo "In de functie doeietsbeter(): ". $a . "<br>";
}

echo $a . "<br>";
doeiets();
doeietsbeter();

?>
```

Figuur 2.17: Globale variabelen

In het voorbeeld zal de functie `doeiets()` geen waarde uitschrijven voor $a. De functie `doeietsbeter()` zal dat echter wel doen.

Daar we de variabelen waarmee een functie moet werken normaal meegeven als parameter lijkt dit in de eerste plaats overbodig. Het voorbeeld in figuur 2.18 geeft echter beter weer wat er allemaal met globale variabelen bereikt kan worden.

We gaan globale variabelen vooral gebruiken om configuraties door te geven en te gebruiken doorheen het volledige programma. In het voorbeeld werd de taal veranderd midden in het programma, maar meestal zal deze taal slechts 1 maal ingesteld worden in het begin van het script.

2.7 Embedded zei u toch?

Tot slot geven we een voorbeeld (figuur 2.19) hoe we PHP kunnen mixen met HTML om een mooier resultaat te bekomen dat hetgeen we tot nu toe gezien hebben. We nemen terug de tafels van vermenigvuldiging zoals in het voorbeeld bij de for-lussen, en passen het zo aan dat het een mooie uitvoer geeft in de browser.

We zien dat er twee stukken php-code in de HTML zitten. De functies die gebruikt worden

```
<?php


function begroeting($naam){

  global $taal;

  if ($taal == "nl"){
echo "We heten " . $naam . " welkom op deze PHP inleiding! <br>";
  } else {
echo "We welcome " . $naam . " to this PHP introduction! <br>";
  }
}

$taal = "nl";
begroeting("Bernard");

$taal = "en";
begroeting("Bernard");

?>
```

Figuur 2.18: Globale variabelen nuttig gebruiken

moeten ook niet in hetzelfde blok zitten als de aanroep van de functies. Dit maakt het mogelijk om veel functies vooraf te declareren, nog voor we aan de HTML begonnen zijn. Dit wordt dan meestal bekomen met enkele `require()`-statements.

```
<html>
  <head>
<title>
  De tafels van vermenigvuldiging
</title>
  </head>
  <body bgcolor="#FFFFFF">

<?php
function vermenigvuldig($getal1, $getal2){
  return $getal1 * $getal2;
}
?>
<h1>Vermengvuldigen</h1>

<table width="90%" border=1>

<?php
// de tabel uitschrijven

for($i=0 ; $i<=10; $i++){
  echo "<tr align=\"center\">"; // de rij beginnen

  for($j=0; $j<=10; $j++)
  echo "<td>" . vermenigvuldig($i,$j) . "</td>";

  echo "</tr>"; // de rij beeindigen
}

    ?>

  </table>
  </body>
</html>
```

Figuur 2.19: PHP-code in HTML-code inbakken

Hoofdstuk 3

Internetprogrammatie

Wanneer we programmeren in een gewone omgeving, zoals bij standaard C of Java programma's hebben we één programma dat blijft doorwerken tot alles gedaan is. Wanneer we echter een E-commerce systeem in PHP willen programmeren hebben we een bijkomend probleem. De gebruiker van het E-commerce systeem gaat telkens als hij ergens klikt een pagina opvragen van de server, maar nergens wordt het verband aangetoond met de vorige pagina's die de gebruiker reeds heeft aangemaakt.

Dit voorbeeld duidt aan dat we hier en daar anders gaan moeten denken wanneer we in PHP een samenhangend geheel willen programmeren. In dit hoofdstuk gaan we even enkele belangrijke punten aanstippen.

3.1 Vertrouw nooit uw medemens

De titel kan een beetje grof overkomen, maar het geeft perfect weer wat we willen aantonen. Alle data die van de gebruiker doorgestuurd wordt naar je script moet je wantrouwen. Dit houdt in dat alles wat doorgestuurd wordt vanaf de machine van de surfer als "verdacht" moet bestempeld worden. Een klein voorbeeldje maakt alles duidelijk (het voorbeeld is fictief!):

Stel, ergens op een webpagina heb je een zoek-mogelijkheid ingebouwd. De persoon vult iets in in een veldje en dat wordt doorgestuurd naar je programma. In PHP komt de inhoud van het veldje in een variabele te staan die de naam "zoekopdracht" heeft. Deze zoekopdracht wordt doorgegeven aan een denkbeeldige SQL database in de vorm van de opdracht

```
$query = "SELECT url FROM paginas WHERE inhoud like '%$zoekopdracht%'";
DoDatabaseQuery($query);
```

De variabele "zoekopdracht" wordt vervangen door het woord dat de gebruiker opgaf, zoals bijvoorbeeld het woord "PHP".

Wanneer de gebruiker echter in het veld de tekst `foo'; DELETE FROM pagina WHERE url='` ingeeft, dan ben je al je informatie kwijt. In de praktijk zou dit voorbeeld niet gewerkt hebben, maar veel security-fouten worden gemaakt door het grote vertrouwen dat de programmeur

stelt in de gebruiker van de pagina's, of het controleren niet strikt genoeg te programmeren en teveel af te gaan op eerlijke gebruikers.

Een tweede voorbeeld zal enkele andere zaken duidelijk maken. Stel dat de surfer kan kiezen via een combobox uit verschillende files. Het PHP script toont dan de inhoud van de file. Wanneer we in de HTML-code van de aanroepende pagina de combobox als volgt opstellen

```
<SELECT NAME=bestandsnaam>
<OPTION VALUE=file1.txt>
<OPTION VALUE=file2.txt>
<OPTION VALUE=file3.txt>
</SELECT>
```

dan kan de gebruiker deze sourcecode opslaan op zijn PC en aanpassen zodat deze file eruit gaat zien als

```
<SELECT NAME=bestandsnaam>
<OPTION VALUE=file1.txt>
<OPTION VALUE=file2.txt>
<OPTION VALUE=obscurefile.txt>
</SELECT>
```

Wanneer de programmeur ervan uitging dat enkel de filenamen "file1.txt" tot "file3.txt" konden doorgegeven worden in de variabele "bestandsnaam", dan kan hier een ernstige fout gemaakt worden. De malafiede persoon kan gemakkelijk enkele files zien die normaal niet voor zijn ogen bestemd zijn.

Deze voorbeelden zijn volledig verzonnen, maar tonen wel aan dat zelfs data die in een HTML file opgeslagen werd (desnoods via de "hidden" parameter), ook kan aangepast worden. Daarom geldt er maar 1 regel: *Vertrouw nooit je gebruikers.*

Deze fouten worden meestal gemaakt in kleine scriptjes die snel moeten gemaakt worden en eingelijk maar weinig doen, te weinig eigenlijk.

3.2 Paswoorden en dergelijke

Wanneer we een gebruiker willen controleren op zijn "echtheid" dan moet er op de een of andere manier een passwoord over het netwerk verstuurd worden.

Ofwel gaan we dit passwoord moeten sturen over dit netwerk (wat dus onderschept kan worden door iemand anders), ofwel encrypteren we het eerst aan de kant van de gebruiker, en sturen we de geëncrypteerde versie door. Deze tweede oplossing lijkt veiliger, maar is het zeker niet. De geëncrypteerde versie kan onderschept worden, en door de hacker gebruikt worden om binnen te raken op de site. Hoe de encryptie gebeurt aan de kant van de gebruiker laten we hier in het midden. Enkele voorbeelden zijn javascript of java-applets.

De conclusie is simpel. Wanneer we een echt veilige site willen, moeten we wel werken met geëncrypteerde verbindingen (https://foo.bar.com/etc). Er is geen ontkomen aan. Wanneer we echter minder gevoelige data willen beschermen kan een eigen systeem de gegevens beschermen. Meer over beveiliging van gegevens vinden we verder.

3.3 De browser en de server

3.3.1 De browseroorlog

Wanneer we webpagina's willen maken die door iedere browser correct worden weergegeven zitten we met een klein probleempje. Door de vele verschillende tussen de browsers die momenteel bestaan, zijn er veel problemen in compatibiliteit. We moeten er rekening mee houden dat zogenaamde industriestandaarden als "cookies" door iedere browser verschillend wordt geïnterpreteerd, en dat we dus op veel problemen kunnen stuiten. Het is daarom geen overdreven luxe om je webpagina te testen op verschillende browsers, en verschillende versies van deze browsers. Denk er ook aan om bijvoorbeeld de browsers "links" en "lynx" onder Linux eens te testen. Velen moeten hierop terugvallen wanneer er snel iets moet aangepast worden via een terminal.

3.3.2 De server

PHP is een taal die op verschillende platformen en onder verschillende webservers kan draaien. Daar de programmeur meestal niet kan werken op de site waar die gehost wordt, is het wel aan te raden dat de programmeur in zijn ontwikkel-omgeving zo gelijk mogelijk probeert op te zetten als de omgeving waar de site uiteindelijk gaat komen. Het is soms moeilijk dit te bekomen en enkel voor grote projecten belangrijk. Denk er aan dat een database onder Windows zich soms anders gaat gedragen dan een database onder Linux.

De meeste webhosting providers die PHP aanbieden doen dit meestal met een Apache server. Wanneer je deze configuratie als standaard neemt heb je een goede kans om scripts te schrijven die compatibel zijn met andere server (omdat deze configuratie zich een beetje als standaard heeft opgedrongen).

Hoe alles verloopt tussen de browsers en de server bij het sturen van een webpagina is eigenlijk onbelangrijk voor onze lessenreeks, en wordt hier ook niet behandeld. Wanneer we onderwerpen bespreken die hierop slaan zullen we over dit onderwerp meer uitleg geven.

3.3.3 Informatie uitwisselen

Enkele zaken die we wel moeten weten zijn de volgende.

1. Vraag

   De browser stuurt naar de server een aanvraag. In de aanvraag staat de URL van de gewenste pagina en cookies die de browser bewaard heeft. De server gaat deze file

opzoeken op zijn harddisk. Dan gaat hij aan de hand van de extensie bepalen wat voor een file het is. Wanneer er nog bijkomende handelingen moeten gedaan worden gaan die eerste gedaan worden –een php-file moet nog uitgevoerd worden–... In deze vraag kan de informatie in een form ofwel in de URL zitten, ofwel in de aanvraag. Dit hangt af van de methode die gebruikt werd. Meestal wordt "GET" gebruikt en komt de informatie dus in de URL terecht. Een andere manier is "POST", deze wordt meer gebruikt voor grotere hoeveelheden data. Een ander voordeel van post is het feit dat de doorgestuurde data niet in de URL doorgegeven wordt, en dus niet "zichtbaar" is voor de gebruiker. Dit kunnen we gebruiken om paswoorden door te sturen naar de server. Combinatie van beide is ook mogelijk, PHP kan hiermee overweg.

2. Headers

   Wanneer een pagina doorgestuurd moet worden naar de browser wordt eerste een header samengesteld. Deze header bevat informatie over de soort file dat de browser gaat krijgen, cookies die gezet en verwijderd moet worden, en extra informatie wanneer de file op een andere plaats moet gezocht worden. Daarbij kunnen nog andere zaken gezet worden zoals de vraag om de pagina niet te cachen.

3. Content-type Na de header kan er nog een vermelding komen van de inhoud van de file. Meestal wordt dit gebruikt om de inhoud toch nog te veranderen. Wanneer een browser dit tegenkomt moet hij zich hieraan houden. Dit zijn echter enkel vastgelegde regels, iets waar niemand zich moet aan houden...

4. De uiteindelijke pagina Dit vereist weinig uitleg. De pagina wordt doorgestuurd in zijn uiteindelijke vorm. Wanneer we een PHP script hebben opgevraagd wordt de uitvoer van het script en de HTML die tussen het script staat (of omgekeerd) doorgestuurd.

3.4 Invoer van de gebruikers

Een veelgebruikte manier om informatie van een gebruiker te vragen is via een html-formulier. Laat ons beginnen met een voorbeeldje van een html-pagina die de naam van iemand vraagt, en wanneer hij op de "verzend-knop" drukt deze naam dan weergeeft op de **volgende** pagina.

Opmerking! De vorige zin moeten we goed indachtig zijn! We kunnen slechts aan de gegevens op het moment dat de pagina terug geladen wordt. Hetzelfde geld voor cookies die we later zullen zien.

Wanneer we een form gebruiken in een HTML-document, dan wordt wanneer we de verzend-knop indrukken de inhoud van het formulier genomen en verwerkt in de de aanvraag die in de parameter `action` staat van het formulier. Een kort voorbeeldje van een formulier in een html document wordt gegeven in figuur 3.1.

We zien dat in het voorbeeld de file form1.php opgeroepen wordt wanneer we op de "verzend"-knop drukken. De enige parameter die van de gebruiker gevraagd wordt (via een input-veld) gaat dan meegegeven worden aan het PHP-script.

Waar komt dat nu te staan in het PHP-script kan je je afvragen... Wel, de uitleg hiervan is vrij gemakkelijk. In bovenstaand voorbeeld hebben we een naam gegeven aan het tekstveld

```
<html>
  <head>
    <title>Een formulier</title>
  </head>

  <body>
    <h1>Een eerste formulier</h1>

<form method="GET" action="form1.php">
  Geef uw naam in :
  <input name="gebruikersnaam" value="naam?"><br>
  <input type="submit" value="Verzenden">
</form>

  </body>
</html>
```

Figuur 3.1: Een form in een html-document

(nl "gebruikersnaam"). In het opgeroepen PHP-script komt dit zichtbaar als een globale variabele met de naam $gebruikersnaam. Heel simpel dus, en dat is wat PHP en formulieren nu net zo gemakkelijk maakt.

```
<html>
  <head>
    <title>Het resultaat</title>
  </head>

  <body>
    <h1>Het resultaat</h1>

<?php
// de variabele van het formulier kunnen we hier zomaar gebruiken!
echo "U gaf " . $gebruikersnaam . " in!";

?>


  </body>
</html>
```

Figuur 3.2: Een form in een html-document

Verder in de cursus gaan we nog veel meer formulieren gebruiken. Om af te sluiten geven we nog een uitgebreid voorbeeld die gebruikt maakt van enkele nieuwe functies. De code wordt weergegeven in figuur 3.3.

Eerst en vooral hebben we willekeurige getallen nodig. Daarvoor gebruiken we de functie `rand(min,max)` die een getal tussen min en max teruggeeft (min en max inclusief!). Wanneer

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Raad het getal</title>
  </head>

  <body bgcolor="#FFFFFF">
    <h1>Raad het getal</h1>

<?php
// de bedoeling is dat de computer een getal in gedachten neemt
// en dat de gebruiker dit vindt. We geven hints met hoger en
// lager...

srand((double) microtime() * 100000);

// eerst kijken we als er reeds een te raden getal bestaat
if (! isset($te_raden_getal))
{
  // blijkbaar bestaat het nog niet... Een nieuw creeeren
  $te_raden_getal = rand(1,100);
} else
{
  if (isset($antwoord)){
// kijken als het antwoord te groot of te klein (of correct is)
if ($antwoord > $te_raden_getal){
  $bericht = "Uw getal is te groot!";
} elseif ($antwoord < $te_raden_getal){
  $bericht = "Uw getal is te klein!";
} else {
  $bericht = "Uw getal is correct!";
  $te_raden_getal = rand(1,100);
}
  }
}

echo $bericht . "<br>"

?>

<form method="GET" action="raad.php">
  Doe een gok:
  <input name="antwoord" value="<?php echo $antwoord?>"><br>
  <input type="hidden" name="te_raden_getal" value="<?php echo $te_raden_getal?>">
  <input type="submit" value="Gokken...">
</form>

  </body>
</html>
```

Figuur 3.3: Een getal raden

we echter vergeten de functie `srand()` aan te roepen gaan we telkens dezelfde waarde krijgen. We gebruiken daarom `srand()` met een seed-waarde die gebaseerd is op de tijd. De getallen die `rand()` gaat genereren zijn dan heel willekeurig.

We slaan dit getal op in de variabele `\$te_raden_getal`.

Verder gebruiken we ook een speciale vorm van de if-lus. De tweede keer gebruiken we het keyword `elseif`. Deze zorgt ervoor dat we gemakkelijker de structuur kunnen opbouwen.

We gebruiken ook de functie `isset()`. Deze functie kijkt als een bepaalde variabele reeds gedefineerd is, en een geeft een return-code die ofwel true of false is.

Hoofdstuk 4

Strings

In dit hoofdstuk gaan we nog een beetje dieper in op strings. We zien hoe we strings kunnen manipuleren door erin te knippen en te vervangen. We gebruiken regular expressions om strings te controleren op geldigheid (bv een email-adres of een webpagina). Verder zien we hoe we data moeten klaarmaken voor opslag. Dit laatste gaan we nog even uitbreiden naar andere variabelen...

4.1 String specifieke functies

We kunnen met `sizeof()` de lengte van een string gaan berekenen. Deze functie kan gebruikt worden op alle objecten, maar bij strings krijgen we een resultaat waar we verder mee kunnen werken. De grootte van een integer kan meestal niet verder gebruikt worden.

Bij de array's hebben we al de functies `implode()` en `explode()` gezien.

We kunnen een lange string gaan omzetten in verschillende lijnen door er overal `\\n` tussen te voegen met de functie `wordwrap()`. Deze functie neemt als parameters de string en de breedte van een lijn. Wanneer we een derde parameter meegeven zal niet het newline-character gebruikt worden, maar de meegegeven break-string. Dit is bijvoorbeeld handig om overal de "¡BR¿"-tag in te voegen. Wanneer we echter al een string hebben die gescheiden is door newlines, dan kunnen we de functie `nl2br()` gebruiken. Deze functie is ook heel handig wanneer we een tekstveld (textarea) hebben gelezen van een webpagina.

Om strings te vergelijken kunnen we ofwel de gewone vergelijkingsoperator gebruiken, ofwel de functie `strcmp()` die dezelfde syntax volgt als de functies onder C/C++. Ook `strncmp()` bestaat onder PHP. Deze functie wordt gebruikt om de eerste n chars bitsgewijs te gaan vergelijken.

Om strings een bepaalde lengte te geven kunnen we `str_pad()` gebruiken, die een string kan aanvullen aan beide zijden met een gegeven string. Dit is echter een functie uit PHP 4.0.

Om de teken te kennen die bij een bepaalde ASCII-waarde hoort kunnen we de functie `chr()` gebruiken.

Qua hoofdlettergebruik is PHP ook goed voorzien. Zo kunnen we de string in hoofdletters

omzetten met `strtoupper()` en naar kleine letters met `strtolower()`. De functie `ucfirst()` gaat het eerste karakter in een string omzetten naar een hoofdletter (handig bij zinnen), en `ucwords()` gaat ieder woord voorzien van een hoofdletter.

```
<html>
  <head>
    <title>Hoofdletters</title>
  </head>

  <body>
    <h1>Hoofdletters</h1>
<?php
$beginstring = "dit is een nutteloze zin die VEEL duidelijk maakt.";
echo strtolower($beginstring) . "<BR>";
echo strtoupper($beginstring) . "<BR>";
echo ucfirst($beginstring) . "<BR>";
echo ucwords($beginstring) . "<BR>";
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 4.1: Hoofdlettergebruik in PHP

Stukjes uit strings selecteren gebeurt met de functie `substr()`, zoals we zien in figuur 4.2.

```
<html>
  <head>
    <title>Substr</title>
  </head>

  <body>
    <h1>Substr()</h1>
<?php
echo substr ("Zeus Werkgroep Informatica", 5) . "<BR>";
echo substr ("Zeus WPI", 0, 4);
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 4.2: Hoofdlettergebruik in PHP

Er bestaan nog veel andere functies om strings te berwerken, maar het is onmogelijk om al deze functies hier te bespreken. Alle functies kan je vinden in de manual van PHP, onder de sectie “String functions”

4.2 Invoer van een form

Wanneer we data uit een form lezen mogen we die data niet vertrouwen. Eerst en vooral moeten we erop toezien dat er geen speciale tekens inzitten zoals html-tags en vreemde quotes. In PHP kunnen we extra opties aanzetten zodat alle verdachte quotes reeds gecontroleerd worden, maar toch, "better safe than sorry"...

4.2.1 Whitespace

Wanneer we een gegeven doorgestuurd krijgen kan het zijn dat er vooraan of achteraan nog spaties, tabs of newlines. Spaties in de string zelf zijn echter belangrijk. Met de functies `trim()`, `ltrim()` en `rtrim()` kunnen we respectievelijk de spaties aan beide zijden verwijderen, of enkel aan de linker- of rechterkant. De functie `chop()` kan hier ook gebruikt worden aangezien de werking ongeveer gelijk is aan de werking van `trim()`. Perl-gebruikers moeten echter opletten dat ze de `chop()` van PHP niet verwarren met de `chop()` van Perl.

4.2.2 Speciale chars

Invoer van een gebruiker mogen we nooit vertrouwen op de eerlijkheid van de gebruiker. We moeten ervoor zorgen dat onze functies nooit mislopen door een belachelijke invoer van de gebruiker. Daarvoor bestaan er verschillende functies, maar de meestgebruikte zijn `quotemeta()` en `addslashes()`. De eerste functie gaan voor elk vreemd karakter een extra
toevoegen. `Addslashes()` gaat voor iedere quote een
toevoegen. Dit is bijvoorbeeld handig om data terug te gebruiken in html-tags. Wanneer we in de vorige voorbeelden verkeerde data invullen in de veldjes op de webpagina kan dat verkeerd verwerkt worden. We kunnen dit echter tegengaan door `addslashes()` te gebruiken. We gaan echter wel opmerken dat PHP normaal zo slim is om zelf die speciale slashes in te voegen!

We kunnen het effect van `addslashes()` omkeren door de functie `stripslashes()` te gebruiken.

4.2.3 HTML-tags

Wanneer een gebruiker data kan ingeven die later terug in een html-document tevoorschijn komt, dan willen we er zeker van zijn dat er geen html-tags in verschijnen, aangezien de gieïnterpreteerd worden door de browser, en een malafiede persoon bijvoorbeeld meta-redirect-tags kan invoegen in je pagina. Niet goed dus. De functies `htmlspecialchars()` en `htmlentities()` kunnen speciale-html tekens omzetten naar de juiste html-constructies. Dit kan ingewikkeld klinken, maar een voorbeeldje kan alles duidelijk maken. De code vinden we in figuur 4.4.

Dit is bijvoorbeeld handig om mensen te verhinderen links te posten in een forum die met PHP gemaakt is.

```
<html>
  <head>
    <title>Addslashes</title>
  </head>

  <body>
    <h1>Geef iets in</h1>
<form method="GET" action="<?php echo $PHP_SELF ?>">
  <input name="invoer">
  <input type="submit" value="Ok!">
</form>

<?php
echo "SELECT * FROM iets WHERE data like '%$invoer%'";
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 4.3: Een voorbeeld hoe we addslashes moeten gebruiken, of hoe PHP het voor ons doet!

```
<html>
  <head>
    <title>Htmlspecialchars</title>
  </head>

  <body>
    <h1>Geef nog iets in</h1>
<form method="GET" action="<?php echo $PHP_SELF ?>">
  <input name="invoer">
  <input type="submit" value="Ok!">
</form>

<?php
echo "Zonder: $invoer<BR>";
echo "Met: " . htmlspecialchars($invoer);
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 4.4: htmlspecialchars()

4.2.4 Opslag

Wanneer we strings willen opslaan in een file moeten we zeker zijn dat alle gegevens correct opgeslagen zullen worden. We moeten hetzelfde indachtig zijn wanneer we bijvoorbeeld array's en objecten moeten opslaan. Daarvoor bestaan de functies `serialize()` en `unserialize()`. De eerste functie maakt de data klaar voor opslag en de laatste zet de data terug om naar bruikbare data. Deze functies zijn handig om data op te slaan in bv een file.

Het gebruik van deze functies wordt uit de doeken gedaan bij het gebruik van files.

4.3 Regular expressions en bijhorende functies

Regular expressions zijn een begrip in de *nix wereld. Ze zijn een soort algemeen ondersteunde krachtige zoekstring. De bespreking van hoe regular expressions nu werkt is heel moeilijk uit de doeken te doen, maar er bestaan verschillende goede bronnen over dit onderwerp. Een snelle zoektocht op internet naar "regular expression howto" zou goede resultaten moeten opleveren. Algemeen mogen we aannemen dat een regular expression een beschrijving geeft van hoe een string eruit moet zien. Wanneer we deze regular expressions goed gebruiken kunnen we hiermee snel bv email-adressen controleren op hun geldigheid.

PHP heeft verschillende functies die werken met regular expressions. De bekendste zijn `ereg()` en `eregi()`. Deze functies gaan een string gaan controleren op hun geldigheid volgens een gegeven regular expression. Opioneel kunnen we nog stukken van de regular expression opslaan in een array. De functies `ereg_replace()` en `eregi_replace()` gaan regular expressions gaan vervangen door een andere string.

Een voorbeeldje om alles duidelijk te maken vinden we in figuur 4.5. Een mailadres wordt gevraagd en gecontroleerd op de geldigheid ervan. Om het mailadres te controleren wordt een regular expression gebruikt.

De replace functies werken gelijklopend. De volledige syntax van deze functies zijn te vinden onder het hoofdstuk *Regular Expression Functions* in de PHP manual. Het voorbeeld uit figuur 4.6 werd uit de handleiding genomen.

```
<html>
  <head>
    <title>Regexp</title>
  </head>

  <body>
    <h1>Email-adressen controleren</h1>
    <form method="GET" action="<?php echo $PHP_SELF?>">
      Mailadres: <input name="mailadres">
      <input type="submit" value="controleren">
    </form>

<?php
if(isset($mailadres))
{
  $mailadres = trim($mailadres);
  echo "Het mailadres $mailadres is ";
  if(eregi("^[_a-z0-9-]+(\.[_a-z0-9-]+)*@[a-z0-9-]+(\.[a-z0-9-]+)*$",$mailadres)){
    echo "correct!";
  }else{
    echo "foutief!";
  }
}
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 4.5: Regular expressions om email-adressen te controleren.

```
<?php
$text = "Op http://www.zeus.rug.ac.be/ vinden we de webpagina van Zeus WPI.";

$text = ereg_replace("[[:alpha:]]+://[^<>[:space:]]+[[:alnum:]/]",
                     "<a href=\"\\0\">\\0</a>", $text);

echo $text;

?>
```

Figuur 4.6: Links omzetten

Hoofdstuk 5

Array's en $$

In dit hoofdstuk gaan we wat dieper ingaan op het gebruik van array's onder PHP. Wanneer we bijvoorbeeld een puntenlijst willen bijhouden van een bepaald jaar, dan kunnen we die informatie in verschillende variabelen opslaan, die de vorm $voornaamfamilienaam hebben, maar het wordt al snel onoverzichtelijk. Daarenboven is het programma moeilijk aan te passen wanneer er een persoon bijkomt. Informatie die normaal in lijsten opgeslagen ligt gaan we normaal in array's gaan stockeren. In andere programmeertalen zijn array's ook bekend en verloopt het gebruik ongeveer gelijk met het gebruik in PHP. Er zijn echter enkele toevoegingen in PHP en deze zullen we hier ook proberen toe te lichten.

5.1 Array's in PHP

5.1.1 Array's vullen

Wanneer we in PHP een array willen gebruiken, kunnen we die gewoon initialiseren. Een declaratie is niet nodig, net zoals we andere variabelen niet moeten declareren in PHP. Een voorbeeldje van de initialisatie en het gebruik van arrays vinden we in figuur 5.1.

We kunnen een array echter op een andere manier opvullen. Door gewoon waarden aan de array toe te kennen zonder een index op te geven, wordt de volgende beschikbare index genomen en wordt op die plaats in de array de nieuwe data opgeslagen. We kunnen als het ware achteraan de array nieuwe data toevoegen door die erbij te duwen.

We kunnen ook de functie `array()` gebruiken om een array op te vullen. Deze functie is vooral handig om kleine array's met vaste waarden op te vullen.

We moeten echter geen numerieke indexen aanhouden. In PHP mag de index eender wat zijn. Indexen met reële getallen zijn eerder onbruikbaar, maar wanneer we merken dat we gemakkelijk strings kunnen gebruiken als index worden de mogelijkheden een stuk groter. Merk echter op dat de indexen hoofdlettergevoelig zijn!

Laten we echter even de aandacht vestigen op het voorbeeld in figuur 5.5. Beide constructies werken, maar de eerste is verkeerd. Wanneer we strings als index willen gebruiken, moet die altijd binnen quotes staan. PHP laat de andere constructie toe, maar doordat een string

```
<html>
  <head>
    <title>Arrays</title>
  </head>

  <body>
    <h1>Arrays</h1>
<?php
$punten[1] = 18;
$punten[2] = 15;
$punten[3] = 13.5;
$punten[4] = 12;
$punten[5] = 5;
$punten[6] = 17;

echo $punten[1] . "<BR>";
echo $punten[2] . "<BR>";
echo $punten[3] . "<BR>";
echo $punten[4] . "<BR>";
echo $punten[5] . "<BR>";
echo $punten[6] . "<BR>";
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 5.1: Een eerste array, $punten

```
<html>
  <head>
    <title>Toevoegen</title>
  </head>

  <body>
    <h1>Toevoegen</h1>
<?php
$punten[1] = 19;
$punten[] = 12;
$punten[] = 14;
$punten[] = 5;
$punten[] = 17.5;

echo $punten[1] . "<BR>";
echo $punten[2] . "<BR>";
echo $punten[3] . "<BR>";
echo $punten[4] . "<BR>";
echo $punten[5] . "<BR>";
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 5.2: Array's opvullen

```
<html>
  <head>
    <title>Array()</title>
  </head>

  <body>
    <h1>array()</h1>
<?php
$eentottien = array(1,2,3,4,5,6,7,8,9,10);

echo $eentottien[4]; // uitvoer is 5!
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 5.3: array() gebruiken

```
<html>
  <head>
    <title>Strings als index</title>
  </head>

  <body>
    <h1>Strings als index</h1>
<?php
$punten["Bernard Grymonpon"]=17;
$punten["Tom Poppe"]=16;
$punten["Piet Snot"]=5;

echo $punten["Bernard Grymonpon"] . "<BR>";
echo $punten["Tom Poppe"] . "<BR>";
echo $punten["Piet snot"] . "<BR>"; // geen uitvoer, case-sensitive!
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 5.4: Strings als index

zonder quotes evengoed een constante kan zijn, kunnen we hier fouten maken die moeilijk op te sporen zijn. Het is beter er een goede gewoonte van te maken de string-indexen van arrays altijd met quotes te omsluiten.

Meerdimensionele array's zijn ook mogelijk. De declaratie valt zoals altijd weg en we moeten gewoon de waarden toekennen met de correcte indexen. We kunnen terug strings gebruiken als index. De toevoeging door geen index op te geven is ook mogelijk.

5.1.2 Array's overlopen

In de vorige voorbeelden hebben we telkens de inhoud via vaste indexen uit de array gehaald. We kunnen met behulp van een lus een bepaalde array aflopen, maar dan mogen we geen strings gebruiken als index.

Met behulp van de functie `count()` kunnen we onderzoeken hoeveel indexen er reeds in gebruik zijn. We kunnen dan enkel maar hopen dat deze indexen sequentieel ingevuld zijn.

Wanneer we echter strings gebruikt hebben, is deze constructie onbruikbaar. Hiervoor moeten we de `list()=each()` constructie gebruiken. Deze constructie gaat de array overlopen en telkens de index en de waarde teruggeven. Deze methode kunnen we ook genest gebruiken om meerdimensionele arrays te overlopen. Het voorbeeld in figuur 5.8 zal volstaan, de syntax is eenvoudig te begrijpen...

Wanneer we dit echter twee maal na elkaar doen zou de tweede maal geen data gevonden worden. Dit is het gevolg van de werking van list-each. Iedere array heeft een interne pointer die en bepaald element aanduidt. Wanneer we list-each gebruiken wordt die pointer telkens

```php
<?php

/* fout */
$foo[bar] = "Zeus werkgroep informatica";
echo $foo[bar];

/* goed */
$foo["bar"] = "Zeus werkgroep informatica";
echo $foo["bar"];

/* waarom */
define("bar",15);
$foo[bar] = "Zeus werkgroep informatica";
echo $foo[bar];

?>
```

Figuur 5.5: foo[bar] != foo[“bar”]

```php
<html>
  <head>
    <title>Meerdimensionele arrays</title>
  </head>

  <body>
    <h1>Meerdere dimensies</h1>
<?php
$taal["Bernard"][] = "PHP";
$taal["Bernard"][] = "C";
$taal["Bernard"][] = "C++";
$taal["Bernard"][] = "Java";
$taal["Bernard"][] = "Nederlands";

$taal["Tom"][] = "C";
$taal["Tom"][] = "C++";
$taal["Tom"][] = "Perl";
$taal["Tom"][] = "PHP";
$taal["Tom"][] = "Nederlands";

echo $taal["Bernard"][2] . "<br>"; //C++
echo $taal["Tom"][4] . "<br>"; //C++

?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 5.6: Meerdimensionele arrays

```
<html>
  <head>
    <title>Arrays en count</title>
  </head>

  <body>
    <h1>count(array)</h1>
<?php
$eenarray[] = 10;
$eenarray[] = 11;
$eenarray[] = 12;
$eenarray[] = 13;
$eenarray[] = 14;

for($i=0; $i < count($eenarray); $i++)
{
  echo $eenarray[$i] . "<BR>";
}


?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 5.7: Een for-lus met count

```
<html>
  <head>
    <title>List()=each()</title>
  </head>

  <body>
    <h1>List()=each()</h1>
<?php
$taal["Bernard"][] = "PHP";
$taal["Bernard"][] = "C";
$taal["Bernard"][] = "C++";
$taal["Bernard"][] = "Java";
$taal["Bernard"][] = "Nederlands";

$taal["Tom"][] = "C";
$taal["Tom"][] = "C++";
$taal["Tom"][] = "Perl";
$taal["Tom"][] = "PHP";
$taal["Tom"][] = "Nederlands";

while(list($naam,$talen) = each($taal)){
  echo "<b>$naam</b> kent de volgende talen:<br>";
  while(list($index,$taalnaam) = each($taal[$naam]))
echo "$taalnaam<br>";
  echo "<br>";
}

?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 5.8: Het gebruik van list()=each()

eentje verder geschoven. Wanneer we echter opnieuw willen beginnen moeten we `reset()` gebruiken. Deze functie zet de interne array-pointer terug naar het begin van de array. Andere functies de gebruik maken van deze interne pointer (en die dus ook verplaatsen) zijn `next()` en `prev()`. Deze twee functies hebben echter de slechte eigenschap dat wanneer de waarde van een bepaalde entry in de array, deze waarde wordt teruggegeven, en dat dit dus resulteert in een "FALSE". Niet echt handig, want je bent nog niet op het einde van de array. De list-each-constructie is de meest gebruikte constructie, en de veiligste.

5.2 Andere functies met array's

5.2.1 implode(), explode() en split()

Met de functies `implode()`, `explode()` en `split()` kunnen we strings omzetten naar array's en vice versa. explode() en split() gaan een string onderzoeken naar een bepaalde substring, en gaan de string aan de hand van de substring gaan opsplitsen. Het verschil tussen explode en split is het verschil in de substring. Split gaat de opgegeven string beschouwen als een regular expression[1], explode neemt de string letterlijk. Wanneer we dus geen regular expressions nodig hebben, kiezen we beter voor explode() aangezien die functie merkbaar sneller is. We zien een voorbeeld in figuur 5.9.

```
<html>
  <head>
    <title>Pizza!</title>
  </head>

  <body>
    <h1>Pizza</h1>
<?php
$ingredienten = "kaas ham olijf bodem champignons ananas";
$ingredientenarray = explode(" ",$ingredienten);

sort($ingredientenarray);

$einde = implode($ingredientenarray," en ");
echo "Een pizza bestaat uit $einde";
?>


    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Berre</a></address>
  </body>
</html>
```

Figuur 5.9: explode() en implode()

[1]Meer hierover in het hoofdstuk over string-functies

5.2.2 Sorteren

Zoals we in het vorige voorbeeld hebben we gezien hoe we een array kunnen sorteren. Er bestaan veel verschillende sorteerfuncties, en allemaal doen ze iets anders. De meest gebruikte zijn `sort()` en `asort()`. Sort gaat de elementen gaan rangschikken, maar de indexen veranderen niet mee. Asort gaat ook de indexen verplaatsen.

```
<html>
  <head>
    <title>asort()</title>
  </head>

  <body>
    <h1>asort()</h1>
<?php
$fruit["a"] = "Appel";
$fruit["b"] = "Peer";
$fruit["c"] = "Banaan";
$fruit["d"] = "Citroen";

asort($fruit);
reset($fruit);

while(list($index, $naam)=each($fruit)){
    echo "$index = $naam <br>";
}
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 5.10: sorteren met asort()

De functie `shuffle()` (die maar beschikbaar is vanaf php3.0.8 en php4.0b4!) doet precies wat de naam zegt. De functie zorgt ervoor dat de array op een willekeurige manier wordt "geschud".

Er bestaan nog veel andere functies die gebruikt kunnen worden op array's, maar de belangrijkste hebben we hier opgesomd. Voor een volledig overzicht van alle functie verwijzen we naar de manual van php.net[2].

5.3 Arrays en forms

Wanneer we in een formulier vooraf niet weten hoeveel invulvelden we nodig hebben, dan kunnen we ook arrays gebruiken om dit probleem op te lossen. Door telkens een gelijke naam

[2]Die is te vinden op http://www.php.net/manual

```
<html>
  <head>
    <title>De vuilbakken buitenzetten</title>
  </head>

  <body bgcolor="#FFFFFF">
    <h1>Op kot...</h1>
<?php

$wie = array("Bernard","Grietje","Stefanie","Tine","Veerle");

srand(time());
shuffle($wie);
echo "De maand mei: <br>";
for($i=0; $i<4;$i++){
  echo "Week " . ($i+1) . ": " . $wie[$i] . "<BR>";
}
echo "De gelukkige in mei is: " . $wie[4];

?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 5.11: shuffle()

gevolgd door `[]` in te geven bij het `name=` veld in de `input`-tag, kunnen we ervoor zorgen dat de variabelen die doorgegeven aan het PHP-script reeds in een array zitten.

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Knoeien met arrays</title>
  </head>

  <body bgcolor="#FFFFFF">
    <h1>Reserveer hier uw tickets...</h1>

<form method="GET" action="arrayform.php">
  Geef het aantal personen in :
  <input name="aantal" value="3" size="5"><br>
  <input type="submit" value="Ok, verdergaan">
</form>

    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard Grymonpon</a></address>
  </body>
</html>
```

Figuur 5.12: De html-pagina

Bovenstaande constructie kan maar in enkele gevallen gebruikt worden, maar meestal is het dan de enige oplossing. Dergelijke oplossingen zijn een heel stuk moeilijker in Perl, C/C++ of Java.

5.4 De $$-constructie

Deze constructie is een van de aardigheden in PHP. De uitleg ervan is moeilijk te doen, een voorbeeld (figuur 5.15) verduidelijkt veel meer. Er zijn weinig situaties waar we deze constructie goed kunnen gebruiken, en dus moeten we ze zo veel mogelijk proberen te vermijden.

We zien dat eerste de variabele wordt vervangen door de waarde ervan, en dat daarna dat nogmaals gebeurd. Dit effect kan dieper genest worden wanneer we het wensen, maar je programma komt er heel onoverzichtelijk door. *Het gebruik van deze constructie wordt dan ook afgeraden!*

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Gelieve de namen in te vullen</title>
<!-- Changed by: Berreke, 11-Apr-2002 -->
  </head>
  <body bgcolor="#FFFFFF">
    <h1>De namen</h1>

    <form method="GET" action="arrayformdone.php">
      <table align="center" border="1" cellspacing="0" cellpadding="4">
        <tr>
          <th>Id</th><th>Voornaam</th><th>Familienaam</th><th>Woonplaats</th>
        </tr>
<?php
// een standaardwaarde zetten...
if (!isset($aantal)) $aantal = 3;

for($i=0; $i < $aantal; $i++)
{
  echo "<tr>\n";
  echo "<td align=\"right\">". ($i + 1) . "</td>\n";

  echo "<td align=\"center\">";
  echo "<input name=\"voornaam[]\">";
  echo "</td>\n";

  echo "<td align=\"center\">";
  echo "<input name=\"familienaam[]\">";
  echo "</td>\n";

  echo "<td align=\"center\">";
  echo "<input name=\"woonplaats[]\">";
  echo "</td>\n";
  echo "</tr>\n";
}

?>

      </table>
      <input type="submit" value="Ok, bestellen">
    </form>
  </body>
</html>
```

Figuur 5.13: Het script dat de invul-pagina opbouwt

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Ok, tis in orde</title>
  </head>

  <body>
    <h1>Bestelling werd geplaatst</h1>
<?php
srand(time());

for($i=0; $i < count($voornaam); $i++)
{
  if(isset($voornaam[$i]) && isset($familienaam[$i]) && isset($woonplaats[$i])){
$safe_voornaam = htmlentities(trim($voornaam[$i]));
$safe_familienaam = htmlentities(trim($familienaam[$i]));
$safe_woonplaats = htmlentities(trim($woonplaats[$i]));

// alles in een database plaatsen...
// en verwerken...

echo "De bestelling voor " . $safe_voornaam . " " . $safe_familienaam;
echo " hebben we correct verwerkt. De bus vertrekt uit " . $safe_woonplaats;
echo " om ". rand(1,24) . " uur " . (rand(0,3) * 15) . ".<br>\n";

  }
}
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 5.14: Het script dat de namen verwerkt

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>$$-constructie</title>
  </head>

  <body>
    <h1>$$-constructie</h1>

<?php

$hello = "world";
$world = "Hello";
echo $$hello . " " . $hello ."<br>";

?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard Grymonpon</a></address>
  </body>
</html>
```

Figuur 5.15: De $$-constructie

Hoofdstuk 6

Klassen en objecten

Wanneer we grotere programma's moeten maken is het gebruik van klassen heel handig. Populaire talen als Java en C++ zijn volledig gericht op programmeren met klassen, of *objectgeörienteerd programmeren*. PHP heeft ook ondersteuning voor klassegerichte programmatie. We kunnen klassen aanmaken en bijhorende methodes aanmaken. Overerving kan ook gebruikt worden. Er zijn wel enkele tekortkomingen, maar daarover later meer.

6.1 Een klasse maken

Wanneer we een klasse willen programmeren in een PHP-script moeten we de klasse-definitie gaan opbouwen. We zien dat we binnen een klassedefinitie wel de variabelen moeten aanduiden, maar we moeten nog altijd geen type meegeven. Het is een declaratie zonder typeaanduiding. We merken hier echter ook op dat het zelfs niet nodig is om de variabelen vooraf aan te duiden. Het is echter wel een goede stijl om de meest gebruikte variabelen binnen de klasse vooraf te initialiseren.

Figuur 6.1 geeft een voorbeeld van een klasse voor een persoonsobject. We slaan enkele vaste gegevens op in het object. De variabelen die we daarvoor gebruiken moeten we declareren met `var`.

```
<?php

class Persoon{
  var $naam;
  var $voornaam;
  var $straat;
  var $nr;
  var $postcode;
  var $nummer;
}
?>
```

Figuur 6.1: Variabelen in een klasse

In een klasse kunnen we ook altijd een constructor definieren. Deze methode wordt altijd opgeroepen wanneer we een nieuw object maken uit deze klasse. De constructor in PHP is de methode (functie) binnen de klasse met dezelfde naam als de klassenaam. Figuur 6.2 toont het gebruik van de constructor.

```
<?php

class Persoon{
  var $naam;
  var $voornaam;
  var $straat;
  var $nr;
  var $postcode;
  var $gemeente;

  function Persoon($p_naam,$p_voornaam,$p_straat,$p_nr,$p_postcode,$p_gemeente){
    $naam = $p_naam;
    $voornaam = $p_voornaam;
    $straatnaam = $p_straat;
    $nr = $p_nr;
    $postcode = $p_postcode;
    $gemeente = $p_gemeente;
  }


}
?>
```

Figuur 6.2: Een constructor

De andere methodes (functies die enkel op een object werken) kunnen we gelijkaardig aanmaken.

Een klasse gebruiken we door nieuwe objecten te maken van die klasse, en daar dan de gepaste methodes op te gebruiken.

6.2 Enkele opmerkingen

Alle eigenschappen en methodes van een bepaalde klasse zijn *public*. PHP kan niets private of protected zetten. Dit is een tekort aan PHP. Wanneer we echter onszelf deze restrictie opleggen (geen variabelen rechtstreeks gebruiken uit een object, enkel via methodes werken) kunnen we goede OO programma's maken.

Er bestaat geen destructor. De destructor moeten we zelf maken wanneer we die nodig hebben. We moeten deze eigengemaakte methode dan ook oproepen voor we met `unset()` het object vernietigen.

Er bestaat wel overerving in PHP. De syntax is als volgt:

```php
<?php

class Persoon{
  var $naam;
  var $voornaam;
  var $straat;
  var $nr;
  var $postcode;
  var $gemeente;
  var $safe;

  function Persoon($p_naam, $p_voornaam, $p_straat, $p_nr, $p_postcode, $p_gemeente){
    $this->naam = $p_naam;
    $this->voornaam = $p_voornaam;
    $this->straat = $p_straat;
    $this->nr = $p_nr;
    $this->postcode = $p_postcode;
    $this->gemeente = $p_gemeente;
    $this->safe=false;
  }

  function setNaam($p_naam, $p_voornaam){
    $this->naam = $p_naam;
    $this->voornaam = $p_voornaam;
    $this->safe=false;
    $safe=false;
  }

  function setAdres($p_straat,$p_nr,$p_postcode,$p_gemeente){
    $this->straatnaam = $p_straat;
    $this->nr = $p_nr;
    $this->postcode = $p_postcode;
    $this->gemeente = $p_gemeente;
    $this->safe=false;
  }

  function display(){
    echo "Naam: " . $this->voornaam . "<br>";
    echo "Familienaam: " . $this->naam . "<br>";
    echo "Straat + nr: " . $this->straat . " " . $this->nr . "<br>";
    echo "Postcode + gemeente: " . $this->postcode . " " . $this->gemeente . "<br>";
  }

  function isSafe(){ // overbodige methode!
    return $this->safe;
  }

  function opslaan(){
    $data = serialize($this);
    // file openen
    // $data wegschrijven
    $this->safe=true;
  }
}
?>
```

```
<?php
require("persoon3.php");

?>
<html>
  <head>
    <title>OOP</title>
  </head>

  <body>
    <h1>Persoon</h1>
<?php
$ik = new Persoon("Bernard","Grymonprez","Merelstraat",8,9000,"Gent");
$ik->setNaam($ik->naam,"Grymonpon");

$ik->display();
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 6.4: Hoe gebruiken we de klasse uit persoon3.php

```
class Student extends Persoon{
// extra variabelen

// extra methodes
}
```

Denk er echter wel aan dat de constructor van de onderliggende klasse niet opgeroepen wordt en dat we dat dus zelf moeten doen in de nieuwe constructor.

Men kan nog heel wat meer doen in PHP met klasses, maar dit alles valt buiten het bereik van deze basiscursus. Om de kunst van het objectgericht programmeren onder de knie te krijgen zijn er andere talen die veel striktere regels hebben beter. Voorbeelden hiervan zijn Java en C++. Meer informatie over klassen en objecten in PHP kan gevonden worden in de handleiding van PHP onder het hoofdstuk *Classes and Objects*.

Hoofdstuk 7

Files

Om data op te slaan hebben we ergens een bewaarplaats nodig. Wanneer we veel data willen opslaan zijn we toegewezen op databanken, maar kleine hoeveelheden data kunnen we gemakkelijk in een bestand opslaan. We bekijken ook de mogelijkheden van PHP om file uploads mogelijk te maken.

7.1 Files gebruiken voor dataopslag

7.1.1 Openen en sluiten

Wanneer we een file willen openen gebruiken we de functie `fopen()`. Deze functie neemt als eerste argument de filenaam (dit mag ook een http- of ftp-URL zijn). Het tweede argument is de mode waarin de file moet geopend worden. Willen we alleen lezen, dan gebruiken we mode "r". Wanneer we alleen willen schrijven naar een file, dan gebruiken we ofwel "w", ofwel "a". Bij mode "w" wordt de file eerst nog gewist. Wanneer we willen lezen en schrijven gebruiken we "r+". C/C++ programmeurs zullen deze modes zeker herkennen. De functie `fopen(filenaam, mode)` geeft ons een file-handeler terug. Dit is een integer die we in alle andere file-operaties zullen moeten meegeven.

Wanneer we klaar zijn met de file sluiten we best terug af. Alle data die nog in buffer zat wordt dan weggeschreven naar de schijf (of naar andere buffers), en we zijn zeker dat we niet onopzettelijk onze file gaan vullen met verkeerde data. Om de file te sluiten gebruiken we de functie `fclose(filehandler)`.

Een tweede vorm om een file te openen is de `file(filenaam)` functie. Deze functie neemt slechts 1 argument, een filenaam. Deze file wordt geopend en wordt als een array teruggegeven aan de PHP-script. Wanneer we deze constructie gebruiken kunnen we echter niet schrijven naar een file.

7.1.2 Lezen en schrijven

Wanneer we een file geopend hebben met `fopen()`, dan kunnen we op verschillende manieren lezen uit deze file. Ofwel lezen we karakter per karakter met `fgetc(file-handler)`. Deze functie leest het volgende karater in de file, en geeft het terug als een string. Wanneer we echter meerdere karakters ineens willen lezen, moeten we de functie `fread(handler, size)` gebruiken. Er worden precies 'size' aantal karakters gelezen, ofwel wordt gestop aan het einde van de file.

Een veel betere functie, die een volledige lijn per keer kan lezen is `fgets(filehandler, size)`. Er wordt gelezen tot er een newline of het einde van de file gevonden wordt.

Een laatste functie om een file te lezen is `fpassthru(file-handler)`. Deze functie gaat de volledige file lezen en doorgeven aan de browser. Let er wel op dat er niets meer geparsed wordt!

Wanneer we willen schrijven kunnen we de functies `fputs(filehandler, string[, lengte])` of `fwrite(filehandler, string[, lengte])` gebruiken. De beide functies werken identiek. Wanneer er geen lengte gegeven is, dan wordt de volledige string geschreven naar de file.

7.1.3 Navigeren in files

Wanneer we een file geopend hebben, willen we soms enkele lijnen overslaan, of meteen naar het dertiende karakter springen. Om binnen files te navigeren bestaan er verschillende functies. `rewind(filehandle)` is de simpelste functie onder deze functies. De interne file-pointer wordt teruggezet naar het begin van de file.

`fseek(filehandle, offset)` gaat de interne pointer op een welbepaalde plaats gaan zetten. De parameter offset is de positie vanaf het begin van de file. Wanneer we een bepaalde positie willen weten, moeten we `ftell(filehandle)` gebruiken. Deze functie geeft de huidige positie in een file weer. *Let erop dat fseek een waarde van -1 teruggeeft bij een fout, en anders 0! Dit is strijdig met alle andere PHP-functies.*

Een laatste handige functie is `feof(filehandle)`. Deze functie geeft een boolean terug die weergeeft als het einde van de file al dan niet bereikt is. Deze functie wordt meestal gebruikt met een while-lus om bestanden volledig in te lezen.

7.1.4 Enkele opmerkingen

Wanneer we willen schrijven naar een file moeten we daar schrijftoegang tot hebben. Wanneer PHP echter als een module van Apache draait, dan gaat deze niet gestart zijn als uw eigen usernaam en userid, maar als een speciale user (bv www-data op een debian systeem).

Dit brengt echter een groot veiligheidsrisico met zich mee. Wanneer de user www-data moet kunnen schrijven naar een bepaalde file, moet hij er schrijfpermissie toe hebben. Dit houdt dus in dat je de file schrijfbaar moet zetten voor iedereen. Wanneer dit gebeurt op een systeem waar meerdere gebruikers op zitten, en er toegang tot hebben via ssh of telnet, en in je directory kunnen, dan kan iedereen je file zomaar gaan overschrijven. De enige oplossing

is PHP als een cgi gaan gebruiken, maar daarmee verlies je functionaliteit van cookies en headers. Het is een tweeledig probleem, waar Apache2 misschien een uitkomst kan aan bieden, aangezien een webserver dan als een bepaalde gebruiker kan draaien, om zo gemakkelijker met file om te kunnen gaan.

Er kan ook gebruik gemaakt worden van php-cgiwrap, maar dit moet aan de serverkant geïnstalleerd worden.

Een goede manier om te werken met files is het gebruik van `file()` om te lezen en `fputs()` om te schrijven. Wanneer we voor de andere files kiezen moeten we heel aandachtig zijn om geen verkeerde data in te lezen. De functies `serialize()` en `unserialize()` zijn handig om data klaar te maken voor opslag.

We geven een kort voorbeeld. We maken een klein gastenboekje. Het bestaat uit drie verschillende files. De eerste file is een gewoon een html-form dat om invoer vraagt. Dit formulier wordt verwerkt door een eerste php-file.

```
<html>
  <head>
    <title>Invullen van het gastenboek</title>
  </head>

  <body>
    <h1>Gelieve uw gegevens in te geven</h1>
<form method="GET" action="gastinvul.php">
  Naam: <input name="naam"><br>
  Voornaam: <input name="voornaam"><br>
  Email: <input name="email"><br>
  <input type="submit" value="Ok!">
</form>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 7.1: Het invulformulier

Om het gastenboek bekijken, gebruiken we een ander scriptje. We lezen de file met `file()`.

7.2 File uploads

Soms willen we dat een bepaalde gebruiker van onze webpagina een bestand kan doorsturen naar de server, zodat we dan verder kunnen werken aan dit bestand. Enkele voorbeelden zijn pasfoto's op een ledenpagina, of een CV doorsturen bij het invullen van je gegevens in een databank van een recruteringsbureau. Onder PHP kunnen we heel gemakkelijk file uploads verzorgen. We moeten wel even opletten hoe we alles in de HTML-form inbakken. Let er bijvoorbeeld op dat er geen GET meer gebruikt wordt, maar een POST. We kunnen een HIDDEN element meegeven dat een bepaalde maximumwaarde bepaalt voor de file. Deze

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Ok, ingevuld</title>
  </head>

  <body>
    <h1>Ok!</h1>
<?php
if(isset($naam) && isset($voornaam) && isset($email)){
  $dataarray[] = $naam;
  $dataarray[] = $voornaam;
  $dataarray[] = $email;
  $data = serialize($dataarray);

  if ($file = fopen("gastenboek.txt","a")){
    fputs($file, $data . "\n");
    fclose($file);
    echo "Uw gegevens werden opgeslagen...";
  } else {
    echo "De file kan niet geopend worden!";
  }
}
?>

    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 7.2: De verwerking van het formulier

```
<html>
  <head>
    <title>Ziedaar</title>
  </head>

  <body>
    <h1>Gastenboek</h1>
<?php
$alledata = file("gastenboek.txt");
echo "Aantal entry's : " . count($alledata) ."<BR>";

while(list($key,$data) = each($alledata)){
  $dataarray = unserialize($data);
  echo "Naam + voornaam: " . $dataarray[0] . " " . $dataarray[1] . "<BR>";
  echo "Email: " . $dataarray[2] . "<BR>";
  echo "<HR>";
}
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 7.3: Het script dat het formulier terug uitgeeft

tag moet echter vlak voor de FILE-type INSERT tag staan. Een voorbeeld zal heel veel duidelijk maken.

Wanneer we dan deze file willen verwerken in PHP, krijgen we enkele extra variabelen erbij. Zo kunnen we de grootte van de file en het MIME-type terugvinden. Let erop dat de file normaal ergens op een tijdelijke plaats staat. Deze file moet nog gecopieerd worden wanneer wie die verder willen gebruiken.

Een voorbeeldje om alles duidelijk te maken. De form die hierboven staat gaat enkele gegevens opslaan in een file, en je passfoto ergens plaatsen op de server. We kunnen deze file dan in een ander script gewoon meegegeven als een bestaande tekening.

```
<html>
  <head>
    <title>Upload</title>
  </head>

  <body>
    <h1>File upload</h1>
<form method="POST" action="gast2invul.php" ENCTYPE="multipart/form-data">
  Naam: <input name="naam"><br>
  Voornaam: <input name="voornaam"><br>
  Email: <input name="email"><br>
  <input type="hidden" name="MAX_FILE_SIZE" value="10000">
  Foto: <input type="FILE" name="foto"><br>
  <input type="submit" value="Ok!">
</form>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 7.4: Het upload html-formulier

```
<html>
  <head>
    <title>Ok, ingevuld</title>
  </head>

  <body>
    <h1>Ok!</h1>
<?php
if(isset($naam) && isset($voornaam) && isset($email)){
  $dataarray[] = $naam;
  $dataarray[] = $voornaam;
  $dataarray[] = $email;

  if($foto){
    $ext = substr($foto_name, sizeof($foto_name)-4,4);
$fotonaam = "$naam.$ext";
    if (copy($foto,"images/".$fotonaam)){
      echo "Uw foto werd correct bewaard...<BR>";
  $dataarray[] = $fotonaam;
    } else {
      echo "Ik kon uw foto niet bewaren<BR>";
    }
  } else {
echo "Geen foto ontvangen...<BR>";
  }

  $data = serialize($dataarray);
  if ($file = fopen("gastenboek2.txt","a")){
      fputs($file, $data . "\n");
      fclose($file);
      echo "Uw gegevens werden opgeslagen...";
    } else {
      echo "De file kan niet geopend worden!";
    }
}
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 7.5: Het gastenboek opvullen

```
<html>
  <head>
    <title>Ziedaar</title>
  </head>

  <body>
    <h1>Gastenboek</h1>
<?php
$alledata = file("gastenboek2.txt");
echo "Aantal entry's : " . count($alledata) ."<BR>";

while(list($key,$data) = each($alledata)){
  $dataarray = unserialize($data);
  echo "Naam + voornaam: " . $dataarray[0] . " " . $dataarray[1] . "<BR>";
  echo "Email: " . $dataarray[2] . "<BR>";
  echo "<IMG SRC=\"images/" . $dataarray[3]. "\">";
  echo "<HR>";
}
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 7.6: Het gastenboek bekijken

Hoofdstuk 8

Databases en PHP

Wanneer we heel veel informatie willen opslaan en regelmatig raadplegen of aanpassen, dan wordt het gebruik van files heel omslachtig. We moeten zelf dan allemaal functies schrijven om de data te gaan zoeken en aanpassen. Wanneer we ook nog regelmatig de data willen filteren om slechts enkele resultaten te krijgen, komt het gebruik van files te omslachtig. Dat is het moment dat we aan databases moeten denken. Er bestaan verschillende soorten databases, maar de "relationele databases" zijn de meest gebruikte.

8.1 Databases

Wanneer we het woord databases uitspreken, gaan veel mensen denken aan producten zoals MS-Access. Een database is echter veel minder, en op het zelfde moment ook veel meer dan een pakket als MS-Access. En een goede database hoeft ook niets te kosten; er zijn verschillende gratis oplossingen.

8.1.1 Waarom een database?

De reden waarom werd al licht aangehaald in de inleiding. Wanneer we de data veel moeten manipuleren, of er snel opzoekingen in doen, dan is de oplossing met files heel moeilijk. We moeten zelf allerlei functies gaan voorzien om de data te lezen en weg te schrijven of om in de data te gaan zoeken volgens bepaalde zoekcriteria. Wanneer we dan gaan denken aan de prijslijsten van een groot bedrijf, wordt het gebruik te traag en omslachtig *en* veel te foutgevoelig. Databases doen de opslag voor ons. We krijgen verschillende API's naar bv C/C++, Java of Perl. PHP heeft ook ondersteuning voor databases. We kunnen ongeveer alle handelingen uitvoeren die nodig zijn, gaande van opzoeken van data, tot het maken of het verwijderen van een volledige database.

Verder is het gebruik van een database veel gemakkelijker wanneer de data niet enkel vanuit het PHP-script komt, maar bijvoorbeeld van een applicatie die binnen het bedrijf gebruikt wordt.

Een ander voordeel zijn de schrijfrechten van de files. Wanneer we plain files gebruiken

moeten deze files of directories schrijfbaar zijn voor de webserver, of voor iedereen. Niet de beste oplossing. Een database wordt meestal gebruikt via een TCP/IP-connectie of een socket op de locale machine. De database-server doet alle handelingen, zodat de files slechts eigendom en schrijfbaar moeten zijn voor de databaseserver en niet voor iedereen.

8.1.2 Soorten databases

Er bestaan verschillende soorten databases en binnen iedere soort heb je dan nog de verschillende "merken". We proberen een kleine onderverdeling te maken...

Heden ten dage zijn de meestgebruikte databases "relationele databases". Deze databases zijn door de jaren heen heel veel gebruikt in de bedrijfs- en academische wereld. Veel databases zijn door de jaren heen gegroeid tot heel stabiele en betrouwbare systemen. Deze databases worden meestal aangeduid met de term RDBMS.

Enkele jaren terug kwam er een nieuw idee de kop opsteken. Dit zijn de objectgeorienteerde databases. Deze databases gaan in de richting van object-georienteerde talen door de data telkens als een object voor te stellen en niet als een rij in een tabel. Wanneer we dit combineren met een OO-taal zijn de mogelijkheden enorm. ODBMS is een afkorting voor deze soort databases.

Er zijn ook databases die beide principes gaan verenigen, nl Extended Relation Databases. De benaming voor deze databases is ERDBMS.

We gaan ons echter beperken tot de eerste soort databases, nl RDBMS. Enkele voorbeelden zijn MySQL, msql of MS-Access. Allemaal ondersteunen ze niet volledig de standaard, maar ze zijn goede voorbeelden van relationele databases. Er bestaan nog grotere in de bedrijfswereld, maar deze komen al snel onbetaalbaar voor een kleine website. Enkele voorbeelden zijn SyBase en Oracle.

8.1.3 Gebruik van databases

Wanneer we databases gaan gebruiken, kunnen we terugvallen op een standaard. Deze standaard kreeg de naam "SQL[1]" (de uitspraak is "es-ku-el" en niet "sequel", het is een afkorting) ofwel SQL-92 of SQL2.

SQL is eingelijk een soort van taal waarmee we de database kunnen opbouwen en ondervragen. We kunnen deze taal opsplitsen in 2 delen, namelijk het deel dat we gebruiken voor de opbouw van een database en een deel dat we gebruiken voor het bewerken van de data in de database. Meestal gaan we slechts enkele keren de eerste soort commando's gebruiken, terwijl de tweede soort heel frequent gebruikt worden. De eerste soort wordt Data Definition Language (DDL) genoemd, de tweede krijgt de naam Data Manipulation Language (DML).

Enkele voorbeelden van query's:

```
CREATE DATABASE phptest;
```

[1] Standard Query Language

```
CREATE TABLE cursisten (
   id INTEGER UNSIGNED NOT NULL AUTOINCREMENT PRIMARY KEY,
   naam VARCHAR(50),
   voornaam VARCHAR(50),
   email VARCHAR(60));

CREATE TABLE lessen (
   id INTEGER UNSIGNED NOT NULL AUTOINCREMENT PRIMARY KEY,
   titel VARCHAR(50),
   tijd DATETIME);

CREATE TABLE inschrijvingen(
   cursist_id INTEGER UNSIGNED NOT NULL,
   les_id INTEGER UNSIGNED NOT NULL,
   constraint inschrijvingen PRIMARY KEY(cursist_id, les_id));

INSERT INTO cursisten(naam, voornaam, email)
  VALUES('Poppe','Tom','tom@zeus.rug.ac.be');
INSERT INTO cursisten(naam, voornaam, email)
  VALUES('Goossens','Kristof','goossens@zeus.rug.ac.be');
INSERT INTO cursisten(naam, voornaam, email)
  VALUES('Janssens','Bart','bartjanssens@hotmail.com');

INSERT INTO lessen (titel,tijd)
  VALUES('PHP deel 3','2001-05-02 18:30:00');
INSERT INTO lessen (titel,tijd)
  VALUES('PHP deel 4','2001-05-09 18:30:00');

INSERT INTO inschrijvingen(cursist_id, les_id)
  VALUES('1','1');
INSERT INTO inschrijvingen(cursist_id, les_id)
  VALUES('2','1');
INSERT INTO inschrijvingen(cursist_id, les_id)
  VALUES('1','2');
INSERT INTO inschrijvingen(cursist_id, les_id)
  VALUES('2','2');
INSERT INTO inschrijvingen(cursist_id, les_id)
  VALUES('3','2');

SELECT naam, voornaam FROM cursisten;
SELECT naam, voornaam FROM cursisten WHERE naam='Poppe';
SELECT naam, voornaam FROM cursisten, inschrijvingen
  WHERE inschrijvingen.les_id='2'
  AND inschrijvingen.cursist_id=cursisten.id;

DELETE FROM cursisten WHERE id='1';
```

```
UPDATE lessen SET titel='PHP Hypertext Preprocessor (Deel 2)'
  WHERE id='1';
```

Voor een uitgebreide uitleg over het gebruik van databases en SQL moeten we echter verwijzen naar andere plaatsen, aangezien het onmogelijk is om dit alles te bespreken in een inleidingscursus PHP.

8.1.4 Welk database-systeem

De keuze van het databasesysteem is meestal opgedrongen door de omgeving. Wanneer we enkele hosting-plaatsen gaan bezoeken zien we dat meestal maar 1 database ondersteund wordt. Wanneer we een PHP-script gaan schrijven voor database-gebruik moeten we dus eerst weten welke database we gebruiken.

Er bestaat een standaard om een database te benaderen, namelijk ODBC[2]. Deze API is een standaard die gebruikt kan worden voor veel verschillende databases. Deze is echter een stuk trager omdat iedere aanvraag nogmaals vertaald moet worden naar de functies van de bepaalde database. Wanneer we echter niet weten welke database er aan de andere kant staan, is ODBC onze enige keuze.

Binnen deze cursus gaan we MySQL gebruiken. MySQL is een Open Source project dat heel snel en goed is. De database wordt actief ontwikkeld en is ideaal voor kleine tot middengrote database-applicaties. Het is geschreven volgens een client/server architectuur en is beschikbaar voor verschillende platformen. Bij veel verschillende hosting-providers is MySQL de enige beschikbare database. De omschakeling naar andere databasesystemen zou gemakkelijk moeten zijn.

8.1.5 MySQL, enkele opmerkingen

MySQL ondersteund de SQL92 standaard en ODBC level 0-2 standaard. Men kan MySQL instellen in verschillende talen, waaronder nederlands, frans en engels. MySQL heeft API's naar verschillende talen, waaronder C/C++, Java, Perl en natuurlijk PHP (duh!). Enkele andere eigenschappen van MySQL zijn de grote capaciteit, de snelheid en stabiliteit en de gratis beschikbaarheid onder Unix en OS/2. Voor het gebruik onder Windows moet er een licentie gekocht worden na 30 dagen.

MySQL heeft natuurlijk ook nadelen, zoals het niet ondersteunen van sub-selects in query's en triggers. Views zijn ook niet ondersteund in MySQL.

Ondanks deze tekortkomingen is MySQL een heel goede keuze voor een kleine tot middelmatige database.

[2]Open Database Connectivity

8.2 PHP en MySQL

Er bestaan heel veel functies om een MySQL-database te gebruiken onder PHP. We gaan die niet allemaal bespreken, maar enkel de meest gebruikte. Wanneer we speciale database-handelingen moeten gebruiken kunnen we altijd een gespecialiseerd boek raadplegen om alle functies te leren kennen.

Een standaardscript dat een database gebruikt gaat meestal eerst een connectie leggen met de database, daarna de inhoud van een script gaan verwerken en dan de database aanpassen of aanvullen. Uiteindelijk wordt de connectie terug afgesloten.

We zullen dan ook enkel functies bekijken om deze taken uit te voeren.

8.2.1 Openen, sluiten en database kiezen

Om een connectie te leggen met de database gebruiken we de functie `mysql_connect()` of met `mysql_pconnect()`. De parameters zijn bij beide functies gelijk. We geven de volledige syntax van `mysql_connect()`:

```
int mysql_connect(string hostname [:port] [:/path_to_socket],
                  string [username], string [password]);
```

Zoals we zien hebben we telkens een hostname nodig waarnaartoe we willen verbinden. Normaal gebeurt dit via een TCP/IP connectie. Wanneer de PHP-webserver en de MySQL database op dezelfde machine draaien kan PHP communiceren met de MySQL server via de socket. Meestal is de eerste parameter "localhost". De tweede en de derde parameter spreken voor zich. Wanneer we verbinden met de database moeten we normaal een username en een passwoord opgeven. Ook in een PHP-script is dat niet anders. Let erop dat het passwoord plain-text in de file komt te staan, wat dus eigenlijk onveilig is. Let erop dat andere mensen op je systeem de file kunnen komen lezen. Een echte oplossing voor dit probleem bestaat er niet, tenzij je op de een of andere manier de file die de passwoorden bevat eigendom kan maken van de webserver-user. `mysql_connect()` en `mysql_pconnect()` geven een integer terug. Die integer is een uniek getal dat de connectie met de database weergeeft. Meestal moet dit getal ook bijgehouden worden en gebruikt worden in de volgende database-oproepen. Wanneer de connectie niet gelukt is krijgen we een "false" waarde terug, dus kunnen we de volledige verbindingsconstructie in een if-constructie inpassen.

Let erop dat we perfect twee of meerdere connecties met –al dan niet– verschillende databases kunnen openen in een script.

Het verschil tussen `mysql_connect()` en `mysql_pconnect()` is (naast de 'p'[3] in de naam) miniem. Wanneer we PHP als een module draaien en `mysql_pconnect()` gebruiken gaat de connectie bij het afsluiten of bij het beëindigen van de het script *niet* gesloten worden. De module houdt de verbinding open. Wanneer er later een ander script uitgevoerd wordt dat probeert een verbinding te leggen via `mysql_pconnect()` of `mysql_connect()` met dezelfde parameter-waarden gaat de reeds geopende (en bewaarde) connectie gebruikt worden. Dit

[3] De 'p' staat voor 'persistent'

bespaart de overhead die ontstaat bij het openen en sluiten van connecties. Wanneer de connectie toch lang genoeg niet gebruikt wordt, gaat PHP de connectie uiteindelijk toch dichtgooien. `mysql_pconnect()` moet dus gebruikt worden bij sites die binnen een korte tijdspanne regelmatig een connectie leggen naar een bepaalde database met dezelfde username en password. Drukbezochte sites die veel dynamische pagina's bevatten kunnen dus beter `mysql_pconnect()` gebruiken. Sites waar er maar enkele hits in een dag komen kunnen echter beter `mysql_connect()` gebruiken.

Om een connectie af te sluiten kunnen we de functie `mysql_close()` gebruiken. De enige parameter die nodig is, is de integer die we verkregen hebben hij het openen van de connectie. Wanneer we geen parameter meegeven wordt de laatst geopende connectie afgesloten (ideaal wanneer we maar 1 connectie naar 1 database onderhouden? Nee, denk aan verdere uitbereidingen). `mysql_close()` geeft een "true" terug wanneer de afsluiting gelukt is, of een "false" wanneer `mysql_close()` de connectie niet kon sluiten.

`mysql_close()` zal nooit een connectie sluiten die met `mysql_pconnect()` geopend is, maar zal in dat geval *wel* een "true" teruggeven.

Wanneer we dan een connectie hebben geopend (met `mysql_connect()` of `mysql_pconnect()`) moeten we de correcte database selecteren. Dit gebeurt met de functie `mysql_select_db()`. De twee parameters zijn de naam van de database en de link-integer. We krijgen ofwel een "true" of een "false" terug naargelang het succes van de functie.

8.2.2 Query's en resultaten

Wanneer we een vraag stellen aan de database (bv "Geef ons alle cursisten") moet dat in de taal van de database. Daarvoor gebruiken we SQL. De vraag wordt doorgestuurd naar de database, en ofwel krijgen we een "false" terug, ofwel een positieve integer die een verwijzing bevat naar het resultaat. Het resultaat moet ofwel nog verder verwerkt worden (bijvoorbeeld bij een SELECT), ofwel geeft het gewoon een aanwijzing over het al dan niet slagen van de vraag (bij bv een CREATE-statement).

Wanneer we een resultaat hebben (bv van een SELECT-statement) moeten we meestal het resultaat nog verder gaan verwerken. We kunnen het aantal rijen dat een bepaalde query teruggaf gaan onderzoeken met de functie `mysql_num_rows()`. De enige parameter die nodig is, is de integer die teruggegeven werd van een `mysql_query()` oproep.

```
$result = mysql_query("SELECT DISTINCT * FROM cursisten");
$aantal_cursisten = mysql_num_rows($result);
echo "Er zijn $aantal_cursisten aanwezig in PHP deel 3";
```

De verschillende rijen kunnen we dan nog afzonderlijk gaan onderzoeken. Het gemakkelijkst hiervoor is de functie `mysql_fetch_array()`. Deze functie geeft een associatieve array terug met als indexen de namen van de kolommen zoals die in de database voorkomen. De waarde van een bepaald array-element is de inhoud van het veld in de tabel. De waarde die teruggegeven wordt is dus ofwel een array ofwel een "false". Deze constructie wordt veel gebruikt in combinatie met een while-lus.

```
<html>
  <head>
    <title>Databases</title>
  </head>

  <body>
    <h1>Een eerste voorbeeld</h1>
<?php
// mysql verbinden en database kiezen
$link = mysql_connect("localhost","phpuser","phppass");
if (!$link) die ("Fout: kan niet verbinden met de database");
mysql_select_db("postnummers");

// query uitvoeren
$result = mysql_query("SELECT * FROM postnummers LIMIT 0,30");
if (!$result) die("Fout bij het uitvoeren van de query...");
while($row = mysql_fetch_array($result))
{
  echo " De gemeente " . $row["gemeente"] . " heeft postcode " . $row["postfix"] . "<BR>";
}

mysql_close($link);
?>
    <hr>
    <address><a href="mailto:bernard@zeus.rug.ac.be">Bernard</a></address>
  </body>
</html>
```

Figuur 8.1: Een query uitvoeren...

`mysql_fetch_row()` doet hetzelfde als `mysql_fetch_array()`, maar slaat de namen niet op als strings in de indexen van de array, maar gewoon als getallen gebaseerd op de volgorde van de kolommen in de tabel of query. Velen beweren dat de functie `mysql_fetch_row()` sneller is dan `mysql_fetch_array()`. Dit is echter geen waar! `mysql_fetch_array()` is een stuk gemakkelijker in het gebruik en is dus aan te raden.

Om de interne pointer in de array van resultaten (niet de array die 1 rij teruggeeft, maar de array die de rijen aanwijst) te verplaatsen moeten we de functie `mysql_data_seek()` gebruiken. De eerste parameter is de integer die we terugkregen van de query-vraag. De tweede parameter is het nummer van de rij die we bij een volgende `mysql_fetch_xxx()` instructie willen terugkrijgen. Denk eraan dat we beginnen te tellen vanaf nul. We krijgen een "true" of een "false" terug in de vorm van een integer.

Deze functie wordt echter weinig gebruikt omdat een resultaat meestal slechts eenmaal doorlopen gaat worden om de resultaten te verwerken en de pagina op te bouwen.

Een andere handige functie is `mysql_fetch_object()`. Deze functie gaat geen array teruggeven met de waarden van een bepaalde rij uit een resultaat, maar geeft een object terug met als "properties" de namen en bijhorende waarden uit die bepaalde rij.

8.2.3 Fouten en geheugen

Wanneer we een "false" terugkrijgen van een bepaalde functie kunnen we met de functies `mysql_errno()` en `mysql_error()` het foutnummer en de foutmelding terugkrijgen. Dit kan handig zijn om deze weg te schrijven naar een logfile samen met de query. We kunnen dan later zien waar en wanneer een script gefaald heeft. Om de waarden van `mysql_errno()` correct te kunnen interpreteren moeten we een kijkje gaan nemen in de file `mysqld_error.h`.

Wanneer we echt grote query's uitvoeren (niet de query is lang, maar het resultaat is gigantisch groot (veel data in veel rijen)) kan het beter zijn dat we het geheugen die gebruikt wordt door een resultaat terug vrijgeven wanneer we het niet meer nodig hebben. Daarvoor gebruiken we de functie `mysql_free_result()`. De enige nodige parameter is de resultaat-integer. We krijgen terug een "true" of een "false". Voor grotere result-sets kunnen we ook `mysql_unbuffered_query()` gebruiken.

Een goed voorbeeld hoe we met databases kunnen werken is het Open Source Project 'PHP-MyAdmin'[4]. Het is een MySQL-administratie-programma dat volledig via PHP werkt.

[4]Te vinden op http://phpwizard.net/projects/phpMyAdmin/index.html

```
<html>
  <head><title>Databases</title>
<!-- Changed by: B Grymonpon, 21-Apr-2002 -->
</head>
  <body>
    <h1>Postnummers zelf onderzoeken</h1>
<?php
// mysql verbinden en database kiezen
$link = mysql_connect("localhost","phpuser","phppass");
if (!$link) die ("Fout: kan niet verbinden met de database");
mysql_select_db("postnummers");
?>
      <form method="GET" action="<?php echo $PHP_SELF?>">
        Geef de query in <input name="query" size=30>
        <input type="submit" value="Go!"><BR>
      </form>
<?php
      if(!empty($query)){
        $query = stripslashes($query);

        echo "<b>Query</b>" . htmlentities($query) . "<BR>";

        $result = mysql_query($query);
        if (!$result){
          echo "Er was een fout bij het uitvoeren van je query!<br>";
          echo mysql_errno() . ": " . mysql_error() . "<BR>";
          die("Quiting");
        }
        // blijkbaar is het resultaat juist... uitschrijven die handel
        echo "De resultaten van je query:<br>";
        echo "<TABLE size=90% align=center>";
        while($row = mysql_fetch_array($result))
        {
          echo "<TR>";
          while(list($key, $waarde) = each($row)) echo "<td>$waarde</td>";
          echo "</TR>\n";
        }
        echo "</TABLE>";
      }

mysql_close($link);
?>
  </body>
</html>
```

Figuur 8.2: Zelf query's uitvoeren

Hoofdstuk 9

De Zeus-links pagina

9.1 De vraag

Zeus WPI heeft op hun website een verwijzing naar een pagina waar links naar andere sites komen te staan. Deze pagina zou normaal moeten opgebouwd worden uit een database. Iedereen kan links aanbrengen via een web-interface. Deze worden opgeslagen en gekeurd door een keurteam. Wanneer de link goedgekeurd wordt komt hij uiteindelijk op de webpagina te staan met een stukje commentaar van de plaatser van de link. De links worden verder in categorieën onderverdeeld naargelang het onderwerp.

Aan deze site hebben we dus 3 delen. Eerst en vooral hebben we het gedeelte dat bijna geen interactie vraagt. Dit is de pagina die iedereen te zien krijgt. Er kan een categorie gekozen worden waarna alle links in die categorie getoond worden.

Het tweede deel is het deel waar leden links kunnen toevoegen. Deze worden in de database opgeslagen, maar nog niet getoond op de site.

Het derde deel is het deel van de moderator/administrator. Deze kan links ofwel goedkeuren of afkeuren. Hij kan categorieën toevoegen, wijzigen of verwijderen.

De bedoeling van dit hoofdstuk is het opbouwen van deze applicatie.

Opmerking! Bij de Zeus-site wordt er gebruik gemaakt van shtml-pagina's om de pagina's een uniform uitzicht te geven. We gaan proberen dit toe te passen in onze applicatie. Dit zal echter wel voor een extra belasting zorgen van de website omdat we de pagina's vooraf gaan laten parsen door de webserver.

9.2 De opbouw van de database

De database wordt voorzien van 2 tabellen. De eerste tabel bevat de categorieën. Daarin slaan we telkens de naam op van de categorie en een id dat uniek is voor deze categorie. De tweede tabel bevat de links. Per link houden we de URL en het commentaar bij. Verder houden we een veld bij dat het id van de categorie bevat, een veld dat bijhoudt als de link

al goedgekeurd werd, een datum en een uniek id. In de eerste tabel is de id de primary key. De tweede tabel heeft ook de id als primary key. Er bestaat in deze database een "1 op oneindig"-relatie tussen cat_id in de links-tabel en id in de categorieën tabel. De keuze van de primary key kan enkele database-mensen tegen het hart stoten, maar om een veld auto_increment te krijgen eist MySQL dat dit veld de primary key is.

De creatie van de database gebeurde via de commando-regel:

```
bernard@localhost:~$ mysqladmin -p create zeuslinks
Enter password:
Database "zeuslinks" created.
bernard@localhost:~$ mysql -p zeuslinks
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2 to server version: 3.22.32-log

Type 'help' for help.


mysql> create table categories(
    -> id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> name VARCHAR(100));
Query OK, 0 rows affected (0.00 sec)

mysql> create table links(
    -> id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> cat_id INTEGER UNSIGNED NOT NULL,
    -> url VARCHAR(200),
    -> description MEDIUMTEXT,
    -> date DATETIME,
    -> show ENUM('y','n'));
Query OK, 0 rows affected (0.01 sec)
```

9.3 De hulpfuncties

We hebben een file die ervoor zorgt dat de pagina's niet gecached worden aan de browser-kant. Deze wordt altijd eerst ingevoegd, aangezien deze bij de header moet gevoegd worden (figuur 9.1).

We hebben een file (figuur 9.2) waar al onze configuraties in terecht komen, waaronder de configuratie van de database-host, login en password.

Eerst en vooral gaan we al onze database-functies afzonderlijk plaatsen. Dit zorgt ervoor dat de aanpassing van de applicatie naar een andere database gemakkelijker wordt. Wanneer de file wordt aangeroepen gaan we ook een de database-connectie openen en een functie voorzien voor het afsluiten van de connectie. Deze wordt met de functie `register_shutdown_function()` toegevoegd aan het einde van het script (figuur 9.3).

```
<?php
    Header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
    Header("Last-Modified: ". gmdate("D, d M Y H:i:s") ." GMT");
    Header("Cache-Control: no-cache, must-revalidate");
    Header("Pragma: no-cache");
?>
```

Figuur 9.1: nocache.inc.php

```
<?php
// configuratiefile voor "zeuslinks"
// (c)2001 Bernard Grymonpon

// de configuratie van de database-parameters
$DBcfg["name"] = "zeuslinks";
$DBcfg["login"] = "phples";
$DBcfg["pass"] = "secret";
$DBcfg["host"] = "localhost";
$DBcfg["link"] = 0;

function full_stop($waarom){
  echo "Fout: $waarom";
  end_html();
  die();
}
?>
```

Figuur 9.2: config.inc.php

```
<?php
// database wrapper voor MySQL
// (c)2001 Bernard Grymonpon

function DoQuery($query){
  global $DBcfg;

  if(!$DBcfg["link"]){
    $DBcfg["link"] = mysql_pconnect($DBcfg["host"],$DBcfg["login"],$DBcfg["pass"]);
    if (!$DBcfg["link"]){
      die("Problemen met de database");
    }
    if (!mysql_select_db($DBcfg["name"])){
      die("Problemen met de database");
    }
  }
  $res = mysql_query($query);
  return $res;
}

function FetchArray($res){
  $row = mysql_fetch_array($res);
  return $row;
}

function NumRows($res){
    $row = mysql_num_rows($res);
    return $row;
}

function InsertId(){
    return mysql_insert_id();
}

function CloseDb(){
  global $DBcfg;

  if($DBcfg["link"]) mysql_close($DBcfg["link"]);
}

register_shutdown_function(CloseDb);
?>
```

Figuur 9.3: database.inc.php

We gaan ook het begin en het einde van de pagina in een aparte functie inplakken. Dit zorgt ervoor dat de code in de scripts leesbaarder is, en dat de HTML meer gescheiden wordt van het eigenlijke programma (figuur 9.4).

```
<?php

function start_html($titel = "Zeus Links"){
  echo "<html><head><title>";
  echo $titel;
  echo "</title></head><body bgcolor=\"#FFFFFF\">";
}

function end_html(){
  echo "</body></html>";
}
?>
```

Figuur 9.4: html.inc.php

Al de vorige includes worden in 1 include ingepast, zodat we die gemakkelijk kunnen hergebruiken doorheen het hele project (figuur 9.5).

```
<?php
require("nocache.inc.php");
require("config.inc.php");
require("database.inc.php");
require("html.inc.php");
?>
```

Figuur 9.5: zeuslinks.inc.php

9.4 De toevoeg-module

Deze module moet een gebruiker toelaten een link toe te voegen. Het toevoegen van een URL gaat via 1 enkele pagina waar alle gegevens opgevraagd worden.

9.4.1 De eisen

We zullen alles in 1 pagina verwerken. De aanvraag en de toevoeging wordt in 1 script ingevoegd. Door middel van de controle van de parameters kunnen we zien als er al dan niet een aanvraag tot toevoeging is.

9.4.2 De implementatie

We hebben enkele functies gedefinieerd om het hoofdprogramma korter te houden. We hebben een functie om een volledig formulier te plaatsen. Deze functie wordt aangeroepen wanneer we het lege formulier willen plaatsen in de browser. De functie zorgt ervoor dat de uitlijning een beetje deftig is en dat de oude waarden terug weergegeven worden wanneer deze bestonden.

Wanneer we detecteren dat er iets toegevoegd wordt gaan we invoer van de gebruiker controleren en klaarmaken voor opslag. We moeten erop letten dat de data onze query niet om zeep kan helpen.

Na de toevoeging wordt er een mooie tekst op het scherm geschreven dat de gebruiker meld dat de link toegevoegd werd voor verwerking. Figuur 9.6 toont ons het toevoeg-stuk.

9.5 De administratie-module

De administratiemodule bestaat uit 3 delen. We hebben het deel dat ervoor zorgt dat de categorieën aangepast kunnen worden, het deel dat de links aanpast en een deel dat de toegevoegde links gekeurd worden.

9.5.1 De eisen

Om het project overzichtelijk te houden gaan we de 3 delen in drie verschillende files opsplitsen. De administratie is een stuk moeilijker te implementeren, maar wanneer we het project in kleinere stukken opsplitsen zien we snel dat de adminstratie van de links of de categorieën vrij goed overeen komt.

9.5.2 De implementatie

Eerst bekijken we de adminstratie van de categorieën in figuren 9.7 en 9.8. We zien later dat de implementatie van de links ongeveer gelijklopend is, maar dan met meer velden die gecontroleerd moeten worden.

De moderatie van de links gebeurt via een systeem dat alle links uitschrijft met 3 radiobuttons. We kunnen kiezen voor “Ja”, wat ervoor zorgt dat de link toegevoegd wordt. Verder zorgt “Nee” ervoord dat de link verwijderd wordt en “Hold” dat de beslissing uitgesteld wordt (geen wijziging). We zien de code in figuur 9.9.

9.5.3 Enkele opmerkingen

9.6 Opmerkingen over het volledige project

Wanneer we dit project beter en groter willen uitwerken, kunnen we nog enkele opmerkingen maken. Zo kunnen we klassen introduceren die de data van een categorie of een link bijhouden.

```
<?php
require("zeuslinks.inc.php");
start_html("Een link toevoegen!");

if(isset($url) && $url != ""){
    // er is data beschikbaar om toe te voegen...
    // controle (denk eraan dat magic_quotes aanstaat!)
    if ($description == "" || $categorie == ""){
        full_stop("Gelieve alle velden in te vullen!<br>");
    }
    $cat_query = "SELECT * FROM categories WHERE id='$categorie'";
    $res = DoQuery($cat_query);
    if (NumRows($res) != 1){
        full_stop("Fout bij de keuze van de categorie!<br>");
    } else {
        if(!$row = FetchArray($res)){
            full_stop("Fout bij het ophalen van de categorienaam!");
        } else $cat_name = $row["name"];
  }

    $datetime = date("Y-m-d H:i:s");

    $insert_query = "INSERT INTO links(cat_id,url,description,date,onpage)
VALUES('$categorie','$url','$description','$datetime','n')";
    if(!DoQuery($insert_query))
        full_stop("Fout tijdens het toevoegen van de link!<br>");
    else
        echo "Uw link werd toegevoegd aan de database.<br>\n";
} else {
?>
    <h1>Een link toevoegen</h1>
    <form method="GET" action="<?php echo $PHP_SELF ?>">
      Uw URL: <input name="url" size=40><br>
      Uw categorie: <select name="categorie">
<?php
    $res = DoQuery("SELECT * FROM categories");
    while($row = FetchArray($res))
        echo "<OPTION value=" . $row["id"] . ">" . $row["name"];
?>
      </select><br>
      Uw beschrijving:<br>
      <textarea cols=60 rows=10 name="description">
Hier uw beschrijving...
      </textarea><br>
      <input type="submit" value="Ok, toevoegen">
    </form>
<?php
} // IF ELSE
end_html();
?>
```

Figuur 9.6: add.php

```
<?php

require("zeuslinks.inc.php");

start_html("Administratie - Categorieen");

if(isset($a_name)){
// een categorie toevoegen!
$query = "INSERT into categories VALUES(’’,’$a_name’)";
if (!DoQuery($query))
full_stop("Fout tijdens het toevoegen!");
else {
echo "De categorie werd aangemaakt...";
echo "<a href=$PHP_SELF>Verdergaan</a>";
}
} elseif(!isset($id)){
// geen id gezet, dus gaan we een lijst weergeven...
echo "<h1>Lijst</h1>";
list_cat();
} else {
if(!isset($action)){
    echo "<h1>Informatie over een categorie...</h1>";
show_cat($id);
echo "<a href=$PHP_SELF?action=delete&id=$id>Verwijderen</a><br>";
} elseif ($action=="modify"){
modify_cat($id, $m_name);
      echo "<a href=$PHP_SELF>Verdergaan</a>";
} elseif ($action=="delete"){
delete_cat($id);
      echo "<a href=$PHP_SELF>Verdergaan</a>";
}
}
?>
<h2>Toevoegen</h2>
<form method=get action=<?php echo $PHP_SELF?>>
Geef een naam: <input name=a_name>
<input type=submit value=toevoegen>
</form>
<?php
end_html();
?>
```

Figuur 9.7: De administratie van de categorieën

```
<?php
function delete_cat($id){
    if(!DoQuery("DELETE FROM categories WHERE id='$id'"))
      full_stop("Fout tijdens het verwijderen van de categorie!");
    else
      echo "De categorie werd verwijderd...";
}

function modify_cat($id, $name){
    $query = "UPDATE categories SET name='$name' WHERE id='$id'";
    if(!DoQuery($query))
      full_stop("Fout tijdens het aanpassen van de categorie!");
    else
      echo "De categorie werd aangepast...";
}

function show_cat($id){
    if(!$cat = DoQuery("SELECT * FROM categories WHERE id='$id'"))
      full_stop("Fout tijdens het ophalen van de categorie!");

    if(NumRows($cat) != 1)
      full_stop("Fout tijdens het ophalen van de categorie!");

    $row = FetchArray($cat);
    echo "<form method=GET action=$PHP_SELF>";
    echo "<input type=hidden name=action value=modify>";
    echo "<input type=hidden name=id value=$id>";
    echo "<table>";
    // dubbele associatieve array, maar de oneven waarden
    // nodig... daarom (each && list = each)
    while(each($row) && list($key,$waarde) = each($row)){
      echo "<tr><td><b>$key</b></td>";
      echo "<td><input name=m_$key value=\"$waarde\"></td></tr>\n";
    }
    echo "</table>";
    echo "<input type=submit value=Wijzigen></form>";
}

function list_cat(){
    if(!$cat = DoQuery("SELECT * FROM categories"))
        full_stop("Fout tijdens het ophalen van de categorieen!");

    while($row = FetchArray($cat)){
        $cat_id = $row["id"];
        $cat_name = htmlentities($row["name"]);
        echo "<a href=$PHP_SELF?id=$cat_id>$cat_name</a><BR>\n";
    }
    echo "Aantal categorieen in de database = " .NumRows($cat);
}

?>
```

Figuur 9.8: Hulpfuncties bij admin_cat.php

```
<?php
require("zeuslinks.inc.php");
start_html("Administratie - Moderatie");

if(isset($moderate)){
  // y wordt toegelaten, n gewist en h wordt ongewijzigd gelaten
  for($i=0; $i < count($keys); $i++){
    $id = $keys[$i];

    $count[$moderate[$i]]++;
    switch($moderate[$i]){
    case "y" :
      $query = "UPDATE links SET onpage='y' WHERE id='$id'";
      break;
    case "n" :
      $query = "DELETE FROM links WHERE id='$id'";
      break;
    case "h" :
      break;
    }
    if(isset($query) && !DoQuery($query))
      echo "Fout tijdens het moderaten van link $id!<br>";
  }
  echo "Aantal toegelaten : " . $count[y] . "<br>";
  echo "Aantal weerhouden : " . $count[n] . "<br>";
  echo "Aantal onveranderd : " . $count[h] . "<br>";
  echo "<a href=$PHP_SELF>Nog een keer!</a>";

} else {

  $query="SELECT * FROM links WHERE onpage='n'";
  if (!$res = DoQuery($query))
      full_stop("Fout tijdens het ophalen van de links!");
  if (NumRows($res) > 0){
    echo "<form action=$PHP_SELF method=GET>";
    echo "<table border=1>";
    while($row = FetchArray($res)){
      $id = $row["id"];
      $url = $row["url"];
      $descr=nl2br(htmlentities($row["description"]));
      echo "<tr valign=top>";
      echo "<td>$id<input type=hidden name=keys[] value=$id></td>";
      echo "<td>$url</td>\n <td>$descr</td>\n";
      echo "<td><SELECT name=moderate[]>";
      echo "<option value=y>Ok</option>";
      echo "<option value=n>Delete</option>";
      echo "<option value=h selected>Hold</option>";
      echo "</select>";
      echo "</td></tr>\n";
    }
    echo "</table>";
    echo "<input type=submit value=Ok></form>";
  }
}
end_html();
?>
```

Wanneer we dan ervoor zorgen dat via de constructor onmiddelijk een object kan geladen worden aan de hand van een id, kunnen we snel en gemakkelijk verschillende aanpassingen doen wanneer het nodig zou zijn.

Een andere, maar minder intressante toepassing is het gebruik van de netwerk-functies van PHP om de gegeven link uit te testen door een netwerkverbinding te openen naar die pagina.

Er is ook niets van beveiliging aanwezig. Denk eraan dat we bijvoorbeeld de administrator-module best beschermen via een username/password-combinatie. We kunnen natuurlijk ook zelf onze beveiliging inbouwen. Wanneer we de situatie van Zeus beschouwen hebben we natuurlijk de leden-database waarmee we de controle kunnen doen voor de toegang tot de verschillende delen van dit project.

Andere aanvullingen zijn beschrijvingen bij de categorieën en een betere integratie tussen de verschillende administratie-modules.

Al bij al is dit een goed begin, maar voor het officieel in productie zou gaan moet er nog een en ander veranderen.