

PHP Basis

Beginnen met PHP: HyperText Preprocessor



CompuFirm.nl

INHOUDSOPGAVE

PHP BASIS	1
BEGINNEN MET PHP: HYPERTEXT PREPROCESSOR	1
VOORWOORD	13
1.....	15
INLEIDING TOT SERVER-SIDE SCRIPTING.....	15
1.1 WAT IS PHP?	15
1.2 GESCHIEDENIS.....	15
1.3 PHP VERSIES.....	16
1.4 ZEND.....	17
1.5 LAMP PLATFORM	18
1.6 GEBRUIK EN POPULARITEIT	18
1.7 STATISCHE VS. DYNAMISCHE WEBPAGINA'S.....	19
1.8 OPVRAGEN VAN EEN (STATISCHE) WEBPAGINA.....	20
1.9 OPVRAGEN VAN EEN (DYNAMISCHE) PAGINA	21
1.9.1 <i>Client-side Scripting</i>	21
1.9.2 <i>Server-side Scripting</i>	23
1.9.3 <i>Client-side vs. Server-side</i>	24
1.10 WAT KUNNEN WE MET PHP BEREIKEN?	25
1.11 ANDERE SERVER-SIDE TECHNOLOGIEËN	26
2.....	27
BEGINNEN MET PHP.....	27
2.1 WAT HEEFT U NODIG?	27
2.1.1 <i>(HTML) Editor</i>	27
2.1.2 <i>Webserver</i>	28
2.1.3 <i>PHP Software</i>	30
2.1.4 <i>RDBMS (Database)</i>	31
2.1.5 <i>MySQL beheer</i>	33
2.1.6 <i>Complete oplossingen</i>	34
2.1.7 <i>FTP software</i>	39
2.1.8 <i>Browser</i>	39
2.2 PHP.INI	39
2.2.1 <i>Aanpassingen maken</i>	40
2.3 VERWERKING EN OPBOUW PHP PAGINA'S	43
2.3.1 <i>.php extensie</i>	43
2.3.2 <i>Script delimiters</i>	44
2.3.3 <i>De PHP Scripting Language</i>	46
2.4 UW EERSTE PHP PAGINA.....	47
3.....	49
SCRIPTEN MET PHP	49
3.1 INLEIDING.....	49
3.2 ONLINE MANUAL.....	49
3.3 PHPINFO()	50
3.4 STATEMENTS.....	51

3.4.1 punt-komma	51
3.4.2 Eigen regel	52
3.4.3 Witruimte	52
3.4.4 Meerdere regels	53
3.5 VARIABELEN	53
3.5.1 Wat is een variabele?	53
3.5.2 Variabelen declareren	53
3.5.3 Naamgeving	54
3.5.4 Variabelen initialiseren	56
3.6 DATATYPEN	56
3.6.1 String	57
3.6.2 Integer	58
3.6.3 Floating Point	58
3.6.4 Boolean	59
3.6.5 null	59
3.7 ARRAYS	60
3.7.1 Numerieke array aanmaken	61
3.7.2 Arrays inzien met <code>print_r()</code>	64
3.7.3 Arrays uitlezen	66
3.7.4 Associatieve arrays	68
3.8 OBJECTS	69
3.8.1 Objects met Native PHP Classes	69
3.8.2 Classless objects met <code>stdClass()</code>	70
3.8.3 Object Literals (bestaan niet)	71
3.9 <code>VAR_DUMP()</code>	71
3.9.1 <code>var_export()</code>	73
3.10 DATUMS	73
3.11 VARIABELEN FORMATTEREN	74
3.11.1 Locale zetten	75
3.11.2 Datums	75
3.11.3 Getallen	76
3.11.4 Valuta	77
3.12 CONSTANTEN	78
3.12.1 Constanten zijn (standaard) hoofdlettergevoelig	79
3.12.2 Waarom constanten gebruiken?	79
3.12.3 Configuratiebestanden	79
3.12.4 PHP Constanten	80
3.12.5 Compile-time constants	81
3.13 OMGEVINGSVARIABELEN	81
3.13.1 Superglobals	82
3.13.2 Veiligheid	82
3.13.3 Overzicht	84
3.13.4 Beschikbaarheid	85
3.14 IN DE PAGINA SCHRIJVEN	85
3.14.1 Echo vs. Print	86
3.14.2 <code>printf()</code> en <code>sprintf()</code>	86
3.14.3 Linebreaks in string	87
3.14.4 Switchen tussen HTML en PHP modus	87
3.14.5 Valide HTML uitvoer	87
3.14.6 Logica en presentatie scheiden	88
3.14.7 String Concatenation	90
3.14.8 Strings combineren met HTML	91
3.14.9 Escapen van double quotes	92
3.14.10 Double vs. single quotes	92
3.14.11 Overzicht	98
3.15 COMMENTAAR	99
3.15.1 Eenregelig commentaar	100

3.15.2 Meerregelig commentaar.....	100
3.15.3 Begincommentaar	101
3.15.4 Code 'uitcommentariëren'	102
3.15.5 DocBlocks	103
3.15.6 Tips	103
3.16 CONTROLESTRUCTUREN	103
3.17 SELECTIES	104
3.17.1 Vergelijkingsoperatoren	105
3.17.2 Booleaanse operatoren	106
3.17.3 If.....	107
3.17.4 Else	108
3.17.5 If Else combineren met HTML.....	109
3.17.6 Ternary (Conditional) Operator	110
3.17.7 Nesten	111
3.17.8 Elseif	112
3.17.9 Switch	113
3.18 MATCH (PHP8).....	116
3.19 HERHALINGEN (LUSSEN).....	118
3.19.1 While	119
3.19.2 For	124
3.19.3 Foreach.....	127
3.19.4 Lussen combineren met if ... else.....	131
3.19.5 comma seperated values.....	131
3.19.6 Break en continue.....	132
3.19.7 Nesten	133
3.20 GESTRUCTUREERD PROGRAMMEREN.....	134
3.20.1 Pseudo-code	134
3.21 WISKUNDIGE OPERATOREN	134
3.21.1 Operator precedence.....	135
3.22 PHP TEMPLATE SYNTAX.....	136
3.22.1 Ternary Control operator.....	138
3.22.2 Switch	139
3.22.3 Voorkom extra witruimte	140
3.23 ANDERE UITVOER DAN HTML	140
3.23.1 XML.....	141
3.23.2 JSON.....	142
3.23.3 Plain Text	143
3.24 CODING CONVENTIONS	144
3.24.1 Controle structuren.....	145
3.24.2 Conclusie.....	147
4.....	149
WERKEN MET FORMULIEREN	149
4.1 INLEIDING	149
HTML FORMULIEREN	150
4.2 DATA VERZENDEN	152
4.2.1 Method GET.....	152
4.2.2 Method POST.....	153
4.2.3 Conclusie.....	154
4.3 URL ENCODING	154
4.4 DATA VERWERKEN.....	155
4.4.1 Register_Globals.....	155
4.4.2 \$_GET	155
4.4.3 \$_POST	156
4.4.4 \$_REQUEST.....	156

4.4.5 <i>print_r()</i>	157
4.5 HTML ESCAPE	158
4.6 LOS OPVRAGEN FORMHANDLER.....	160
4.6.1 <i>isset()</i>	161
4.6.2 <i>redirect met header('location: ...')</i>	161
4.6.3 <i>ob_start()</i>	162
4.6.4 <i>Controle</i>	162
4.7 LEEG GELATEN VELDEN	163
4.7.1 <i>Checkboxes en radiobuttons</i>	163
4.8 (SIMPELE) VALIDATIE	164
4.9 HIDDEN FIELDS	166
5.....	171
WERKEN MET DATABASES	171
5.1 INLEIDING	171
5.2 GESCHIEDENIS	171
5.3 WAT IS EEN DATABASE?	172
5.3.1 <i>Structuur</i>	172
5.4 STRUCTURED QUERY LANGUAGE (SQL)	174
5.4.1 <i>De Data Definition Language (DDL)</i>	175
5.4.2 <i>De Data Manipulation Language (DML)</i>	175
5.4.3 <i>Data Control Language (DCL)</i>	175
5.5 PHPMYADMIN	175
5.6 DATABASE AANMAKEN	176
5.7 TABELLEN AANMAKEN.....	176
5.8 AFSLUITING.....	178
6.....	181
DATABASE GEBRUIKEN.....	181
6.1 INLEIDING	181
6.2 MYSQL API KIEZEN	181
6.2.1 <i>MySQL API</i>	182
6.2.2 <i>mysqlnd</i>	183
6.3 MYSQLI	183
6.3.1 <i>Installatie</i>	183
6.3.2 <i>Voordelen</i>	184
6.3.3 <i>Nadelen</i>	184
6.3.4 <i>Migratie MySQL naar MySQLi</i>	184
6.3.5 <i>Meer informatie</i>	185
6.4 STAPPENPLAN	185
6.5 STAP 1: VERBINDEN MET MYSQL SERVER	185
6.6 STAP 1A: CHARSET SPECIFICEREN	187
6.7 STAP 2: DATABASE SELECTEREN.....	189
6.7.1 <i>Verbinden en selecteren combineren</i>	189
6.8 STAP 3: QUERY OPSTELLEN	189
6.8.1 <i>Gegevens selecteren: SELECT</i>	190
6.8.2 <i>SELECT statement in PHP String</i>	190
6.8.3 <i>Case Sensitivity</i>	191
6.8.4 <i>Selecteer alleen de benodigde velden</i>	191
6.8.5 <i>Witruimte in SQL Statement</i>	192
6.8.6 <i>Volgorde</i>	193
6.9 STAP 4: QUERY UITVOEREN.....	193
6.9.1 <i>mysqli_error()</i>	193

6.9.2 Query tonen bij error	195
6.9.3 mysqli_errno()	195
6.9.4 Eén query	196
6.10 STAP 5: GEGEVENS TONEN	196
6.10.1 mysqli_fetch_* functies	197
6.10.2 mysqli_fetch_row()	198
6.10.3 mysqli_fetch_assoc()	198
6.10.4 mysqli_fetch_array()	199
6.10.5 Tonen met while lus	199
6.10.6 In de lus naar HTML mode	201
6.11 STAP 6: GEHEUGEN VRIJGEVEN (OPTIONEEL)	204
6.12 STAP 7: VERBINDING VERBREKEN (OPTIONEEL)	204
6.13 HET HELE SCRIPT	205
6.14 VERDIEPING: GENERIEK TONEN	206
6.15 SELECTIE ORDENEN: ORDER BY	208
6.15.1 Meerdere velden	209
6.15.2 Positie in plaats van veldnaam	209
6.15.3 VERDIEPING: Groeperen	209
6.16 SELECTIE BEPERKEN: FIELDLIST EN WHERE	210
6.16.1 fieldlist	210
6.16.2 WHERE Clause	211
6.16.3 Quotes	211
6.16.4 Case insensitivity	212
6.16.5 Operatoren	212
6.17 CONTROLE OP AANWEZIGHEID RECORDS	214
6.18 AFBEELDINGEN EN HYPERLINKS TONEN	215
6.19 GEGEVENS IN EEN FORMULIER TONEN	216
6.20 GEBRUIK VAN DYNAMISCHE SQL STATEMENTS	216
6.20.1 beveiliging tegen geïsoleerd opvragen	218
6.20.2 Hergebruik medewerkers_tonen.php	219
6.20.3 Sprintf()	220
6.21 GEGEVENS TOEVOEGEN: INSERT	221
6.21.1 Andere formaten	221
6.21.2 Insert in PHP	222
6.21.3 Insert data uit formulier	223
6.21.4 'Testen' query	223
6.21.5 Action query: geen resultset	224
6.21.6 Veel gemaakte fouten	225
6.21.7 VERDIEPING: duplicate records voorkomen	227
6.22 GEGEVENS AANPASSEN: UPDATE	232
6.22.1 WHERE clause	233
6.23 GEGEVENS VERWIJDEREN: DELETE	238
6.23.1 Soft Delete	238
6.23.2 Verwijderen bevestigen	240
6.24 ALTERNATIEVE PRESENTATIE	240
6.24.1 Verwijderen bevestigen	241
6.25 AFSLUITING	247
7	248
MODULARISEREN	248
7.1 INLEIDING	248
7.2 INCLUDES	249
7.2.1 Notatie	249
7.2.2 Paden	249
7.2.3 Include_path	250

7.2.4 HTML insluiten.....	251
7.2.5 PHP code insluiten	252
7.2.6 PHP functies insluiten	253
7.2.7 Variabele includes	254
7.2.8 (Geen) Include Parameters	255
7.2.9 require().....	256
7.2.10 include_once() en require_once().....	257
7.2.11 Waarschuwingen onderdrukken	257
7.2.12 Beveiliging tegen los opvragen.....	258
7.2.13 Auto_prepend_file & auto_append_file	260
7.2.14 PRAKTIJK.....	261
7.3 FUNCTIES	273
7.3.1 Language constructs.....	274
7.3.2 Werking	275
7.3.3 Voorbereiding	276
7.3.4 Opbouw	276
7.3.5 Indeling	277
7.3.6 Aanroepen functie	278
7.3.7 Parameters en Argumenten	279
7.3.8 Type Hinting for Parameters	280
7.3.9 meer/minder argumenten.....	283
7.3.10 flexibel aantal argumenten met func_*_arg*()	283
7.3.11 array als parameter.....	284
7.3.12 JSON string als parameter	285
7.3.13 Variadic functions met de ... operator.....	286
7.3.14 Default argumenten	286
7.3.15 Named Arguments	287
7.3.16 By Value vs. By Reference.....	288
7.3.17 Static.....	290
7.3.18 Bereik (scope)	291
7.3.19 Render Functies	295
7.3.20 Waarden retourneren.....	296
7.3.21 Typed return values.....	297
7.3.22 PHPDoc Blocks	297
7.3.23 Functies: PRAKTIJK.....	298
7.3.24 VERDIEPING: Callback functies	311
7.3.25 VERDIEPING: Anonieme functies (Closures)	313
7.3.26 Aanbeveling en tips	314
7.4 ALTERNATIEVEN.....	315
7.5 SELF-SUBMITTING FORMS	315
7.5.1 Opbouw formulier	317
7.5.2 Logica	317
7.5.3 Controle op Submit button	318
7.5.4 Controle op hidden field	320
7.5.5 Verdieping: Volgorde omdraaien en functie.....	321
7.5.6 Multiple Submits en Form Spoofing.....	322
7.6 REDIRECTING	324
7.6.1 Output buffering.....	325
7.6.2 Debugging	326
7.6.3 Absolute URL	326
7.6.4 Alternatieven	326
7.6.5 Querystring.....	327
7.7 PRAKTIJKVOORBEELD: SELF-REFERENCING PAGE	327
8.....	329
WERKEN MET COOKIES	329

8.1 INLEIDING	329
8.2 COOKIES.....	330
8.2.1 Werking	330
8.2.2 Soorten	330
8.2.3 Beperkingen.....	331
8.2.4 Toepassingen.....	331
8.3 ZETTEN EN UITLEZEN NON-PERSISTENT COOKIE	331
8.3.1 Name en value.....	332
8.3.2 Uitlezen.....	334
8.3.3 Aanwezigheid cookie controleren met <code>isset()</code>	334
8.3.4 Inhoud cookie controleren / veilig weergeven.....	335
8.4 ZETTEN EN UITLEZEN PERSISTENT COOKIE	335
8.4.1 Absolute vervaldatum	336
8.4.2 Relatieve vervaldatum.....	338
8.4.3 Storage Location.....	338
8.5 COOKIE WIJZIGEN.....	340
8.6 COOKIE VERWIJDEREN.....	341
8.7 DEVELOPER TOOLS	341
8.7.1 Met de FireBug Extensie in FireFox.....	342
8.7.2 FireCookie.....	342
8.7.3 Met de Web Developers Toolbar in FireFox.....	343
8.7.4 In FireFox	343
8.7.5 In Microsoft Internet Explorer	344
8.8 SCOPE	346
8.8.1 Path	346
8.8.2 Domain	346
8.8.3 Secure	346
8.9 MEERDERE COOKIES	346
8.10 MEERDERE COOKIES ALS ARRAY UITLEZEN	348
8.11 MEERDERE WAARDEN IN ÉÉN COOKIE.....	349
8.11.1 Waarde kan geen array zijn	349
8.11.2 Explode().....	350
8.11.3 <code>parse_str()</code> en <code>http_build_query()</code>	350
8.11.4 <code>serialize()</code> en <code>unserialize()</code>	351
8.12 COOKIES 'VERSLEUTELN'.....	352
8.13 VOORBEELD: INSTELLING VASTHOUDEN	353
8.14 SAMENVATTING.....	354
9.....	355
WERKEN MET SESSIES	355
9.1 INLEIDING.....	355
9.2 WERKING	355
9.3 SESSIE STARTEN	356
9.4 SESSIE VARIABLE AANMAKEN.....	357
9.4.1 <code>Save_PathValue</code>	358
9.5 SESSIE VARIABLE UITLEZEN.....	358
9.5.1 Inlezen in globale variabele	359
9.5.2 Bereik.....	359
9.5.3 Controle	359
9.5.4 Meerdere gebruikers	360
9.5.5 Toepassingen.....	360
9.5.6 Meerdere Sessievariabelen.....	360
9.5.7 Datatypen.....	360
9.6 SESSIE VARIABLE AANPASSEN	361
9.6.1 Sessie variabelen en <code>register_globals</code>	361

9.7 DUUR VAN DE SESSIE	361
9.8 SESSIE VARIABLE VERWIJDEREN.....	362
9.9 SESSION BEËINDIGEN	362
9.10 CLIENT ZONDER COOKIE SUPPORT, WAT NU?	363
9.11 APPLICATIE VARIABLEN	364
9.12 PRAKTIJK: LOGIN	365
9.12.1 Voorbereidingen	365
9.12.2 Login Form	365
9.12.3 Login Check.....	365
9.12.4 Logout.....	368
9.12.5 Rest Navigatie aanpassen	369
9.12.6 Login tegen de database	369
9.12.7 Userlevels	371
9.12.8 Password hashing.....	372
1.1.1 Salt.....	375
9.12.9 Slow Hash algoritmes	376
9.12.10 Crypt().....	376
9.12.11 Password API.....	376
9.12.12 password_hash.....	376
9.12.13 password_verify().....	377
9.12.14 Alternatieven	378
9.13 LINKS.....	378
10.....	381
BIJLAGEN.....	381
10.1 SCHEMA MINI-APPLICATIE.....	381
10.2 EXTRA OEFENINGEN	382
10.2.1 Formulier Datum Tonen.....	382
10.2.2 Formulier Tabel Maken	383
10.2.3 Database Extra 1	386
10.2.4 Database Extra 2	386
10.2.5 Database Extra 3	387
10.2.6 Database Extra 4	388
10.2.7 Database Extra 5	388
10.2.8 Database Extra 6	388
10.2.9 Database Extra 7	388
10.3 PHP VERVOLG CURSUS	388

CompuFirm.nl

Voorwoord

Tijdens deze cursus wordt een solide basis gelegd voor het bouwen van interactieve websites met PHP. Het accent ligt bij de koppeling van de website met een (MySQL) database. U leert onder andere om met behulp van formulieren de inhoud van een database te tonen en te muteren.

Tijdens deze training zal dan ook duidelijk worden dat tijdens het programmeren in PHP drie talen elkaar telkens overlappen; HTML, SQL en PHP. We verwachten van de cursist geen andere voorkennis dan HTML. Programmeerkennis is niet vereist en alle basale database concepten, inclusief het schrijven van queries in SQL, komen in de cursus aan de orde.

Wij wensen u veel succes en plezier toe tijdens het volgen van deze cursus.

Namens CompuFirm,

Arno Mol
maart 2021

Inleiding tot Server-side Scripting

1.1 Wat is PHP?

PHP is een **gratis, platform onafhankelijke, scripttaal**. U kunt er mee op de **commandline** werken en er zelfs **Windows-achtige programma's** (PHP-GTK) mee schrijven. Maar PHP wordt vooral gebruikt om **dynamische, interactieve websites** te bouwen. De scripts worden uitgevoerd op de webserver en dat maakt PHP een **Server-side** technologie.

In dit hoofdstuk zal u precies duidelijk worden wat Server-side scripting nu eigenlijk inhoudt. Voordat we echter op de technische details in zullen gaan willen we u in deze paragraaf iets meer vertellen over de achtergrond en de voor- en nadelen van PHP.

1.2 Geschiedenis

Het acroniem PHP stond **oorspronkelijk** voor **Personal Home Page tools**.



PHP was in den beginnen namelijk de creatie van één persoon: de Deen **Rasmus Lerdorf**. In de herfst van **1994** zocht deze in Groenland geboren programmeur naar een manier om bij te kunnen houden wie zijn online CV bezochten. Hij bedacht daarvoor een oplossing in de vorm van een aantal in C geschreven CGI binaries. Ze vervingen de Perl scripts waarmee hij tot dan zijn site beheerde. Zijn oplossing viel bij een hoop andere programmeurs erg in de smaak en Rasmus stelde vervolgens op veler verzoek in 1995 de code beschikbaar onder de noemer **Personal Home Page Tools (PHP)**.



Rasmus Lerdorf

Rasmus bleef mogelijkheden toevoegen, waaronder een **Forms Interpreter (FI)** wat leidde tot een **tweede versie: PHP/FI**, waar andere programmeurs ook weer verbeteringen aan toevoegden waardoor PHP nog populairder werd. In 1997 ging Rasmus samenwerken met de Israëliërs **Zeev Suraski** en **Andi Gutmans** die ondermeer een volledig nieuwe scripting engine ontwierpen. En er werden database connecties aan toegevoegd.

Met versie 3 deed de hobby-achtige naam geen recht meer aan de geavanceerde mogelijkheden en werd een verkiezing voor een nieuwe naam uitgeschreven. PHP wordt vanaf dan uitgelegd als een **recursief acroniem** dat staat voor **PHP: Hypertext Preprocessor**. Oftewel:

P = PHP

H = Hypertext

P = Processor

De eerste P staat dus weer voor PHP en dat kunt u ook weer uitsplitsen etc. (Vergelijkbaar met GNU: GNU's Not Unix). Beetje vreemde naam wellicht, maar nerds vinden dit "Droste effect" leuk! En het is in ieder geval beter dan Pretty Happy Programmers dat ook werd voorgesteld...

Ja ja, een van de eerste dingen die mensen zich vaak afvragen is waar PHP voor staat (en daarna wilden ze dat ze het niet gevraagd hadden ;-). Meer over de geschiedenis van PHP vindt u op <http://php.net/manual/en/history.php> en <http://en.wikipedia.org/wiki/PHP>.

1.3 PHP versies

Sinds de herfst van 1994, toen het idee voor PHP ontstond, zijn de volgende versies van PHP verschenen:

Versie	Naam	Jaar	Zend Engine
1.0	Personal Home Page Tools	begin 1995	-
2.0	PHP/FI (Forms Interpreter)	1995-1997	-
3.0	PHP: Hypertext PreProcessor	1997-2000	-
4.0	"	midden 2000	Zend Engine 1
4.1	"	december 2001	"
4.2	"	april 2002	"
4.3	"	december 2002	Zend Engine 2
5.0	"	juli 2004	"
5.2	"	november 2006	"
5.3	"	juni 2009	"
5.4	"	maart 2012	"
5.5	"	juni 2013	" met Zend Optimiser+
5.6	"	augustus 2014	"
7.0	"	december 2015	Zend Engine 3
7.1	"	december 2016	"
7.2	"	november 2017	"
7.3	"	december 2018	"
7.4	"	november 2019	"
8.0	"	november 2020	" + JIT

Tip: U kunt uw code snel testen voor PHP versies vanaf 4.3+ op <https://3v4l.org/>.

PHP wordt steeds verder door ontwikkeld door zo'n 1000 programmeurs die verenigd zijn in het **PHP Development Team**. Zij lossen bovendien eventuele problemen razendsnel op.

U kunt de laatste versie altijd downloaden via <http://www.php.net>, de officiële PHP website.

Vandaag de dag (maart 2021) is PHP dus een **Open Source** project en zijn we aanbeland bij versie **8.0.3**. De 5.6 versie werd tot 31 december 2018 actief bijgewerkt maar heeft nu ook het End of Life stadium bereikt...

De PHP versie wordt gegeven in de **x.y.z** notatie, bekend als semantic versioning (SemVer). De x staat voor de **major** versie, de y voor de **minor** versie en de z voor de **release** versie.

Er is overigens geen versie 6, men is meteen overgestapt naar PHP 7. PHP 6 was lange tijd de gedroomde opvolger van PHP 5 maar het is vanwege unicode problematiek nooit van de grond gekomen en om associaties met deze mislukking te voorkomen slaat men dit versienummer uiteindelijk over...

PHP 7 is door de nieuwe Zend Engine 3 ruim **2x sneller** dan PHP 5.6. Bovendien gebruikt het **minder geheugen**. Het is vrijwel geheel **backwards compatible** met de laatste PHP 5 versies op enkele zaken na. Zo is een grote verandering dat **de oude mysql bibliotheek** standaard **niet meer beschikbaar** is! We komen daar later nog uitgebreid op terug.

PHP7 kent wel veel **nieuwe mogelijkheden**, maar daar zullen we in deze basis cursus niet aan toe komen... Zie voor een overzicht: <http://php.net/manual/en/migration70.php>.

PHP8 bevat ook weer **vele verbeteringen** waar we tijdens deze cursus niet aan toe zullen komen...

Let op: U moet dus altijd goed opletten met **welke versie** u werkt. Ook binnen een zelfde versie kunnen de verschillen groot zijn! Tijdens de cursus zullen wij steeds met de meest recente versie werken, maar het meeste van de basisfunctionaliteit die wij in deze cursus bespreken is sinds versie 5.4 eigenlijk nauwelijks veranderd...

Weet u niet zeker met welke versie u werkt bij uw provider? Controleer het dan met de `phpinfo()` functie. Zie **3.3**.

Tip: U kunt de PHP versie ook tonen in uw code met de constante `PHP_VERSION` of de functie `phpversion()`.

1.4 Zend

Zeev Suraski en Andi Gutmans hebben in 1999 een eigen bedrijf opgericht:

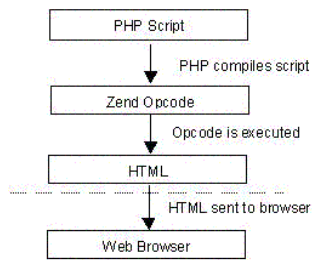


<http://www.zend.com>

Zend is uiteraard een samentrekking van hun voornamen. Zeev en Andi zijn aan de scripting engine blijven sleutelen en hun zogenaamde **Zend Engine** verscheen voor het eerst in PHP 4. De Zend Engine is Open Source en is, inmiddels in versie 3, nog steeds de motor achter PHP.

De Zend engine is een zogenaamde compiler en runtime engine die uw PHP code in het geheugen laadt en omzet naar Zend **opcodes** (operation codes). Deze opcodes worden vervolgens uitgevoerd en het resultaat wordt als HTML naar de client gestuurd. Althans dit is het default gedrag, maar dat is uiteraard aan te passen zoals we later zullen zien!

In een diagram ziet de basale werking er zo uit:



Zend levert naast de Zend engine ook allerlei commerciële tools voor PHP developers. Bovendien bieden zij de enige **PHP certificering** aan. Lange tijd was er ook het **Zend Framework**, dat een van de populairste frameworks was naast Laravel en Symfony (samen ook wel de grote 3 genoemd) voor het ontwikkelen van PHP applicaties. Zend is inmiddels met het framework gestopt maar het is geforkt en gaat nu door het leven onder de naam **Laminas**.

1.5 LAMP Platform

Ook de overige onderdelen van het meest gebruikte PHP platform zijn Open Source:

- Linux als besturingssysteem,
- Apache als Webserver,
- MySQL als database en
- PHP voor Server-side scripting

Dit platform wordt vaak kortweg aangeduid met de afkorting **LAMP**. PHP is overigens een officiële module, `mod_php`, van de Apache webserver (<http://www.apache.org>).

LAMP staat in algemener zin ook voor het Open Source Web Platform: Linux - Apache - MySQL en PHP, *Python* of *Perl*. Voor meer informatie: <http://www.onlamp.com>

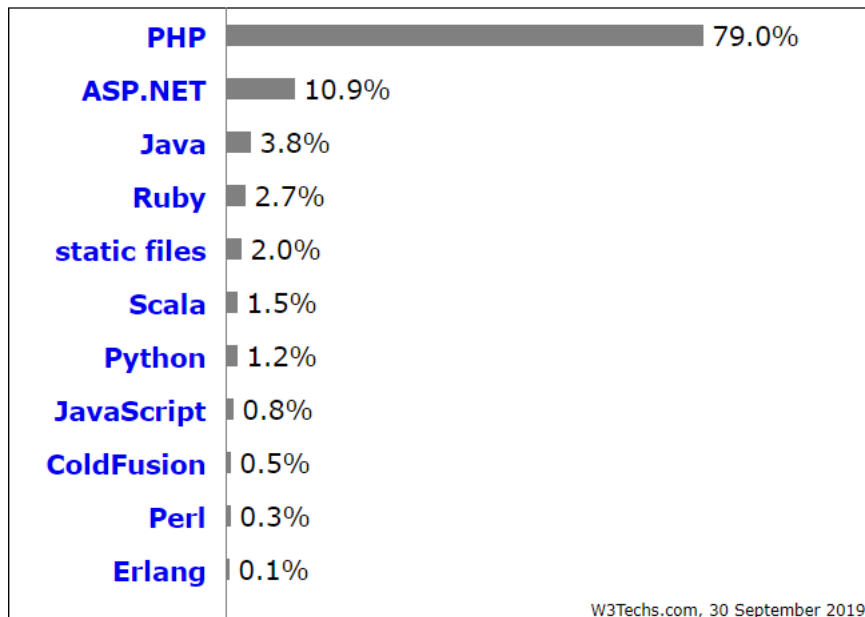
Maar u kunt PHP ook uitstekend op een **Microsoft Windows platform** draaien. Met de zelfde onderdelen als hierboven wordt dan van **WAMP** gesproken. Maar u kunt net zo makkelijk **Internet Information Services** als webserver gebruiken (**WIMP**) en/of **SQL Server** als database (**WISP**).

Vanaf PHP5 heeft u echter ook standaard de beschikking over een **ingebouwde test webserver** en **SQLite** als **standaard** aanwezige (**flat file**) **database**: u heeft dan eigenlijk simpelweg door PHP te downloaden al een **WPS** omgeving...

1.6 Gebruik en populariteit

PHP wordt ondersteund door een groot aantal Internet Service Providers (ISP's). PHP is één van de populairste en makkelijkste manieren om een dynamische website te bouwen.

W3Techs (http://w3techs.com/technologies/overview/programming_language/all) komt met de volgende cijfers:



79% van alle websites waarvan ze de Server-side technologie kennen gebruikt PHP!

Geschat wordt dat er wereldwijd zo'n 4,5 miljoen PHP ontwikkelaars actief zijn.

Sommige van de grootste en drukst bezochte sites ter wereld maken (deels) gebruik van PHP:
FaceBook, Yahoo, Wikipedia.

Heel veel sites maken gebruik van open source software die in PHP geschreven is: CMS systemen als **WordPress, Joomla! & Drupal**, webshops als **OsCommerce & Magento**, forums als **phpBB** en **vBulletin**. Noem ze maar.

Voordat we echter daadwerkelijk met PHP aan de slag gaan zullen we in dit hoofdstuk eerst nog wat dieper ingaan op de theorie en achtergronden van dynamische webpagina's en Client-side scripting.

1.7 Statische vs. dynamische Webpagina's

De HTML pagina's die u tot nu toe gemaakt heeft waren zeer waarschijnlijk **statische** pagina's. U heeft uw webpagina's lokaal ontwikkeld door HTML code in een editor aan te maken en het resulterende document als `.htm` of `.html` bestand op te slaan. U kon ze eenvoudig lokaal testen door ze vanuit uw bestandsysteem in een browser op te vragen.

Toen de pagina's rijp waren voor publicatie heeft u ze geupload naar een webserver. Daar blijven ze vervolgens ongewijzigd staan totdat ze door een (vaak veel te lang uitgestelde) update overschreven worden.

Voor een update moeten de pagina's weer met de hand worden aangepast en de wijzigingen moeten weer worden opgeslagen. Dit is niet alleen tijdrovend maar een bijkomend probleem is dat niet iedereen binnen de organisatie dit kan of mag doen.

Bij statische pagina's heeft u als auteur dus al van tevoren exact vastgelegd hoe de inhoud van de pagina er uit komt te zien, nog voordat een bezoeker ook maar één pagina heeft gezien. Net zoals in een boek blijft de inhoud van dergelijke pagina's feitelijk altijd hetzelfde ongeacht door wie, waar, wanneer en hoe de pagina's bekeken worden. Pas bij een herdruk (het uploaden van een herziene versie van de pagina) zal nieuwe informatie beschikbaar zijn.

Daar is op zichzelf natuurlijk niets mis mee, u kunt zo op een snelle en prettige manier informatie aanbieden. Toch worden statische pagina's steeds meer iets van het verleden. De moderne websurfer verwacht altijd het laatste nieuws, bij voorkeur voor haar op maat gesneden en het liefst met de mogelijkheid om erover mee te kunnen discussiëren!

Wanneer we soortgelijke webapplicaties met interactieve elementen willen gaan bouwen dan redden we het niet met alleen HTML. We zullen de *hardcoded HTML* moeten *vervangen door instructies* die HTML genereren *nadat* de pagina werd opgevraagd. Dergelijke pagina's worden ook wel **dynamische** webpagina's genoemd. En laat PHP nu een van de populairste en meest toegankelijke manieren zijn om dynamische websites te bouwen!

Bij dynamische pagina's speelt ook de onderhoudsproblematiek een stuk minder. De pagina's worden altijd automatisch opgebouwd aan de hand van de laatst beschikbare informatie binnen de organisatie, afkomstig uit (al bestaande) databases. Diegenen die toch al verantwoordelijk waren voor het bijhouden van deze informatie kunnen dat gewoon blijven doen zonder dat zij zich druk hoeven te maken over de wijze waarop de data in de website wordt gebruikt. De dynamische pagina communiceert zelf met de database.

Voordat we dieper ingaan op hoe deze technieken precies werken en hoe PHP daar in past, zullen we eerst herhalen hoe het opvragen van een (statische) webpagina in zijn werk gaat.

1.8 Opvragen van een (statische) webpagina

Alle toepassingen op het internet zijn gebaseerd op een **Client-server mechanisme**. Zo ook het World Wide Web. Het opvragen en tonen van een webpagina dient dan ook van twee kanten belicht te worden: die van **de browser (de client)** en **de webserver (de server)**.

Let op: Onderstaande is een gesimplificeerde samenvatting van een webtransactie. Het mag duidelijk zijn dat de realiteit complexer in elkaar steekt.

Een webbrowser is het programma dat een client gebruikt om webpagina's op te vragen en het ontvangen document te tonen.

Een webserver is een programma dat op verzoek webpagina's aan kan bieden: *it serves pages on request*. Het maakt de webserver daarbij normaal gesproken niet uit om wat voor een soort bestand het gaat of wat de inhoud van dat bestand is. Hun enige taak is het aangevraagde document te leveren.

Stel, u wilt de homepage van CompuFirm bekijken. U typt dan in de adresbalk van uw browser <http://www.compufirm.nl>. In deze zogenaamde **Uniform Resource Locator (URL)** staat geen bestandsnaam. We laten het daarmee aan de webserver over om te bepalen wat het standaarddocument is dat moet worden getoond. In ons geval is de naam van het standaarddocument `index.php`. Dat is een heel gebruikelijke naam, maar het had ook een andere kunnen zijn (bijvoorbeeld `default.asp` of `index.html`). De systeembeheerder die de webserver beheert kan dat namelijk zelf instellen.

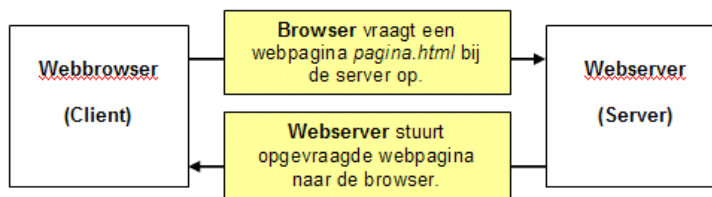
Op het moment dat we op [enter] drukken gaat de browser op zoek naar het (unieke) IP-nummer behorende bij www.compufirm.nl. Dat is 109.72.86.5. Als u dus de homepage van CompuFirm opvraagt dan vraagt u eigenlijk om <http://109.72.86.5/index.php>.

HTTP staat daarbij voor **HyperText Transfer Protocol** en dit is, simpel gesteld, een verzameling afspraken over de wijze waarop bestanden zoals webpagina's aangevraagd en toegezonden moeten worden.

Hoe werkt dat protocol dan? De aanvraag van de browser wordt verzonden als een zogenaamd **HTTP Request**. Hierin staat de naam van het bestand dat de browser toegezonden wil krijgen. We zullen de opbouw van dit request later in de cursus uitgebreider bestuderen. Indien de webserver het request kan honoreren omdat hij

1. het bestand heeft kunnen lokaliseren in zijn bestandssysteem én
2. de client over voldoende rechten beschikt om het bestand ook te mogen inzien

dan zal de server (een kopie van) de inhoud van het bestand in de vorm van een **HTTP Response** naar de browser zenden. De response is een *stream* (een stroom) gegevens. In dit geval bestaat de stream uit de HTML tekst waaruit de webpagina is opgebouwd. In een schema ziet dat er als volgt uit:



Het maakt de server overigens niets uit om wat voor een soort document het gaat. Het kan gaan om een HTML bestand, maar net zo goed om een GIF afbeelding of een Flash filmpje. De enige taak van de server is om het door te spelen aan de browser.

De browser verwerkt de instructies *sequentieel*, oftewel in de volgorde waarin ze in de broncode voorkomen. Eventueel in de pagina opgenomen afbeeldingen worden zo op dezelfde manier door de browser aangevraagd en door de webserver verzonden.

In de browser worden de HTML tekst en de afbeeldingen weer omgezet in een webpagina. De in de browser ingebouwde *HTML interpreter* zorgt ervoor dat de HTML-tags juist geïnterpreteerd worden en er een netjes opgemaakte pagina getoond kan worden.

1.9 Opvragen van een (dynamische) pagina

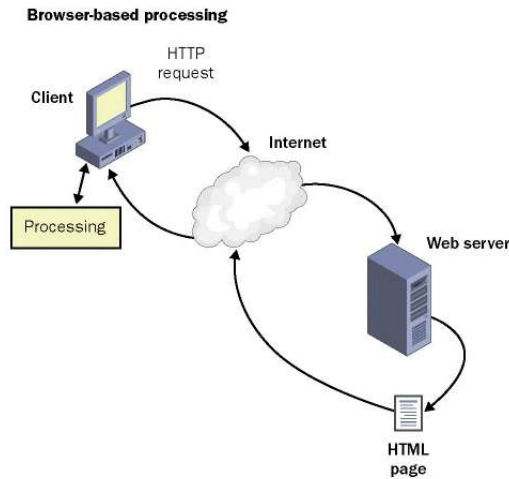
Er zijn in feite **twee verschillende manieren** om dynamische webpagina's te kunnen aanbieden. In het **Client-side model** creëert de browser de dynamische webpagina's en in het **Server-side model** draagt de webserver daar zorg voor.

Bij het opvragen van een dynamische webpagina moet daarom onderscheid gemaakt worden tussen pagina's waarbij gebruik wordt gemaakt van Client-side scripting en pagina's die gebruik maken van Server-side scripting.

1.9.1 Client-side Scripting

Als u wel eens met JavaScript gewerkt hebt dan weet u dat u in een HTML pagina stukken programmacode kunt opnemen om zo de functionaliteit van HTML uit te breiden. Deze zogenaamde **scripts** worden uitgevoerd door de browser tijdens (en eventueel na) het laden van de pagina.

Inleiding tot Server-side Scripting



Script vs. Programma

Een script is een reeks instructies. U kunt het vergelijken met een script voor een toneelstuk. Daarin staat ook vrij letterlijk wat de betrokkenen moeten doen.

Scripts die u schrijft met JavaScript of PHP blijven altijd leesbaar (*human readable*) ze worden geïnterpreteerd door een scripting engine. Deze geeft uiteindelijk aan een applicatie of het OS door wat er moet gebeuren. Wanneer u een dergelijk script wilt wijzigen is het een kwestie van uw code aanpassen en opnieuw opslaan.

Een programma is ook een reeks instructies. Maar bij programma's die u schrijft in programmeertalen als Java of Visual Basic moet uw broncode eerst gecompileerd worden naar machinecode. Hiervoor is een apart stuk software, een zogenaamde *compiler*, nodig. Uw code wordt hierdoor een uitvoerbaar, binair bestand waarna het gelijk door het OS kan worden uitgevoerd. Deze manier van werken is in eerste instantie wat bewerkelijker maar in de uitvoering wel sneller omdat de code niet meer geïnterpreteerd hoeft te worden. Bovendien zijn dergelijke programmeertalen veel krachtiger: ze kunnen meer.

We zullen in deze cursus de termen script en programma door elkaar heen gebruiken. In beide gevallen bent u aan het programmeren!

De webbrowser moet dus niet alleen weten hoe hij HTML, tekst en afbeeldingen juist weergeeft maar ook hoe hij JavaScript moet verwerken. Daarvoor gebruikt een JavaScript *enabled* browser zijn ingebouwde JavaScript Scripting Engine die de JavaScript commando's herkent en kan uitvoeren. Deze situatie waarbij de browser scripts uitvoert noemen we **Client-side Scripting**. Logisch, want de scripts worden Client-side (aan de kant van de client) uitgevoerd.

Sterker nog, aan de andere kant van de transactie, de kant van de server (Server-side) heeft de webserver geen besef van het feit dat de pagina die hij zojuist verzonden heeft behalve HTML ook scripts bevatte. De webserver hoeft er ook niks mee te doen: ze worden immers toch door de browser uitgevoerd!

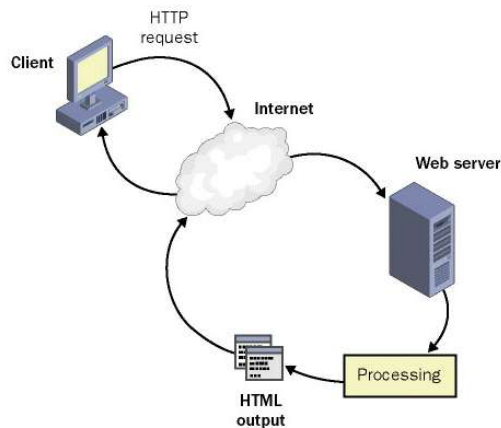
De browser maakt gebruik van modules of plug-ins om de instructies te verwerken. De instructies kunnen tussen de HTML code staan of in een apart bestand, bijvoorbeeld een `.js` bestand. Dit laatste heeft meestal de voorkeur omdat u op deze manier de code centraal kunt onderhouden. Bovendien hoeft een dergelijk bestand slechts één keer door de browser opgehaald te worden omdat het daarna uit *cache* gelezen wordt.

1.9.2 Server-side Scripting

PHP en een aantal andere bekende technieken maken echter gebruik van **Server-side scripting**. De scripts worden in dat geval Server-side (aan de kant van de server) uitgevoerd. In deze situatie heeft de browser juist geen besef van het feit dat er scripts op de server zijn uitgevoerd. De browser krijgt namelijk alleen het resultaat van deze scripts te zien en dat is de door het script gegenereerde HTML!

Elke webbrowser kan HTML weergeven en gebruikers hoeven dus ook geen speciale software te installeren om PHP pagina's te kunnen bekijken.

Server-based processing

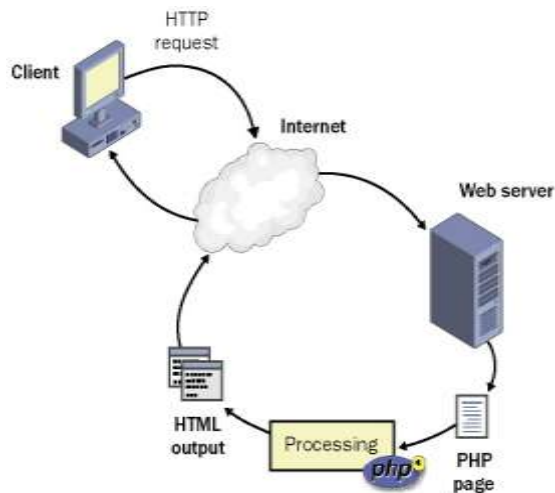


We hebben echter net gezien dat een webserver eigenlijk een betrekkelijk eenvoudig stuk software is. Het zit te wachten op een request van een browser en wanneer er een request binnenkomt dan vervult hij dat request en sluit vervolgens weer de verbinding. Een webserver kan verder niet zo veel. Hij kan geen formulier invoer verwerken of contact leggen met een database.

Een webserver kan echter wel uitgebreid worden met extra software waardoor dit wel mogelijk wordt. Afhankelijk van het soort bestand kijkt de server dan of hij het bestand simpelweg naar de browser moet verzenden of dat hij het bestand door moet sluisen naar de uitbreiding zodat die het bestand kan verwerken. Het resultaat hiervan wordt dan doorgegeven aan de webserver die het weer naar de browser stuurt.

Hieruit volgt dat met PHP (en de andere technieken) niets kan worden weergegeven wat in principe ook niet met gewone HTML kan worden weergegeven! Het grote verschil zit hem in hoe de HTML aangemaakt wordt.

Maar hoe weet de webserver nu dat een pagina scripts bevat die hij moet uitvoeren? In eerste instantie is de **extensie** van de pagina het signaal dat de server in actie moet komen. Een PHP pagina bijvoorbeeld heeft als extensie geen `.html` maar `.php`. Hierdoor weet een *PHP enabled server* dat hij de pagina niet gelijk naar de browser moet zenden. In plaats daarvan stuurt de webserver de pagina eerst langs een geschikte scripting engine die op de server is geïnstalleerd. Deze zal de in de PHP pagina opgenomen scripts verwerken en vervolgens het resultaat naar de client zenden.



Die scripts staan tussen speciale **script delimiters**, zeg maar markeertekens. In PHP zijn dat `<?php` en `?>`. Ze geven het begin en einde van een PHP codeblok aan. De scripting engine weet dan dat hij de instructies die hier tussen staan moet verwerken in plaats van ze meteen naar de browser door te zenden.

Aangezien het resultaat pure HTML is ziet een PHP pagina er voor de browser precies hetzelfde uit als elke andere webpagina. Alle andere code (tekst, HTML, Client-side scripts) in het document (buiten de script delimiters) wordt gewoon onveranderd (*as-is*) doorgegeven.

Voor andere Server-side technologieën werkt dit op een vergelijkbare manier. Bij ASP bijvoorbeeld heeft een webpagina een `.asp` extensie en staat de code die door de scripting engine moet worden uitgevoerd tussen `<%` en `%>`.

1.9.3 Client-side vs. Server-side

De twee technieken verschillen in eerste instantie natuurlijk van elkaar door de plek waar ze uitgevoerd worden. Server-side scripts worden op de server uitgevoerd en Client-side scripts worden door de browser verwerkt. Maar juist hierdoor verschillen ze op nog een aantal andere belangrijke punten van elkaar!

1.9.3.1 Browserafhankelijkheid

Omdat Server-side scripts pure HTML genereren is het mogelijk browser neutrale code te schrijven. U bent niet afhankelijk van de browser zoals bij Client-side scripting. Daar is het maar de vraag of de browser wel JavaScript ondersteunt. Zelfs als dat het geval is, is het nog de vraag of het wel de juiste versie is. En als alles klopt kan het nog mis gaan omdat de gebruiker kan besluiten de JavaScript-ondersteuning van zijn browser uit te schakelen. Met Server-side scripts kennen we dergelijke problemen niet. Zaken als **browser-detectie** of **formulier-validatie** kunt u dan ook veel beter op de Server afhandelen.

Let wel: u kunt met Server-side scripting natuurlijk wel Client-side scripts meesturen waardoor bovenstaande problemen zich alsnog kunnen voordoen. Ook kunt u het voordeel van neutrale HTML ongedaan maken door toch browserspecifieke *markup* te genereren.

Een ander voordeel van Server-side scripting is dat u functionaliteit beschikbaar kunt maken op de server. Zo kunt u bijvoorbeeld een component installeren die van cijfers grafieken kan maken en deze vervolgens als een gewone afbeelding kan versturen. Elke browser kan dit vervolgens gewoon tonen zodat niet elke client apart een Java *applet* hoeft te downloaden om die grafiek te kunnen aanmaken.

1.9.3.2 Zichtbaarheid code

Wanneer u de broncode van een dynamische webpagina in de browser opvraagt ziet u niets anders dan HTML code. Bij Client-side JavaScript (CSJS) ziet u wel de JavaScript code omdat deze naar de browser gezonden werd om daar te worden verwerkt. Een hypotheekberekening zou dan ook helemaal gevolgd kunnen worden en al uw harde werk zou in één keer voor hergebruik gekopieerd kunnen worden.

Bij een PHP pagina kunt u de oorspronkelijke scripts die de pagina creëerden niet zien. Ze werden immers op de server uitgevoerd en alleen het resultaat werd naar de browser gezonden! U kunt PHP dan ook niet leren door de code van PHP sites te bekijken. Alleen wanneer u rechtstreeks toegang zou hebben tot het bestandssysteem op de webserver (bijvoorbeeld via FTP) is het mogelijk om de scripts in te zien.

1.9.3.3 User events

Client-side scripts kunnen rechtstreeks reageren op door gebruikers geïnitieerde *events* in de browser. Denk bijvoorbeeld aan een MouseOver event. Server-side scripts kunnen dat niet! Omdat ze op de server worden uitgevoerd is er altijd een *roundtrip* naar de server nodig. Voor een reactie moet dus op een link of een formulierknop geklikt worden.

1.9.3.4 Mogelijkheden

Met **Client-side scripting** kunt u geweldige dingen doen, zeker in combinatie met het Document Object Model (DOM) en Cascading StyleSheets (CSS). U kunt hele rijke gebruikers interfaces maken, maar de werking van Client-side script **beperkt** zich **tot de browser**. Met andere woorden u kunt geen resources buiten de browser benaderen!

Met **Server-side scripting** kunt u wel **externe resources benaderen**. U kunt gebruik maken van het **filesysteem** op de webserver (bestanden en mappen openen, aanpassen, verplaatsen), e-mail verzenden via een **mailserver**, queries doen naar een **database server**. Kortom, alles wat een webapplicatie moet kunnen doen...

1.10 Wat kunnen we met PHP bereiken?

Tja, wat niet? Met PHP (en in feite alle vormen van Server-side scripting) kunt u ondermeer:

- Dynamisch de inhoud van een webpagina aanpassen
- Gebruikers keuzes laten maken en afhankelijk van hun wensen bepaalde informatie wel of niet tonen
- (browser neutrale) HTML genereren. PHP kan dus in iedere browser bekeken worden!
- Afhankelijk van de browserversie een passende pagina tonen (dit werkt ook als JavaScript uitstaat!)
- Bezoekersaantallen bijhouden met tellers
- (Delen van uw site) afschermen voor onbevoegde gebruikers
- Uw broncode beschermen. PHP is veilig omdat uw code niet in de browser bekeken kan worden
- De status van individuele gebruikers over meerdere pagina's vasthouden (door cookies en sessie variabelen)
- Banners bovenaan uw pagina laten roteren
- Data die per HTML formulier wordt verzonden aannemen, valideren en verwerken
- Automatisch een nieuwsbrief versturen naar geregistreerde bezoekers

En natuurlijk:

- Data uit databases of andere bronnen (zoals tekstbestanden) halen en het resultaat in de browser tonen. Bovendien kunt u deze data online bijwerken.
- Een Content Management Systeem (CMS) bouwen

- Een Web Shop bouwen
- Etc, etc...

Kortom, u kunt nu eindelijk zelf beginnen met het **maken van interactieve websites** zoals u ze ook op het web tegenkomt!

Overigens kunt u heel klein beginnen en zonder al teveel te programmeren al heel veel plezier hebben van PHP. Bijvoorbeeld door een standaard *header* en *footer* voor elke pagina van uw website in te sluiten, dynamisch datums te zetten en met constanten of variabelen te werken voor gegevens die u op meerdere pagina's gebuikt maar ook snel op een centrale plek wilt kunnen aanpassen... Deze manier van werken werd voorheen vaak met Server-Side Includes (SSI) geregeld, maar met PHP gaat een en ander veel makkelijker!

1.11 Andere Server-side technologieën

PHP is niet de enige manier om dynamische webpagina's te genereren. Hieronder volgt een korte opsomming van een aantal veel gebruikte technologieën. We hebben hier echter niet de mogelijkheid om uitgebreid op hun specifieke voor- en nadelen in te gaan.

Bovendien heeft u uw keuze al gemaakt, toch? U volgt deze cursus ten slotte omdat u PHP wilt leren! Toch is het goed om te weten wat de alternatieven zijn. Vaker dan u denkt zult u uw keuze moeten verantwoorden tegenover mensen die een andere techniek gebruiken. Qua mogelijkheden lijken de technieken in ieder geval sterker op elkaar dan dat ze van elkaar verschillen!

Server-side technologie	Maakt gebruik van
CGI (Common Gateway Interface)	Perl, Tcl, C, C++, shell scripts
SSI (Server-Side Includes)	SSI directives
ColdFusion	CFML, server-side ActionScript, Java
Server Side JavaScript (SSJS)	JavaScript
Node.js	JavaScript
ASP	VBScript, Jscript of PerlScript
ASP.NET	VB.NET of C#
JSP	JSP scriptlets & Java Servlets
PHP	PHP
RoR (Ruby On Rails)	Ruby
Django	Python

Vaak is de belangrijkste factor bij de keuze van een technologie het **platform** (besturingssysteem) dat binnen de organisatie wordt gebruikt. Wanneer een bedrijf over een Windows NT of Windows 2000 netwerk beschikt dan is de keuze voor ASP voor het opzetten van een intranet snel gemaakt. Als de provider waarbij u tot nu toe uw webpagina's heeft ondergebracht op Linux draait dan is de keuze voor PHP voor die nieuwe dynamische website snel gemaakt.

Een andere belangrijke factor is **(voor)kennis**. Een Visual Basic programmeur zal snel met ASP uit de voeten kunnen. Voor een Java programmeur is JSP een logisch vertrekpunt en een ervaren Perl programmeur zal in PHP veel herkennen, om maar eens wat voorbeelden te noemen.

Voor beginnende (web)programmeurs is PHP in ieder geval prima toegankelijk en één van de meest krachtige technieken. Aan de slag dus!

Beginnen met PHP

2.1 Wat heeft u nodig?

In dit hoofdstuk bespreken wat u allemaal nodig heeft om PHP pagina's te kunnen schrijven en uit te kunnen voeren:

- Editor
- Webserver
- PHP software
- RDBMS (Database)
- FTP Software
- Browser

Aan het einde van dit hoofdstuk snapt u hoe de onderdelen samenwerken en heeft u uw eerste PHP script geschreven en uitgevoerd.

2.1.1 (HTML) Editor

Als eerste hebben we natuurlijk een programma nodig om onze PHP pagina's mee te schrijven. U kunt daar uitstekend de HTML editor voor gebruiken waarmee u al uw (statische) HTML pagina's schreef, zolang u maar met de hand code in kunt voeren.

PHP bestanden zijn immers gewoon tekstbestanden en een simpele teksteditor als **Kladblok** (Notepad) die u standaard in elke Windows versie vindt, voldoet in principe al, maar is voor het betere programmeerwerk niet echt geschikt. Om meerdere documenten te bewerken moet u bijvoorbeeld meerdere instanties van Kladblok openen, wat niet echt prettig werkt.

Vervelender is echter dat Kladblok geen regelnummering kent en dat is bij het opsporen van fouten (*debugging*) een kleine ramp. Stelt u zich voor dat u straks de volgende foutmelding krijgt:

parse error on line 113

Dat wordt dan een hoop telwerk! Ook een groot gemis is het ontbreken van *syntax color highlighting*, oftewel het in verschillende kleuren kunnen weergeven van valide HTML en PHP instructies.

In de praktijk worden daarom onder andere deze **commerciële** editors veel gebruikt, die al deze features (en nog veel meer) wel bezitten:

- **Adobe Dreamweaver**
<http://www.adobe.com/products/dreamweaver/>
- **phpDesigner**
<http://www.mpsoftware.dk/phpdesigner.php>
- **PhpStorm**
<http://www.jetbrains.com/phpstorm/>
- **Zend PHP Studio**
<http://www.zend.com/en/products/studio/>
- **NuSphere PhpEd**
<http://www.nusphere.com/>

Beginnen met PHP

U kunt meestal een trial versie downloaden en een maandlang proberen of de editor aan uw wensen voldoet.

Er zijn gelukkig ook een aantal fantastische, **gratis** ontwikkelomgevingen beschikbaar:

- **CodeLobster**
<http://www.codelobster.com/>
- **Eclipse for PHP Developers**
<http://www.eclipse.org/>
- **PHPEclipse**
<http://www.phpeclipse.com/>
- **Maguma OpenStudio**
<http://sourceforge.net/projects/openstudio>
- **Aptana**
<http://www.apтана.com/php>
- **Komodo Edit**
<http://www.activestate.com/komodo-edit/>
- **NetBeans**
<http://www.netbeans.org/>

Dit zijn allen complete ontwikkelomgevingen (Integrated Development Environments oftewel IDE's) voor PHP en kennen dus nog veel meer mogelijkheden dan een editor als Dreamweaver om snel PHP applicaties te kunnen bouwen. Ze hebben echter geen van allen een Design View waarin u visueel kunt werken.

Een buitenbeentje is **Web Matrix** van **Microsoft**. Hoewel u het misschien niet zou verwachten is dit een uitstekende PHP editor. De editor komt zelfs met een test webserver waar u uw scripts lokaal op kunt draaien en snel in een van uw browsers kunt bekijken. Er zijn zelfs mobiele emulators. Bovendien kunt u ook makkelijk open source applicaties als Joomla! en dergelijke uit een gallery kiezen en installeren... Tot slot heeft u tal van handige extra's zoals het loggen van requests. U kunt Web Matrix gratis downloaden via <http://www.microsoft.com/web/webmatrix/>.

Voor een uitgebreid overzicht van PHP editors en reviews: <http://www.php-editors.com/>

We raden u echter aan eerst zoveel mogelijk met de hand te coderen om zo een solide basis te verwerven voordat u met deze programma's aan de slag gaat. Tijdens deze cursus is de editor ondergeschikt aan het PHP leren programmeren. Op elke cursus PC staan **Programmer's Notepad**, **Dreamweaver** of **NetBeans** om het nodige typewerk te veraangemen.

Pas wanneer u precies weet wat u aan het doen bent wordt het tijd om deze programma's te testen om te zien of u er productiever mee kunt werken. Met een solide basis bent u in ieder geval in staat om de code zelf handmatig aan te kunnen passen!

2.1.2 Webserver

PHP kan op de **commandline** worden uitgevoerd, maar het is groot geworden omdat het zo makkelijk met HTML is te combineren. Uiteraard moeten we dan wel beschikken over een webserver die onze opdrachten gaat verwerken en de pagina's kan serveren. U heeft verschillende opties:

2.1.2.1 Apache

Op een Linux server zal dit bijna altijd een **Apache** webserver zijn omdat PHP daarmee het best geïntegreerd is. PHP is immers een officiële Apache module!



U kunt de webserver downloaden via <http://httpd.apache.org/>.

2.1.2.2 IIS

PHP kan echter ook prima draaien op een Windows machine waarop Microsoft's **Internet Information Server (IIS)** is geïnstalleerd.



<http://www.iis.net/>

Ziet u het php file icon links onder?

2.1.2.3 Nginx

Het relatief nieuwe **Nginx** (spreek uit Engine X) is een gratis, open-source, high-performance web server.



<http://nginx.org/>

Het project is in 2002 gestart door Igor Sysoev. De eerste public release kwam uit in 2004. Nginx host op dit moment maar liefst 12.18% (22.2M) van alle actieve sites.

Nginx staat bekend om zijn high performance, stability, rich feature set, simple configuration en low resource consumption. In tegenstelling tot traditionele servers maakt Nginx geen gebruik van threads om requests af te handelen. Het maakt gebruik van een veel betere, schaalbare event-driven (asynchrone) architectuur, die weinig geheugen nodig heeft.

Nginx wordt o.a. gebruikt door grote sites als Netflix, Pinterest, Airbnb, WordPress.com en GitHub.

2.1.2.4 Internal Test Server

Vanaf PHP 5.4 kunt u gebruik maken van de meegeleverde webserver. U hoeft dan geen webserver als Apache, IIS of Nginx op uw eigen computer te installeren. Alleen PHP downloaden vanaf <http://php.net> is dan in principe voldoende om te kunnen beginnen met dynamisch PHP Web development. Met het tevens standaard in PHP aanwezige SQLite heeft u immers gelijk ook al een database oplossing beschikbaar. Dat gezegd hebbende, komt die situatie natuurlijk absoluut niet overeen met de omgeving waarin u uw scripts in productie zou willen brengen, maar het is een goede start!

U start de built-in test server op via de commandline. Navigeer eerst naar uw project en geef dan het volgende commando:

```
$ php -S localhost:8080
```

Op de console ziet u dan iets als:

```
$ php -S localhost:8080
PHP 7.1.16 Development Server started at Mon May 14 09:24:56 2018
Listening on http://localhost:8080
Document root is D:\Dropbox\Oefenbestanden\PHP
Press Ctrl-C to quit.
```

Beginnen met PHP

Open nu een browser en navigeer naar <http://localhost:8080>. Als u een `index.php` heeft dan wordt deze gelijk getoond. Anders moet u de naam van het bestand opgeven. Er is geen directory listing zoals bij Apache...

Tip: U kunt mogelijk in de terminal ook met Ctrl+klik de URL openen!

Requests worden vervolgens gelogd:

```
[Tue Mar 10 20:54:01 2015] ::1:54338 [200]: /test.php
[Tue Mar 10 20:54:32 2015] ::1:54346 [200]: /test.php?naam=Arno
[Tue Mar 10 21:00:50 2015] ::1:54419 [200]: /test.php?naam=Piet
```

De server blijft actief tot u afsluit met Ctrl+ C.

U kunt uiteraard ook een andere poort dan 8080 kiezen, mits deze niet in gebruik is natuurlijk. Bovendien kunt u met de flag `-t` ook een specifieke directory opgeven waaruit de bestanden moeten worden geserveerd, bijvoorbeeld:

```
$ php -S localhost:8000 -t public/
```

Let op: De server is zoals de naam al aangeeft **puur bedoeld voor testen!** U kunt hem uiteraard **niet voor productie** gebruiken. De ingebouwde server heeft **geen support voor Apache .htaccess** bestanden, dus als uw applicatie deze gebruikt dan is het niet handig om met de ingebouwde server te testen. Er bestaan overigens wel *workarounds* maar het blijft aanmodderen...

Zie voor meer informatie de manual: <http://php.net/features.commandline.webserver> en <https://lornajane.net/posts/2012/php-5-4-built-in-webserver>.

Sommige editors zoals NetBeans, en Web Matrix komen met een eigen test webserver die ook PHP ondersteunt. In tegenstelling tot Apache of IIS draaien deze echter niet continu op poort 80 maar wordt steeds onder een random poort opgestart als u een pagina wilt testen.

2.1.3 PHP Software

Op de webserver moet de PHP software geïnstalleerd zijn, zodat niet alleen statische HTML pagina's geserveerd kunnen worden maar ook de dynamische PHP scripts.

U kunt de software downloaden via <http://www.php.net/downloads.php>.

2.1.3.1 CGI of module

PHP is in verschillende varianten beschikbaar. PHP is beschikbaar als een **CGI executable** of als een **Apache module** of **ISAPI applicatie** voor IIS. ISAPI staat voor Internet Server Application Programming Interface en is het door Microsoft geïntroduceerde alternatief voor CGI. CGI, ofwel de Common Gateway Interface is het protocol waarmee webserver's kunnen communiceren met andere, achterliggende, applicaties.

Het nadeel van CGI is dat voor elke gebruiker een nieuw proces moet worden opgestart. Bij een drukke server kost dit veel resources.

Een module of ISAPI applicatie wordt daarentegen in dezelfde geheugenruimte als de webserver geladen wanneer de applicatie voor de eerste keer wordt aangeroepen. Hierna blijft de applicatie in het geheugen aanwezig en kan aanvragen sneller afhandelen dan een CGI-applicatie omdat de applicatie niet steeds in het geheugen hoeft te worden geladen.

2.1.4 RDBMS (Database)

Dynamische pagina's worden meestal ingezet om recente data afkomstig uit een database te kunnen tonen in een webpagina. U dient daarvoor uiteraard de beschikking over een database te hebben. Meestal is dat al het geval en moeten gegevens uit een bestaande database online gebracht worden. U moet dan weten om wat voor een soort database het gaat en hoe u die kunt benaderen. Meestal zult u daarvoor met de systeem- of databasebeheerder moeten overleggen.

Om vanuit PHP gegevens uit de database te kunnen lezen en te kunnen muteren gaat u gebruik maken van **SQL (Structured Query Language)**. Veel databases hebben een eigen SQL dialect maar de meest gebruikte en meest krachtige opdrachten zijn eigenlijk bij alle databases gelijk. In **hoofdstuk 5** leest u meer over SQL statements.

Er zijn namelijk nogal wat merken en typen databases op de markt. Bekende zijn bijvoorbeeld MySQL, PostgreSQL, Microsoft Access, Microsoft SQL Server, Oracle, Centura, Informix en IBM DB/2. Deze verschillen nogal in mogelijkheden (en prijskaartje) maar het goede nieuws is dat u ze allemaal op vrijwel dezelfde manier vanuit PHP kunt benaderen.

Database toegang vanuit PHP is over het algemeen heel erg snel omdat PHP voor de meest populaire databases speciale functiebibliotheken kent die *native connections* mogelijk maken omdat de API (Application Programming Interface) rechtstreeks aangesproken wordt. U kunt voor toegang tot oudere of exotische databases ook altijd nog terugvallen op ODBC. In **hoofdstuk 6 en 7** leest u meer over database toegang en het werken met databases.

2.1.4.1 MySQL

In combinatie met PHP wordt voor vanaf *scratch* te ontwikkelen webapplicaties bijzonder vaak voor **MySQL** gekozen, de meest populaire open-source database.



PHP en MySQL worden vaak zelfs in een adem genoemd, alsof ze standaard bij elkaar horen. En dat is misschien ook wel een beetje zo. Ze zijn samen opgegroeid en uiterst populair geworden. MySQL is de tweede meest gebruikte database ter wereld (bron: <http://db-engines.com>). Ze hebben alleen eigenaar Oracle nog voor zich.

Vele open-source applicaties zoals bijvoorbeeld Joomla! en WordPress maken standaard (exclusief) gebruik van MySQL.

Wij zullen tijdens de cursus ook MySQL gebruiken. Het is namelijk bijzonder snel en stabiel. De database server kan op dezelfde machine als de webserver geïnstalleerd zijn of op een aparte (database)server. In de cursussituatie zijn Apache en MySQL op dezelfde (Windows) machine geïnstalleerd.

Leuk om te weten: U spreekt MySQL officieel uit als "My S-Q-L" (/maɪ ˌɛskjuːˈɛl), maar vaak wordt ook "My Sequel" (/maɪ ˈsiːkwəl/) gezegd. MySQL is genoemd naar de dochter van de (Zweedse) ontwikkelaar Michael "Monty" Widenius, die My heet. Het SQL deel staat uiteraard gewoon voor Structured Query Language (dat overigens ook officieel uitgesproken wordt door de letters los te noemen...). Ook twee andere projecten van Monty, MariaDB en MaxDB, zijn naar zijn kinderen genoemd!

Maar realiseert u zich vooral dat u ook tal van andere opties heeft! Dat kunnen **commerciële oplossingen** zijn zoals **Oracle** of **MS SQL Server** die u misschien al gebruikt in uw organisatie.

Er zijn naast MySQL echter ook een aantal andere **open-source** oplossingen:

2.1.4.2 MariaDB

Sinds MySQL door Oracle is overgenomen zijn de originele ontwikkelaars van MySQL een nieuw project begonnen om er voor te zorgen dat het RDBMS altijd gratis beschikbaar zal blijven:



<https://mariadb.org/>

Het is een *drop in replacement* voor MySQL en voor alle praktische doeleinden werkt het precies hetzelfde als MySQL: <https://kb.askmonty.org/en/mariadb-versus-mysql-compatibility/>. Alles wat u hier leert over MySQL is direct toepasbaar op MariaDB. MariaDB maakt ook gebruik van de mysql(i) bibliotheek. Het zou zo maar kunnen dat u denkt te werken met MySQL maar dat het bij nadere bestudering om MariaDB blijkt te gaan!

De laatste versies van XAMPP (zie hierna) gebruiken MariaDB in plaats van MySQL!

De populariteit van MariaDB neemt gestaag toe. Zeker sinds Wikipedia aangaf van MySQL naar MariaDB te gaan migreren.

Een andere minder bekende replacement voor MySQL is **Percona Server** (<http://www.percona.com/software/percona-server>). Het is sneller en heeft meer mogelijkheden. MySQL teams van Facebook, Google, LinkedIn en Twitter werken aan een fork, **WebScaleSQL** genaamd dat geoptimaliseerd wordt voor dergelijke grote websites.

2.1.4.3 SQLite

Vanaf PHP5 wordt standaard SQLite meegeleverd.



<https://sqlite.org/>

SQLite is een *serverless* SQL interface naar een flat file. U hoeft niets te installeren, u kunt gelijk aan de slag met SQL opdrachten tegen een `.sqlite` of `.db` bestand!

SQLite is een perfecte vervanging voor MySQL in kleine tot middelgrote applicaties, bijvoorbeeld wanneer er geen MySQL support aanwezig is bij de provider. U kunt er vrijwel alles mee wat u ook met MySQL kan!

Eigenlijk heeft u met PHP 5 dus al een complete testomgeving met de interne Webserver en SQLite als database (op Windows is dat zeg maar het WPS platform).

Eventuele alternatieven voor SQLite zijn:

- **textSQL**
<http://sourceforge.net/projects/txtsql/>
- **Gladius DB**
<http://sourceforge.net/projects/gladius/>

- **Flat-file SQL**
<http://sourceforge.net/projects/fsql/>

2.1.4.4 PostgreSQL

Een andere populaire open source optie is het robuuste **PostgreSQL** (<http://www.postgresql.org>). Ook wel 's werelds meest geavanceerde open source database genoemd.



2.1.5 MySQL beheer

Om uw MySQL/MariaDB database(s) te kunnen beheren heeft u verschillende opties:

2.1.5.1 CLI

Allereerst kunt u natuurlijk via SSH op de commandline werken, maar echt makkelijk is dat niet.

2.1.5.2 MySQL WorkBench

Als u liever met een vriendelijke visuele interface werkt dan kunt u de **MySQL Workbench** gebruiken die u gratis voor verschillende besturingssystemen kunt downloaden via <http://www.mysql.com/downloads/workbench/>.

Maar dit desktop programma kunt u meestal niet gebruiken om de bij uw provider gehoste databases te beheren omdat die externe verbindingen in het algemeen niet toe staan.

2.1.5.3 HeidiSQL

Een zeer handige en gratis tool is HeidiSQL. U kunt er niet alleen data in MySQL/MariaDB mee bekijken en bewerken, maar ook in PostgreSQL en Microsoft SQL Server!



<https://www.heidisql.com>

2.1.5.4 phpMyAdmin

Vrijwel elke provider biedt voor het beheer van MySQL **phpMyAdmin** aan. Dit is een PHP webapplicatie om online MySQL databases te kunnen beheren.



<http://www.phpmyadmin.net/>.

Wij zullen tijdens de cursus deze standaard tool gebruiken.

Compacte, prettige (en ook gratis) alternatieven zijn de volgende scripts:

- **Adminer**
<http://www.adminer.org/>

- **SQL Buddy**
<http://sqlbuddy.com/>

2.1.6 Complete oplossingen

Het afzonderlijk downloaden en installeren van Apache, PHP, MySQL en phpMyAdmin is heel leerzaam en vergroot uw inzicht in hoe de onderdelen samenwerken. Bovendien kunt u zelf kiezen met welke versies u wilt werken. In de praktijk blijkt echter dat het vaak best lastig is om alles netjes werkend te krijgen.

Als u zonder problemen gelijk aan de slag wil, dan zijn er gelukkig ook complete oplossingen beschikbaar.

2.1.6.1 XAMPP

Een populaire oplossing is **XAMPP**:



<http://www.apachefriends.org/>

Dit bevat alle onderdelen (en meer) in een handige package dat zich probleemloos laat installeren. De X staat voor cross platform (het is beschikbaar voor Windows, Mac en Linux). De A staat voor Apache, de M stond oorspronkelijk voor MySQL maar in latere versies voor MariaDB, de eerste P voor PHP en de tweede officieel voor Perl, maar u kunt het ook zien als de P van phpMyAdmin.

Wij zullen tijdens deze cursus in principe gebruik maken van XAMPP. Wanneer u, na het opstarten van in ieder geval de Apache service in het XAMPP Control Panel, in uw browser <http://localhost/> intypt dan komt u op de (lokale) homepage van de XAMPP installatie uit:



NB.: Merk op dat in de adresbalk <http://localhost/dashboard/> staat. Dat komt doordat in index.php van de webroot een redirect naar deze map staat...

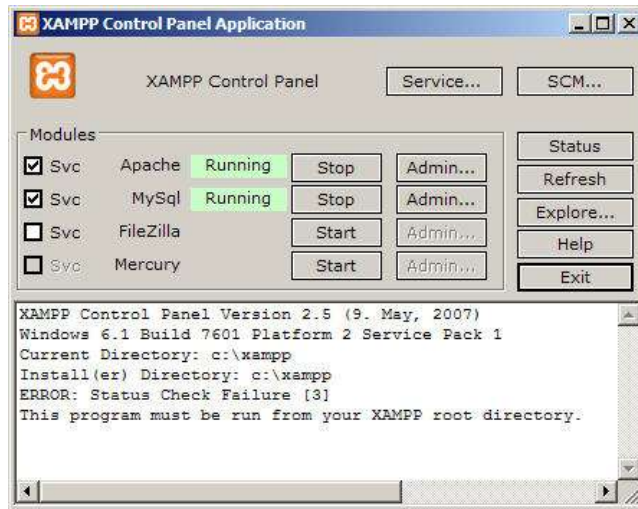
We maken tijdens de cursus een **map** cursus aan in de **webroot** van **XAMPP** (C:\xampp\htdocs\). De documenten in deze folder:

C:\xampp\htdocs\cursus

zijn dan in de browser op te vragen via

<http://localhost/cursus/>

Na een eventuele wijziging in uw `php.ini` moet u Apache herstarten. U doet dat via de Stop knop op het XAMPP Control Panel:



Wanneer Apache gestopt is kunt u hem opnieuw starten. Mocht dit problemen opleveren dan kunt u het ook snel via de **commandline** met deze **batchfiles** doen:

C:\XAMPP\apache_stop.bat
C:\XAMPP\apache_start.bat

Let op: XAMPP is bedoeld als ontwikkeltool en verder **niet beveiligd**. Als u het op uw eigen machine wilt installeren lees dan vooral <http://www.apachefriends.org/en/xampp-windows.html#1221>

Tip: Veel vragen die u mogelijk hebt worden beantwoord op https://www.apachefriends.org/faq_windows.html

2.1.6.2 WampServer

Een andere oplossing om lokaal PHP te kunnen draaien is **WampServer**, vaak (onterecht) kortweg als **WAMP** aangeduid. Deze draait, zoals de naam al aangeeft alleen op Windows:

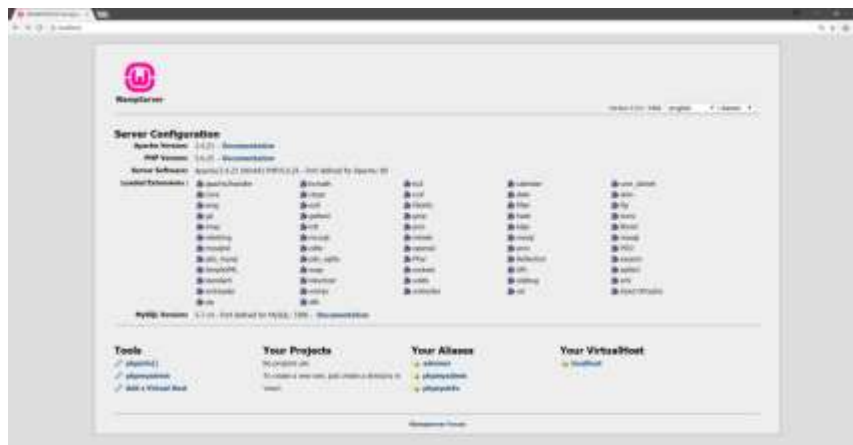


<http://www.wampserver.com/>

De 64-bits versie installeert standaard in `c:\wamp64\`. U kunt dat echter gerust wijzigen in `c:\wamp\`. Voorafgaand aan de installatie krijgt u advies over de voorwaarden die nodig zijn voor een geslaagde installatie. Zorg er in ieder geval voor dat uw VC redistributable versies op orde zijn. WampServer wil standaard gebruik maken van Internet Explorer voor het tonen van de interface en Notepad voor het editen van configuratiebestanden. Als u daarvoor liever een andere browser of editor gebruikt dan kunt u dat tijdens de installatieprocedure wijzigen.

Als u WampServer start dan worden alle services tegelijk opgestart. Wanneer u dan in uw browser <http://localhost/> intypt dan komt u op de (lokale) homepage van de WampServer installatie uit:

Beginnen met PHP



Als u dus tijdens de cursus (of thuis) WampServer wilt gebruiken, dan plaatst u de documenten in deze folder:

C:\wamp\www\cursus

En deze zijn dan in de browser op te vragen via

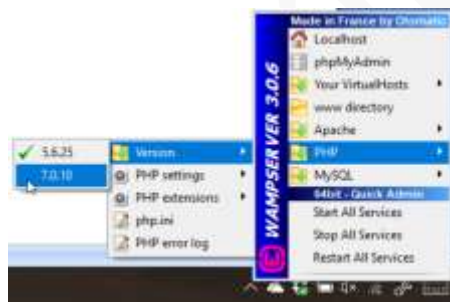
<http://localhost/cursus/>

2.1.6.3 Welke te kiezen?

Eigenlijk maakt het niet zoveel uit. Beiden zijn makkelijker te installeren en te configureren dan alle losse onderdelen afzonderlijk.

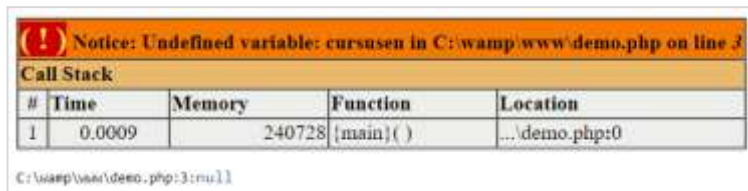
XAMPP gebruikt **meer recente versies van PHP**. Bovendien kan ook **andere componenten** installeren (zoals een mail server, FTP server en Perl) maar die onderdelen heeft u vaak niet nodig.

WampServer maakt het makkelijk om **verschillende versies** van Apache, MySQL en PHP naast elkaar te gebruiken. Standaard wordt PHP5 gebruikt, als u liever met PHP7 werkt dan kunt u dat via de manager in de system tray snel aanpassen:



Op een vergelijkbare manier zijn ook heel veel andere instellingen te maken. Bijvoorbeeld het aan of uitzetten van PHP extensies en andere instellingen waarvoor u anders php.ini zou moeten aanpassen. Het is op zich makkelijk maar verbergt ook de werking van hoe een en ander op de achtergrond werkt...

WampServer heeft ook **Adminer** als alternatief voor phpMyAdmin en **phpsysinfo** voor systeeminformatie. Ander verschil is ook dat XDebug standaard geïnstalleerd is. Foutmeldingen en vardumps zien er daardoor anders uit dan u wellicht gewend bent:



In plaats van:

Notice: Undefined variable: cursussen in C:\xampp\htdocs\demo.php on line 3
NULL

En bij een `var_dump()`:

```
C:\wamp\www\demo.php:3:
array (size=4)
  0 => string 'HTML' (length=4)
  1 => string 'CSS' (length=3)
  2 => string 'JavaScript' (length=10)
  3 => string 'PHP' (length=3)
```

In plaats van:

```
array(4) { [0]=> string(4) "HTML" [1]=> string(3) "CSS" [2]=> string(10) "JavaScript" [3]=> string(3) "PHP" }
```

Of de geformatteerde versie die we tijdens de cursus veel gebruiken:

```
array(4) {
  [0]=>
    string(4) "HTML"
  [1]=>
    string(3) "CSS"
  [2]=>
    string(10) "JavaScript"
  [3]=>
    string(3) "PHP"
}
```

Wat u prettiger vindt is vooral een kwestie van persoonlijke voorkeur...

In WampServer kunt u overigens ook makkelijk **virtual hosts aanmaken** via een wizard.

WampServer neemt door dit alles wel **meer ruimte** in dan XAMPP (2Gb vs 600Mb).

Meer over de verschillen tussen XAMPP en WampServer (en alternatieve oplossingen) leest u op:

- <https://stackoverflow.com/questions/8664901/what-is-the-difference-between-xampp-or-wamp-server-iis>
- <https://stackoverflow.com/questions/3247331/is-xampp-for-windows-preferred-over-wampserver-if-so-why>
- <https://stackoverflow.com/questions/5580342/windows-xampp-vs-wampserver-vs-easyphp-vs-alternative>
- <http://www.tutorialgear.com/xampp-vs-wamp-which-one-is-best-for-you/>

2.1.6.4 Laragon

Sinds kort bestaat er voor Windows een **super snel** en **handig alternatief** voor XAMPP en WAMPServer genaamd Laragon:

Beginnen met PHP



<https://laragon.org/>

Het installeert zich standaard in C:\laragon. De webroot is dan standaard C:\laragon\www, maar dit kunt u makkelijk wijzigen. Een hele handige feature is **auto virtual hosts**. Als u dan bijvoorbeeld een map aanmaakt als C:\laragon\www\cursus dan wordt deze automatisch ook beschikbaar onder <http://cursus.test> in plaats van het gebruikelijke <http://localhost/cursus>. Dergelijke virtual hosts kunt u ook aanmaken in XAMPP en WAMPServer, maar dat gaat daar heel wat omslachtiger. En dit scheelt een hoop gedoe met include paden e.d.

U kunt in Laragon heel makkelijk werken met **verschillende PHP versies**. Een kwestie van downloaden en in de bin map zetten en volia u kunt deze versie gelijk kiezen en gebruiken.

Laragon heeft Composer aan boord en kan via **Quick App** snel diverse PHP projecten zoals WordPress, Drupal, Laravel of Symfony lokaal installeren en is daarna zoals gezegd ook nog eens heel snel in het lokaal werken met deze pakketten.

Laragon installeert niet alleen Apache, PHP en MySQL maar standaard ook al:

- Nginx
- SSL
- PostgreSQL
- MongoDB
- Redis
- Memcached
- HeidiSQL
- Adminer

PhpMyAdmin ontbreekt maar is zeer eenvoudig toe te voegen.

Laragon heeft ook een handige **MailCatcher** functie die verstuurd mailtjes onderschept en kan indien gewenst ook mail verzenden via uw Google account.

En nog veel meer.... Kortom, Laragon is echt super handig!

2.1.6.5 MAMP

Wanneer u op een **Mac** werkt dan kunt u XAMPP gebruiken. Een mooi alternatief is MAMP dat in een gratis en een betaalde Pro versie bestaat:



<https://www.mamp.info>

Bijzonder hier is dat u in plaats van Apache ook voor Nginx als webserver kunt kiezen!

2.1.7 FTP software

Om onze PHP pagina's uiteindelijk te kunnen uploaden naar een webserver hebben we FTP software nodig. Bijvoorbeeld WS_FTP of FileZilla. Tijdens de cursus zullen we dat echter niet doen. We maken en testen de pagina's lokaal.

2.1.8 Browser

Tenslotte moeten we de door ons gemaakte pagina's kunnen testen met een browser. Het maakt in principe niet uit welke browser u daarvoor gebruikt (want PHP is immers browser onafhankelijk!). Er staan op de cursuscomputers recente versies van Microsoft Internet Explorer, Chrome en FireFox.

U doet er verstandig aan om **FireFox** (www.mozilla.com/firefox/) als uw default browser te gebruiken. U zult tijdens het ontwikkelen bijzonder veel plezier beleven aan de handige **Add-ons** die u daarin kunt gebruiken.

Vrijwel onmisbaar zijn de volgende Add-ons:

- **Html Validator**
- **Web Developer**
- **JSView/Quick Java**
- **Tamper Data** (voor inzicht in XSS attacks)

Meer informatie over deze en andere (ontwikkel) add-ons vindt u op <https://addons.mozilla.org/nl/firefox>.

2.2 php.ini

De exacte werking van PHP wordt bepaald door het **configuratiebestand** `php.ini`. Dit configuratie bestand wordt ingelezen wanneer PHP opstart. Als PHP draait als server module dan gebeurt dit eenmalig, bij het opstarten van de webserver. Als PHP als CGI applicatie draait dan wordt het bestand bij elke aanvraag ingelezen. Dit houdt dus in dat als **PHP als** een server **module** draait en u **wijzigingen** aanbrengt in `php.ini`, u de **webserver** moet **herstarten** om de wijzigingen door te voeren!

Wanneer u een bestand met daarin enkel de functie `phpinfo()` op de server zet en opvraagt kunt u zien wat de locatie van het `php.ini` bestand is, in XAMPP is dat bijvoorbeeld:

PHP Version 5.3.8 	
System	Windows NT WIN-R1VLF3J952S 6.1 build 7600 (Windows 7 Enterprise Edition) i586
Build Date	Aug 23 2011 11:47:20
Compiler	MSVC9 (Visual C++ 2008)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--disable-isapi" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=D:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=D:\php-sdk\oracle\instantclient11\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet" "--with-mcrypt=static" "--disable-static-analyze"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\php\php.ini

Bovendien kunt u iets verderop onder het kopje Core uitlezen wat de instellingen van `php.ini` zijn:

Core

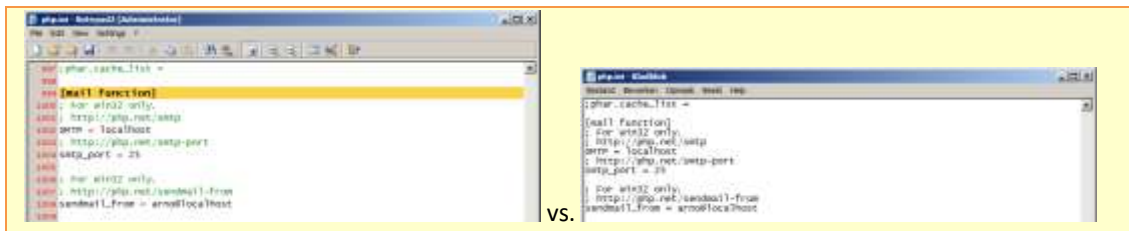
PHP Version		5.3.8
Directive	Local Value	Master Value
<code>allow_call_time_pass_reference</code>	Off	Off
<code>allow_url_fopen</code>	On	On
<code>allow_url_include</code>	Off	Off
<code>always_populate_raw_post_data</code>	Off	Off
<code>arg_separator.input</code>	&	&
<code>arg_separator.output</code>	&	&
<code>asp_tags</code>	Off	Off
<code>auto_append_file</code>	<i>no value</i>	<i>no value</i>
<code>auto_globals_jit</code>	On	On

Zo kunt u snel zien hoe de omgeving waarop u moet gaan werken in elkaar steekt. Het uploaden van het `phpinfo` bestandje en het analyseren van de informatie erin is meestal een van de eerste dingen die een PHP programmeur zal doen als hij met een nieuwe omgeving te maken krijgt.

2.2.1 Aanpassingen maken

Als u een eigen webserver draait kunt u `php.ini` natuurlijk gewoon direct aanpassen. De `php.ini` van uw **provider** kunt u natuurlijk **niet aanpassen**!

Tip: het is aan te bevelen het `php.ini` bestand **niet met Kladblok** te **openen** (standaardinstelling onder Windows) maar **met** een editor als **NotePad2**, **NotePad++** of **Programmer's Notepad**. Deze hebben **color syntax highlighting** voor `.ini` files en dat maakt ze een stuk **makkelijker** te **bewerken** omdat u doordat commentaar groen wordt weergegeven en instellingen zwart meteen ziet welke instellingen wel en welke niet actief zijn...



Toch zijn veel van de instellingen zijn op script niveau in te stellen met de functie `ini_set()`, of op directory niveau met een `.htaccess` file, maar dit geldt niet voor instellingen die de werking van PHP zelf beïnvloeden...

Zie voor een lijst met alle settings en hoe ze zijn in te stellen de volgende URL:
<http://php.net/manual/en/ini.list.php>:

Name	Default	Changeable
<code>allow_call_time_pass_reference</code>	"1"	PHP_INI_PERDIR
<code>allow_url_fopen</code>	"1"	PHP_INI_SYSTEM
<code>allow_url_include</code>	"0"	PHP_INI_ALL

De waarde `PHP_INI_SYSTEM` betekent dat het alleen op server niveau in `php.ini` zelf is aan te passen. Als PHP als Apache module is geïnstalleerd moet u Apache **herstarten** om de wijzigingen door te voeren.

2.2.1.1 Op directory niveau met .htaccess

De waarde `PHP_INI_PERDIR` betekent dat een instelling ook op directory niveau is aan te passen. De meest gangbare manier is om dit met een `.htaccess` bestandje te doen. De instelling overschrijft dan de waarden uit `php.ini` voor de folder waarin het is geplaatst. De wijziging is na plaatsing meteen van kracht. Apache moet wel geconfigureerd zijn om dit toe te staan, maar bij vrijwel elke provider is dat het geval.

De algemene notatie is:

php_value *ini_setting* *value*

NB.: In plaats van het keyword `php_value` kunt u ook `php_flag` gebruiken, maar deze is alleen voor booleaanse waarden, dus om iets aan of uit te zetten. Bijvoorbeeld:

```
php_value short_open_tag On
# is het zelfde als
php_flag short_open_tag On
```

Voor booleaanse waarden (aan/uit) kunt u ook 1 (aan) of 0 (uit) gebruiken. De waarden mogen tussen dubbele quotes staan, maar het is niet nodig. Maar u kunt met `php_flag` **geen andere waarden** opgeven! Voor instellingen die een andere waarden dan aan/uit vragen moet u dus geen `php_flag` gebruiken!

```
#let op: dit werkt niet!
php_flag error_reporting 30719

#let op: dit werkt wel:
php_value error_reporting 30719
```

Let op: Providers die met **cPanel** werken voor de instellingen maken gebruik van **suPHP**. Dan levert elk `.htaccess` bestand een 500 Internal Server Error op. Bij deze providers moet u juist een **custom php.ini** bestandje in een map zetten! Dat geldt ook voor providers die PHP als **CGI** aanbieden.

2.2.1.2 Op scriptniveau met `ini_set()`

De waarde `PHP_INI_ALL` betekent dat het naast de eerder genoemde manieren ook op scriptniveau is in te stellen.

U kunt hiervoor de algemene functie `ini_set()` gebruiken:

```
<?php
// zet error_reporting level maximaal
ini_set('error_reporting', E_ALL);
?>
```

Merk op dat `E_ALL` **niet tussen quotes** moet worden gezet, het is geen string maar een **PHP constante** (meer hierover in **Hoofdstuk 3**). Voor **veel** van deze **settings** is ook een **eigen functie** beschikbaar, zo ook voor het instellen van de `error_reporting`:

```
<?php
// zet error_reporting level maximaal
error_reporting(E_ALL);
?>
```

Bovenstaande notatie doet precies hetzelfde. `ini_set()` is dus de algemene variant. Er is ook een `ini_get()` om de huidige waarde voor een instelling uit te lezen:

```
<?php
// toon huidige error_reporting level
echo ini_get('error_reporting'); // geeft 32767 in PHP 5.4
                                //      30719 in PHP 5.3
                                //      6143 in PHP 5.2
                                //      2047 in PHP 5.1 en eerder
?>
```

De waarden voor `error_reporting` zijn zogenaamde **bitmasks**, vandaar dat u getallen krijgt... De verschillen komen doordat in nieuwere versies meer niveaus zijn toegevoegd.

2.2.1.3 `display_errors` en `error_reporting`

Een bekend scenario is dat bij een provider gehoste pagina's geen fouten laten zien. U krijgt enkel een blanco pagina. Wat nu?

Om alle meldingen zichtbaar te maken bij een provider die geen fouten laat zien kunt u de volgende 2 manieren gebruiken:

1. Zet de instellingen met `ini_set()` bovenaan (elk) script:

```
<?php
// laat fouten zien
ini_set('display_errors', 'On'); // of 1

// zet error_reporting level maximaal
error_reporting(E_ALL);

// uw code:
...
```

2. Zet de instellingen in een `.htaccess` file:

```
php_flag display_errors On
php_value error_reporting 30719
```

Let op: U moet voor de waarde van `error_reporting` in een `.htaccess` een getal gebruiken. U kunt nu niet de PHP constanten gebruiken!

Tip: Een handige tool om uw `error_reporting` met getallen in te stellen is de **Ultimate PHP error_reporting wizard** die u vindt op <http://www.bx.com.au/tools/ultimate-php-error-reporting-wizard>:

The screenshot shows the 'Ultimate PHP error_reporting wizard' interface. It includes a table of error levels with checkboxes, and sections for Binary, Decimal, Constant, PHP, and .htaccess code snippets.

Value	Constant	
1	E_ERROR	☒
2	E_WARNING	☒
4	E_PARSE	☒
8	E_NOTICE	☒
16	E_CORE_ERROR	☒
32	E_CORE_WARNING	☒
64	E_COMPILE_ERROR	☒
128	E_COMPILE_WARNING	☒
256	E_USER_ERROR	☒
512	E_USER_WARNING	☒
1024	E_USER_NOTICE	☒
2048	E_STRICT	☒
4096	E_RECOVERABLE_ERROR	☒
8192	E_DEPRECATED	☒
16384	E_USER_DEPRECATED	☒
E_ALL		☒
None		☐

Binary: 0111111111111111

Decimal: 32767

Constant: E_ALL

PHP: <php error_reporting(E_ALL); >

.htaccess: php_value error_reporting 32767

This tool is a reference for PHP developers needing to set PHP's error reporting level. To use this tool, tick the checkboxes next to the errors you want PHP to report on and the tool will generate the various different representations. To set the error reporting level via PHP, copy and paste the "PHP" section's contents. To set the error reporting level via Apache, copy and paste the ".htaccess" section's contents into a ".htaccess" file. You can also use the ".htaccess" section in Apache's VHost configuration file (httpd.conf). Visit the author's blog.

Maar de fouten worden natuurlijk met een reden niet getoond! Fouten geven namelijk teveel informatie prijs aan kwaadwillende gebruikers terwijl goedwillende gebruikers er niets mee kunnen. Daarom is het op een productie site beter om de fouten niet te tonen maar ze te loggen. Ook dat kan op page level niveau, maar `.htaccess` is bij uitstek geschikt voor deze taak:

```
php_value log_errors on
php_value error_log /home/path/public_html/domain/PHP_errors.log
```

U moet er uiteraard wel voor zorgen dat de logfile bestaat en dat deze schrijfrechten heeft!

Tip: Een leuke tour door `php.ini` vindt u op <http://www.sitepoint.com/a-tour-of-php-ini/>.

2.3 Verwerking en opbouw PHP pagina's

In **hoofdstuk 1** heeft u een algemene indruk gekregen hoe een dynamische webpagina gegenereerd wordt. In deze paragraaf richten we ons op de specifieke details voor PHP.

2.3.1 .php extensie

Wanneer u op een PHP enabled webserver gebruik wilt gaan maken van PHP dan zult u uw pagina's allereerst een zinvolle extensie moeten geven zodat de webserver weet dat de Server-side scripts in de pagina's eerst verwerkt moeten worden. Vanaf PHP4 is `.php` de standaardextensie geworden (in vorige versies werden ook `.phtml` en `.php3` gebruikt).

Overigens is het ook mogelijk om de extensie zelf te bepalen als u de webserver zelf in beheer heeft, bijvoorbeeld omdat u niet wilt laten zien dat u PHP gebruikt. Wij zullen tijdens deze cursus echter de standaard `.php` extensie gebruiken.

Het geven van een `.php` extensie is in principe alleen zinvol voor pagina's die daadwerkelijk Server-side scripts bevatten. Pagina's die alleen uit HTML (en Client-side scripts) bestaan kunt u gewoon een `.html` extensie blijven geven. Er is overigens geen merkbaar snelheidsverlies wanneer u een dergelijke statische pagina toch een `.php` extensie zou geven. PHP ziet het meteen wanneer er geen PHP codeblokken in de pagina staan en geeft de pagina zonder verdere verwerking gelijk door. Om het onderscheid tussen statische en dynamische pagina's extra te benadrukken zullen wij tijdens de cursus statische pagina's altijd een `.html` extensie geven.

2.3.2 Script delimiters

In tegenstelling tot traditionele CGI scripts waar u alle benodigde HTML met print statements moet genereren, kunt u PHP code in een bestaand HTML document opnemen (*embedden*). Maar hoe weet PHP nu welke delen van een PHP pagina Server-side scripts bevatten die door de scripting engine geïnterpreteerd moeten worden?

De scripts binnen de pagina moeten daarvoor altijd worden omsloten met de zogenaamde *script delimiters*:

```
<?php
// code ...
?>
```

De **script delimiters markeren het begin en het einde** van een **PHP codeblok** en geven daarmee aan dat de code die ertussen staat als PHP moet worden geïnterpreteerd. De instructies die hier tussen staan worden verwerkt door de *scripting engine*. Het resultaat van deze verwerking is meestal HTML dat vervolgens naar de client wordt gezonden. Alle andere code buiten de script delimiters (tekst, HTML, Client-side scripts) wordt onveranderd (*as-is*) doorgezonden.

Let op: U kunt met PHP overigens ook andere uitvoer dan HTML genereren, bijvoorbeeld XML, CSS of JavaScript. Zie **3.18 Andere uitvoer dan HTML** voor meer informatie.

Let op: In HTML bestaat een element uit een *opening tag* en een bijbehorende *closing tag*. Deze is gelijk aan de opening tag voorafgegaan door een *forward slash*. Dit geldt ook voor opname van Client-side scripts in HTML. Zo kunt u JavaScript opnemen door de code tussen `<script>` en `</script>` te zetten. In PHP zet u de code echter als het ware *binnen één tag*. Na het kleiner-dan-teken komt een vraagteken en nogmaals voor het groter-dan-teken:

```
<?php> // FOUT! </?php>
<php> // FOUT! </php>
<?php> // FOUT! </?>
<?php // GOED! ?>
```

2.3.2.1 closing tag weglaten

U mag bij PHP only scripts die geen uitvoer hebben de closing tag weglaten. Sterker nog, het is in de meeste frameworks zelfs conventie dit te doen. Zie ook <http://www.php-fig.org/psr/psr-2/>: "The closing `?>` tag *MUST* be omitted from files containing only PHP".

Dit voorkomt het per ongeluk insluiten van trailing whitespace en eventuele problemen die dergelijke witruimte kan veroorzaken in de response.

Sommige programmeurs vinden het wel prettig om het bestand met commentaar af te sluiten om zo duidelijk te maken dat het bestand compleet is (en niet per ongeluk afgekap):

```
<?php
...
// end of file
```


2.3.2.2 Wisselen tussen PHP en HTML

U mag zo vaak als u wilt wisselen tussen HTML-modus en PHP-modus. Dit levert geen performance verlies op, zoals bijvoorbeeld bij Classic ASP scripts wel het geval is.

Realiseert u zich echter wel dat er niet zomaar HTML instructies in een PHP-codeblok kunnen staan! Dit geeft meteen een foutmelding:

```
<?php <strong>...</strong> ?>
```

namelijk:

Parse error: syntax error, unexpected '<' in demo.php

PHP instructies in een HTML blok daarentegen hebben geen effect en zullen zichtbaar worden weergegeven!

2.3.2.3 Script delimiter notatie

Er bestaan nog 3 andere manieren om script delimiters te noteren naast de variant die we u al eerder hebben laten zien. Deze notatie die ook wel de **Standard** of **XML Style** wordt genoemd is echter **de enige juiste**. We zullen deze dan ook steeds gebruiken in de cursus.

Voor de volledigheid noemen we hier alle mogelijke notaties, zodat u ze kunt herkennen (en eventueel vervangen) wanneer u ze in code van anderen tegenkomt.

notatie	naam
<?php ... ?>	Standard Style / XML Style
<? ... ?>	Short Style / SGML Style
<% ... %>	ASP Style
<script language="php"> ... </script>	Script Style

De **ASP Style** en **Script Style** zijn ooit geïntroduceerd om te kunnen gebruiken in combinatie met WYSIWYG editors die geen ondersteuning boden voor de PHP Standard Style. Ze zijn echter **vervallen in PHP7**.

De PHP Short Style is uiteraard bedoeld om iets minder te hoeven typen, met name als u ze gebruikt in combinatie met de **PHP Shorthand** notatie `<?= ... ?>` die hetzelfde doet als `<?php echo ...; ?>`.

Het probleem met de Short Style (en de vervallen ASP Style) is echter dat deze afhankelijk zijn van deze instellingen in `php.ini`. Ze moeten daar de waarde `On` hebben en u ziet al in het bijbehorende commentaar dat dit niet wordt aangeraden:

```
; Allow the <? tag. Otherwise, only <?php and <script> tags are
; recognized.
; NOTE: Using short tags should be avoided when developing
; applications or libraries that are meant for redistribution,
; or deployment on PHP servers which are not under your control,
; because short tags may not be supported on the target server.
; For portable, redistributable code, be sure not to use short
; tags.
short_open_tag = Off

; Allow ASP-style <% %> tags.
asp_tags = Off
```

Beginnen met PHP

Het `php.ini` bestand is niet door u aan te passen wanneer u uw PHP scripts bij een provider host. Als de provider de instellingen ooit wijzigt (bijvoorbeeld bij een PHP upgrade) of wanneer u uw scripts bij een andere provider met andere instellingen onderbrengt, dan zit u behoorlijk in de problemen. Want wanneer deze instellingen de waarde `Off` hebben dan wordt in één keer al uw **PHP code** voor de eindgebruiker **zichtbaar**! Dergelijke code is met andere woorden niet erg portabel!

De Short Style zult u waarschijnlijk wel veelvuldig tegenkomen, omdat deze in het verleden heel gangbaar was. Een bekend probleem met deze notatie is dat u hem niet kunt gebruiken in scripts waarin u met XML Declarations en XML Processing Instructions werkt. Deze staan immers ook tussen `<?` en `>?` delimiters. Zie ook **Andere uitvoer dan HTML**.

Wat is dan ook de enige conclusie die u na dit verhaal kunt trekken? Juist, **gebruik alleen de Standard Style**!

Update: de verkorte `echo` notatie is de laatste tijd (weer) erg populair geworden vanwege het gebruik in PHP **Templates** (zie later in de cursus). Vanaf **PHP 5.4** zijn de **short echo statements** losgekoppeld van de short open style en staan ze **default aan**! U kunt dan dus simpelweg

```
<?= $var ?>
```

schrijven in plaats van

```
<?php echo $var; ?>
```

Wij zullen dat later in de cursus in de templates ook doen. Merk op dat witruimte en puntkomma niet nodig zijn:

```
<?=$var?>
```

doet hetzelfde als

```
<?= $var; ?>
```

of

```
<?= $var ?>.
```

De laatste notatie, **met spaties** maar **zonder puntkomma** is de **meest gebruikte** notatie en die zullen wij dan ook tijdens de cursus gebruiken!

2.3.3 De PHP Scripting Language

De Server-side scripts in PHP pagina's zijn geschreven in, jawel, PHP, een eigen scripting language. Deze heeft een syntaxis die sterk lijkt op C/Perl/Java/JavaScript. Als u al eens met JavaScript gewerkt heeft dan zal een en ander u zeker bekend voorkomen!

De opdrachten die u in PHP geeft, worden uiteindelijk door de Zend scripting engine geïnterpreteerd en uitgevoerd. Om PHP pagina's te kunnen maken dient u zich dus de PHP taal aan te leren. We zullen daar in het volgende hoofdstuk dan ook uitgebreid op ingaan. Maar eerst gaat u uw eerste PHP pagina schrijven, zodat de tot nu toe besproken theorie ook echt gaat leven!

2.4 Uw eerste PHP pagina

Eindelijk gaat u dan uw eerste PHP pagina maken! In deze eerste oefening gaan we het tijdstip afdrukken waarop de bezoeker onze webpagina oproept. Dit is weliswaar een hele simpele dynamische pagina maar wel één die precies duidelijk maakt hoe PHP werkt!

Een dergelijke pagina zou met alleen HTML onmogelijk te maken zijn.

Oefening 2.1	Start uw favoriete editor op en neem de volgende code over: <pre><!DOCTYPE html> <html> <head> <title>Mijn eerste PHP Pagina</title> </head> <body> <h1>Welkom</h1> Deze PHP pagina werd gegenereerd op: <?php echo date('d/M/Y - H:i:s'); ?> </body> </html></pre> Sla het bestand op als <code>tijdstip.php</code> en controleer het PHP bestand door met de browser naar het volgende adres te surfen: http://localhost/tijdstip.php Let op: Denk aan de <code>.php</code> extensie, anders zal de pagina niet werken. Bekijk in de browser de broncode van uw pagina en zie dat de PHP code <code><?php echo date('d/M/Y - H:i:s'); ?></code> vervangen is door de huidige datum/tijd. Het is net alsof u het er zelf in had getypt... Klik hierna eens op de reload knop (of gebruik de functietoets [F5]) en zie dat de tijd weer actueel wordt, omdat het PHP script weer opnieuw wordt uitgevoerd door de script-engine op de webserver. De opdracht <code>echo</code> schrijft de huidige datum die door de opdracht <code>date</code> wordt geleverd naar de pagina. Het masker <code>('d/M/Y - H:i:s')</code> zorgt daarbij voor de specifieke notatie. Kijk voor meer mogelijke notaties eens op http://www.php.net/date.

Als bovenstaande oefening gelukt is en u nog tijd over heeft dan kunt u de onderstaande verdiepingsoefening(en) maken.

Verdieping 2.1a	Voeg aan bovenstaande pagina een Client-side script toe dat de huidige tijd op de server laat zien in een waarschuwingsvenster wanneer de pagina wordt geladen. Hint: gebruik de HTML onload eventhandler en een JavaScript alert functie.
Verdieping 2.1b	Als u voldoende JavaScript kennis heeft, probeer dan ook eens de tijd die PHP in de pagina heeft gezet als variabele te gebruiken in een JavaScript dat na een druk op een knop de PHP tijd laat zien én het tijdstip waarop op de knop werd geklikt.

Verdieping 2.1c	<i>U kunt de vorige oefening nog verdiepen door het verschil tussen de 2 tijdstippen te berekenen. U kunt in PHP een timestamp maken met de <code>time()</code> functie. Maar pas op: deze wordt gegeven in seconden terwijl de JavaScript <code>getTime()</code> method met milliseconden werkt!</i>

Let op: U kunt een lokaal opgeslagen `.php` pagina **niet rechtstreeks vanuit** het **bestandssysteem** van uw harde schijf opvragen (door er dubbel op te klikken bijvoorbeeld).

U moet een PHP pagina **altijd via de webserver** opvragen zodat de scripts verwerkt kunnen worden. Wat er bij dubbelklikken gebeurt, hangt af van de configuratie van uw systeem, oftewel aan welke applicatie is de `.php` extensie gekoppeld? Dat kan een editor zijn: de pagina start dan bijvoorbeeld op in Dreamweaver of NetBeans.

Nu u de basisprincipes kent, wordt het hoog tijd om u in de *syntaxis* van PHP te verdiepen.

Scripten met PHP

3.1 Inleiding

U weet inmiddels hoe PHP werkt en u hebt inmiddels ook begrepen dat de scripts in PHP pagina's in de eigen PHP scriptingtaal geschreven worden. De PHP syntax is voor een groot deel afkomstig uit de programmeertaal **C**, maar ook de programmeertaal **Perl** heeft een grote invloed gehad op PHP. Met het steeds meer OOP worden van de taal is ook de nodige **Java** syntax in de PHP terug te vinden. Bekendheid met deze talen is natuurlijk een groot voordeel, maar ook als uw programmeerervaring zich beperkt tot **JavaScript** dan zullen veel basisnotaties u reeds bekend voorkomen! Wij gaan er tijdens deze cursus echter vanuit dat u helemaal nog geen programmeerervaring hebt en zullen dan straks ook bij de basis beginnen. U kunt dit hoofdstuk dan ook zien als een korte inleiding in programmeren!

PHP kent zeer veel gespecialiseerde, op het schrijven van webapplicaties toegespitste functies. In de PHP manual worden er **meer dan 10.000** beschreven!

Op www.php.net/manual/en/indexes.functions.php vindt u alle functies alfabetisch op rij. Uiteraard hoeft u deze niet allemaal uit uw hoofd te leren. Vele functies zijn zo specialistisch dat u ze misschien zelfs helemaal nooit zult gebruiken...

Overigens heeft u ook nooit zomaar al deze functies ook gelijk tot uw beschikking. Over welke functies u kunt beschikken, hangt helemaal af van welke extensies en modules er geactiveerd zijn in uw PHP omgeving. Er bestaat, heel toepasselijk, een ingebouwde functie om te laten zien over welke functies u kunt beschikken: `get_defined_functions()`. Deze zal echter laten zien dat u bij een gemiddelde installatie toch al gauw over 1450+ functies beschikt.

Voor de liefhebber:

```
$functions = get_defined_functions()['internal'];
sort($functions);
print_r($functions);
```

Merk op dat `echo` en `include` er niet bij staan. Dit zijn geen functies maar *language constructs*...

In dit hoofdstuk zullen we de belangrijkste zaken van PHP doornemen. We gaan er daarbij zoals gezegd vanuit dat u géén programmeer ervaring heeft. Voor cursisten met ervaring in andere programmeertalen zal het onderstaande voor een deel gesneden koek zijn. PHP heeft echter een paar kenmerken (en eigenaardigheden) die ook voor hen interessant zijn. Bovendien hopen we door middel van verdiepingsoefeningen en extra informatie we dit essentiële hoofdstuk ook voor hen aantrekkelijk gemaakt te hebben.

3.2 Online Manual

Uitleg is altijd dichtbij! Door de naam van een functie achter de URL van de officiële PHP site (<http://www.php.net>) te plakken (bijvoorbeeld <http://www.php.net/date>) komt u direct op de juiste pagina in de **online manual** uit!

Als u de Nederlandse mirror pakt op <http://nl.php.net> dan is deze informatie ook nog eens grotendeels Nederlandstalig! Deze site is meestal ook sneller. Wilt u ook **offline** de documentatie door kunnen nemen, dan kunt u de manual ook **downloaden**. Kies dan een voor u prettig formaat op

<http://www.php.net/download-docs.php>. Voor het gebruik op Windows machines is het (makkelijk doorzoekbare) .chm formaat een aanrader!

De online manual bevat echter waardevolle **annotaties**: tips en suggesties van PHP programmeurs, vaak met handige codefragmenten die u kunt kopiëren en hergebruiken in uw eigen code. De PHP manual is er beroemd om!

Wanneer u de beschikbare functies in de manual bekijkt dan ziet u daarbij ook vanaf welke versie van PHP deze functie beschikbaar is:



U kunt zo'n functie dan ook alleen gebruiken als uw versie van PHP aan de minimale eisen voldoet. Soms is het ook zo dat er in latere versies extra mogelijkheden aan toegevoegd zijn of juist functionaliteit is vervallen of aangepast.

Let op: Uw PHP versie en de instellingen in `php.ini` bepalen dus in grote mate wat u met PHP kunt doen. Om die reden is het altijd belangrijk precies te weten met welke PHP versie u werkt en wat de instellingen in `php.ini` zijn!

3.3 phpinfo()

Een handige manier om uw versie en instellingen te achterhalen is met de opdracht `phpinfo()`. Dit is een speciale functie die niet alleen output geeft over de versie van PHP en tal van instellingen, maar ook laat zien welke functiebibliotheken beschikbaar zijn en wat de waarden van de verschillende omgevingsvariabelen zijn.

Maak maar eens een bestand `phpinfo.php` (of een andere willekeurige naam) aan waarin u enkel het volgende zet:

```
<?php phpinfo(); ?>
```

Plaats het op uw webserver en vraag het bestand op. Dit kleine stukje code laat u alles zien wat u wilt weten over uw PHP installatie:



System	Windows NT ARNO-PC 6.1 build 7601 (Windows 7 Ultimate Edition Service Pack 1) i586
Build Date	Sep 18 2013 12:59:11
Compiler	MSVC11 (Visual C++ 2012)
Architecture	x86
Configure Command	cscrip/nologo configure.js "--enable-snapshot-build" "--disable-isapi" "--enable-debug-pack" "--without-mssql" "--without-pdo-mssql" "--without-pi3web" "--with-pdo-oci=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8=C:\php-sdk\oracle\instantclient10\sdk,shared" "--with-oci8-11g=C:\php-sdk\oracle\instantclient11\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--with-mcrypt=static" "--disable-static-analyze" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\Windows\php.ini

Let op: u hoeft bij de aanroep **geen** `echo` statement op te geven. U hoeft de functie ook niet binnen een HTML pagina aan te roepen: de functie print een complete HTML pagina!

U kunt zo snel zien met welke versie van PHP u werkt en hoe deze geconfigureerd is omdat u de in de `php.ini` gemaakte instellingen kunt uitlezen.

Let op: laat geen `phpinfo.php` bestanden op uw server staan. Er staat relatief gevoelige informatie in, waar kwaadwillenden hun voordeel mee zouden kunnen doen. Maak er een goede gewoonte van het gelijk weer weg te halen als u gezien heeft wat u wilde zien...

Tip: Als u alleen het nummer van de PHP versie zou willen zien en niet alle configuratie informatie dan kunt u simpelweg volstaan met:

```
echo phpversion(); // geeft bijv. 5.5.4
// of
echo PHP_VERSION; // idem
```

Let op: De code voorbeelden in dit hoofdstuk staan **voor de leesbaarheid niet** steeds **tussen script delimiters**. In uw **eigen scripts** moet PHP code **natuurlijk wel** altijd tussen script delimiters staan!

3.4 Statements

In PHP geeft u uw instructies, net als in elke andere high level programmeertaal, door middel van zogenaamde **statements**.

3.4.1 punt-komma

Een statement in PHP moet **altijd eindigen** op een **puntkomma (;)**.

Als u de puntkomma vergeet te noteren krijgt u gelijk een duidelijke foutmelding, bijvoorbeeld iets als:

Parse error: syntax error, unexpected T_VARIABLE in **test.php** on line **3**

Merk op dat het regelnummer in dit geval meestal niet de regel is waar het probleem zit. Dat is meestal de regel **erboven** (dus in het voorbeeld regel 2). Omdat u hier de puntkomma vergeten

bent, zal PHP denken dat de opdracht doorloopt en dan struikelen over code die eigenlijk in een nieuw statement had moeten staan.

Denk er dus goed aan elk statement met een `;` af te sluiten! U zult het zeker in het begin best eens vergeten, maar op een gegeven moment wordt het heel vanzelfsprekend.

NB.: Er is overigens **één uitzondering** op de regel dat elk statement met een puntkomma moet worden afgesloten. Als het statement de **laatste regel** is in een **PHP codeblok** dan mag u de **puntkomma weglaten**. De **eind scriptdelimiter** `?>` geldt namelijk **ook** als het **einde** van een **statement**.

Uit de PHP manual: *"The closing tag of a block of PHP code automatically implies a semicolon; you do not need to have a semicolon terminating the last line of a PHP block".*

Zie: <http://php.net/manual/en/language.basic-syntax.instruction-separation.php>

Toch kunt u zich maar beter aanwennen altijd puntkomma's te zetten! Op die manier is een codeblok altijd makkelijk uit te breiden met extra statements.

Veel ontwikkelaars vinden het echter wel heel makkelijk om in **templates** waar ze toch al de **short echo** gebruiken de puntkomma ook maar weg te laten. In plaats van `<?php echo $firstname; ?>` of `<?= $firstname; ?>` gebruiken zij dan `<?= $firstname ?>` of zelfs `<?=$firstname?>`. De spaties kunnen namelijk ook weggelaten worden... En deze blokjes zullen niet snel uitgebreid worden met extra statements!

3.4.2 Eigen regel

Een statement hoeft niet per se op een eigen regel te staan, maar dat is wel veel leesbaarder. Vergelijk deze twee fragmenten maar eens.

Eerst alle statements op een regel:

```
$x=1;$y=21;$sum=$x+$y;echo $sum;
```

Vervolgens met elk statement op zijn eigen regel:

```
$x=1;
$y=21;
$sum=$x+$y;
echo $sum;
```

Beide fragmenten doen hetzelfde (namelijk 22 wegschrijven) maar hoewel het tweede meer ruimte inneemt is het veel leesbaarder en makkelijker aan te passen.

Die 'dollaridingen' in de code zijn overigens variabelen en daar gaan we het zo meteen over hebben.

3.4.3 Witruimte

Met wat extra witruimte wordt het nog veel leesbaarder:

```
$x  = 1;
$y  = 21;
$sum = $x + $y;

echo $sum;
```

Witruimte kost geen extra processortijd en maakt de code absoluut niet langzamer. Blanco regels binnen de PHP blokken worden namelijk volledig genegeerd. Het kost u misschien iets meer tijd om

de code op te maken, maar dat is een eenmalige inspanning en het maakt de code veel beter leesbaar!

Het gebruik van commentaar zou de code nog beter leesbaar maken. Het gebruik van commentaar is zelfs zo belangrijk dat we er later in dit hoofdstuk apart op terugkomen (zie **3.10 Commentaar**).

3.4.4 Meerdere regels

In PHP kan een lang statement voor de leesbaarheid op meerdere regels genoteerd worden. Linebreaks in een editor zijn voor PHP gewoon witruimte en worden genegeerd.

Onderstaande code levert dus geen enkel probleem op:

```
$sum = $x +
      $y +
      $z;
```

De puntkomma sluit het statement af.

Ook binnen een string (zie 3.3.5 en 3.7) mag u afbreken. De witruimte wordt dan wel onderdeel van de string, maar dat is wanneer de string (X)HTML of SQL bevat over het algemeen geen probleem. We komen hier later in de cursus nog op terug.

3.5 Variabelen

Eén van de belangrijkste concepten in elke programmeertaal is de mogelijkheid om informatie (tijdelijk) op te slaan zodat u er later in het script weer gebruik van kunt maken. Dit doet u met variabelen. Variabelen zijn de bouwblokken van elk programma. In PHP kunt u variabelen aanmaken, er vervolgens waarden aan toe wijzen en die in uw script gebruiken om er bijvoorbeeld (wiskundige) berekeningen mee uit te voeren of stukken tekst te ontleden.

3.5.1 Wat is een variabele?

Een variabele is een klein stukje geheugenruimte waaraan u een naam gegeven hebt. Indien u het lastig vindt u hier iets bij voor te stellen dan kunt u een variabele ook zien als een lade in een ladekast. Elke lade voorziet u van een etiket (de naam van de variabele) en in de lade kunt u spullen zoals cijfers, letters e.d. stoppen (de waarde van de variabele).

Door in uw script naar de naam van de variabele te verwijzen kunt u de opgeslagen waarde uitlezen of aanpassen. U haalt als het ware een lade leeg om iets met de inhoud te gaan doen. Een variabele is leeg totdat u er data in stopt (hoewel er bij het aanmaken al wel geheugenruimte gereserveerd wordt).

3.5.2 Variabelen declareren

Het aanmaken van variabelen heet **declareren**. U moet dit in veel talen **expliciet** doen voordat u een variabele gaat gebruiken. In de meeste programmeertalen en scriptingtalen declareert u daarom variabelen met een speciaal keyword (bijvoorbeeld `var` in JavaScript of `Dim` in VBScript). In strongly typed languages als Visual Basic, C# of Java moet u dan ook nog eens aangeven van welk **datatype** de variabele is.

In PHP is dat allemaal **niet nodig**. U kunt een variabele simpelweg declareren door er een naam voor te kiezen. Variabelen beginnen in PHP altijd met een dollar-teken, zoals u straks zal leren, dus u zou een variabele `aantal` als volgt kunnen declareren:

```
$aantal;
```

U heeft nu een variabele gedeclareerd zonder er een waarde aan toe te kennen. Als u op deze manier **meerdere** variabelen zou willen declareren dan moet u ze met **puntkomma's** scheiden:

```
$aantal; $prijs;
```

Let op: U mag geen komma's gebruiken!

```
$aantal, $prijs; // niet doen!
```

Het gebruik van een komma geeft een fatal error:

```
Parse error: syntax error, unexpected ',' in demo.php
```

Maar: **het simpelweg declareren van variabelen wordt echter gezien als een slechte manier van werken!** Het is beter om variabelen altijd gelijk een beginwaarde te geven. Dit proces heet **initialiseren**, de variabele wordt daarbij **impliciet gedeclareerd**:

```
$aantal = 5;
```

Ook als u het juiste aantal nog niet weet moet u de variabele toch een waarde geven. U kunt er dan voor kiezen om hem bijvoorbeeld eerst op 0 te zetten of de variabele bewust leeg te laten:

```
$aantal = NULL;
```

U mag in PHP overigens meerdere variabelen in een keer dezelfde waarde geven:

```
$aantal = $prijs = NULL;
```

Impliciet declareren is in de praktijk dus heel erg makkelijk. U kunt gewoon variabelen aanmaken wanneer u ze nodig heeft. Een nadeel is wel dat PHP bij het maken van een typefout automatisch een nieuwe, lege variabele aanmaakt en dit kan dan weer tot onvoorspelbare resultaten leiden.

3.5.2.1 Error_reporting()

Door bovenaan uw pagina's `error_reporting(E_ALL);` op te nemen zal PHP een waarschuwing geven wanneer een variabele wordt gebruikt voordat u er een waarde aan heeft toegewezen. Dit is in de ontwikkelfase heel handig! U kunt het ook aangeven in `php.ini`.

Wanneer u zichzelf ook nog aanleert om elke variabele die u in uw pagina gaat gebruiken alvast bovenaan de pagina te introduceren door er een initiële waarde aan toe te kennen dan kan er bijna niets meer fout gaan!

Laten we voordat we het toekennen van waardes bespreken eerst eens kijken naar de naamgeving van variabelen.

3.5.3 Naamgeving

In PHP begint elke variabelenaam, net als in Perl, met een `$`-teken. De naam moet **beginnen met een letter of een underscore**, waarna deze (optioneel) **aangevuld** kan worden met andere **letters, cijfers** en **underscores**.

Als programmeur moet u steeds geschikte namen voor uw variabelen bedenken. U moet daarbij een balans zien te vinden tussen een **logische beschrijvende naam** en een **acceptabele lengte**. Te lang is

niet handig, maar vermijd afkortingen als `$a` of `$x`. Over twee maanden weet u echt niet meer wat u daarmee bedoelde... Laat staan een programmeur na u.

Aan de andere kant is het gebruik van `$i` voor een teller in een lus, juist perfect en zou een lange naam als `$teller_voor_in_de_lus` juist onwerkbaar zijn.

In bovenstaand voorbeeld ziet u dat bij een variabelenaam die uit meerdere delen is opgebouwd de delen worden gescheiden door **underscores**. Dit wordt in de praktijk veel gedaan, deze notatie wordt ook wel **join_lower** of **snake_case** genoemd. Spaties kunnen namelijk niet gebruikt worden! Een andere veel voorkomende manier is de zogenaamde **camelCap notatie** waarbij de variabele met een kleine letter begint maar de volgende delen in de naam met een hoofdletter beginnen.

Het is echt heel belangrijk om goede namen te kiezen voor uw variabelen. Dat maakt dat u code veel beter leesbaar, begrijpbaar en onderhoudbaar wordt! Een goede naam is een naam die betekenisvol is, dat wil zeggen een die goed omschrijft wat de variabele bevat. Vraag u dus altijd af wat de variabele gaat bevatten. Bedenk dan een logische beschrijvende naam. Wordt de naam langer dan pak hem beet 15 karakters kijk dan of u hem af kan korten zonder (te veel) aan betekenis te verliezen.

Veruit het beste is om variabelen (en commentaar) in **Engels** te houden. Dit maakt uw code in principe voor iedereen ter wereld leesbaar. En is dus verplichte kost als u in een internationale organisatie opereert. Voordeel is dat Engelse termen vaak ook herkenbaarder en compacter zijn. Een nadeel kan wel zijn dat er een groter risico is dat u voor een gereserveerd keyword kiest (maar daar wordt u dan snel genoeg op geattendeerd met een foutmelding!).

Dit gezegd hebbende zullen we tijdens deze Nederlandstalige cursus toch regelmatig gebruik maken van Nederlandse namen en commentaren.

Samenvattend:

- Gebruik **alleen cijfers en letters**!
- Gebruik **geen** speciale tekens als **spaties, punten of hekjes** of horizontale **streepjes** (hyphens)! Underscores (`_`) mogen wel.
- De naam kan alleen met een letter of underscore beginnen, dus niet met een getal!
- Letters zijn de uppercase en lowercase letters van A t/m Z en a t/m z, maar ook alle ASCII waarden tussen 127 en 255. Theoretisch zou u dus ook accenten e.d. kunnen gebruiken (zo is `$ëïé` is een geldige naam) maar dit wordt sterk afgeraden!
- Gebruik **geen keywords** (gereserveerde woorden). Eventueel mag het wel als deel van de naam.
- Gebruik unieke namen: dezelfde naam kan niet verwijzen naar twee verschillende variabelen!
- Gebruik beschrijvende namen zodat u snel kunt zien wat de variabele bevat, vrijwel altijd zal dat een zelfstandig naamwoord zijn.

Let op: PHP is **hoofdlettergevoelig** (*case sensitive*) wat betreft de naamgeving van variabelen! Dat betekent dat de variabele `$leeftijd` een andere is dan `$Leeftijd` of `$LEEF TIJD`. Dit kan tot onverwachte resultaten leiden als u per ongeluk een typefout maakt.

Dit geldt overigens **niet** voor de andere opdrachten die u in PHP kunt geven. De **namen** van PHP **functies** en **controlestructuren** zijn **niet hoofdlettergevoelig**. Dat is in de praktijk wel makkelijk omdat een kleine inconsequentie in de schrijfwijze u niet meteen in de problemen brengt. Maar het is uiteraard maar een kleine moeite om een consequente stijl aan te houden, en dat is wel zo netjes!

In onderstaande tabel ziet u een aantal voorbeelden van (on)geldige variabele namen:

Geldig	Ongeldig
\$naam	naam
\$naam	\$1
\$naam1	\$24hrs
\$naam_1	\$first.name
\$NAAM	\$first name
\$ Naam	\$first-name
\$FirstName	\$ first name
\$first_name	\$(first) name

Of een naam geldig is of niet, weet u overigens meteen wanneer u uw script laat uitvoeren. De parser geeft bij ongeldige variabele namen meteen een fatal error!

Veel programmeurs vinden het prettig om het datatype van de variabele in de naam aan te geven. Zie de opmerking na de bespreking van de datatypen.

3.5.4 Variabelen initialiseren

Het voor de eerste keer toewijzen van een waarde aan een variabele heet **initialiseren**. U doet dat zoals u hierboven al had gezien met het `=`-teken, de zogenaamde *toewijzingsoperator*. De variabele waaraan de waarde wordt toegewezen staat altijd links!

```
$mijn_variabele = 2;
```

U mag in PHP ook meerdere variabelen in een keer initialiseren:

```
$voornaam = $achternaam = "";
```

Na de initiële toewijzing kunnen de opgeslagen waarden natuurlijk gedurende de loop van het programma veranderen, daarvoor zijn het tenslotte variabelen. Maar daar gaan we het straks over hebben. Het initialiseren van een variabele is echter een belangrijke stap omdat u hiermee ook aangeeft met wat voor een soort gegevens u (in eerste instantie) gaat werken.

3.6 Datatypen

Een datatype, of gegevenstype, is de classificatie van gegevens in een bepaalde categorie volgens zijn kenmerken. Zo worden alfanumerieke karakters geclassificeerd als strings en gehele getallen als integers.

Bij de meeste programmeertalen moet u gelijk bij het declareren van een variabele aangeven om wat voor een soort variabele het gaat, oftewel welk type data er in kan worden opgeslagen.

Bij PHP hoeft dat niet. Sterker nog, het kan zelfs niet! Een variabele is dus niet van een bepaald datatype maar kan elk datatype bevatten (net zoals in JavaScript). Een dergelijke scripttaal wordt daarom ook wel een *untyped language* genoemd, dit in tegenstelling tot een programmeertaal als bijvoorbeeld Java die (*strongly*) *typed* is. Een betere naam is misschien **loosely typed** of **dynamic typed language**. PHP heeft namelijk geen expliciete gedeclareerde datatypes. PHP bepaalt bij het inlezen van de variabele zelf het datatype, afhankelijk van de data en de context. Het bepaalt de gegevenstypen door de kenmerken van de geleverde gegevens te analyseren.

PHP ondersteunt impliciet de volgende **8 datatypen**:

PHP kent **4 eenvoudige** datatypen (ook wel *scalars* genoemd):

- **String** (tekst)
- **Integer** (heel getal)

- **Floating point** (gebroken getal)
- **Boolean** (Waar of niet waar)

PHP kent **2 complexe** of **compound** datatypen (die zo heten omdat ze kunnen worden samengesteld uit variabelen van meer dan één datatype):

- **Array**
- **Object**

Bovendien is er een datatype dat gebruikt wordt voor verwijzingen naar externe bronnen buiten PHP:

- **Resource**

Dit datatype komen we bijvoorbeeld tegen bij het werken met **bestanden** en **databases**.

Tot slot is er een speciaal datatype dat de **afwezigheid van data** aangeeft:

- **NULL**

We hebben in totaal dus **8 basis** (ook wel *primitive*) **datatypen** en afhankelijk van het soort informatie dat in de variabele wordt gestopt (of juist niet), wordt deze dus van een bepaald type.

3.6.1 String

Teksten of tekenreeksen zoals ze ook wel genoemd worden, kunnen elke denkbare combinatie van letters, cijfers en speciale tekens bevatten. In het Engels worden ze met de term *string* aangeduid. Dat is bij programmeurs zo ingeburgerd dat wij die term tijdens deze cursus ook gebruiken.

Stel u wilt in een variabele `$naam` de naam Karin opslaan. In PHP noteert u dat als volgt:

```
$naam = "Karin";
```

Let op: In PHP kunnen strings zowel tussen *single* als *double quotes* staan:

```
$naam = 'Karin';
```

Bovenstaand statement geeft precies hetzelfde resultaat. In **Double vs. single quotes** gaan we uitgebreid in op de verschillen. Voorlopig kunt u onthouden dat als u geen bijzondere wensen hebt en alleen tekst in een string wilt zetten dat het gebruik van **single quotes** een **goed uitgangspunt** is.

Als er echter een **single quote** in de **tekst** staat dan moet u deze **escapen** met een backslash of gebruik maken van **dubbele quotes**:

```
$safe = 'Karin\'s Place'; // escapen met backslash
$safe = "Karin's Place"; // of omsluiten met dubbele quotes
```

Merk op dat een **getal** dat **tussen quotes** staat, bijvoorbeeld `'1234'`, ook een **string** is. Het is dan verstandig deze waarde eerst **expliciet te converteren naar een integer** voordat u er mee aan de slag gaat. U leest daar meer over in **3.7 Variabelen Converteren**. Overigens is PHP in veel situaties in staat tot **impliciete conversie**.

Ook `'abc1234'` en `'$@#&*'` zijn voorbeelden van strings.

Maximale lengte?

De lengte van een string is vanaf PHP7 op 64-bits systemen bijna **onbeperkt**. In theorie kan de maximale grootte namelijk 9223372036854775807 bytes zijn (9.2 EB oftewel Exabyte). Op 32-bits systemen en in eerdere versies van PHP was de maximale grootte overigens toch ook al een respectabele 2GB (2147483647 bytes). U kunt dus zonder problemen hele grote bestanden inlezen!

In de praktijk loopt u meestal eerst tegen de limiet op van het totale geheugen dat PHP kan toewijzen voor alle variabelen in een bepaalde scriptuitvoering. De geheugenlimiet is standaard ingesteld op 128 MB maar deze kunt u verhogen indien nodig. In de normale praktijk zal dit echter vrijwel nooit voorkomen...

Zie voor een uitgebreide discussie: <https://stackoverflow.com/questions/3189040/what-is-the-maximum-length-of-a-string-in-php>

3.6.2 Integer

Numerieke variabelen kunnen van het type integer of van het type floating point zijn. Wanneer een getal een **geheel getal** is, dat wil zeggen geen waarde achter de komma heeft dan is het een **integer** (int). Heeft een getal wel **waarden achter de komma** dan is het een **floating point** (float).

Een integer kan **positief** zijn, maar ook **negatief**.

De **minimale waarde** en **maximale waarde** van een integer hangen af van het Operating System. Op een 32-bits systeem ligt de waarde van een integer tussen -2 miljard en +2 miljard ($\pm 2.147.483.648$). Op een 64-bits loopt dat op tot $\pm 9.223.372.036.854.775.808$. Oftewel 9 triljoen, 223 miljard, 372 biljoen, 36 miljard, 854 miljoen, 775 duizend en 808.

Sinds PHP 7.0 kunt u de min en max waarden uitlezen met de ingebouwde constanten `PHP_INT_MIN` en `PHP_INT_MAX`. We behandelen constanten later in dit hoofdstuk.

Wanneer we een geheel getal toekennen aan een variabele dan zal de variabele als datatype integer krijgen. Door het getal 1967 in de variabele `$jaartal` te stoppen, of beter gezegd: door de waarde 1967 toe te kennen aan `$jaartal`, wordt deze variabele een numerieke variabele van het datatype integer.

```
$jaartal = 1967;
```

Behalve de gebruikelijke decimale notatie (base 10) kent PHP ook de hexadecimale (base 16) en de octale (base 8) notatie:

```
$getal = 255;    // decimaal
$getal = 0xFF;   // hexadecimaal (begint met 0x)
$getal = 0377;   // octaal (begint met 0)
```

In alle drie de gevallen zou een `var_dump($getal)` als resultaat `int(255)` geven. Hoewel de andere notaties soms handig kunnen zijn, zullen we er in deze cursus verder niet op ingaan...

3.6.3 Floating Point

Een **floating point value** is een getal met decimalen achter de komma. Alleen werkt u in PHP niet met een komma maar met een `.` als scheidingsteken:

```
$lengte = 1.78;
```

Het maximal aantal cijfers achter de komma dat u kunt gebruiken zonder precisie verlies is 15. Sinds PHP 7.2 kunt u deze waarde uitlezen met de ingebouwde constante `PHP_FLOAT_DIG`. De minimale

waarde van een float is 2.2250738585072E-308 en de maximale waarde is 1.7976931348623E+308. Sinds PHP 7.2 kunt u deze waarden uitlezen met de ingebouwde constanten `PHP_FLOAT_MIN` en `PHP_FLOAT_MAX`.

3.6.4 Boolean

Een **boolean** is misschien wel het eenvoudigste datatype. Een variabele van dit datatype kan maar 2 mogelijke waarden bevatten: Ze is `true` of `false`, oftewel waar of niet-waar.

In dit voorbeeld krijgt de boolean variabele `$getrouwd` de waarde `true`:

```
$getrouwd = true;
```

De waarden `true` en `false` zijn **case-insensitive**, `True` of `TRUE` mag in principe dus ook. U zou de de schrijfwijze die u het prettigst vindt kunnen kiezen (wees dan wel consistent!), maar de **PSR-12: Extendend Coding Style** (<https://www.php-fig.org/psr/psr-12/>) stelt dat:

All PHP reserved keywords and types MUST be in lower case.

En dat zullen wij vanaf nu dus ook doen!

Booleans zijn **niet bedoeld om te printen**. Als u het toch zou doen dan levert `true` het getal 1 op en `false` een lege string (""), die dus niet zichtbaar is. U gebruikt Booleans vooral in **controlestructuren** om te **controleren** of aan bepaalde **condities** voldaan wordt. We bespreken dat later in dit hoofdstuk in **paragraaf 3.12** en verder.

3.6.5 null

Wanneer u de variabele al wel hebt gedefinieerd maar nog geen (definitieve) waarde hebt om er aan toe te kennen dan kunt u als waarde `null` opgeven:

```
$naam = null;
```

U zou het ook zo kunnen formuleren: de variabele `$naam` verwijst op dit moment nog nergens naar. Pas als u er een waarde aan toekent is dat wel het geval.

U kunt een variabele met `null` initialiseren om er vervolgens later een waarde aan toe te kennen, maar u kunt het ook later in het script doen bij een variabele die al een waarde heeft (en hierna dus niet meer):

```
$name = 'Arno';
$name = null; // naam heeft nu geen waarde meer
```

Let op: `null` betekent de **afwezigheid van een waarde**. Het is daarmee iets anders dan een lege string '', het getal 0 of de boolean `false`. Dat zijn namelijk allemaal wel waarden (met hun eigen datatype). Wat dat betreft is het `null` concept binnen PHP volledig vergelijkbaar met dat in databases.

Met `var_dump()`, dat we later in dit hoofdstuk nog uitgebreid zullen bespreken, kunt u dat ook goed zien:

```
$test = "";
var_dump($test); // geeft string(0) ""

$test = 0;
var_dump($test); // geeft int(0)
```

```
$test = false;
var_dump($test); // geeft bool(false)
```

maar:

```
$test;
var_dump($test); // geeft NULL

$test = NULL;
var_dump($test); // geeft NULL
```

Het verschil is dat u, wanneer u een gedeclareerde variabele zonder enkele waarde in uw code probeert te gebruiken, een foutmelding krijgt:

```
$name;
echo $name;
```

Dit geeft de melding:

Notice: Undefined variable: name in **demo.php**

Als u `NULL` toekent dan zou dit geen problemen opleveren:

```
$name = null;
echo $name;
```

De variabele is nu namelijk wel gedefinieerd, ook al bevat hij geen waarde. Maar dit is natuurlijk niet hoe u de `null` wilt gebruiken. Met `null` geeft u immers als het ware aan dat er een waarde verwacht wordt maar dat die op het moment van noteren nog niet beschikbaar is. Later in uw script kunt u daar dan ook op controleren met de standaard in PHP ingebouwde `is_null()` functie:

```
if (is_null($naam)) {
    echo 'naam heeft nog geen waarde...';
} else {
    echo $naam;
}
```

Of simpelweg op de waarde:

```
if ($naam === NULL) {
    echo 'naam heeft nog geen waarde...';
} else {
    echo $naam;
}
```

Als de naam wel een waarde heeft gekregen dan wordt de code in de else branch uitgevoerd. Later in de cursus leert u meer over condities en if/else controlestructuren.

De notatie is **hoofdletter ongevoelig** (`NULL` mag ook) maar de meeste programmeurs gebruiken **hoofdletters**.

3.7 Arrays

Een array (letterlijk: een *reeks*) is een datatype waarin **meerdere waarden** kunnen worden opgeslagen onder één variabele naam. Als een variabele een **lade** is waar u een waarde in kunt

stoppen, dan is een array in die analogie een **ladekast**. Men spreekt ook wel over een **matrix** of een **container variable** die meerdere **elementen** kan bevatten.

In een array heeft ieder element dus **dezelfde naam** (de naam van de array), maar de naam wordt aangevuld met een **index**, in het Engels ook wel **key** genoemd. Hierdoor zijn alle waarden in de array nog steeds apart benaderbaar.

Waarom is dat nuttig? Zoals we straks zullen zien maken arrays het juist ook mogelijk om de inhoud te filteren, te sorteren en er allerlei logica op toe te passen die met losse variabelen niet mogelijk was geweest...

U voelt vast wel aan dat als we meerdere namen in variabelen willen zetten, dat dit niet handig is:

```
// niet handig!
$naam1 = 'Arno';
$naam2 = 'Piet';
...
$naam9 = 'Truus';
```

Het wordt heel lastig om nu iets met alle namen tegelijk te doen... Dat de namen van de variabelen in bovenstaand voorbeeld al een soort van teller meekregen is overigens een sterke indicatie dat een array het juiste opslagformaat is voor deze variabelen!

Het gebruik van arrays is in eerste instantie misschien even **wennen** maar ze zijn vooral **heel erg nuttig**. Doordat u maar **een variabele** hebt (de array), terwijl die toch **meerdere naam-waarde** paren kan bevatten (de afzonderlijke elementen met hun eigen key en value), kunt u meestal met **minder code** af. Dat maakt de code beter leesbaar en kan een hoop werk schelen. Zeker bij taken waar veel **herhalingen** nodig zijn, omdat u dan alle elementen in de array makkelijk in een **lus** kunt **benaderen**, zoals we straks zullen zien. Ook kunt u met een array makkelijk een **hele verzameling data** als een **argument aan een functie** doorgeven.

De index kan een **getal** zijn of een **string**. Arrays met een numerieke index heten simpelweg **numeriek geïndexeerde arrays**. En u heeft er wellicht al eens mee gewerkt in een andere programmeertaal want vrijwel elke programmeertaal kent dergelijke arrays. In PHP (en Perl) wordt echter ook veel gebruik gemaakt van **user-defined keys**: u kunt voor de index zelf een handige beschrijvende of anderszins betekenisvolle naam kiezen en deze als **string** toekennen. Dergelijke arrays worden **associatieve arrays** genoemd. Dit type array wordt heel veel gebruikt in PHP en komt in andere talen (anders dan Perl) eigenlijk niet voor. Deze talen kennen wel vergelijkbare constructies maar die worden dan met **hash**, **map** of **dictionary** aangeduid.

Bij elke key in de array hoort steeds een bepaalde **value**. Dat kan een enkele waarde zijn zoals een string of een getal, maar het kan ook weer een array zijn. Dergelijke arrays, waarbij elementen zelf ook weer arrays zijn, worden **multi-dimensionale arrays** genoemd.

Arrays worden veel gebruikt in PHP. Een aantal standaard ingebouwde variabelen, de zogenaamde **superglobals** (`$_GET`, `$_POST`, `$_SERVER` etc.) zijn namelijk **associatieve arrays** zoals we later in dit hoofdstuk zullen zien. Ze worden automatisch aangemaakt bij elk request en gevuld met de beschikbare elementen. In het database hoofdstuk zullen we bovendien ontdekken dat **database records** min of meer standaard **als een array** worden **aangeleverd**.

3.7.1 Numerieke array aanmaken

Het aanmaken en vullen van arrays wordt meestal in één stap gedaan. In PHP kan een array op nogal wat verschillende manieren tot stand komen. We laten hieronder een aantal manieren zien waarop u een array handmatig kunt aanmaken en vullen. In de praktijk zult u echter ook vaak werken met arrays die al gevuld worden aangeleverd. Denk daarbij aan de superglobals maar ook aan de inhoud van een bestand of zoals we straks zullen zien: data uit de database!

Een array waarin u de namen van de dagen van het weekend opslaat, kunt u in PHP als volgt declareren en initialiseren:

```
$weekend[] = 'zaterdag'; // index = 0
$weekend[] = 'zondag';   // index = 1
```

In bovenstaand voorbeeld zetten we de dagen van de week gelijk in de gewenste volgorde en geven we zelf geen index op.

Zoals u ziet, hoeft u in PHP een array, net als de andere datatypen, niet eerst expliciet aan te maken. De eerste keer dat u de array gebruikt wordt hij automatisch aangemaakt. In dit voorbeeld wordt de array `$weekend` (er van uitgaande dat hij nog niet bestaat) dus aangemaakt op het moment dat de string `'zaterdag'` als eerste element wordt toegevoegd. De **rechte haken** `[]` achter de naam van de variabele `$weekend` zijn daarbij wel **essentieel**. Als u die weg zou laten dan was het een gewone (scalar) variabele geworden...

Merk op dat de **index** van het **eerste element** in PHP de **waarde 0** krijgt, net zoals bij veel andere programmeertalen zijn numerieke arrays **zero based**. Dat is misschien even wennen als u hier nog niet eerder mee te maken heeft gehad, maar het went gelukkig snel. De index van elk volgend element in de array wordt steeds met de waarde 1 opgehoogd. Het tweede element in de array heeft dan als index 1 en voor het derde element is dat dus 2, en zo verder... Bij dergelijke arrays is de index dus **sequentieel** (opvolgend).

Het laatste element heeft als index het aantal elementen minus 1:

```
$index_highest = count($weekend) - 1;
echo $index_highest; // geeft 2-1 = 1
```

De functie `count()` geeft het aantal elementen in de array. We komen er later nog op terug...

Er zit in principe geen grens aan het aantal elementen dat u in een array kunt zetten. Een mogelijke beperking bij enorme aantallen zou de `memory_limit` setting in `php.ini` kunnen zijn. Deze staat standaard op een 8 MB maar u kunt dit eventueel nog verhogen.

U kunt later altijd nog items aan de array toevoegen. Arrays in PHP zijn dus per definitie **dynamisch** (ze kunnen "groeien"). Dus als weekends ooit uit meer dan 2 dagen gaan bestaan zit u goed ;-)

U kunt numeriek geïndexeerde arrays ook zelf van een index voorzien:

```
$weekend[0] = 'zaterdag';
$weekend[1] = 'zondag';
```

Of als u het prettiger zou vinden om met 1 te beginnen:

```
$weekend[1] = 'zaterdag';
$weekend[2] = 'zondag';
```

U kunt ook de laatste dagen van de week als index gebruiken:

```
$weekend[6] = 'zaterdag';
$weekend[7] = 'zondag';
```

In dergelijke scenario's zullen de ontbrekende indicies niet worden aangemaakt. Een statement als dit:

```
echo $weekend[0];
```

geeft nu iets als:

Notice: Undefined offset: 0 in **demo.php** on line 4

Wanneer u echter op deze manier arrays vult dan bent u in feite associatieve arrays aan het aanmaken. We bespreken deze later in dit hoofdstuk.

Let op: als er al een **element met dezelfde index** in de array had gestaan dan **wordt** deze **overschreven!**

3.7.1.1 array()

Een makkelijker manier om in één keer een array aan te maken en waarden aan de variabelen toe te kennen is met behulp van het language construct `array()`:

```
$weekend = array('zaterdag', 'zondag');
```

PHP maakt ook nu weer zelf de indicie aan, te beginnen bij 0. U kunt echter ook een ander startpunt afdwingen met:

```
$weekend = array(1 => 'zaterdag', 'zondag');
```

De **arrow operator** die u hiervoor gebruikt is een *isgelijk*-teken gevolgd door een *groterdan*-teken. U zult het straks bij de associatieve arrays ook gebruiken. Nu krijgt u bij `$weekend[1]` de waarde `'Zaterdag'`.

3.7.1.2 Array literals

Vanaf PHP 5.4 kan het nog eenvoudiger met de **array literal syntax** die u misschien ook van JavaScript kent:

```
$weekend = ['zaterdag', 'zondag'];
```

U kunt deze notatie ook over meerdere regels uitsmeren voor een betere leesbaarheid:

```
$weekend = [
    'zaterdag',
    'zondag'
];
```

3.7.1.3 explode()

Een andere manier om een array snel te vullen is met de `explode()` functie. U geeft als argumenten een (reeks) scheidingsteken(s) en de te bewerken string:

`explode(separator, string)`

De string wordt geretourneerd als array met evenveel elementen als er stukken string gescheiden zijn door de delimiter:

```
// string:
$dagen = 'zaterdag zondag';
$weekend = explode(' ', $dagen); // array
```

Ook dit levert net als de vorige manieren een **numeriek geïndexeerde** array op met twee items op.

De tegenhanger van `explode()` is `implode()` die van een array een string maakt en die we zo meteen zullen tegenkomen.

NB.: Wanneer u in de vervolgcursus **reguliere expressies** hebt leren gebruiken dan kunt u ook de nog krachtiger functie `preg_split()` en zijn tegenhanger `preg_join()` toepassen.

3.7.1.4 range()

Mocht u ooit behoefte hebben aan een numeriek geïndexeerde array waarin de elementen een **openvolgende waarde** hebben, dan is de `range()` functie heel handig:

```
// vul array met getallen van 1 t/m 10
$numbers = range(1, 10);

// vul array met hoofdletters A t/m Z
$letters = range('A', 'Z');
```

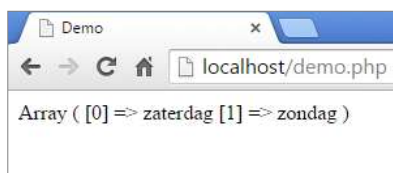
3.7.2 Arrays inzien met print_r()

U kunt de inhoud van een array snel controleren met de functie `print_r()`. De `_r` in de functienaam staat voor *recursive*, de functie kan namelijk ook multi-dimensionale arrays (waarover later meer) van begin tot eind weergeven!

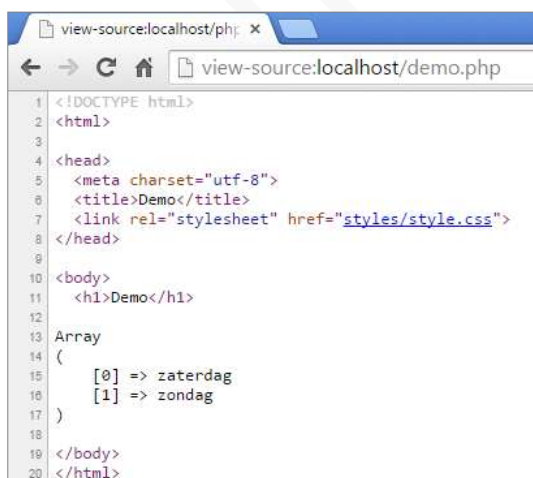
U gebruikt de functie als volgt:

```
print_r($weekend);
```

Dit geeft:



Alles staat nu in de browser op één regel, wat bij grotere arrays nogal onoverzichtelijk wordt. Bekijk u echter de broncode in de browser dan staan de elementen van de array wel keurig onder elkaar:



U kunt deze beter leesbare uitvoer ook **direct in de browser** te zien krijgen wanneer u de functieaanroep in een HTML `<pre>` element zet:

```
echo '<pre>';
print_r($weekend);
echo '</pre>';
```

`print_r()` heeft een optioneel tweede argument `$return` waarmee u kunt aangeven dat er niet rechtstreeks geprint moet worden. Standaard staat deze op `false` maar als u hem op `true` zet dan wordt de uitvoer als string geretourneerd. Dan kan u bovenstaande ook op één regel doen:

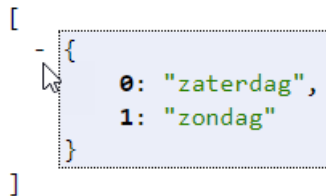
```
echo '<pre>' . print_r($weekend, true) . '</pre>';
```

In beide gevallen krijgt u nu het gewenste resultaat **in de browser** te zien:



Doordat de uitvoer in een HTML `<pre>` element is opgenomen wordt de witruimte in de uitvoer nu wel gehonoreerd (en niet teruggebracht tot maximaal 1 spatie). U ziet dat de browser daarbij ook automatisch omschakelt naar een Monospace font om een correcte positionering van de karakters te kunnen garanderen (een i neemt evenveel ruimte in als een w).

Tip: Voor een instant fraaie weergave van een `print_r()` kunt u ook de **Chrome extension PHPView** gebruiken:



Door op het - teken te klikken kunt u de array items weggelassen, wat handig kan zijn bij grote geneste arrays.

Let op: er is dus **geen echo nodig** omdat `print_r()` standaard zijn output gelijk print.

Als u het wel zou combineren met een `echo` dan krijgt u een 1 achter de uitvoer omdat `print_r()` net als `print()` de waarde `true` retourneert na de uitvoer... Om dezelfde reden moet u het ook niet concatenen zonder als tweede argument `true` mee te geven...

```
// niet doen!
echo '<pre>';
echo print_r($weekend);
echo '</pre>';

// niet doen!
echo '<pre>' . print_r($weekend) . '</pre>';
```

Dit geeft namelijk in beide gevallen:



Let op: denk er bij het uitprinten van door gebruikers gevulde arrays zoals `$_GET` en `$_POST` aan dat u ook de uitvoer van `print_r()` moet **escapen**! We komen hier later in de cursus nog op terug.

U kunt nog gedetailleerder zien wat er in een array zit met het straks te bespreken `var_dump()`.

Vaak zult u na het uitprinten de rest van het script niet willen uitvoeren. Dit kunt u realiseren met `die()` of het iets vriendelijker klinkende `exit()`:

```
print_r($weekend);
die();
// rest van de code hieronder wordt niet uitgevoerd...
```

Het kan ook in 1x:

```
die(print_r($weekend));
// rest van de code hieronder wordt niet uitgevoerd...
```

In een later stadium zullen we een eenvoudige functie maken die dit allemaal wat handiger voor ons regelt.

3.7.3 Arrays uitlezen

U kunt een array **niet** meteen wegschrijven met `echo`:

```
// Let op: werkt niet!
echo $weekend;
```

Dit geeft:

Notice: Array to string conversion in **demo.php**
Array

U moet de waarden in dergelijke arrays benaderen door de numerieke index, de zogenaamde **key**, tussen **rechte haken** achter de variabele naam te zetten. Het is daarbij belangrijk dat u zich realiseert dat arrays in PHP (net als in veel andere programmeertalen) **zero based** zijn, wat wil zeggen dat het eerste item in de array de index 0 heeft gekregen en dat vanaf daar verder wordt geteld:

```
echo $weekend[0]; // zaterdag
echo $weekend[1]; // zondag
```

Wanneer u een niet bestaande index opgeeft:

```
echo $weekend[2];
```

dan krijgt u de waarde `NULL` terug. Er zal, afhankelijk van het niveau van `error_reporting`, of niks te zien zijn, of een waarschuwing als deze:

Notice: Undefined offset: 2 in **demo.php**

3.7.3.1 Accolades

Overigens mag u **ook** altijd **accolades** gebruiken in plaats van de rechte haken:

```
echo $weekend[0]; // zaterdag
echo $weekend{0}; // zaterdag
```

Maar dit is vrij **ongebruikelijk**. **Rechte haken** zijn de **standaard** en maakt het voor iedereen meteen duidelijk dat het om een array gaat.

3.7.3.2 Variabelen

U kunt ook **variabelen** gebruiken om naar de index van de array te verwijzen, zolang het maar een geldige index oplevert:

```
$x = 0;
echo $weekend[$x + 1]; // zondag
```

We zullen dit principe later gebruiken in `for()` lussen om met een teller `$i` door numeriek geïndexeerde arrays te lopen.

3.7.3.3 implode()

Als u alle waarden uit de array wilt wegschrijven dan kunt u gebruik maken van de `implode()` functie, de tegenhanger van de eerder besproken `explode()` functie. Met `implode()` kunt u de elementen uit een array converteren naar een string. Ze worden met een optionele delimiter achter elkaar gezet:

`implode(separator, array)`

Een voorbeeld:

```
$dagen = implode(" ", $weekend);
echo $dagen;
```

geeft:

zaterdag zondag

Zou u de spatie echter door een spatie en een komma vervangen:

```
$dagen = implode(" ", $weekend);
echo $dagen;
```

Dan komt de uitvoer er dus zo uit te zien:

zaterdag,zondag

Merk op dat bij het laatste element (`zondag`) de separator niet herhaald wordt, m.a.w. er komt geen komma achter te staan! Bij een `for` of `foreach` lus was dat wel het geval geweest!

3.7.3.4 Lussen

Arrays worden meestal door middel van `for` of `foreach` **lussen** uitgelezen. Deze controlestructuren bespreken we later in dit hoofdstuk in **3.17**.

3.7.4 Associatieve arrays

Arrays kunnen in plaats van een numerieke key ook een **string** als key hebben. Dergelijke arrays worden in PHP **associatieve arrays** genoemd. In andere talen worden dergelijke arrays vaak *hash*, *hashtable* of *map* genoemd. Het grote voordeel van associatieve arrays is dat ze **beschrijvend** zijn en omdat u de keys als programmeur **zelf** kunt **kieszen** kunt u zo duidelijker maken wat voor een waarde er bij hoort. De in PHP ingebouwde superglobals zijn voorbeelden van associatieve arrays.

We kunnen van `$weekend` ook een associatieve array maken. Dit kan ook weer op drie manieren:

```
$weekend['za'] = 'zaterdag';  
$weekend['zo'] = 'zondag';
```

Met de `array()` functie:

```
$weekend = array('za' => 'zaterdag', 'zo' => 'zondag');
```

Met de **array literal**:

```
$weekend = ['za' => 'zaterdag', 'zo' => 'zondag'];
```

Merk op dat deze twee laatste notities gebruik maken van de `=>` operator. De officiële naam is de **double arrow operator** en deze wordt in PHP alleen in associatieve arrays gebruikt om values aan een key toe te wijzen. Niet te verwarren met de **greater than or equal to operator** `>=` die u in condities gebruikt (en ook precies andersom noteert!) of de **arrow operator** `->` die u bij objecten gebruikt om naar methods en properties te verwijzen. In veel andere talen gebruikt u daarvoor de **dot operator** `.` maar die is in PHP al in gebruik al **concatenation operator**...

Uitlezen gaat als volgt:

```
echo $weekend['za']; // zaterdag  
echo $weekend['zo']; // zondag
```

De index mag natuurlijk ook een variabele zijn, iets als dit werkt prima:

```
$dag = 'za';  
echo $weekend[$dag];
```

Dat is alsof u `$weekend['za']` aanriep...

Let op: als de **index** een **string** is dan moet deze wel **tussen quotes** staan! Enkele of dubbele quotes maakt niet uit, de gewone regels zijn van toepassing. Maar `$weekend[za]` mag dus niet! PHP ziet `za` dan als een ongedefinieerde constante... Meer hierover leest u in het onderdeel **Constanten**.

Normaal gesproken zult u voor het uitlezen van associatieve arrays vaak een `foreach` lus gebruiken. Zie **foreach**.

3.7.4.1 Verschillende indicie

U kunt bij het aanmaken van een array zelfs **verschillende indicie door elkaar heen** gebruiken:

```
$weekend[1] = 'zaterdag';  
$weekend['zo'] = 'zondag';
```

Maar dit is **bad practice** en kunt u beter **niet doen**! Bovenstaande array is in feite een associatieve array waarin de eerste key een integer is en de tweede een string.

3.7.4.2 Verschillende datatypen

Elementen in een array kunnen van hetzelfde datatype zijn, zoals de strings in `$weekend`, maar u kunt alle datatypen gebruiken:

```
docent = [
    'voornaam' => 'Arno',
    'leeftijd' => 48,
    'lengte' => 1.76,
    'plaats' => 'Amsterdam',
    'getrouwd' => false,
    'gelukkig' => true
];
```

Een **element** kan **zelf** ook weer een **array** kan zijn. We spreken dan over **multi-dimensionale arrays**. Hier gaan we in de **vervolgcursus** mee aan de slag, want over (het werken met) arrays valt nog heel **veel meer te vertellen** dan we op dit moment kunnen bespreken...

3.7.4.3 context

Merk op dat u met associatieve arrays als het ware een *context* schept voor uw variabelen. Als u in een script een `$title` gebruikt voor uw webpagina en later op deze pagina een `$title` wilt gebruiken voor een vacature dan gaat dit voor problemen zorgen. Met arrays kunt u naam-waarde paren die bij elkaar horen organiseren:

```
$page['title'] = 'Job Overview';

$job['title'] = 'PHP Developer';
$job['description'] = 'As a PHP Developer you will ... ';
```

Dat werkt veel prettiger!

3.8 Objects

PHP ondersteunt Object Oriented Programming (OOP). In versie 4 was de ondersteuning nog minimaal, maar vanaf versie 5 is het zoals u zou verwachten bij moderne programmeertalen.

De basis voor OOP is een datatype dat **object** wordt genoemd. Een object wordt geïntanceerd vanuit een **class** en is kort door de bocht een verzameling van variabelen waarin data wordt opgeslagen (**properties** genoemd) en bijbehorende functies (**methods** genoemd) waarmee deze data gemanipuleerd kan worden.

Een class bevat de blauwdruk voor de mogelijkheden die alle van de class geïntanceerde objecten zullen hebben. U kunt classes zelf aanmaken maar vanaf PHP 5 zijn een aantal classes standaard beschikbaar:

3.8.1 Objects met Native PHP Classes

Vanaf PHP 5 zijn een aantal classes standaard beschikbaar. U kunt er gelijk mee aan de slag, u hoeft dus niet eerst zelf een class te definiëren.

3.8.1.1 DateTime

Een voorbeeld van een standaard beschikbare class is de `DateTime` class. U kunt deze bijvoorbeeld op de volgende manier gebruiken:

```
$date = new DateTime('1968-12-06');
echo $date->format('d/m/Y');
```

Let op: de naam is niet `Date` zoals in JavaScript!

Met het keyword `new` maakt u een nieuwe instantie aan van de class `DateTime`. De instantie bevat de datum 6 december 1968 en wordt opgeslagen in een variabele die `$date` wordt genoemd. Vervolgens kunt u van de ingebouwde functionaliteit gebruik maken om deze datum te formatteren. Merk het gebruik van de **arrow operator** `->` op. In veel andere talen (Java, VB.NET, C#) wordt de **dot operator** `.` gebruikt om naar de methods en properties van een object te verwijzen, maar in PHP is de punt al in gebruik als **concatenation operator** om strings en variabelen aan elkaar te kunnen 'plakken' (waarover later meer).

U kunt dezelfde functionaliteit in PHP echter ook nog gewoon **procedureel** coderen:

```
$date = date_create('1968-12-06');  
echo date_format($date, 'd/m/Y');
```

Een overzicht van alle functionaliteit vindt u op <http://php.net/manual/en/class.datetime.php>.

3.8.1.2 mysqli

Een ander bekend voorbeeld is de standaard `mysqli` class. U kunt deze procedureel gebruiken maar ook op de OOP manier:

```
// procedureel  
$conn = mysqli_connect('localhost', 'root', 'geheim', 'cursus');  
echo mysqli_get_server_info($conn);  
  
// object oriented  
$db = new mysqli('localhost', 'root', 'geheim', 'cursus');  
echo $db->server_info;
```

In beide gevallen wordt verbinding gemaakt met de database `cursus` op de MySQL database server en de versie getoond, alleen de manier van noteren verschilt. De uitkomst is bijvoorbeeld iets als:

```
5.5.5-10.1.13-MariaDB
```

Waarmee duidelijk wordt dat we blijkbaar de MariaDB variant van MySQL gebruiken! We komen hier bij het onderdeel **Werken met databases** op terug!

3.8.2 Classless objects met `stdClass()`

Als u zelf objecten wilt aanmaken om simpelweg variabelen te kunnen onderscheiden, net zoals we eerder hebben besproken bij de associatieve arrays, dan kunt u daarvoor gebruik maken van de native `stdClass`. Hiermee kunt u zogenaamde **classless objects** aanmaken. U hoeft dan dus niet eerst zelf een class te definiëren. Aanmaken van een dergelijk object gaat als volgt:

```
$page = new stdClass();
```

De standard class heeft nog geen properties en methods, u moet ze zelf toekennen. Dit gaat *on-the-fly*.

3.8.2.1 Properties

Properties zijn eenvoudig toe te kennen:

```
$page = new stdClass();  
$page->title = 'Vacature overzicht';  
  
$job = new stdClass();  
$job->title = 'PHP Web developer';
```

Daarna kunt u de properties uitlezen:

```
echo $page->title; // Vacature overzicht
echo $job->title; // PHP Web developer
```

3.8.2.2 Methods

Het werken met methods (functies) in classless objects is op zijn zachtst gezegd nogal *tricky*. We gaan er hier dan ook verder niet op in. U leert er meer over in de vervolgcursussen en dan zult u ook in gaan zien dat u beter zelf classes kunt gaan definiëren om veel flexibeler te kunnen werken...

3.8.3 Object Literals (bestaan niet)

U kunt in PHP op dit moment **niet** objecten aanmaken met **object literals** `{}` zoals dat in JavaScript wel kan...

U kunt eventueel wel iets vergelijkbaars doen door de array als object te **casten**:

```
$objWeekend = (object)['za' => 'zaterdag', 'zo' => 'zondag'];
echo $objWeekend->za; // geeft zaterdag
```

Meer ideeën hierover vindt u op <http://stackoverflow.com/questions/455800/does-php-feature-short-hand-syntax-for-objects>.

Update: in **PHP7+** kunt u met zogenaamde **Anonymous Classes** wel eenvoudig methods aan een class-loos object toevoegen. Deze manier van werken lijkt sterk op hoe methods in JavaScript object literals kunnen worden toegevoegd. We gaan hier in de cursus **PHP OOP & MVC** mee aan de slag.

De OOP manier van werken wordt weliswaar steeds meer gemeengoed (in frameworks!), maar het wordt eigenlijk pas vooral zinvol als u de traditionele, procedurele manier van werken al beheerst. We komen er in de vervolgcursus **PHP Vervolg** en de cursus **PHP OOP & MVC** uitgebreid op terug!

3.9 var_dump()

Met de functie `var_dump()` kunt u snel zien wat een variabele voor een waarde bevat en van welk datatype die waarde is:

```
$waarde = '13';
var_dump($waarde);
```

geeft:

```
string(2) "13"
```

Oftewel, de waarde is een string van 2 karakters, namelijk "13". Maar als u er een integer van maakt:

```
$waarde = 13;
var_dump($waarde);
```

dan is dit ook meteen duidelijk:

```
int(13)
```

Voor arrays werkt het ook:

```
$weekend = array('Zaterdag', 'Zondag');
var_dump($weekend);
```

Dit geeft:

```
array(2) { [0]=> string(8) "Zaterdag" [1]=> string(6) "Zondag" }
```

U kunt dit, net als bij `print_r()` weer verhelderen met het `<pre>` element:

```
echo '<pre>';
var_dump($weekend);
echo '</pre>';
```

geeft:

```
array(2) {
  [0]=>
    string(8) "Zaterdag"
  [1]=>
    string(6) "Zondag"
}
```

U heeft hier overigens **niet** het **probleem met de extra 1** van `print_r()`, als u `var_dump()` met een `echo` zou combineren. Er is ook geen tweede argument om onderscheid te maken tussen rechtstreekse uitvoer en het retourneren van een string.

U kunt **meerdere variabelen** tegelijk dumpen door ze als argumenten mee te geven:

```
var_dump($voornaam, $achternaam);
```

Wanneer het u alleen om de opbouw van de array gaat dan is `print_r()` overzichtelijker... Zeker voor het tonen van `$_GET`, `$_POST` en `$_SERVER` arrays omdat die vanwege hun aard (HTTP berichten) toch enkel uit (veel) strings bestaan.

U kunt `var_dump()` ook heel goed voor **objecten** gebruiken:

```
$datum = new DateTime('1968-12-06');
echo '<pre>';
var_dump($weekend);
echo '</pre>';
```

U krijgt dan de **properties** te zien:

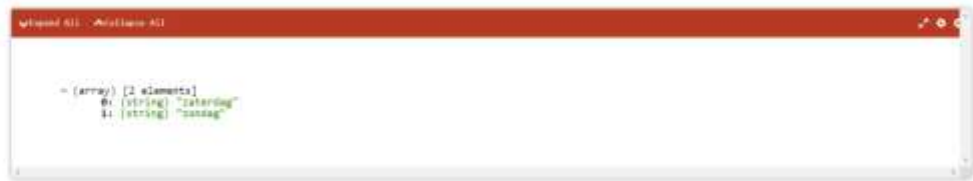
```
object(DateTime)#2 (3) {
  ["date"]=>
    string(19) "1968-12-06 00:00:00"
  ["timezone_type"]=>
    int(3)
  ["timezone"]=>
    string(16) "Europe/Amsterdam"
}
```

Wanneer u de **PHP extensie XDebug** gebruikt dan ziet een `var_dump` er anders uit, bijvoorbeeld zo:

```
C:\wamp\www\demo.php:3:
array (size=4)
  0 => string 'HTML' (length=4)
  1 => string 'CSS' (length=3)
  2 => string 'JavaScript' (length=10)
  3 => string 'PHP' (length=3)
```

U kunt ook de **Chrome extension var_masterpiece** gebruiken voor een in/uitklapbare weergave:

```
var_dump([0] => straggli("zaterdag")[1] => straggli("zondag"))
```



Mogelijk handig bij grotere samengestelde arrays en objecten.

3.9.1 var_export()

Er is nog een derde functie in dit genre, namelijk `var_export()`. Het grote verschil met de andere twee is dat deze functie een variabele op zo'n manier als string teruggeeft dat u er met PHP zo weer mee aan de slag zou kunnen. De weekend array uit de eerdere voorbeelden wordt zo getoond:

```
array (
    'za' => 'zaterdag',
    'zo' => 'zondag',
)
```

Merk op dat dit precies de notatie is die u zou gebruiken wanneer u de array zelf aan zou maken...

3.10 Datums

Datums zijn in PHP geen apart datatype. Het kan een object zijn zoals hierboven, of een string of een integer. De functies `mkdate()` en `strtotime()` werken namelijk met **timestamps**: het aantal seconden dat verstreken is sinds de **unix epoch**, middernacht 1 januari 1970:

```
// maak 1 april 2000 aan
$datum = mktime(0,0,0,4,1,2000);

// of makkelijk met string:
$datum = strtotime('2000-4-1');

// datum is een integer:
var_dump($datum); // geeft int(954540000)
```

Een datum voor de epoch levert een negatieve timestamp op:

```
$datum = strtotime('1945-5-5');
var_dump($datum); // geeft int(-778212000)
```

Let op: U kunt met deze functies **geen datums voor 13 December 1901 en na 19 januari 2038** aanmaken **op 32 bits systemen** vanwege de beperking in grootte die een 32-bit signed integer kan hebben (van -2147483647 tot +2147483647). Dit laatste wordt ook wel het Jaar 2038 probleem genoemd (vaak afgekort als y2k38). Met de DateTime class in PHP >= 5.2 kan het gelukkig wel.

Nogmaals de naamgeving variabelen

Veel programmeurs vinden het prettig om het datatype in de naam van de variabele op te nemen. Ze vinden de code hierdoor beter leesbaar en beter onderhoudbaar. Wanneer een variabele bijvoorbeeld een string met daarin een voornaam bevat dan wordt de naam `$sVoornaam` of `$strVoornaam`. De boolean voor getrouwd zou met `$bGetrouwd` of `$blnGetrouwd` worden aangeduid. En als de variabelen **global** zijn, dan wordt er door sommigen zelfs de prefix `g_` voor gezet: `$g_sVoornaam` en `$g_bGetrouwd`.

Andere prefixes:

Numeriek `n`, `nr`

- integer `i`, `int`

- float `f`, `flt`

String `s`, `str`

Boolean `b`, `bln`

Array `a`, `arr`

Object `o`, `obj`

Resource `r`, `rsc`

Deze werkwijze, die *Hungarian Notation* wordt genoemd, stamt uit de begindagen van DOS. Toen introduceerde de Chief Architect van Microsoft, Dr. Charles Simonyi, een systeem voor de naamgeving van variabelen (*naming convention*) waarbij een voorvoegsel (*prefix*) aan de variabelenaam werd toegevoegd om aan te geven van welk datatype de desbetreffende variabele is.

Dit systeem werd heel populair binnen Microsoft en raakte bekend onder de naam Hungarian notation omdat Simonyi oorspronkelijk uit Hongarije kwam en de variabelenamen er door de voorvoegsels ook wat uitheems uit gingen zien.

We zullen dit systeem tijdens de cursus **niet** gebruiken. In een untyped language als PHP is het nut beperkt en vaak worden de namen er niet leesbaarder op. Maar als u het prettig vindt, ga dan vooral uw gang!

Oefening 3.1

Ontwerp een PHP script met de naam `variabelen.php` dat de volgende variabelen op een duidelijke manier declareert en vult met de waarde die erachter vermeld staat.

Betekenis variabele (verzin zelf een geschikte naam)	Inhoud variabele
Straatnaam waar het huis staat	Zwaluwlaan
Huisnummer van deze woning	39a
Vrije verkoopwaarde van het huis	€ 390.000,00
Sofinummer kostwinner	169067210
Geboren op	18 mei 1970

Laat het script vervolgens de namen en bijbehorende waarden van alle variabelen onder elkaar in de browser tonen.

3.11 Variabelen formatteren

Wanneer u datums, getallen en valuta in variabelen heeft gezet dan heeft u vaak ook speciale wensen over hoe u deze precies wilt weergeven, oftewel wil **formatteren**. Een datum wilt u soms kort noteren (01/04/2000) of juist helemaal uitschrijven (1 april 2000) .

Voor het opmaken van datums kunt u dan de `date()` functie gebruiken die u al eerder zag maar dan met een zelf aangemaakte datum als tweede argument, bijvoorbeeld:

```
$datum = strtotime('2000-4-1');  
echo date('l, j F Y', $datum);
```

De uitvoer is dan:

Saturday, 1 April 2000

Dus in het **Engels**, wat de **standaard locale** voor PHP is (ook al draait het op een Nederlandstalige Windows machine).

Maar waarschijnlijk wilt u uw datums in het Nederlands weergeven, net als u een miljoen wilt weergeven als 1.000.000,00 in plaats van het Engelse 1,000,000.00. En wilt u er een € symbool voor in plaats van een \$. Allemaal taalafhankelijke instellingen. Om dit voor elkaar te krijgen moet u om te beginnen de locale op Nederlands zetten:

3.11.1 Locale zetten

U kunt de locale wijzigen met de functie `setlocale()`. Deze neemt als eerste argument een constante die aangeeft wat u precies wilt beïnvloeden. Wat u meestal zult willen is alles in 1x regelen met `LC_ALL`. Deze constante beïnvloedt de notatie van datums, valuta, duizendtallen etc. U zou er eventueel ook voor kunnen kiezen om deze apart in te stellen met constanten als `LC_TIME`, `LC_MONETARY` en `LC_NUMERIC`. Zie <http://php.net/manual/en/function.setlocale.php> voor details.

Met het tweede argument geeft u aan wat de locale moet worden. Hoe u de locale precies moet noteren kan per systeem verschillen. Als u een Nederlandse locale op een **Linux** machine wilt instellen lukt dat meestal zo:

```
setlocale(LC_ALL, 'nl_NL'); // charset is ISO-8859-1
setlocale(LC_ALL, 'nl_NL@euro'); // charset is ISO-8859-15
setlocale(LC_ALL, 'nl_NL.UTF-8'); // charset is UTF-8
```

Voor **Windows** wordt meestal een van deze twee gebruikt:

```
setlocale(LC_ALL, 'nld_nld');
// of
setlocale(LC_ALL, 'Dutch_Netherlands');
```

Als u op een Windows machine een lege string opgeeft dan wordt de locale gehaald uit de Land en Taal instelling van het Configuratiescherm:

```
setlocale(LC_ALL, '');
```

Voor portabele code kunt u ook meerdere kommagescheiden waarden opgeven. De eerste die herkend wordt, wordt dan gebruikt. De functie retourneert deze waarde ook zodat u hem in een variabele kunt opslaan die u bijvoorbeeld kunt hergebruiken om een aangepaste locale weer terug te zetten, of om simpelweg te laten zien welke gekozen werd:

```
$locale = setlocale(LC_ALL, 'nl_NL', 'nl_NL.UTF-8', ...,
                    'nld_nld', 'Dutch_Netherlands');
echo $locale; // geeft gekozen locale
```

3.11.2 Datums

Helaas heeft het aanpassen van de locale **geen invloed** op de weergave van `date()` omdat deze *locale unaware* is. Dat geldt helaas ook voor de `date_format()` functie. Voor het gewenste resultaat moet u de functie `strftime()` gebruiken.

3.11.2.1 strftime()

U gebruikt de functie als volgt:

```
setlocale(LC_ALL, 'nl_NL.UTF-8', 'Dutch_Netherlands');
$datum = strtotime('2000-4-1');

// LET OP: Werkt alleen onder Linux!
echo strftime('%A, %e %B %Y', $datum);
```

```
// LET OP: Werkt alleen onder Windows!  
echo strftime('%A, %#d %B %Y', $datum);
```

Dit geeft de datum eindelijk in het Nederlands:

```
zaterdag, 1 april 2000
```

Let op: de speciale karakters in de **formatstring** zijn voor `strftime()` anders dan die voor `date()`. Bovendien werkt `'%e'` dat het nummer van de dag zonder *leading zero* zou moeten geven **onder Windows niet**. U zou dan `'%#d'` moeten gebruiken, wat onder Linux weer niet werkt... Voor meer informatie zie <http://php.net/manual/en/function.strftime.php>.

3.11.3 Getallen

Als u een decimaal getal wilt afdrukken dan zal de locale nu ook van invloed zijn op de decimale punt:

```
setlocale(LC_ALL, 'Dutch_Netherlands');  
$getal = 18.95;  
echo $getal;
```

dan is de uitvoer:

```
18,95
```

Dus met een komma zoals we in Nederland gewend zijn. Als `$getal` echter de waarde `19.00` heeft wordt enkel 19 afgedrukt. En grote bedragen worden niet per duizendtal gescheiden met een punt. Als u toch decimalen achter de komma wil of duizendtallen scheiden dan kunt u de `number_format()` functie gebruiken.

3.11.3.1 `number_format()`

De functie `number_format()` neemt het op te maken getal als verplichte eerste argument en retourneert een formatted string. Het heeft de volgende algemene notatie:

`number_format($number, $decimals, $decimal_point, $thousands_separator)`

U kunt hem met 1, 2 of alle 4 argumenten gebruiken. Voor het aantal decimalen is de standaardwaarde 0, voor de decimale punt is dat, jawel, een punt en de duizendtal separator is een komma. Dat is dus de Engelse manier van noteren, ook al staat de locale op Nederlands. Zet u het aantal decimalen op 2 dan krijgt u dus automatisch een punt, tenzij u met het derde argument aangeeft dat het een komma moet zijn. Maar als u een derde argument gebruikt moet u verplicht ook het vierde opgeven!

Samenvattend:

```
setlocale(LC_ALL, 'Dutch_Netherlands'); // heeft geen invloed op werking  
$getal = 1000000.00;  
echo $getal; // 1000000  
echo number_format($getal); // 1,000,000  
echo number_format($getal, 2); // 1,000,000.00  
echo number_format($getal, 2, ','); // Wrong parameter count warning!  
echo number_format($getal, 2, ',', '.'); // 1.000.000,00
```

Een alternatief voor het gebruik van decimalen is het werken met `printf()` of `sprintf()` dat wel *locale aware* is als u de juiste parameters gebruikt. `printf()` drukt de geformatteerde string gelijk af terwijl `sprintf()` deze retourneert. In onderstaand voorbeeld gebruiken we `sprintf()`:


```
setlocale(LC_ALL, 'Dutch_Netherlands'); // heeft invloed op werking %f
$getal = 1000000.00;

echo sprintf('%F', $getal); // 1000000.000000
echo sprintf('%f', $getal); // 1000000,000000
echo sprintf('%.2F', $getal); // 1000000.00
echo sprintf('%.2f', $getal); // 1000000,00
```

Het eerste argument is een format string waarin u gewone karakters (hier *directives* genoemd) die gewoon worden doorgegeven kunt combineren met *conversion specifications* die de waarde in het tweede argument formatteren. Ze beginnen altijd met een %-teken en de specifier daarna geeft aan hoe het tweede argument behandeld moet worden. In bovenstaand voorbeeld gaan we uit van f en F die het als een float behandelen. Het getal na de punt geeft in dit geval de precision aan. Maar er zijn heel veel andere mogelijkheden. Zie daarvoor de manual op <https://www.php.net/sprintf>.

Merk op dat %F (met een hoofdletter) **niet locale aware** is. Bovendien kunt u met de `*printf()` functies **geen duizendtallen** scheiden.

Tip: Een ander handig wel *locale aware* alternatief is het gebruik van de **NumberFormatter** class, die we hierna bij bedragen zullen bespreken.

Let op: Bij het gebruik van decimalen vindt in beide gevallen **afrounding** plaats, vergelijkbaar met wat de functie `round()`:

```
setlocale(LC_ALL, 'Dutch_Netherlands');
$getal = 123.456;

echo number_format($getal, 2, ',', '.'); // 123,46
echo sprintf('%.2f', $getal); // 123,46
```

Vaak wilt u getallen met decimalen gebruiken voor het weergeven van bedragen:

3.11.4 Valuta

Om een bedrag weer correct weer te geven wilt u er het *currency symbol* aan mee geven. U kunt dat simpelweg doen door dat er hard voor te zetten:

```
setlocale(LC_ALL, 'Dutch_Netherlands');
$bedrag = 123.45;
echo '€ ' . number_format($bedrag, 2, ',', '.'); // € 123,46 (UTF-8!)
echo '&euro; ' . number_format($bedrag, 2, ',', '.'); // € 123,46
```

Het rechtsreeks inkloppen van een euro symbool (Windows: alt+0128) werkt alleen goed als u UTF-8 gebruikt als charset voor uw webpagina's (zoals zou moeten).

3.11.4.1 money_format()

Voor het weergeven van bedragen bestaat echter ook een speciale functie `money_format()` die weer een heel eigen notatie kent:

```
setlocale(LC_ALL, 'nl_NL.UTF-8');
$bedrag = 123.45;
echo money_format('%i', $bedrag); // EUR 123,46
echo money_format('%2n', $bedrag); // € 123,46
```

Helaas is het zo dat de functie **niet bestaat onder Windows**. Als dat voor u belangrijk is dan zijn er diverse custom functies te vinden in de documentatie op <http://php.net/money-format>. U zou deze als volgt kunnen toepassen:

```
// als de money_format functie niet bestaat (onder Windows)
if (! function_exists('money_format')) {
    // definieer dan een eigen functie met dezelfde naam
    function money_format($format, $number)
    {
        ...
    }
}
// gebruik money_format op de normale manier
```

3.12 Constanten

Constanten worden gebruikt voor data die meerdere malen op een pagina of site worden herhaald en waarvan de waarde, in tegenstelling tot variabelen, niet hoeft te veranderen binnen de pagina of site.

Stel dat u vaak een BTW tarief op uw pagina's gebruikt dan kunt u dat tarief als een constante bovenaan uw pagina declareren:

```
define('BTW_TARIEF', 1.21);
```

Let op: de naam van de constante is een **string** en moet dus **tussen quotes** staan! Veel auteurs weten dit niet en het staat zelfs in sommige boeken fout! Dat het toch goed lijkt te gaan komt doordat het `error_reporting` level blijkbaar laag staat, want dit zou u zien als u hem op maximaal zet (zoals hoort tijdens het ontwikkelen):

```
error_reporting(E_ALL);
...
// FOUT: BTW_TARIEF moet een string zijn
define(BTW_TARIEF, 1.21);
```

Dit geeft:

Notice: Use of undefined constant BTW_TARIEF – assumed 'BTW_TARIEF' in `test.php`

Het is gebruikelijk de naam van constanten in **hoofdletters** te typen en woorden in de naam van elkaar te scheiden met *underscores* (`_`). Deze notatie wordt ook wel met **ALL_CAPS** aangeduid. Hiermee zijn ze makkelijker te onderscheiden van variabelen.

Let op: constanten beginnen dus nooit met een `$`-teken!

Het uitlezen gaat als volgt:

```
echo BTW_TARIEF;
```

en geeft:

```
1.21
```

Een alternatieve (en iets omslachtiger) manier is om de constante uit te lezen met de functie `constant()`:

```
echo constant('BTW_TARIEF');
```

3.12.1 Constanten zijn (standaard) hoofdlettergevoelig

In PHP zijn constanten **standaard**, net als variabelen, **hoofdletter gevoelig**:

```
echo btw_tarief; // gaat mis
```

geeft:

Notice: Use of undefined constant btw_tarief - assumed 'btw_tarief' in **test.php**
btw_tarief

Als u wilt toestaan dat constanten ook in lowercase aangeroepen mogen worden dan kunt u bij het aanmaken gebruik maken van een derde boolean argument `$case_insensitive`. Standaard heeft deze de waarde `false` waardoor de constanten dus hoofdlettergevoelig zijn. Als u als waarde `true` opgeeft dan maakt de case niet uit:

```
define('BTW_TARIEF', 1.21, true);  
echo btw_tarief; // gaat goed
```

3.12.2 Waarom constanten gebruiken?

De belangrijkste reden om constanten te gebruiken in plaats van variabelen is dat u de waarde van een constante niet per ongeluk kunt wijzigen in de loop van uw programma. Als u toch zou proberen een andere waarde toe te wijzen dan negeert PHP deze nieuwe waarde:

```
define('BTW_TARIEF', 1.21);  
define('BTW_TARIEF', 1.19);  
echo BTW_TARIEF; // blijft 1.21
```

geeft:

Notice: Constant BTW_TARIEF already defined in **test.php**
1.21

Dit geldt natuurlijk alleen voor de duur van het programma (*at runtime*). U kunt de waarde uiteraard wel in uw code aanpassen (*at designtime*) om het script daarna vervolgens met de vernieuwde waarde uit te laten voeren!

Uw code wordt niet alleen beter leesbaar door het gebruik van constanten maar ook veel beter onderhoudbaar. Stel u voor wanneer in uw code elke prijs inclusief BTW steeds met 1.21 vermenigvuldigd zou worden. Weet u dan straks nog waar dit zogenaamde **magic number** voor staat? En als u dat nog wel zou weten, hoe handig is het dan om dit getal aan te passen als het BTW tarief verandert? Met het gebruik van een constante weet u echter precies wat er waarom in de code gebeurt en hoeft u, wanneer het BTW tarief op een gegeven moment verandert, dit maar op één plek aan te passen!

3.12.3 Configuratiebestanden

Constanten worden dan ook veel gebruikt in **configuratiebestanden**. Hierin worden dan allerlei constanten opgenomen met daarin waarden die door de applicatie heen gebruikt worden. Denk aan zaken als bedrijfsnaam, bestandsnaam voor het logo, teksten voor foutmeldingen etc. Kortom, alles waarvan het makkelijk is om het op één plek snel te kunnen wijzigen. Veel open source software maakt handig gebruik van dergelijke configuratiebestanden.

We komen er in de vervolgcursus uitgebreid op terug maar u kunt in PHP eenvoudig een ander bestand insluiten met het `include` construct. Als u uw constanten in een bestand zet dat u bijvoorbeeld `config.php` noemt:

```
<?php
// dit is config.php
define('BTW_TARIEF', 1.21);
...
?>
```

Dan kunt u het bestand met include bovenaan al uw pagina's insluiten en daarin de constanten gebruiken:

```
<?php
// sluit config.php in
include 'config.php';

// gebruik de daarin gedefinieerde constanten
echo BTW_TARIEF;
?>
```

Als het BTW tarief dan mocht wijzigen dan hoeft u het alleen in config.php aan te passen. Hierna wordt overal waar u de constante BTW_TARIEF gebruikt, automatisch het nieuwe tarief gebruikt.

Oefening	Definieer een constante <code>COMPANY_NAME</code> en neem deze op in de footer van uw script:
	Hieronder ziet u de constante <code>COMPANY_NAME</code>: © 2011-2013 CompuFirm
Verdieping	Gebruik voor het tweede jaartal de <code>date()</code> functie, zodat altijd het huidige jaar getoond wordt. U kunt dit eventueel ook met behulp van een constante <code>THIS_YEAR</code> doen.
Verdieping	Voeg aan het script de constante BTW tarief toe en toon een \$bedrag van 100 inclusief en exclusief 21% BTW:
	€ 100,00 (excl. BTW) € 121,00 (incl. BTW) <i>Maak voor de opmaak gebruik van de <code>number_format()</code> functie.</i>

3.12.4 PHP Constanten

PHP kent ook een aantal voorgedefinieerde constanten. `E_ALL` in de `error_reporting()` functie die we eerder in dit hoofdstuk tegenkwamen is er een voorbeeld van. Andere zijn bijvoorbeeld `TRUE` en `FALSE`. Constanten als deze hoeft u niet eerst te declareren maar kunt u zo gebruiken. We zullen ze verder bespreken wanneer we ze tegenkomen.

Mocht u nieuwsgierig zijn naar het totaal overzicht, u krijgt het met deze code:

```
echo '<pre>';
print_r(get_defined_constants());
echo '</pre>';
```

Het `<pre>` element is bedoeld om de uitvoer leesbaar te maken in de browser. U zult deze notatie nog vaker tegenkomen:

```
Array
(
    [E_ERROR] => 1
    [E_RECOVERABLE_ERROR] => 4096
    [E_WARNING] => 2
    [E_PARSE] => 4
    [E_NOTICE] => 8
    [E_STRICT] => 2048
    [E_DEPRECATED] => 8192
    [E_CORE_ERROR] => 16
    [E_CORE_WARNING] => 32
    [E_COMPILE_ERROR] => 64
    [E_COMPILE_WARNING] => 128
    [E_USER_ERROR] => 256
    [E_USER_WARNING] => 512
    [E_USER_NOTICE] => 1024
    [E_USER_DEPRECATED] => 16384
    [E_ALL] => 32767
    [DEBUG_BACKTRACE_PROVIDE_OBJECT] => 1
    [DEBUG_BACKTRACE_IGNORE_ARGS] => 2
)
```

3.12.5 Compile-time constants

Verwarrend genoeg heeft PHP ook een aantal **Compile-time constants** (ook wel **Magic constants** genoemd). Deze hebben geen vaste waarde vooraf maar worden bij de compileer fase die vooraf gaat aan het uitvoeren van uw script gezet (en zijn dan voor de duur van het script constant). Dergelijke constanten zijn omsloten door **2 underscores** aan beide kanten. Denk bijvoorbeeld aan `__DIR__` waarmee u het pad van de huidige directory waarin het script draait krijgt (vanaf PHP 5.3):

```
echo __DIR__; // bijv C:\laragon\www\compufirm
```

3.13 Omgevingsvariabelen

Behalve de door ons zelf bedachte (*user defined*) variabelen zijn er ook een aantal variabelen automatisch beschikbaar. Deze variabelen worden **omgevingsvariabelen** of **servervariabelen** genoemd. Bij elk request worden ze 'gezet', waarna we ze in onze scripts kunnen gebruiken.

Een deel van de variabelen bestaat uit gegevens over de client en het request, het andere deel bestaat uit gegevens over de server. Zo kunt u bijvoorbeeld het merk en de versie van de browser van de eindgebruiker uitlezen:

```
echo $_SERVER['HTTP_USER_AGENT'];
```

Wanneer een pagina met deze code wordt opgevraagd door een gebruiker met Internet Explorer 10 op Windows 7 dan is dit het resultaat:

```
Mozilla/5.0 (compatible; MSIE 10.0; Windows NT 6.1; WOW64; Trident/6.0)
```

Zou diezelfde gebruiker de pagina opvragen in een andere browser, bijvoorbeeld Firefox 23 dan zou de uitvoer er zo uit kunnen zien:

```
Mozilla/5.0 (Windows NT 6.1; WOW64; rv:23.0) Gecko/20100101 Firefox/23.0
```

Maar ook het merk en de versie van de server zijn op deze manier zichtbaar te maken:

```
echo $_SERVER['SERVER_SOFTWARE'];
```

Als de PHP applicatie op het WIMP platform draait (Windows, IIS, MySQL, PHP) dan zou dit zoiets op kunnen leveren:

Microsoft-IIS/5.1

Zou de applicatie echter op het WAMP platform draaien (Windows, Apache, MySQL, PHP) dan had de uitvoer zo kunnen zijn:

Apache/2.4.2 (Win32)

Merk op dat in het eerste voorbeeld ('HTTP_USER_AGENT') informatie wordt gegeven over de software van de **client**, in het tweede voorbeeld ('SERVER_SOFTWARE') over de software op de **server**. In beide gevallen staat er echter `$_SERVER` voor!

3.13.1 Superglobals

`$_SERVER` is een zogenaamde **superglobal**. Dat wil zeggen dat PHP deze variabele automatisch aanmaakt en dat deze variabele overal in uw script beschikbaar is.

Er zijn meer superglobals in PHP:

Superglobal	Omschrijving
<code>\$_COOKIE</code>	Bevat cookie informatie
<code>\$_ENV</code>	Bevat omgevingsvariabelen (waarbij de inhoud sterk afhankelijk is van het OS)
<code>\$_FILES</code>	Bevat geupload(e) bestand(en).
<code>\$_GET</code>	Bevat GET request data (via hyperlink of form).
<code>\$_POST</code>	Bevat POST request data (via form).
<code>\$_REQUEST</code>	Bevat combinatie van GET, POST en COOKIE data.
<code>\$_SERVER</code>	Bevat CGI variabelen waarin informatie uit het HTTP Request en de HTTP Response.
<code>\$_SESSION</code>	Bevat sessie data (per gebruiker).

In het volgende hoofdstuk bespreken we `$_GET`, `$_POST` en `$_REQUEST`. In de vervolg cursus komen daar `$_COOKIE`, `$_SESSION` en `$_FILES` bij.

De superglobals zijn **associatieve arrays**. Wat u voor nu in ieder geval al moet onthouden is dat de notatie van superglobals nauw luistert:

- Na het `$`-teken komt een underscore (`_`)
- De naam van de superglobal bestaat uit HOOFDLETTERS
- Hierna komt een paar rechte haken (`[]`)
- Hier binnen staat de naam van de index, ook deze is in HOOFDLETTERS.
- De naam van de index is een string en deze staat dus tussen (enkele of dubbele) quotes

Het is niet per se nodig, maar wel handig om superglobals eerst in gewone variabelen op te slaan:

```
$browser = $_SERVER['HTTP_USER_AGENT'];
```

Vervolgens gebruikt u gewoon `$browser`. Dit maakt uw code niet alleen leesbaarder maar is ook een goede voorbereiding op eventuele controles en bewerkingen die u later uit veiligheidsoverwegingen wilt toevoegen...

3.13.2 Veiligheid

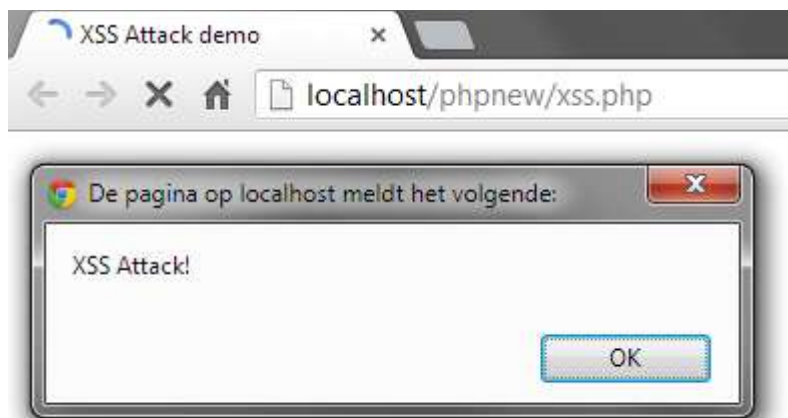
Dit is de eerste keer dat u gegevens naar de pagina schrijft die u zelf niet hebt geïnitieerd. Dit is een potentieel veiligheidsrisico! Het deel van de server variabelen dat door de browser in het request wordt meegezonden zijn door kwaadwillenden aan te passen. Zo kunnen ze mogelijk JavaScript code in uw pagina's injecteren. Dergelijke aanvallen worden **Cross Site Scripting (XSS) Attacks** genoemd en deze kunnen tot allerlei problemen leiden. Het is een beetje vroeg om hier in

deze basiscursus uitgebreid op in te gaan, maar het is heel belangrijk dat u zich nu al realiseert dat u constant moet denken aan veiligheid.

Stel dat een kwaadwillende gebruiker de waarde van `$_SERVER['HTTP_USER_AGENT']` had vervangen door een stuk JavaScript dan had de variabele `$browser` deze ook bevat. Dat is vergelijkbaar met dit voorbeeld:

```
$userinput = "<script>alert('XSS Attack!')</script>";
echo $userinput;
```

Als u dit zou uitvoeren dan wordt de code door de browser als JavaScript geïnterpreteerd en uitgevoerd:



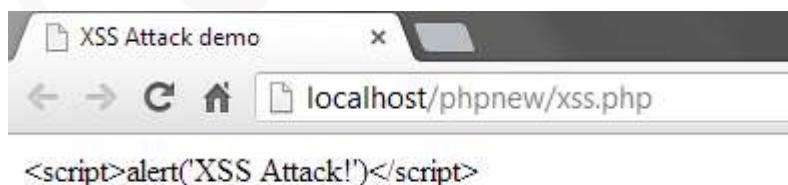
U kunt de meeste problemen vrij eenvoudig voorkomen door de **probleemkarakters** te **escapen**. U doet dat met de ingebouwde functie `htmlspecialchars()`.

```
$userinput = "<script>alert('XSS Attack!')</script>";
echo htmlspecialchars($userinput);
```

Nu worden de `<` en `>` tekens vervangen door hun HTML entiteiten `<` en `>`; waardoor ze niet langer als HTML instructies door de browser worden geïnterpreteerd. Uw uitvoer ziet er dan zo uit:

```
&lt;script&gt;alert('XSS Attack!')&lt;/script&gt;
```

Dit wordt vervolgens in de browser gewoon als tekst weergegeven:



In de **PHP Vervolg** cursus gaan we veel uitgebreider in op deze en andere security issues!

Het injecteren van de HTTP Requestheaders is eenvoudig te realiseren met de Firebug Addon **Tamper Data** (<https://addons.mozilla.org/nl/firefox/addon/tamper-data/>).

U leest meer over XSS op:

- https://en.wikipedia.org/wiki/Cross-site_scripting
- <https://www.owasp.org/index.php/XSS>
- <https://excess-xss.com/>

Op <http://www.thegeekstuff.com/2012/02/xss-attack-examples/> staan enkele PHP voorbeelden uitgewerkt.

Op [https://www.owasp.org/index.php/XSS_\(Cross_Site_Scripting\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet) leest u alles over het voorkomen van XSS!

3.13.3 Overzicht

Een volledig overzicht van beschikbare omgevingsvariabelen vindt u op de PHP site:

<http://php.net/manual/en/reserved.variables.server.php>.

Een snelle manier om ze in te zien is door `phpinfo()` te gebruiken. Zoek vervolgens het kopje **PHP Variables**, daaronder ziet u wat er zoal aan omgevingsvariabelen gezet is voor het betreffende script.

PHP Variables	
Variable	Value
<code>PHP_SELF</code>	/php/test.php
<code>\$_SERVER["HTTP_HOST"]</code>	localhost
<code>\$_SERVER["HTTP_USER_AGENT"]</code>	Mozilla/5.0 (Windows; U; Windows NT 6.0; nl-NL; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5 (.NET CLR 3.5.30729)
<code>\$_SERVER["HTTP_ACCEPT"]</code>	text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
<code>\$_SERVER["HTTP_ACCEPT_LANGUAGE"]</code>	nl
<code>\$_SERVER["HTTP_ACCEPT_ENCODING"]</code>	gzip, deflate
<code>\$_SERVER["HTTP_ACCEPT_CHARSET"]</code>	ISO-8859-1,utf-8;q=0.7,*;q=0.7

U kunt deze simpelweg kopiëren en nadat u er een `$`-teken voorgezet hebt kunt u ze meteen gebruiken in uw scripts...

U kunt het overzicht voor uw server natuurlijk ook weergeven met `print_r()` of `var_dump()`:

```
echo '<pre>';
print_r($_SERVER);
echo '</pre>';
```

Maar denk er dan aan dat u dan ook weer gevoelig bent voor XSS! Dat is natuurlijk meestal geen echt probleem omdat deze functie bedoeld is voor debugging en eindgebruikers er in principe nooit mee te maken krijgen. Maar ook voor u zelf is het beter om het zo te doen:

3.13.3.1 Veilig gebruik van `print_r()` en `var_dump()`

U kunt `print_r()` makkelijk beveiligen tegen XSS injecties door de tweede, optionele, return parameter op `true` te zetten. De functie retourneert dan de uitvoer in plaats van hem meteen te printen:

```
echo '<pre>';
echo htmlspecialchars(print_r($_SERVER, true));
echo '</pre>';
```

Dan krijgt u eventueel ingevoerde HTML/JavaScript code in de browser te zien in plaats van dat deze wordt uitgevoerd! Dit is ook heel nuttig wanneer er alleen opening tags gebruikt zouden zijn. In de traditionele manier van tonen had u dan niets gezien en wellicht gedacht dat er geen waarde was...

Of de opvolgende waarden werden dan opeens vet of cursief getoond. Merk op dat wanneer u in de broncode van de webpagina kijkt u de code wel ziet staan!

Omdat `var_dump()` niet als een string kan retourneren, is het lastiger om de uitvoer te escaperen. Het kan echter wel door **tijdelijk** een **buffer** te gebruiken:

```
echo '<pre>';
ob_start();
var_dump($_SERVER);
$buffer = ob_get_clean();
echo htmlspecialchars($buffer);
echo '</pre>';
```

Wanneer u later in de cursus zelf functies gaat schrijven dan kunt u deze twee fragmenten mooi in een eigen `dump()` functie opnemen!

3.13.4 Beschikbaarheid

Let op: de items in `$_SERVER` zijn niet op elke webserver hetzelfde! Dat komt omdat het de taak van de webserver is om de waarden in de array te zetten, niet die van PHP. Dit treft dus ook andere Server-side scripting languages. In de praktijk komt het er op neer dat een waarde die Apache beschikbaar stelt niet per definitie ook beschikbaar is in Internet Information Server (IIS).

Zie voor een goed overzicht van de beschikbare waarden voor beide webrowsers:

<http://koivi.com/apache-iis-php-server-array.php>

Oefening 3.2	<i>Bestudeer eerst de output van <code>phpinfo()</code> door de opdracht in een apart bestand te zetten dat u <code>phpinfo.php</code> noemt. Zoek op hoe u het IP nummer (adres) van de cursusPC waar u achter zit kunt benaderen en toon dit in een pagina <code>ipnummer.php</code>.</i>

3.14 In de pagina schrijven

U weet nu hoe u variabelen en constanten kunt declareren en initialiseren, maar wat kunt u er nu mee? Een eerste stap is het gebruiken van de variabelen om hun waarde in de pagina te schrijven. Straks zullen we zien hoe u variabelen kunt gebruiken in expressies en zo berekeningen kunt uitvoeren en hoe u ze in controlestructuren gebruikt om het verloop van uw programma's te beïnvloeden.

In uw eerste script heeft u al gebruik gemaakt van één van de manieren om in een pagina te schrijven:

```
<?php echo date('d M Y - H i s'); ?>
```

`date()` is een speciale functie in PHP die de huidige datum en tijd op de server retourneert. En met de opdracht `echo` liet u die waarde in de pagina wegschrijven. U kunt op dezelfde manier de waarde van een variabele wegschrijven:

```
<?php
$tekst = 'een paar woorden tekst';
?>
...
<p>In $tekst staat: <?php echo $tekst; ?></p>
```

Wat u in feite doet is schrijven in de HTTP Response stream. Op deze manier output naar de browser sturen is één van de meest fundamentele aspecten van het ontwikkelen in PHP.

U kunt zo niet alleen de waarde van een variabele wegschrijven maar ook stukken tekst of HTML of een combinatie van dit alles.

3.14.1 Echo vs. Print

In PHP heeft u meerdere manieren om string output weg te schrijven. Behalve het eerder genoemde `echo` kunt u ook `print` gebruiken:

```
echo 'tekst ...';  
// dit doet precies hetzelfde:  
print 'tekst ...';
```

Dit zijn de twee bekendste manieren. Het zijn beide geen functies maar **language constructs**. Dat betekent dat u ze, zoals hierboven, beiden **zonder ronde haken** mag gebruiken. Het grote verschil tussen de twee is dat `print` altijd de integer 1 retourneert om aan te geven dat het printen gelukt is en dat `echo` altijd maar een argument accepteert.

Omdat `print` een waarde retourneert kunt u `print` gebruiken in expressies als hieronder (zie voor de syntax **3.12.6**):

```
// LET OP: dit kan alleen met print:  
$x > 0 ? print 'positief' : print 'negatief';
```

Wanneer u hier `echo` op dezelfde zou proberen te gebruiken zou dat een foutmelding opleveren! Maar dit mag natuurlijk wel:

```
echo $x > 0 ? 'positief' : 'negatief';
```

En leest waarschijnlijk ook prettiger...

`echo` heeft in tegenstelling tot `print` de mogelijkheid om **meerdere strings** weg te schrijven door ze met een **komma** te scheiden:

```
// dit kan alleen met echo:  
echo 'tekst ...', 'meer tekst ...';
```

Dit kan soms best nuttig zijn en is ook weer marginaal sneller dan **concatenatie** of **substitutie** (zie verderop in dit hoofdstuk).

De meeste programmeurs gebruiken `echo`. En wij zullen dat ook consequent doen in deze cursus. Het zijn vaak C programmeurs die voor `print` kiezen. Zie ook hieronder:

3.14.2 printf() en sprintf()

`print` kent daarentegen een aantal varianten die u kunt gebruiken om strings te combineren met variabelen en deze tegelijkertijd te **formatteren**, zoals het rechteekts uit de programmeertaal C afkomstige `printf()` dat met zogenaamde **conversion specifiers** (zeg maar placeholders) werkt:

```
$aantal = 2;  
$prijs = 50;  
  
printf('Deze %d artikelen kosten &euro; %.2f', $aantal, $prijs);
```

Dit geeft:

```
Deze 2 artikelen kosten € 50.00
```

Zoals u ziet geeft u met het % karakter aan dat een specificator begint. Met de letter d geeft u aan dat het om een *digit* gaat en met de letter f dat het om een *floating point value* gaat. Met de notatie . 2 geeft u aan dat u 2 decimalen achter de komma wilt tonen. Dit is slechts een eenvoudig voorbeeld, maar u kunt zich vast voorstellen dat er veel meer mogelijk is! Veel beginners vinden dit soort notaties echter vrij lastig...

Er is ook een `sprintf()` functie die niet direct print maar alleen de geformatteerde string retourneert:

```
$aantal = 2;
$prijs = 50;

echo sprintf('Deze %d artikelen kosten &euro; %.2f', $aantal, $prijs);
```

Merk op dat u de geformatteerde string dan nog wel zelf moet printen!

Zie <https://www.php.net/printf> en <https://www.php.net/sprintf> voor meer informatie.

3.14.3 Linebreaks in string

U mag `echo` en `print` statements over meerdere regels 'uitsmeren', de puntkomma geeft immers het einde van het statement aan:

```
echo 'Deze tekst bestaat uit
meerdere regels. De enters worden
in de uitvoer opgenomen en zijn
zichtbaar in de HTML broncode...
(maar in de weergave van de browser merkt u daar niets van...)' ;
```

Bij het opstellen van SQL statements later in de cursus zult u hier veel plezier van hebben.

Omdat `echo` het meest gebruikt wordt, zullen wij dat tijdens deze cursus ook doen.

3.14.4 Switchen tussen HTML en PHP modus

U kunt **zoveel PHP codeblokken** aanmaken **als u maar wilt** en zo **steeds** binnen een script de **PHP** modus en **HTML** modus **afwisselen**. U heeft hierbij **geen performance loss** zoals bijvoorbeeld bij Classic ASP wel het geval zou zijn.

3.14.5 Valide HTML uitvoer

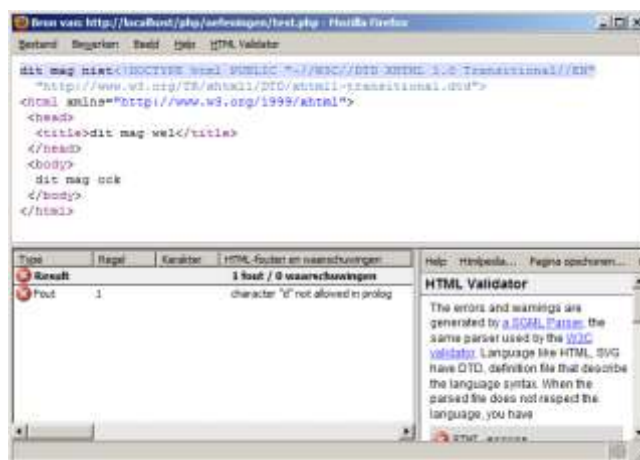
U mag de blokjes in principe overall in uw document opnemen, alleen moet u er wel voor zorgen dat de uitvoer een valide HTML document blijft opleveren! Zo mag u wel iets wegschrijven in de titel, maar u mag bijvoorbeeld geen uitvoer geven voor de DocType Declaration:

```
<?php echo 'dit mag niet'; ?>
<!DOCTYPE html>
<html>
  <head>
    <title><?php echo 'dit mag wel'; ?></title>
  </head>
  <body>
    <?php echo 'dit mag ook'; ?>
  </body>
</html>
```

U kunt een en ander uitstekend controleren met de onmisbare HTML Validator Add-on in FireFox. Die zou bij het opvragen van bovenstaand script dit laten zien:



En wanneer u door zou klikken:



Zo krijgt u meteen inzicht in wat er niet goed gaat in uw pagina. Echt heel handig!

Let op: We zullen in code voorbeelden het nieuwe, compacte HTML5 DOCTYPE gebruiken.

3.14.6 Logica en presentatie scheiden

Het is verstandig om u al zo snel mogelijk aan te wennen uw PHP logica en de HTML presentatie van de gegevens zoveel mogelijk te scheiden. Dit maakt uw code veel beter leesbaar en makkelijker aan te passen. En het maakt de overgang naar **Templates** straks veel makkelijker.

U kunt dat nu al zo veel mogelijk nastreven door eerst bovenaan het script voor te bereiden en dan pas in de HTML pagina tot uitvoer over te gaan, waarbij u het HTML deel zoveel mogelijk als een template ziet waar u met PHP dynamisch gegevens in kan injecteren.

Dus in plaats van:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Voorbeeld</title>
  </head>
  <body>
    <?php
      $voornaam = 'Arno';
      echo "<p>Hoi $voornaam</p>";
    ?>
  </body>
</html>
```

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

Kunt u **beter** zoiets doen:

```
<?php
// hier zoveel mogelijk voorbereiden, maar nog geen uitvoer geven
$voornaam = 'Arno';
?>
<!DOCTYPE html>
<html>
  <head>
    <title>Voorbeeld</title>
  </head>
  <body>
    <p>Hoi <?php echo $voornaam; ?></p>
  </body>
</html>
```

Vanaf PHP 5.4 kunt u dat, ook als de short open tag notatie uitstaat, veilig en compact zo noteren:

```
<?php
$voornaam = 'Arno';
?>
<!DOCTYPE html>
<html>
  <head>
    <title>Voorbeeld</title>
  </head>
  <body>
    <p>Hoi <?=$voornaam?></p>
  </body>
</html>
```

Als de groet ook dynamisch moet zijn (zoals in een latere oefening waar deze tijdsafhankelijk is) dan zet u die ook in een variabele:

```
<?php
$groet = 'Hoi';
$voornaam = 'Arno';
?>
<!DOCTYPE html>
<html>
  <head>
    <title>Voorbeeld</title>
  </head>
  <body>
    <p><?=$groet?> <?=$voornaam?></p>
  </body>
</html>
```

Belangrijk is in ieder geval dat u voorkomt dat er HTML in de variabelen terechtkomt...

Dit is natuurlijk een simpel voorbeeld maar bij complexere scripts werkt deze zogenaamde *separation of concerns* heel prettig. Eerst halen we data op en bereiden die eventueel voor. Daarna geven we het aan de HTML pagina zodat die het kan renderen.

In de HTML beperkt u zich tot de **presentatie**: het wegschrijven van variabelen, eventueel gecombineerd met eenvoudige presentatie logica zoals if/else en loop controlestructuren die we later in dit hoofdstuk zullen bespreken.

Op deze manier is het in een later stadium makkelijker om de logica en de presentatie daadwerkelijk fysiek in aparte bestanden te scheiden. Het HTML deel wordt dan een **template** of **view** genoemd die wordt ingesloten. Een programmeur kan dan aan de PHP code werken en een ontwerper onafhankelijk van hem of haar aan de HTML code.

Een ander groot bijkomend voordeel is dat op deze manier de *syntax highlighting*, *code completion*, *auto indent*, etc. van de editor voor de HTML code blijft werken!

Later gaan we veel uitgebreider op deze zaken in.

3.14.7 String Concatenation

Wanneer u de waarde van een variabele weg wilt schrijven in combinatie met een stuk tekst of HTML dan gebruikt u de punt (.) om de verschillende delen aan elkaar te 'koppelen':

```
<?php
$voornaam = 'Arno';
echo 'Hoi' . $voornaam . '!';
?>
```

De officiële Engelse naam hiervoor is **string concatenation** en de punt wordt in PHP de *concatenation operator* genoemd. In het Nederlands wordt dit soms met 'plakken' of 'rijgen' aangeduid en heeft men het over de plak- dan wel rijg-operator.

Let op: In het voorbeeld staat na 'Hoi ' een extra spatie, anders wordt de voornaam er gelijk achter geplakt!

Veel ontwikkelaars vinden het prettig om spaties om de punten heen te zetten zodat de punten beter opvallen:

```
<?php
$voornaam = 'Arno';
echo 'Hoi ' . $voornaam . '!';
?>
```

Dit heeft geen gevolgen voor de uitvoer. Alleen eventuele spaties in de strings zelf worden zichtbaar. De punten vallen zo inderdaad wat meer op, maar een nadeel is dat het wat meer ruimte kost. In de voorbeelden in dit boek zullen we de concatenatie dan soms ook zonder extra witruimte tonen...

3.14.7.1 Shorthand

Iets wat u in de praktijk vaak tegen zult komen is dat een bestaande string aangevuld moet worden met een nieuw deel. Meestal gebeurt dat conditioneel of in een lus, we zullen daar later in de cursus meerdere voorbeelden van zien. Een eenvoudig(er) voorbeeld is dit:

```
$groet = 'Hoi';
$naam = 'Arno';

$groet = $groet . ' ' . $naam;

echo $groet; // geeft Hoi Arno
```

Hier kent u aan `$groet` eerst de oude waarde van `$groet` toe en plakt er dan het nieuwe deel aan. Maar omdat dit zo vaak voorkomt bestaat er in PHP ook een verkorte notatie voor:

```
$groet = 'Hoi';
$naam = 'Arno';
```

```
$groet .= ' ' . $naam;

echo $groet; // geeft Hoi Arno
```

U plakt er als het ware gelijk het nieuwe deel mee aan `$groet`.

Let er wel op dat u de shorthand operator exact zo noteert. Er mag **geen witruimte** tussen de punt en het `=`-teken staan:

```
// LET OP: dit gaat mis!
$groet .= ' ' . $naam;
```

Dit geeft de volgende melding:

Parse error: syntax error, unexpected '='

3.14.7.2 Echo met komma gescheiden parameters

Een **andere methode** om meerdere strings en/of variabelen te combineren is door gebruik te maken van het feit dat u met één echo statement **meerdere parameters** mee kunt geven, **gescheiden door een komma**:

```
<?php
$voornaam = 'Arno';
// Een echo statement kan meerdere parameters wegschrijven.
// De parameters worden dan door komma's gescheiden:
echo 'Hoi ', $voornaam, '!';
?>
```

Deze notatie is wat minder bekend, maar wel heel snel!

3.14.8 Strings combineren met HTML

Als u bovenstaand voorbeeld in een paragraaf zou willen zetten dan kan dat op verschillende manieren. U zou de HTML tags buiten het PHP codeblok kunnen zetten, zoals in bovenstaand voorbeeld is gebeurd:

```
<p>
<?php
$voornaam = 'Arno';
echo 'Hoi ' . $voornaam . '!';
?>
</p>
```

Let op: U mag de HTML nooit rechtstreeks in een PHP codeblok plaatsen! Dit gaat gelijk mis:

```
<?php
// Plaats nooit rechtstreeks HTML in een PHP blok!
$voornaam = 'Arno';
<p> echo 'Hoi ' . $voornaam . '!'; </p>
?>
```

Dit geeft meteen een foutmelding als deze:

Parse error: syntax error, unexpected '<' in **test.php** on line 4

Maar u zou HTML wel met `echo` in een string weg mogen schrijven:

```
<?php
// HTML als string wegschrijven mag wel!
$voornaam = 'Arno';
echo "<p>Hoi ".$voornaam."</p>";
?>
```

Doordat u nu de opmaak in een keer mee wegschrijft is het makkelijker te zien wat er gebeurt. Bovendien kunt u de uitvoer makkelijk even uitschakelen door het `echo` statement uit te commentariëren.

Een (groot) **nadeel** is wel dat u op deze manier PHP en HTML in PHP codeblokken aan het combineren bent. Dit maakt aanpassingen lastig en een klein foutje kan er voor zorgen dat gelijk niets meer werkt. Bovendien kan de HTML meestal niet meer als HTML door editor bewerkt worden. Denk aan zaken als automatisch afsluiten of code aanvullen. We zullen zo een manier zien die het makkelijker maakt om PHP en HTML samen te laten werken.

3.14.9 Escapen van double quotes

Waar u ook aan moet denken is dat wanneer u op deze wijze te werk gaat het gebruik van attributen speciale aandacht vergt. In HTML behoren de waarden van attributen tussen dubbele aanhalingstekens te staan. In PHP wordt het begin en einde van een string echter ook met een double quote gemarkeerd. Bij een paragraaf met het attribuut `class="greeting"` zou de eerste double quote als het einde van de string worden gezien.

De oplossing bestaat eruit dat u de double quotes moet *escapen*. Of anders gezegd u moet er voor zorg dragen dat de double quote niet meer gezien wordt als een speciaal markeringsteken. In PHP doet u dat door het teken vooraf te laten gaan door een *backward slash* (`\`):

```
$voornaam = 'Arno';
echo "<p class=\"greeting\">Hoi ".$voornaam."</p>";
```

Let op: Als u vreemde resultaten krijgt zonder expliciete foutmeldingen dan ligt dat vaak aan het feit dat in de HTML uitvoer een double quote wel geopend is maar niet afgesloten. Kijk dan ter controle meteen in de HTML broncode in de browser (Menu beeld → Bron). Dit is één van de handigste hulpmiddelen bij het *debuggen* (fouten opsporen en verhelpen).

3.14.10 Double vs. single quotes

Bij bespreking van bovenstaand voorbeeld heeft u misschien gedacht dat het handiger zou zijn om de string niet door double maar door single quotes te laten omsluiten. Nu, dat mag!

```
$voornaam = 'Arno';
echo '<p class="greeting">Hoi '.$voornaam.'</p>';
```

U hoeft nu de double quotes niet te escapen. Maar als er een single quote in voorkomt dan moet het alsnog:

```
echo 'Arno\'s tweede voorbeeld';
```

Er is echter meer aan de hand. Binnen een string die met double quotes wordt omsloten, zal PHP de waarden van ingesloten variabelen **substitueren (interpoleren)**, oftewel vervangen door hun waarde:

```
$voornaam = 'Arno';
echo "Hoi $voornaam";
```

geeft:


```
Hoi Arno
```

Zou u echter single quotes gebruiken:

```
$voornaam = 'Arno';
echo 'Hoi $voornaam';
```

dan geeft dit:

```
Hoi $voornaam.
```

Hoewel het onderstaande wel werkt, is het echter onzinnig om zoiets te doen:

```
// let op: onzinnig!
echo "$voornaam";
```

U doet dat natuurlijk gewoon zo:

```
echo $voornaam;
```

3.14.10.1 Double quotes

Bij een string met double quotes vindt er dus meer *preprocessing* plaats:

- Variabelenamen (beginnend met een dollarteken \$) worden vervangen door de string representatie van hun waarden. Is de variabele nog niet geïnitieerd dan wordt als waarde een lege string gebruikt.
- Bepaalde karakterreeksen (beginnend met een backslash \) worden vervangen door speciale karakters.

De karakter reeksen die worden vervangen zijn deze:

sequentie	wordt vervangen door
\n	Een newline character
\r	Een carriage-return karakter
\t	Een tab karakter
\\$	Een dollarteken
\"	Een dubbele quote
\\	Een backslash

De eerste drie maken het mogelijk om extra witruimte te creëren (zichtbaar in de HTML broncode!) en de laatste drie maken het mogelijk om de karakters die een speciale betekenis hebben in een string met double quotes toch gewoon te kunnen weergeven.

```
$min = 1;
$max = 10;
echo "Het minimum is $min.\n Het maximum is $max.";
```

geeft in de browser:

```
Het minimum is 1. Het maximum is 10.
```

Maar in de HTML *broncode*:

```
Het minimum is 1.  
Het maximum is 10.
```

NB.: Om in de browser de volgende regel ook echt op de volgende regel te tonen moet u ook nog een `<br>` element gebruiken. We komen hier later nog op terug.

U kunt een **array** zoals `$_SERVER["REMOTE_ADDR"]` of `$_SERVER['REMOTE_ADDR']` **niet** zonder meer in de string **plaatsen**. In beide gevallen zou dat een foutmelding opleveren. Als u de array echter **tussen accolades** plaatst, dan gaat het wel goed:

```
echo "<p>IP-adres: {$_SERVER['REMOTE_ADDR']}</p>";
```

Wat overigens, verwarrend genoeg, in dit soort gevallen ook mag, is het **weglaten** van de **quotes** voor de index:

```
echo "<p>IP-adres: $_SERVER[REMOTE_ADDR]</p>";
```

Maar bij concatenatie mag u dit dus juist **nooit** doen!

Tip: sommige developers vinden het prettig om **gewone** variabelen in double quoted strings ook tussen **accolades** te zetten, ook al is dat strict gezien niet nodig. De accolades fungeren dan als een soort **placeholders** en maken het **duidelijker** dat er variabele waarden in de string worden geplaatst:

```
echo "<p>Hoi {$naam}!</p>";
```

3.14.10.2 Single quotes

In een string die wordt omsloten met single quotes gebeurt veel minder. Alles wordt in principe letterlijk weergegeven:

```
$min = 1;  
$max = 10;  
echo "Het minimum is $min.\n Het maximum is $max.";
```

wordt in de browser getoond als:

```
Het minimum is $min.\n Het maximum is $max.
```

U kunt zonder problemen gebruik maken van double quotes in een single quoted string. Wilt u echter een single quote of een backslash weergeven dan moet u deze met een backslash escaperen:

```
echo 'Arno\'s voorbeeld';
```

3.14.10.3 Wanneer welk type quote?

Het is niet altijd even makkelijk om te bepalen wanneer u welk type quote moet gebruiken. Voor tekst die gewoon letterlijk moet worden weergegeven is de single quote het meest geschikt omdat er dan geen preprocessing hoeft plaats te vinden.

De single quote is ook handig voor het wegschrijven van HTML waarin veel double quotes voorkomen. U hoeft dan niet steeds alle double quotes met backslashes te escaperen:

```
echo '<p align="left">Het maximum is 10</p>';
```

Als er in de string echter de waarde van een **variabele** moet worden geplaatst dan moet u dat **met concatenatie operators** doen:

```
echo '<p align="left">Het maximum is ' . $max . '</p>';
```

Veel programmeurs vinden dit echter ook **makkelijker lezen**, zeker als er gebruik gemaakt wordt van een editor die color syntax highlighting toepast. Het is dan namelijk **erg duidelijk** waar variabelen worden gebruikt. Bovendien zullen zij argumenteren dat code die op deze manier geschreven is sneller wordt uitgevoerd.

Interpolatie is namelijk (wat) **trager dan concatenatie**: <http://phptect.com/php/quotes.html> en het definitieve artikel over single vs. double quotes: <http://www.scriptorama.nl/algemeen/quotes-battle>.

Voor de verwerkingssnelheid van een string die geen variabelen bevat maakt het echter geen verschil of ze tussen single of double quotes staat.... Maar een vuistregel zou kunnen zijn dat u dergelijke statische strings juist altijd tussen single quotes zet. Dit is wat wij ook in de voorbeelden in dit boek doen...

Een mogelijk nadeel is dat het gebruik van *newline characters* om uw HTML opmaak netjes te houden wat lastiger wordt:

```
echo '<p>Het maximum is ' . $max . '</p>' . "\n";
```

Maar u zou dit natuurlijk ook zo mogen noteren:

```
echo '<p>Het maximum is ' . $max . "</p>\n";
```

Of zo:

```
// definieer op centrale plek, bijv. in config.php
define('N', "\n");

// en gebruik vervolgens zo:
echo '<p>Het maximum is ' . $max . '</p>' . N;
```

Naast het gebruik van single of double quotes is er nog een derde manier in PHP om een string te specificeren en dat is door de zogenaamde *HereDoc syntax* te gebruiken:

3.14.10.4 HereDoc syntax

Een heel handige manier om HTML met veel double quotes weg te schrijven waarbij ook nog eens substitutie/interpolatie van variabelen plaatsvindt én newline characters gehandhaafd worden, is met de *HereDoc syntax*:

```
<?php
$html =<<< END
<p class="normal">Het maximum is $max</p>
<p class="normal">Nog een alinea</p>
<p>etc..</p>
END;
echo $html;
?>
```

In bovenstaand voorbeeld wordt een variabele `$html` gevuld met meerdere regels HTML totdat het speciale eindlabel `END` bereikt is.

Let op:

De notatie van het HereDoc statement luistert nogal nauw, neem daarom de volgende aanwijzingen goed door:

De naam van het eindlabel mag u zelf verzinnen, u had bijvoorbeeld ook iets als `EOH` kunnen gebruiken (End Of HTML).

Achter de definitie van het eindlabel (`=<<< END`) komt **geen puntkomma** te staan. Sterker nog, u moet hier gelijk een [enter] geven, want er mag zelfs **geen witruimte achter** staan, anders krijgt u de volgende foutmelding:

Parse error: syntax error, unexpected T_SL in **D:\Sites\test.php** on line 3

Bij afsluiting van het stuk HTML moet het eindlabel helemaal aan het begin van een nieuwe regel geplaatst worden, waarna u het gehele statement afsluit met een puntkomma. Op deze regel mogen geen andere karakters voorkomen. Dus ook geen witruimte ervoor of erachter! Anders krijgt u de volgende foutmelding:

Parse error: syntax error, unexpected \$end in **D:\Sites\test.php** on line 8

Het eindlabel mag in principe wel op tussenliggende regels voorkomen, zolang het maar niet aan het begin van een regel staat en niet wordt afgesloten met een puntkomma.

Merk op dat deze notatie de volgende **voordelen** kent:

- **Escapen van quotes niet nodig**
U hoeft enkele/dubbele quotes niet te escapen met backslashes!
- **Concatenatie niet nodig**
U kunt variabelen gewoon tussen de HTML zetten, ze worden dan automatisch geïnterpoleerd, u hoeft ze niet te concateneren!
- **Newline characters niet nodig**
U hoeft geen newline characters toe te voegen aan het einde van de regels. De gewone [enters] volstaan!

Merk wel op dat als u het eind label gelijk na de laatste regel plaatst en daarna volgt nog meer uitvoer, dat deze daar alsnog gelijk achtergeplaatst wordt:

```
<?php
$html =<<< END
<p>tekst..</p>
END;
echo $html;
echo "<p>nog meer tekst..</p>";
?>
```

Bovenstaande code geeft als HTML uitvoer:

```
<p>tekst..</p><p>nog meer tekst..</p>
```

De oplossing bestaat er uit om voor het eindlabel nog een extra enter te geven:

```
<?php
$html =<<< END
<p>tekst..</p>
```

```
END;
echo $html;
echo "<p>nog meer tekst..</p>";
?>
```

Dit geeft in uw uitvoer:

```
<p>tekst..</p>
<p>nog meer tekst..</p>
```

De HereDoc syntax gedraagt zich eigenlijk precies hetzelfde als een double-quoted string, maar omdat de string niet omsloten wordt door double quotes kunt u ze gewoon in de string gebruiken zonder dat u ze hoeft te escaperen!

Arrays zoals `$_SERVER[]` vormen ook hier een probleem. U kunt ze **niet zonder** meer in de HTML code **plaatsen**:

```
<?php
// dit gaat mis!
$html =<<< END
<p>IP-adres: $_SERVER['REMOTE_ADDR']</p>
END;

echo $html;
?>
```

Dit geeft de volgende foutmelding:

Parse error: syntax error, unexpected T_ENCAPSED_AND_WHITESPACE, expecting T_STRING or T_VARIABLE or T_NUM_STRING in **test.php** on line 4

U kunt dit op dezelfde manier oplossen zoals u dat in een dubbel quoted string zou doen, namelijk door de array **tussen accolades** te zetten:

```
<?php
// zo gaat het goed!
$html =<<< END
<p>IP-adres: {$_SERVER['REMOTE_ADDR']}</p>
END;

echo $html;
?>
```

Als **alternatief** kunt u ook de **quotes weglaten**:

```
<?php
// zo gaat het ook goed!
$html =<<< END
<p>IP-adres: $_SERVER[REMOTE_ADDR]</p>
END;

echo $html;
?>
```

Let op: de quotes weglaten werkt alleen hier, doe het niet elders in uw code! U zou dan, bij maximale error_reporting, de volgende melding krijgen:

Notice: Use of undefined constant REMOTE_ADDR - assumed 'REMOTE_ADDR' in **test.php**

U mag de HereDoc syntax natuurlijk ook in combinatie met een `echo` statement gebruiken, u laat dan het `=`-teken weg:

```
<?php
echo <<< END
<p align="left">Het maximum is $max</p>
<p align="left">Nog een alineaa</p>
<p>etc..</p>

END;
?>
```

U kunt met de HereDoc syntax eenvoudig met *copy & paste* HTML tussenvoegen en er variabelen in plaatsen! Deze manier van werken is heel handig bij grote stukken HTML waarin veelvuldig variabelen moeten worden gezet, zoals bij formulieren.

Let op: we gebruikten in dit voorbeeld HTML in de string maar u kunt het natuurlijk voor elke situatie gebruiken waarin tekst met variabelen moeten worden gemixt. Bijvoorbeeld in een **SQL statement**:

```
$sql =<<< EOF
INSERT INTO COMPANY (voornaam, achternaam, afdeling, toestelnummer)
VALUES ('$voornaam', '$achternaam', '$afdeling', $toestelnummer);
EOF;
```

Nadelen zijn er natuurlijk ook. U kunt (net als bij gewone double quoted strings) **geen constanten** gebruiken. Bovendien kunt u ook **geen controlestructuren** in uw HereDoc blok opnemen.

De HereDoc syntax heeft overigens **nog een ander nadeel**: bij sommige editors raakt de **PHP color-syntax highlighting in de war**. Het lijkt dan net of u iets niet goed heeft gedaan, maar als u de pagina opvraagt ziet u dat alles werkt zoals het hoort.

3.14.11 Overzicht

Zoals u inmiddels wel begrepen hebt, is het op heel veel manieren mogelijk om met PHP HTML code te genereren...

Een (niet uitputtend) overzicht:

```
<?php
// In PHP:
$naam = 'Arno';

// single quotes: u moet concateneren
echo "<p class='groet'>Hoi " . $naam . " !</p>";

// dubbele quotes: u moet dubbele quotes escapen
echo "<p class=\"groet\">Hoi " . $naam . " !</p>";

// dubbele quotes: u kunt interpoleren:
echo "<p class=\"groet\">Hoi $naam!</p>";

// dubbele quotes: u kunt newlines gebruiken
echo "<p class=\"groet\">Hoi $naam!</p>\n";

// mixed:
echo "<p class='groet'>Hoi " . $voornaam . " !</p>\n";

// single quotes: newline met constante
echo "<p class='groet'>Hoi " . $naam . " !</p>" . PHP_EOL;

// of korter ha definitie eigen constante
```

```

define('N', "\n"); // of
define('N', PHP_EOL);
echo '<p class="groet">Hoi ' . $naam . '!</p>' . N;

// HereDoc
echo <<< EOH
<p class="groet">Hoi $voornaam!</p>
EOH;

// Terug naar HTML:
?>
<p class="groet">Hoi <?php echo $voornaam; ?></p>

<!-- met shorthand (Af te raden! Tenzij PHP >= 5.4) -->
<p class="groet">Hoi <?=$voornaam?></p>

```

De **shorthand** notatie is **af te raden** omdat deze **niet portabel** is. De werking is afhankelijk van de `short_open_tag` directive in `php.ini`. Wanneer deze de waarde `off` krijgt wordt de PHP code tussen `<?` En `?>` **niet** langer **uitgevoerd**. Deze wordt dan **voor iedereen zichtbaar** in de HTML code meegestuurd!

Tenzij u PHP `>= 5.4` gebruikt dan is werkt de `<?= ?>` notatie altijd, ongeacht de instellingen in `php.ini`. Deze manier van werken, waarbij HTML gewoon as-is wordt genoteerd en we PHP alleen gebruiken om met schort echo's data te injecteren waar dat nodig is, is veruit het handigst. Het is ook de manier waarop in moderne PHP frameworks gewerkt wordt.

3.15 Commentaar

Goede documentatie is bijna net zo belangrijk als het programma zelf! Tegenwoordig wordt geschat dat slechts 10% van de kosten van een programma bestaat uit het schrijven van de code en de overige 90% uit het onderhoud van de code. Goede documentatie maakt dat onderhoud een stuk makkelijker. Zeker als er meerdere mensen aan het onderhoud werken! Want het is slechts zelden dat een programma van begin tot eind door dezelfde programmeur(s) wordt onderhouden.

Elk non-triviaal script zou u daarom van commentaar moeten voorzien. U gebruikt het om aan te geven hoe u op het moment van schrijven over de op te lossen problemen dacht. Het commentaar is dan ook alleen bedoeld voor u zelf en andere programmeurs. Het maakt uw code als het goed is letterlijk leesbaarder. De scripting engine negeert het volledig.

Idealiter zou u bovenaan elke pagina moeten aangeven wat het doel van de pagina is en vervolgens voorziet u elk codeblok dat later vragen op zou kunnen roepen van *inline* commentaar.

Naderhand kunt u (of een collega) dan door enkel de commentaren door te nemen snel het verloop van het script volgen. De ervaring leert namelijk dat het na een aantal weken al behoorlijk lastig kan zijn om uw eigen code te begrijpen. Laat staan voor anderen die later uw code moeten onderhouden! Bovendien zult u merken dat uw programmeerstijl steeds verfijnder wordt. Op een later tijdstip zou u het zelfde probleem wellicht op een andere manier oplossen.

Een vuistregel is dat u elk codeblok waarbij u er niet aan zou moeten denken om het van commentaar te voorzien juist zou moeten becommentariëren! Doe het in elk geval voor alle code die later lastig te doorgronden kan zijn of waar u voor een afwijkende oplossing gekozen heeft.

Uit de code alleen kunt u op zijn best halen wat het doet, niet wat het verondersteld wordt te doen!

Tijdens deze cursus zijn de meeste scripts rijkelijk van commentaar voorzien. Dit is niet alleen om het goede voorbeeld te geven maar ook ter versterking van het leer effect.

Let er wel altijd goed op dat het commentaar de code altijd goed beschrijft. Met andere woorden: als u de **code aanpast**, zult u ook het bijbehorende **commentaar** moeten **aanpassen**! **Verouderd commentaar is erger dan geen commentaar!**

3.15.1 Eenregelig commentaar

Binnen PHP code kunt u commentaar plaatsen na twee *forward slashes* (`//`). Deze tekens moeten voor aan elke regel staan waar commentaar geplaatst wordt. Hier een voorbeeld:

```
<html>
<body>
  <?php
    // Dit is commentaar in de PHP code.
    // Let op: elke regel begint met twee slashes
  ?>
  <!-- Dit is commentaar binnen HTML -->
  etc...
</body>
</html>
```

In tegenstelling tot HTML commentaar zal het PHP commentaar in de uiteindelijke broncode van de webpagina **niet** meer aanwezig zijn.

Commentaar hoeft niet per sé op een eigen regel te staan. U kunt het ook achter een statement plaatsen om het te verduidelijken:

```
<?php
$verzonden = false; // initieer boolean
?>
```

Let op: U kunt soms in voorbeelden op het internet ook een `#` tegenkomen in plaats van `//` als het begin van een commentaarregel. Dit is de in Perl en Shells gebruikelijke manier, maar hoewel deze gewoon in PHP wordt ondersteund, wordt het gebruik van het hekje (*octothorpe*, *hash*, *pound*- of *number sign*) toch afgeraden in de meeste coding standards!

3.15.2 Meerregelig commentaar

Bovendien kunt u in PHP ook heel makkelijk meerdere regels als commentaar aanduiden door ze te omsluiten met `/*` en `*/` (*slash-star*, *star-slash*):

```
<?php
/*
  Dit is commentaar in de PHP code.
  De tweede regel is dat ook...
*/
?>
```

Dit wordt voor extra leesbaarheid ook wel met extra sterretjes genoteerd:

```
<?php
/**
 * Dit is commentaar in de PHP code.
 * De tweede regel is dat ook...
 */
?>
```

Er is ook een speciale manier van commentariëren die van deze notatie gebruik maakt, **PHPDoc** geheten, die bedoeld is voor het **automatisch** kunnen **documenteren van code**. U gebruikt het vooral om functies en classes mee te documenteren. Dit wordt in de vervolgcursussen behandeld...

3.15.2.1 HTML blokken uitcommentariëren

Met `/* */` kunt u zelfs `<?HTML<?php` blokken overlappen:

```
<?php
echo 'PHP actief';
/*
... uitgecommentarieerde PHP code
?>
<div>
  HTML blok (maar nu uitgecommentarieerd door PHP)
</div>
<?php
... nog meer uitgecommentarieerde PHP code
*/
echo 'PHP weer actief';
?>
```

3.15.2.2 Niet nesten

Let op dat de `/* */` notatie **niet genest** kan worden!

Als u bijvoorbeeld code heeft als deze:

```
echo 'regel 1';
/*
echo 'regel 2';
echo 'regel 3';
*/
echo 'regel 4';
```

En u zou nu het hele blok uit willen schakelen:

```
// Dit gaat dus mis!

/*
echo 'regel 1';
/*
echo 'regel 2';
echo 'regel 3';
*/
echo 'regel 4';
*/
```

Dan opent de eerste `/*` het commentaar, de tweede `/*` wordt genegeerd en de eerste `*/` sluit het commentaar af. U krijgt dan een error op de tweede `*/` die immers nu gewoon in de uit te voeren PHP code staat:

Parse error: syntax error, unexpected '*' in test.php on line 24

3.15.3 Begincommentaar

Een voorbeeld van een begincommentaar zou er zo uit kunnen zien:

```
<?php
include('common.php');
/*
*****WEBSITE: http://www.server.com *****
ALGEMEEN
Bestandsnaam:   voorbeeld.php
Doel:           illustreren van commentaarkop
Relaties:       common.php voor standaard
                functies
```

```
Invoer:      geen
Uitvoer:     HTML Skelet
Auteur:      Arno Mol (arno@ioc.nl)
Datum:       26/03/2017
```

REVISIE GESCHIEDENIS

```
24/03/2017   AM eerste opzet
26/03/2017   AM invoer/uitvoer toegevoegd
```

WENSEN

```
- Plek toevoegen voor paginaspecifieke functies
*****
*/
```

```
// Begin HTML pagina
begin_page('Voorbeeld commentaarkop');

// Doe verder niets....

// Sluit HTML pagina
end_page();
?>
```

3.15.4 Code 'uitcommentariëren'

Het is heel gebruikelijk en heel handig om met behulp van commentaar er voor te zorgen dat stukken code (tijdelijk) niet worden uitgevoerd. U schakelt ze als het ware even uit. Men noemt dit, met een enigszins krom woord, ook wel 'uitcommentariëren':

```
<?php
// deze code wordt niet uitgevoerd:
// $verzonden = false;
?>
```

Let op: In de praktijk komt het nogal eens voor dat code blokken worden uitcommentarieerd om ze vervolgens te herschrijven of anderszins aan te passen. De oude code blijft daarna vaak staan, ter referentie, maar u kunt deze 'dode' code het beste verwijderen... Om oude code terug te kunnen zien zou u een tool voor **versie beheer** moeten gebruiken!

3.15.4.1 Commentaar in HTML

Zoals u inmiddels weet zijn de PHP instructies normaal gesproken nooit zichtbaar in de HTML uitvoer. Alleen het resultaat is zichtbaar (wanneer dat geprint wordt). Wanneer u in PHP iets uit commentarieert wordt het zelfs niet eens uitgevoerd!

Maar wat gebeurt er eigenlijk als u HTML commentaar om een PHP blok zet?

```
<?php $naam = 'Arno'; ?>
<!-- <p>Hoi <?php echo $naam; ?></p> -->
```

De PHP code wordt dan natuurlijk gewoon normaal uitgevoerd in de paragraaf! Het resultaat is:

```
<!-- <p>Hoi Arno</p> -->
```

Dat wordt weliswaar niet zichtbaar in de browser getoond maar iedereen die de moeite neemt om in de page source te kijken kan het zien staan... Dat is vooral vervelend wanneer de variabele \$naam niet gezet zou zijn:

```
<!-- <p>Hoi <br />
<b>Notice</b>: Undefined variable: naam in <b>D:\demo\groet.php</b> on line
<b>12</b><br />
```

```
</p> -->
```

Merk op dat u de uitvoer wel kunt uitcommentarieren door in het PHP blok PHP commentaar te gebruiken en zo het hele echo statement niet uit te laten voeren:

```
<!-- <p>Hoi <?php // echo $naam; ?></p> -->
```

Het resultaat is dan:

```
<!-- <p>Hoi </p> -->
```

Let op dat u dit **voor** het echo statement doet! Proberen alleen `$naam` uit te commentarieren geeft een **parse error** waardoor de pagina niet meer getoond kan worden! Merk op dat dit **vooral bij** een **shorthand echo verleidelijk** is, maar dit **kan dus niet!**

```
<!-- <p>Hoi <?= // $naam; ?></p> -->
```

Dit geeft (afhankelijk van uw instellingen) een **blanco page** of de volgende **zichtbare error**:

```
Parse error: syntax error, unexpected '?' in D:\demo\groet.php on line 12
```

Uw enige optie is dan eigenlijk herschrijven naar het eerder getoonde echo statement, of gewoon simpelweg het hele blokje even weg halen...

3.15.5 DocBlocks

In PHP kennen we een speciale vorm van commentaar dat DocBlocks wordt genoemd. Deze worden vooral gebruikt om functies (en methods en classes) zo gedetailleerd te beschrijven dat u er met behulp van speciale tools automatisch online documentatie mee kunt genereren. We bespreken DocBlocks later in de cursus wanneer we zelf functies leren schrijven.

3.15.6 Tips

- Geef altijd commentaar tijdens het programmeren. Doe het niet pas aan het eind van een project. Dan schiet het er meestal bij in...
- Ga niet letterlijk omschrijven wat elk statement doet. Veel statements zijn zelfverklarend genoeg.
- Commentaar mag de code natuurlijk niet tegenspreken. Denk eraan dat u ook het commentaar wijzigt als u de code aanpast. Oud commentaar is nog erger dan geen commentaar!
- In principe is het mogelijk om teveel commentaar te geven: hele verhalen met af en toe een regel code. Dat is uiteraard ook niet de bedoeling. Meestal kunt u met een regel commentaar af voor meerdere regels code.
- Plaats alleen commentaar wanneer code complex of ondoorzichtig is.
- Geef uitgebreid commentaar aan het begin van een bestand of functie. Dan hoeft u niet eerst alle code te doorlopen om te zien wat deze doet.
- Binnen een team: schrijf het commentaar in een taal die alle programmeurs goed beheersen. Vaak wordt dan voor Engels gekozen. Zorg voor een standaard zodat al het commentaar in dezelfde stijl is.

3.16 Controlestructuren

De code die u tot nu toe heeft geschreven was **lineair** (sequentieel), wat betekent dat PHP de opdrachten uitvoert in de volgorde waarin ze in het script staan. Dus van links naar rechts en van onder naar boven. Het verloop van een programma wordt ook wel met de term *flow* aangeduid.

In de meeste scripts wilt u echter beslissingen kunnen nemen zodat onder verschillende voorwaarden verschillende acties uitgevoerd kunnen worden. Dit is noodzakelijk om adequaat te kunnen reageren op gebruikersinvoer (wat wordt behandeld in het volgende hoofdstuk). U kunt die beslissingen nemen met speciale instructies die **controlestructuren** worden genoemd.

Een controlestructuur is een programmastructuur die het mogelijk maakt dat uw script beslissingen neemt die gebaseerd zijn op de input die het krijgt. De controlestructuren zelf veroorzaken geen zichtbare veranderingen in de data of output. Ze besluiten alleen welke acties worden gedaan en in welke volgorde.

Met controlestructuren kunt u dus het verloop van het programma beïnvloeden. Er zijn 3 belangrijke typen controlestructuren:

- **Selecties** (Vergelijkingen of vertakkingen. In het Engels: *Conditional Logic* of *Branching Logic*)
- **Iteraties** (Herhalingen of lussen, in het Engels: *Looping Logic*)
- **Procedures & Functies** (*Jumping Logic*)

Deze controlestructuren geven u controle over of, wanneer en hoe vaak bepaalde instructies worden uitgevoerd. In dit hoofdstuk bespreken we selecties en iteraties. Functies worden aan het einde van deze cursus kort besproken in hoofdstuk 7, maar worden uitgebreid behandeld in de **cursus PHP Vervolg**.

3.17 Selecties

Selecties zijn controlestructuren die een **voorwaardelijke test** gebruiken om te bepalen welke code moet worden uitgevoerd. Ze vormen één van de belangrijkste onderdelen in elke programmeertaal.

Een voorwaardelijke test is een test waarbij een voorwaarde (of een combinatie van voorwaarden) wordt geëvalueerd. Een voorwaarde is daarbij een Booleaanse expressie die naar `true` of `false` evalueert. Andere mogelijkheden zijn er niet, er bestaat niet zoiets als "misschien".

De meest simpele variant is een test die controleert op de waarde van een Booleaanse variabele (het `if`-statement wordt later in deze paragraaf uitgebreid behandeld):

```
if ($getrouwd == true) {  
    echo "U bent getrouwd";  
}
```

Omdat standaard gekeken wordt of de test `true` is kunt u dit korter noteren, iets wat in de praktijk heel veel gebeurt:

```
if ($getrouwd) {  
    echo "U bent getrouwd";  
}
```

U had overigens ook op `false` kunnen controleren:

```
if ($getrouwd == false) {  
    echo "U bent niet getrouwd";  
}
```

Of korter:

```
if (! $getrouwd) {
    echo "U bent niet getrouwd";
}
```

Het uitroepteken staat hier voor logisch NOT en ontkent dus de voorwaarde.

In PHP kunnen ook andere variabelen dan Booleans direct in een conditie gebruikt worden:

- Elke string die langer is dan 0 karakters evalueert naar `true`, met uitzondering van `"0"` dat naar `false` evalueert.
- Elk getal anders dan 0 evalueert naar `true`

```
$a = "";           // if ($a) evalueert naar false
$b = "false";      // if ($b) evalueert naar true
$c = "0";          // if ($c) evalueert naar false

$x = 1;           // if ($x) evalueert naar true
$y = 0;           // if ($y) evalueert naar false
$z = -1;          // if ($z) evalueert naar true
```

Het hierboven beschreven gedrag is misschien niet altijd even voor de hand liggend. In de PHP manual is het echter goed gedocumenteerd:

<http://www.php.net/manual/nl/types.comparisons.php>

Wat in de praktijk ook vaak voorkomt is dat voor de tests twee waarden vergeleken worden:

3.17.1 Vergelijkingsoperatoren

Binnen de te testen condities gebruikt u zogenaamde **vergelijkingsoperatoren** om twee of meer waarden met elkaar te vergelijken.

In PHP kunt u de volgende vergelijkingsoperatoren gebruiken:

operator	voorbeeld	True	False
<code>==</code>	<code>\$A == \$B</code>	als A en B gelijk zijn	als A en B ongelijk zijn
<code>===</code>	<code>\$A === \$B</code>	als A en B gelijk zijn en van hetzelfde datatype zijn (vanaf PHP4)	als A en B ongelijk zijn (of niet van hetzelfde datatype zijn)
<code>></code>	<code>\$A > \$B</code>	als A groter is dan B	als A kleiner is dan B
<code><</code>	<code>\$A < \$B</code>	als A kleiner is dan B	als A groter is dan B
<code>!=</code>	<code>\$A != \$B</code>	als A en B ongelijk zijn	als A en B gelijk zijn
<code>!==</code>	<code>\$A !== \$B</code>	als A en B ongelijk zijn of niet van hetzelfde datatype zijn (vanaf PHP4)	
<code>>=</code>	<code>\$A >= \$B</code>	als A groter dan of gelijk is aan B	als A kleiner is dan B
<code><=</code>	<code>\$A <= \$B</code>	als A kleiner dan of gelijk is aan B	als A groter is dan B

Stel, `$a = 3` en `$b = 4` dan evalueert

```
($a == $b)  naar  false
($a > $b)   naar  false
($a < $b)   naar  true
($a != $b)  naar  true
```

```
($a >= $b)   naar   false
($a <= $b)   naar   true
```

Een voorbeeld:

```
if ($cijfer < 6) {
    echo "onvoldoende";
}
```

3.17.1.1 Klassieke fout met enkel -=teken

Let op het dubbele -=teken! Een **klassieke fout** is om in vergelijkingen een **enkel -=teken** te gebruiken:

```
// LET OP: geeft geen error maar is altijd waar!
if ($getrouwd == true) {
    echo "U bent getrouwd";
} else {
    echo "U bent NIET getrouwd";
}
```

In de code hierboven zal de *else branch* nooit worden uitgevoerd. In talen als Visual Basic wordt een enkel -=teken ook als vergelijingsoperator gebruikt, maar in PHP is dit de **toewijzings operator**. Bovenstaande conditie is daardoor altijd waar, want het lukt altijd om de waarde `true` toe te kennen aan de variabele `$getrouwd` ...

Het lastige is dat u bij dergelijke logische fouten geen foutmeldingen krijgt en dat ze lastig op te sporen zijn. Omdat deze fout nogal eens gemaakt wordt hebben sommige programmeurs zich aangewend om de conditie **om te draaien**:

```
// geeft nu een error!
if (true = $getrouwd) {
    echo "U bent getrouwd";
} else {
    echo "U bent NIET getrouwd";
}
```

U wordt nu geattendeerd op de foutieve toepassing door de volgende foutmelding:

Parse error: syntax error, unexpected '='

Het is immers niet mogelijk om de waarde van de variabele `$getrouwd` toe te kennen aan de boolean `true`! Als u bovenstaande fout vaak maakt is het best een overweging om uw condities op deze manier op te stellen!

3.17.2 Booleaanse operatoren

Behalve vergelijingsoperatoren kunt u in tests ook zogenaamde **Booleaanse operatoren** gebruiken. Ze worden vaak ingezet om samengestelde Booleaanse expressies (gevormd door vergelijingsoperatoren) op te lossen. Ze geven als resultaat wederom `true` of `false`.

De meest gebruikte zijn:

operator	voorbeeld	True	False
&& (AND)	\$A && \$B	als A en B beiden waar zijn	als A of B niet waar is
 (OR)	\$A \$B	als of A of B waar is	als A en B beiden niet waar zijn

! (NOT)	!\$A	als A niet waar is	als A waar is

Stel:

```
$a = 3;
$b = 4;
$c = 5;
$d = 6;
```

Dan evalueren deze condities als volgt:

```
($a < $b) && ($c < $d); // true
($a < $b) && ($c > $d); // false
($a < $b) || ($c > $d); // true
(! ($c > $d));          // true
```

Nog een voorbeeld:

```
if (($praktijk > 6) && ($theorie > 6)) {
    echo "geslaagd";
}
```

Voor de leesbaarheid vinden veel programmeurs het prettig om na de **NOT operator !** een **spatie** te gebruiken:

```
if (! $success) {} // duidelijker!
if (!$success) {}
```

Voor de werking maakt de spatie natuurlijk niet uit... U leest de uitroepteken simpelweg als *not*: *if not succes then...*

De *logical not operator !* wordt ook wel de *bang operator* genoemd.

3.17.3 If

Het `if` statement uit bovenstaande voorbeelden is één van de meest fundamentele en krachtige statements in elke programmeertaal. Als u al eens eerder geprogrammeerd heeft dan zult u het zeker kennen. De algemene notatie is

```
if (conditie) {
    // statements die worden uitgevoerd als conditie waar is
}
```

U kunt het `if` statement gebruiken om stukken code alleen dan uit te laten voeren als aan een bepaalde conditie voldaan wordt. In zijn meest simpele vorm ziet het er zo uit:

```
if ($a > $b) echo 'A is groter dan B';
```

U kunt dit lezen als: **Als** de waarde van variabele `$a` groter is dan die van variabele `$b` **dan**, en alleen dan, moet de tekst `'A is groter dan B'` naar het scherm worden geschreven. Is `$a` kleiner dan `$b` of gelijk aan `$b` dan gebeurt er niets. Het script zal verder gaan met de volgende regel.

De **conditie** staat **altijd tussen haakjes**. Dat is **verplicht** en maakt in één oogopslag duidelijk welk deel de conditie is. Bij samengestelde condities zet u alle afzonderlijke condities ook nog eens tussen haakjes!

In bovenstaand voorbeeld wordt slechts één statement uitgevoerd. Als u **meerdere statements** wilt laten uitvoeren als aan de conditie voldaan wordt, dan **moet** u het **codeblok tussen accolades { }** plaatsen:

```
if ($a > $b) {  
    echo 'A is groter dan B';  
    // ken de waarde van $a toe aan $max  
    $max = $a;  
}
```

Zou u de accolades weglaten:

```
// LET OP: niet zo doen!  
  
if ($a > $b)  
    echo 'A is groter dan B';  
    // ken de waarde van $a toe aan $max  
    $max = $a;
```

Dan gebeurt er iets anders dan u denkt. De waarde van `$a` wordt in dit geval **altijd** aan `$max` toegekend, ongeacht de uitkomst van de vergelijking! **Alleen** het **eerste statement**, de echo regel is nu **conditioneel**. In feite is dit wat er staat:

```
if ($a > $b) {  
    echo 'A is groter dan B';  
}  
  
// ken de waarde van $a toe aan $max  
$max = $a;
```

Het gebruik van accolades maakt het veel duidelijker wat waar bij hoort en wanneer het wordt uitgevoerd. Gebruik ze daarom altijd, ook als u maar een statement gaat uitvoeren! Bovendien kunt u het daarna dan altijd makkelijk uitbouwen naar meerdere statements indien dat nodig blijkt te zijn...

Let op de manier waarop de code tussen de accolades **inspringt**. Doe dit **altijd**. Het verhoogt de leesbaarheid van uw scripts enorm! Of u hiervoor tabs of spaties wilt gebruiken, hangt van uw persoonlijke voorkeur af of van de afspraken in uw team.

3.17.4 Else

Wanneer u wilt dat er bepaalde code wordt uitgevoerd als er **niet** aan de conditie wordt voldaan dan kunt u daarvoor het `else` statement gebruiken:

```
if ($a > $b) {  
    echo "A is groter dan B";  
} else {  
    echo "A is kleiner dan of gelijk aan B";  
}
```

NB.: U had dit ook op één regel mogen zetten en dan had u de accolades weg kunnen laten. Bij tweevoudige selecties als deze komt dat de leesbaarheid echter niet ten goede!

Nu wordt ook duidelijk waarom men het bij `if ... else` statements ook wel over *branching logic* heeft: de alternatieven zijn **vertakkingen** (*branches*) in het programmaverloop.

Oefening 3.3

Voeg aan de pagina variabelen.php een if ... else controlestructuur toe waarmee u controleert of een variabele \$getrouwd waar is of niet. Schrijf in beide situaties een toepasselijke boodschap naar het scherm.

3.17.5 If Else combineren met HTML

U kunt if else statements (en andere controlestructuren) ook combineren met stukken HTML:

```
<?php if ($verzonden) { ?>
    <h1 class="ok">Uw bestelling is succesvol verzonden.</h1>
    <p>Bedankt en tot ziens!</p>
<?php } else { ?>
    <h1 class="error">Helaas, er is een fout opgetreden!</h1>
    <p>Probeer het nogmaals.</p>
<?php } ?>
```

De if en else branches zijn dan in HTML modus en dat is makkelijk noteren zo. U hoeft dan geen hele lappen HTML te echoën en de HTML blijft ook makkelijk bewerkbaar in de Design modus van een WYSIWYG editor als bijvoorbeeld Dreamweaver.:



Let op: Denk er aan dat u alle begin- en sluit-accolades goed neerzet!

Tip:

Het is handig om eerst de structuur neer te zetten (bijvoorbeeld met een snippet!) en dan de HTML er tussen te plaatsen. Verder is het handig om de PHP blokken helemaal links tegen de kantlijn te zetten en de HTML blokken iets in te laten springen. Zo kunt u goed zien welke code wanneer wordt uitgevoerd.

3.17.5.1 Template Syntax

PHP kent speciaal voor het gebruik in HTML templates een **alternatieve template syntax** voor controle structuren. In plaats van de openingsaccolades gebruikt u dan een **dubbele punt**. En op het einde schrijft u **endif** gevolgd door een **puntkomma**:

```
<?php if ($verzonden): ?>
    <h1 class="ok">Uw bestelling is succesvol verzonden.</h1>
    <p>Bedankt en tot ziens!</p>
<?php else: ?>
    <h1 class="error">Helaas, er is een fout opgetreden!</h1>
    <p>Probeer het nogmaals.</p>
<?php endif; ?>
```

Merk op dat u **geen** endif of puntkomma vóór voor de else nodig heeft. We komen aan het einde van dit hoofdstuk nog terug op de template syntax. U kunt deze ook voor namelijk ook voor de andere, nog te bespreken, controlestructuren gebruiken!

3.17.6 Ternary (Conditional) Operator

In de praktijk komt vergelijkbare code als in onderstaand codefragment vaak voor:

```
if ($a > $b) {  
    $max = $a;  
} else {  
    $max = $b;  
}
```

Afhankelijk van een bepaalde conditie krijgt een variabele één van twee waarden toegekend.

PHP kent voor deze situatie een alternatieve schrijfwijze met gebruik van de **ternary operator** `?` `:`, ook wel **conditional operator** of voluit **ternary conditional operator** genoemd. De operator ontleent zijn naam aan het feit dat hij 3 argumenten gebruikt waarvan de eerste een conditie is. De overige twee argumenten geven aan wat er moet gebeuren na evaluatie van de conditie.

In algemene ziet een statement dat gebruik maakt van de ternary operator er zo uit:

conditie ? true-statement : false-statement

Er wordt getest op een conditie, als deze waar is dan wordt het true-statement uitgevoerd en zo niet dan wordt het false-statement uitgevoerd. Bovenstaand fragment kan dan als volgt worden herschreven:

```
$max = $a > $b ? $a : $b;
```

Op deze manier kunt u heel compact noteren, wat zeker heel handig is bij het initialiseren van meerdere formulier variabelen, zoals we in het volgende hoofdstuk zullen zien.

U kunt de notatie trouwens ook prima gebruiken in combinatie met `echo`:

```
echo $count > 0 ? 'Op voorraad' : 'Niet leverbaar';
```

Overigens is niet iedereen even gelukkig met de compacte notatie. Het is best lastig leesbaar als u de notatie niet gewend bent. Gebruik dit construct dan ook alleen als u zich er comfortabel bij voelt. Is dat (nog) niet het geval dan kunt u altijd de langere `if ... else` notatie gebruiken... Maar ook als u het niet zelf gaat gebruiken is het goed om het te herkennen wanneer u het in code van anderen tegenkomt!

Overigens hebben de langere `if ... else` notaties (net als de straks te bespreken `switch`) wel een ander voordeel: ze zijn vrij makkelijk uit te breiden naar het testen van meerdere condities. Hoewel u ternary operator statements ook kunt nesten, komt dit de leesbaarheid meestal niet ten goede!

3.17.6.1 Shorthand Ternary Operator `?:` (aka isseor of Elvis operator)

In expressies als deze:

expr1 ? expr2 : expr3

Dan evalueert dit naar `expr2` als `expr1` **true** (of beter gezegd *truthy*) is en naar `expr3` als `expr1` **false** is, zoals we net hebben gezien.

Sinds PHP 5.3 is het mogelijk om het middelste deel van de ternary operator weg te laten:

expr1 ?: expr3

Dit retourneert `expr1` als `expr1` **true** is en anders `expr3`.

De shorthand notatie wordt ook wel de **issetor** (als in `isset-or`) genoemd. Of grappiger: de **Elvis operator**. Dit omdat wanneer u de operator zou kantelen er een gelijkenis is met de kuif van Elvis:



Maar helaas kunt u hem beter niet zo gebruiken:

```
$id = $_GET['id'] ?: 0;
```

In plaats van het traditionele :

```
$id = isset($_GET['id']) ? $_GET['id'] : 0;
```

De korte notatie geeft namelijk alsnog de `E_STRICT` warning:

Notice: Undefined index: id in **demo.php**

3.17.6.2 null coalescing operator ?? (PHP 7)

Met de null coalescing operator (ook wel null coalesce operator) heeft u dit probleem niet. Coalesce betekent *samenvloeien* of *zich verenigen*.

```
$id = $_GET['id'] ?? 0;
```

Deze geeft namelijk geen foutmelding als de eerste operand niet bestaat of null is!

Maar wanneer u de beveiliging met `intval()` probeert in te bouwen gaat het alsnog mis:

```
$id = intval($_GET['id']) ?? 0;
```

Want dit geeft weer:

Notice: Undefined index: id in **demo.php**

Voor meer informatie over de verschillen tussen de `?:` en `??` operators zie:

<https://stackoverflow.com/questions/34571330/php-ternary-operator-vs-null-coalescing-operator>

3.17.7 Nesten

Soms moet er aan **meerdere condities** worden voldaan voordat u een bepaald stuk code wilt laten uitvoeren. U kunt dan meerdere `if` statements *nesten*:

```
if ($praktijk > 6) {
    if ($theorie > 6) {
        echo 'geslaagd';
    }
}
```

Het tweede `if`-statement wordt alleen uitgevoerd als de eerste waar is. Deze notatie komt de leesbaarheid meestal niet ten goede. Het is dan beter om de condities te combineren met een logische operator:

```
if (($praktijk > 6) && ($theorie > 6)) {  
    echo 'geslaagd';  
}
```

Een andere situatie waarin u moet gaan nesten is wanneer u behoefte heeft aan **meerdere vertakkingen** (meer dan twee). U zou bijvoorbeeld in het `else` statement op een tweede conditie kunnen testen:

```
if ($a > $b) {  
    echo 'A is groter dan B';  
} else {  
    if ($a < $b) {  
        echo 'A is kleiner dan B';  
    } else {  
        echo 'A is gelijk aan B';  
    }  
}
```

Er zijn nu 3 mogelijkheden maar er wordt nog steeds maar één van de alternatieven uitgevoerd!

Door consequent in te springen blijft het geheel toch redelijk goed leesbaar. Het wordt daardoor ook meteen duidelijk dat elke `if` correct wordt afgesloten met een `}`. Toch wordt een dergelijke werkwijze bij enkele niveaus diep nesten en lange codeblokken vrij onoverzichtelijk.

Omdat deze situatie echter vaak voorkomt bestaat er een aparte constructie met de naam `elseif`:

3.17.8 Elseif

`elseif` is, zoals de naam al doet vermoeden een samentrekking van `else` en `if`. U kunt het gebruiken om meervoudig geneste `if` statements compacter te kunnen noteren waardoor de code beter leesbaar wordt. Het eerste voorbeeld uit de vorige paragraaf kunt u als volgt herschrijven:

```
if ($a > $b) {  
    echo 'A is groter dan B';  
} elseif ($a < $b) {  
    echo 'A is kleiner dan B';  
} else {  
    echo 'A is gelijk aan B';  
}
```

In principe zit er geen beperking aan het aantal `elseif` statements maar een `elseif` kan niet na een `else` statement gebruikt worden. Het laatste statement kan eventueel wel een `else` zijn. `elseif` vereist geen eigen `}`, maar het eerste `if`-statement natuurlijk wel. U hoeft nu dus nog maar één keer een `}` te gebruiken!

Oefening 3.4a *Maak een pagina `begroeting.php` waarin u, afhankelijk van de huidige tijd, een passende begroeting toont. Maak daarbij gebruik van de wetenschap dat de PHP functie `date` ook de tijd in uren uit een tijdstip kan isoleren:*

```
$uur = date('H');
```

Gebruik daarbij de volgende intervallen:

Tussen 0 en 6 Goedenacht

Tussen 6 en 12 Goedemorgen

Tussen 12 en 18 Goedemiddag

Tussen 18 en 0 Goedenavond

Oefening 3.4b *Vergroot het effect van de pagina op de vorige oefening door verschillende (achtergrond)kleuren te gebruiken. Denk bijvoorbeeld aan een zwarte pagina met witte tekst voor de nacht... Hint: gebruik een variabele `$css_class`.*

3.17.9 Switch

Wanneer u in uw code de waarde van een variabele op een (groot) aantal mogelijke waarden moet testen dan kunt u dit natuurlijk doen met opeenvolgende `if`-statements voor alle mogelijke waarden, of met een `if ... elseif ... else` constructie. Maar de eerste oplossing is niet heel erg efficiënt omdat altijd alle condities moeten worden getest. Wanneer u gebruikt maakt van `elseif` dan worden ze getest totdat er een gevonden wordt die waar is. Maar de notatie is niet heel overzichtelijk.

Een elegante oplossing voor het vergelijken van een waarde met een groot aantal waarden is de `switch` constructie. Hierbij hoeft er maar één keer een conditie getest te worden. Waarna op basis van de uitkomst gezocht wordt naar een match met een van de binnen de switch opgegeven cases. Als er een match wordt gevonden dan wordt het bij die case gespecificeerde codeblok uitgevoerd.

Zo kan snel de juiste optie uit een lijst met mogelijkheden worden uitgevoerd. De `switch` is behalve snel ook **makkelijk** in het gebruik, heel **leesbaar** en makkelijk **uitbreidbaar**.

N.B.: Voor een check op slechts twee mogelijke uitkomsten is een `if`-statement wel sneller...

In onderstaand voorbeeld wordt op een aantal mogelijke waarden voor de variabele `$beroep` gecontroleerd:

```
switch ($beroep) {
    case 'manager':
        // toon relevante productinfo voor managers
        break;
    case 'developer':
        // toon relevante productinfo voor developers
        break;
    case 'trainer':
        // toon relevante productinfo voor trainers
        break;
}
```

Wanneer de waarde van de variabele `$beroep` met een van de cases overeenkomt dan wordt het bij die case behorende codeblok uitgevoerd. Bij een case kunnen meerdere statements worden uitgevoerd. Deze hoeven niet tussen accolades te staan.

Wanneer er geen match gevonden wordt dan gaat het script verder met de eerstvolgende regel na het einde van de `switch`.

3.17.9.1 break

Let op: Het `break` statement is erg belangrijk! Zou u het weglaten dan worden de codeblokken van **alle cases na** de match alsnog uitgevoerd! PHP gaat namelijk **door met** het uitvoeren van de **code totdat** het een `break` statement tegenkomt! PHP ziet het **begin** van een **nieuwe case niet automatisch** als het **einde** van de **vorige case**. Bij veel andere programmeertalen is dit wel het geval...

Van het voorgaande kunt u handig gebruikmaken als voor meerdere cases dezelfde actie moet worden uitgevoerd. U zet ze dan gelijk onder elkaar **zonder** `break` ertussen:

```
...
case 'insert':
case 'update':
    toon_formulier();
    break;
...
```

U mag de cases overigens ook achter elkaar zetten, eventueel met het codeblok en de `break` erachter:

```
...
case 'insert': case 'update': toon_formulier(); break;
...
```

In situaties als deze, waarbij u voor meer dan één mogelijke waarde steeds dezelfde code wilt laten uitvoeren, bespaart het gebruik van een `switch` u een hoop dubbele code (en verkleint daarmee de kans op fouten)!

Maar denk er wel aan dat de cases elkaar normaal gesproken wederzijds moeten uitsluiten: er mag altijd maar één mogelijke match zijn.

3.17.9.2 default case

Wanneer u wilt kunt u een vangnet inbouwen door een `default`: optie aan de reeks cases toe te voegen. Zo kunt u een `default` actie opgeven (of onverwachte of foutieve invoer afvangen):

```
switch ($beroep) {
    case 'manager':
        // toon relevante productinfo voor managers
        break;
    case 'developer':
        // toon relevante productinfo voor developers
        break;
    case 'trainer':
        // toon relevante productinfo voor trainers
        break;
    default:
        // toon standaard productinfo
        break;
}
```

De `default` case hoeft overigens niet perse aan het einde te staan, u kunt hem ook aan het begin plaatsen als uw code daardoor voor u leesbaarder wordt...

U kunt de `switch` constructie heel goed gebruiken in combinatie met een **selectbox** (keuzelijst) in een HTML formulier zoals we in het volgende hoofdstuk zullen zien. Hierin staan immers een vast aantal opties waarvan er slechts één door de gebruiker gekozen kan worden.

We oefenen nu niet met een `switch`, dat doen we bij de formulieren waar we een switch zullen gebruiken in combinatie met een selectbox.

3.17.9.3 Expressie

In principe probeert u met een switch een waarde te matchen met een case. Vaak zal u in de switch een variabele zetten en in cases de mogelijke waarden die deze variabele kan hebben.

In de oefening met de tijdsafhankelijke begroeting zijn er echter nogal wat waarden die dezelfde groet moeten opleveren en dat ziet er nogal chaotisch uit:

```
switch ($uur) {
    case 0: case 1: case 2: case 3: case 4: case 5:
        $groet = 'goedenacht';
        break;
    case 6: case 7: case 8: case 9: case 10: case 11:
        $groet = 'goedemorgen';
        break;
    case 12: case 13: case 14: case 15: case 16: case 17:
        $groet = 'goedemiddag';
        break;
    case 18: case 19: case 20: case 21: case 22: case 23:
        $groet = 'goedenavond';
        break;
    default:
        $groet = 'Hoi';
}
```

U kunt in de switch echter ook een **expressie** opnemen, zolang de uitkomst maar een waarde oplevert. Iets als dit mag dus ook:

```
// haal uur uit het huidige tijdstip
$uur = date('H');

// deel uur door 6 en rond af naar boven
switch (ceil($uur/6)) {
    // code voor de 4 mogelijke uitkomsten
    case 1:
        // code voor 0-6
        break;
    case 2:
        // code voor 6-12
        break;
    case 3:
        // code voor 12-18
        break;
    case 4:
        // code voor 18-24
        break;
}
```

3.17.9.4 Reverse switch

Het omgekeerde mag ook: u geeft in de switch een vaste waarde op en voor de cases gebruikt u expressies. Een dergelijke switch wordt ook wel **reverse switch** of een switch met **conditional cases**:

```
switch (true) {
    case ($uur >= 0 && $uur < 6):
        $groet = 'goedenacht';
        break;
    case ($uur >= 6 && $uur < 12):
        $groet = 'goedemorgen';
        break;
    case ($uur >= 12 && $uur < 18):
```

```

        $groet = 'goedemiddag';
        break;
    case ($uur >= 18 && $uur < 24):
        $groet = 'goedenavond';
        break;
    default:
        $groet = 'Hoi';
}

```

Overigens ook leuk om dan in de cases iets te doen als:

```

switch(true) {
    case in_array($uur, range(0, 5)): $groet = 'goedenacht'; break;
    case in_array($uur, range(6, 11)): $groet = 'goedemorgen'; break;
    case in_array($uur, range(12, 17)): $groet = 'goedemiddag'; break;
    case in_array($uur, range(18, 23)): $groet = 'goedenavond'; break;
    default:
        $groet = 'Hoi';
}

```

Met reverse switches kunt u een aantal verschillende soorten controles toch overzichtelijk in een switch bij elkaar plaatsen:

```

switch (true) {
    case $x == 'hoi' : echo '$x heeft de waarde hoi'; break;
    case is_numeric($x) : echo '$x is een getal'; break;
    case $x < 10 : echo '$x is kleiner dan 10'; break;
}

```

Wat wellicht iets makkelijker leest dan het equivalent met if-else-if statements:

```

if ($x == 'hallo') {
    echo '$x heeft de waarde hallo';
} else if (is_numeric($x)) {
    echo '$x is een getal';
} else if ($x < 10) {
    echo '$x is kleiner dan 10';
}

```

Maar dat is natuurlijk ook een kwestie van smaak en/of gewenning!

3.18 match (PHP8)

Vanaf PHP8 heeft `switch()` een grotere broer gekregen, namelijk `match()`. In tegenstelling tot `switch` dat een statement is, is `match` een expressie en deze kan **veel korter genoteerd** worden:

- Er is geen `break` statement nodig
- U kunt tegen meerdere (komma gescheiden) waarden controleren!
- Omdat het een waarde retourneert, hoeft u maar 1x een waarde toe te kennen

Een eenvoudig voorbeeld laat dat snel zien:

```

$number = 1;

$letter = match($number) {
    0, 1 => 'a',
    2 => 'b',
    3 => 'c'
};

echo $letter; // 'a'

```


Met een switch was daarvoor meer code geweest:

```
$number = 1;

switch($number) {
    case 0:
    case 1:
        $letter = 'a';
        break;
    case 2:
        $letter = 'b';
        break;
    case 3:
        $letter = 'c';
        break;
}

echo $letter; // 'a'
```

Merk ook op dat u bij de match wel en bij de switch juist geen ; achter de eind-accolade } zet...

Een **belangrijk verschil** is verder dat een switch een loose comparison doet (het gebruikt ==) waar match een **strict comparison** doet (het gebruikt ===):

```
$input = '0';

$output = match($input) {
    0 => 'number',
    '0' => 'letter'
};

echo $output; // 'letter'
```

Vergelijk dat met de switch variant:

```
$input = '0';

switch($input) {
    case 0:
        $output = 'number';
        break;
    case '0':
        $output = 'letter';
        break;
}

echo $output; // 'number'
```

Over wat handiger is kan natuurlijk per situatie verschillen maar het is in ieder geval belangrijk ervan op de hoogte te zijn. Als u straks in de oefening uw tijdsafhankelijke begroeting naar een match variant gaat omzetten bijvoorbeeld...

U kunt net als in switch ook een **default waarde retourneren** wanneer er geen overeenkomst gevonden wordt:

```
$number = 4;

$letter = match($number) {
    0, 1 => 'a',
    2 => 'b',
    3 => 'c',
    default => 'no match found...'
```

```
};  
  
echo $letter; // 'no match found...'
```

Maar bij match wordt er een `UnhandledMatchError` **exception** gethrowd als er **geen overeenkomt** is en u **geen default** hebt opgegeven. Dit maakt hem nog wat strikter:

```
$number = 4;  
  
$letter = match($number) {  
    0, 1 => 'a',  
    2 => 'b',  
    3 => 'c',  
};  
  
echo $letter; // UnhandledMatchError
```

De output is nu iets als:

```
Fatal error: Uncaught UnhandledMatchError: Unhandled match value of type int in demo.php:10
```

Samenvattend:

Switch	Match
Statement	Expression
Loose comparison	Strict comparison
Kan meerdere waarden evalueren (afhankelijk van gebruik break)	Evalueert één waarde
Staat gebruik block statements toe	Enkel single-line expressions mogelijk

Meer informatie vindt u op:

- <https://www.php.net/manual/en/control-structures.match.php>
- <https://stitcher.io/blog/php-8-match-or-switch>

Oefening	<i>Herschrijf de eerder gemaakte tijdsafhankelijke begroeting waarbij u gebruik maakt van de mogelijkheden van <code>match()</code>. Gebruik ook <code>default</code> en denk aan de strict comparison!</i>

3.19 Herhalingen (Lussen)

In sommige gevallen kan het noodzakelijk zijn dat u een bepaalde bewerking een (specifiek) aantal keren moet herhalen. Denk maar eens aan het uitrekenen van de totale rente gedurende de resterende looptijd van een lening, of het afdrukken van de informatie uit een selectie op een database. Voor dergelijke situaties gebruikt u lussen (*loops*).

PHP kent een drietal veelgebruikte controlestructuren om lussen mee te maken:

1. **while / do while**
2. **for**
3. **foreach**

Wanneer gebruikt u nu welke? De vuistregel is de keuze eerst af te laten hangen van de wetenschap of u *van tevoren* al weet hoe vaak de code herhaald dient te worden.

Weet u het aantal herhalingen niet en moet u steeds na het einde van een lus bepalen of er nog een volgende doorgang moet plaatsvinden, dan gebruikt u het best een `while` lus.

Als u van tevoren al weet hoe vaak een lus doorlopen moet worden en het aantal op kunt slaan in een variabele, dan kunt u het beste een `for` lus gebruiken.

De `foreach` lus wordt gebruikt voor het doorlopen van arrays. U kunt hier ook een `for` lus voor gebruiken maar de `foreach` lus is eenvoudiger in het gebruik en bovendien sneller!

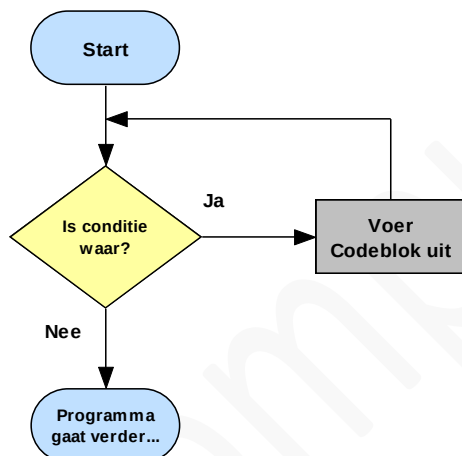
3.19.1 While

De meest eenvoudige lus in PHP is de `while` lus. De algemene notatie luidt:

```
while (conditie) {
    // code die herhaald moet worden zolang conditie waar is
}
```

Bij elke lusdoorgang wordt gecontroleerd of aan een opgegeven logische conditie wordt voldaan. Zo lang aan de conditie voldaan wordt, wordt de code in de lus uitgevoerd. Als de conditie niet langer waar is, dan wordt de code in de lus niet (meer) uitgevoerd en gaat het script verder met de statements na de eindaccolade. U zou de `while` lus dan ook kunnen zien als een zich herhalende `if...`

In een flowchart:



In de praktijk worden `while` lussen vaak gebruikt om door een tekstbestand of de resultaten van een database query te lopen totdat het einde bereikt is. In **pseudo code**:

```
while (there are still rows to read from a database) {
    read in row;
    move to the next row;
}
```

We zullen daar in **Hoofdstuk 6 Werken met databases** ook gebruik van maken.

Of om op de CLI net zo lang om correcte invoer te vragen als nodig is:

```
echo 'Voer een getal in.' . PHP_EOL;
$getal = readline('> ');

while (! is_numeric($getal)) {
```

```

    echo "'$getal' is geen getal, probeer het opnieuw..." . PHP_EOL;
    $getal = readline('> ');
}

```

Maar om te beginnen maken we enkel gebruik van een tellertje `$i`:

```

$i = 1;

while ($i <= 3) {
    echo $i;
    $i = $i + 1;
}
echo ' klaar!';

```

Bovenstaand script zal de getallen 1 t/m 3 en vervolgens klaar! afdrukken:

```
123 klaar!
```

In bovenstaand voorbeeld wordt een teller variabele op de waarde 1 gezet. In de conditie van de `while` lus wordt gecontroleerd of de teller kleiner of gelijk is aan 3. Dat is bij de eerste lusdoorgang natuurlijk het geval. In de lus wordt dan de waarde van de teller weggeschreven (maar uiteraard kunt u hier ook iets heel anders doen). Daarna wordt de teller met 1 opgehoogd en wordt de conditie opnieuw gecontroleerd. De waarde van de teller is nu 2 en dat is nog steeds kleiner of gelijk aan 3. Dus wordt de code in de lus opnieuw uitgevoerd. Er wordt 2 weggeschreven en de teller wordt opgehoogd naar 3. Dat is niet kleiner maar nog wel steeds gelijk aan 3. Dus wordt 3 weggeschreven en de teller opgehoogd naar 4. Nu wordt er niet langer aan de conditie voldaan en wordt de code in de lus niet langer herhaald.

3.19.1.1 Ophogen/verlagen van de teller

Het ophogen (of verlagen) van de teller met 1 komt heel vaak voor en wordt vaak afgekort met behulp van de **increment** (verhogen) en **decrement** (verlagen) **toewijzingsoperatoren**:

operator	naam	gebruik	effect	vergelijkbaar met
<code>++</code>	increment	<code>\$i++;</code>	telt 1 op bij de waarde van variabele <code>\$i</code>	<code>\$i = \$i + 1;</code> <code>\$i += 1;</code>
<code>--</code>	decrement	<code>\$i--;</code>	trekt 1 af van de waarde van variabele <code>\$i</code>	<code>\$i = \$i - 1;</code> <code>\$i -= 1;</code>

De code wordt dan:

```

$i = 1;

while ($i <= 3) {
    echo $i;
    $i++;
}
echo ' klaar!';

```

3.19.1.2 Ontkenning

Merk op dat u ook kunt **testen op** het **niet waar** zijn van een conditie met behulp van de ontkenningsoperator `!`:

```

while (! conditie) {
    // code die herhaald moet worden zolang conditie NIET waar is
}

```

3.19.1.3 Do While

Bij een `while` lus wordt de code in het codeblok **geen enkele keer** uitgevoerd als gelijk bij het **begin** al **niet aan de conditie voldaan** wordt:

```
// deze lus wordt nooit uitgevoerd
$i = 4;
while ($i <= 3) {
    echo $i;
    $i++;
}
```

De `do ... while` lus is een **variant** waarbij de **lus minimaal één keer doorlopen wordt** ook al wordt niet aan de conditie voldaan. De syntax ziet er zo uit:

```
do {
    // code die minimaal één keer moet worden uitgevoerd
} while (conditie);
```

Denk aan de **puntkomma** aan het **einde**! Als we dat toepassen op het vorige voorbeeld:

```
// deze lus wordt toch een keer uitgevoerd
// ook al wordt niet aan de conditie voldaan
$i = 4;
do {
    echo $i;
    $i++;
} while ($i <= 3);
```

Dit geeft:

```
4
```

In een `do ... while` lus wordt de **conditie pas aan het eind getest** en dit garandeert dat de code in de lus minimaal één keer wordt uitgevoerd. Wanneer de conditie naar `false` evalueert stopt de lus alsnog. Dit is bijvoorbeeld handig voor tests die afhankelijk zijn van de uitkomst van de code in de lus.

3.19.1.4 Pas op voor oneindige lussen

Ergens in de te herhalen code moeten wel statements zijn opgenomen die de conditie kunnen veranderen, anders zou u een **oneindige lus** krijgen: de lus blijft zichzelf "eindeloos" herhalen omdat de conditie altijd waar blijft:

```
// PAS OP: oneindige lus!
$i = 1;
while ($i <= 3) {
    // dit codeblok wordt 'oneindig' keer uitgevoerd
    // als de waarde van $i hier niet verandert...
}
```

PHP zal het script overigens na verloop van tijd wel afbreken, namelijk als de `max_execution_time` bereikt is. Deze staat echter standaard op 30 seconden, weliswaar niet echt oneindig, maar probeert u het vooral te vermijden!

3.19.1.5 Uitvoer onder elkaar

Als u de getallen onder elkaar wilt tonen in een browser dan kunt u dat bijvoorbeeld met een `<br>` element doen:

```
$i = 1;

while ($i <= 3) {
    echo $i . '<br>';
    $i++;
}
echo ' klaar!';
```

Dit geeft:

```
1
2
3
klaar!
```

Maar in de broncode staat alles nog steeds achter elkaar:

```
1<br>2<br>3<br> klaar!
```

Dat is zeker bij wat complexere lussen erg lastig debuggen. Het is daarom beter om ervoor te zorgen dat ook in de HTML broncode de doorgangen netjes onder elkaar komen te staan.

3.19.1.6 De `\n` escape sequentie

U kunt dat doen met het al eerder besproken `\n`. Dit staat voor een escape sequentie die een New Line begint:

```
$i = 1;

while ($i <= 3) {
    echo $i . "<br>\n";
    $i = $i + 1;
}
echo 'klaar!';
```

De uitvoer in de browser blijft hetzelfde maar de broncode ziet er nu beter uit:

```
1<br>
2<br>
3<br>
klaar!
```

Let op: `\n` wordt alleen uitgevoerd in een string die wordt omsloten met **dubbele quotes** (zie 3.9.8).

3.19.1.7 De `PHP_EOL` constante

U kunt een new line ook realiseren met de ingebakken PHP constante `PHP_EOL`:

```
$i = 1;

while ($i <= 3) {
    echo $i . '<br>' . PHP_EOL;
    $i = $i + 1;
}
echo 'klaar!'
```

Deze constante geeft automatisch voor elk platform (Windows, linux/Unix, Mac) de juiste sequentie voor een nieuwe regel en u kunt hem ook eenvoudig achter een single quoted string plakken.

3.19.1.8 String opbouwen in de lus

U kunt er ook voor kiezen om in de lus een string op te bouwen om die later in een keer weg te kunnen schrijven:

```
$i = 1;

while ($i <= 3) {
    $msg = $i . '<br>';
    $i = $i + 1;
}

echo $msg;
```

De code hierboven geeft de volgende uitvoer:

```
3
```

Dat komt omdat de waarde van `$msg` steeds wordt overschreven in de lus. Als u alle waarden in de string wilt hebben dan moet u ze in de lus aan de oude waarde van de variabele "plakken" zodat de string steeds verder aangroeit:

```
$msg = $msg . $i . '<br>';
```

Dit soort constructies komt zo vaak voor dat er een shorthand notatie voor bestaat:

```
$msg .= $i . '<br>';
```

Bij de eerste lusdoorgang is `$msg` echter nog *undefined*. U moet de variabele voor de lus initialiseren. De code ziet er dan uiteindelijk zo uit:

```
$msg = ''; // lege string
$i = 1;

while ($i <= 3) {
    $msg .= $i . '<br>'; // concatenatie met shorthand operator
    $i = $i + 1;
}

echo $msg;
```

Dit geeft het gewenste resultaat:

```
1
2
3
```

3.19.1.9 Template syntax

U kunt voor een while lus ook de template syntax gebruiken, maar het voorbeeld met de teller is daar niet heel geschikt voor:

```
<?php $i = 1; ?>
<?php while ($i <= 3): ?>
    <?= $i ?><br>
<?php $i++; ?>
<?php endwhile; ?>
klaar!
```

Straks zullen we betere voorbeelden met `for` en `foreach` zien.

3.19.2 For

Met een `for` lus kunt u eveneens code een aantal keren uitvoeren. De `for` lus staat door zijn gebruik van tellers ook wel bekend als **counting loop**. De algemene notatie voor een `for` lus is:

```
for (start-expressie; conditie; einde-lus-expressie) {  
    // de te herhalen code  
}
```

De start-expressie (initialisatie) wordt één keer uitgevoerd, aan het begin van de lus. Meestal wordt hiermee aan een teller een startwaarde gegeven. Daarna wordt de conditie geëvalueerd. Als deze `true` oplevert wordt de te herhalen code uitgevoerd. Hierna wordt de einde-lus-expressie uitgevoerd. Meestal wordt hier de teller verhoogd of verlaagd. Hierna begint de lus weer opnieuw met het controleren van de conditie. Zolang deze nog waar blijft zal de lus steeds opnieuw worden doorlopen. Een voorbeeld:

```
for ($i = 1; $i <= 3; $i++){  
    echo $i;  
}
```

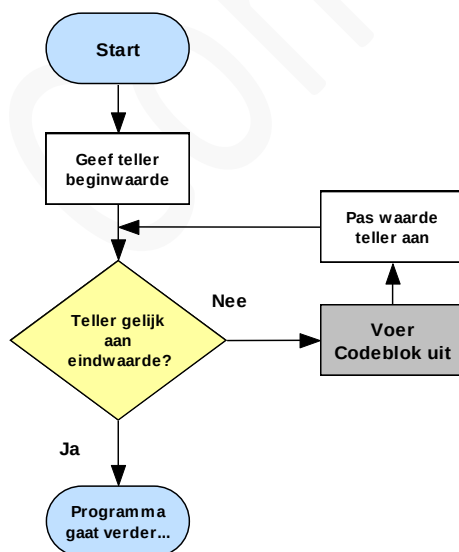
Bovenstaand script zal de getallen 1 t/m 3 afdrukken:

123

Het werkt als volgt: Een variabele `$i` wordt geïnitieerd met de waarde 1. De waarde hiervan is kleiner dan de eindwaarde 3 en dus wordt de code binnen de lus uitgevoerd. In dit geval wordt simpelweg de waarde van `$i` naar het scherm geschreven.

Na het bereiken van de `}` wordt de lus opnieuw ingegaan en hierbij wordt `$i` met 1 opgehoogd, want `$i++` is een verkorte notatie voor `$i = $i + 1`. Dit wordt in de praktijk heel vaak zo geschreven. Op dezelfde manier mag u ook `$i--` gebruiken in plaats van `$i = $i - 1`.

Variabele `$i` heeft nu de waarde 2 en dat is nog steeds lager dan de waarde in de einde-check. De code wordt dus nogmaals uitgevoerd. Na doorloop is de waarde 3. Hiermee wordt de lus voor de laatste maal uitgevoerd. `$i` is nu immers gelijk aan de eindwaarde en de lus wordt daarom verlaten! In een flowchart:



Merk op dat een `for` lus te herschrijven is als een `while`:


```
for ($i = 1; $i <= 3; $i++) {
    echo $i;
}
```

Deze doet precies hetzelfde als:

```
$i = 1;
while ($i <= 3) {
    echo $i;
    $i++;
}
```

Oefening 3.5a *Maak een pagina fontloop.php die het volgende toont:*

6

5

4

3

2

1

Als bovenstaande oefening gelukt is en u nog tijd over heeft dan kunt u de onderstaande verdiepingsoefening(en) maken. U vindt de uitwerking achter in dit cursusboek in de bijlagen:

oefening 3.5b *Maak een variant op bovenstaande pagina die u headingloop.php noemt en die met H1 t/m H6 werkt.*

3.19.2.1 Compact noteren

Hoewel de meeste coding style guides voorschrijven dat u de ruimte om de operators ook in de for lus zou moeten eerbiedigen, kiezen veel programmeurs toch liever voor een wat compactere notatie. Dus in plaats van:

```
for ($i = 1; $i <= 3; $i++) {
    echo $i;
}
```

wordt het dan:

```
for ($i=1; $i<=3; $i++){
    echo $i;
}
```

Dit benadrukt iets beter de 3 verschillende onderdelen. Aan u de keuze of u streng in de leer wilt zijn of niet!

Tijdens de cursus zullen we de notaties waarschijnlijk door elkaar gebruiken... Al zou het natuurlijk beter zijn om in ieder geval consequent te zijn!

3.19.2.2 Meerdere tellers

Met for lussen is qua syntax heel veel mogelijk. U kunt delen weglaten of juist meerdere tellers gebruiken. Deze kunnen elk hun eigen conditie en einde-lus expressie hebben:

```
for ($i=1, $j=10; $i<=10, $j>0; $i++, $j=$j-2) {  
    echo 'i=' . $i . ' & ' . $j . '<br>';  
}
```

In het voorbeeld lopen twee tellers. Een teller `$i` wordt in stappen van 1 opgehoogd naar 10 terwijl de teller `$j` in stappen van 2 verlaagd wordt. De lus stopt wanneer er niet langer aan een van de twee condities wordt voldaan:

```
i=1 & j=10  
i=2 & j=8  
i=3 & j=6  
i=4 & j=4  
i=5 & j=2
```

NB: Stel dat u teller `$i` helemaal tot 10 wilt laten lopen dan kunt u de condities samenvoegen tot `$i<=10 || $j>0`. Teller `$j` loopt dan door tot `-8`.

3.19.2.3 For lus voor Arrays

Bij een numeriek geïndexeerde array die een bekend aantal elementen bevat, kunt u een `for` lus gebruiken om de inhoud van de array naar het scherm te schrijven:

```
$weekend = ['Zaterdag', 'Zondag']; // 2 elementen  
  
for ($i=0; $i<2; $i++){  
    echo $weekend[$i] . "<br>\n";  
}
```

Het is hierbij uiteraard belangrijk ervoor te zorgen dat de teller op 0 begint en dat deze niet hoger kan worden dan de waarde van de hoogste index in de array.

3.19.2.4 Count/SizeOf

Maar wat nu als u niet van tevoren weet hoe groot een array `$arr` met een numerieke index is of wat de hoogste index is?

Met `count($arr)` of het alias `sizeof($arr)` kunt u het aantal elementen in de array bepalen en er met een `for` lus door heen lopen:

```
for ($i=0; $i<count($arr); $i++){  
    echo $arr[$i] . "<br>\n";  
}
```

Een `for` lus is echter **sneller** als u eerst de **grootte** van de array **buiten de lus** berekent, zodat het niet bij elke lusdoorgang opnieuw berekend hoeft te worden (terwijl het toch niet verandert):

```
$size = count($arr);
for ($i=0; $i<$size; $i++){
    echo $week[$i] . "<br>\n";
}
```

Een andere manier om hetzelfde te noteren is om de variabele voor de grootte net als de teller te initialiseren:

```
for ($i=0, $size=count($arr); $i<$size; $i++){
    echo $week[$i] . "<br>\n";
}
```

Ook nu wordt de `count()` functie maar 1x uitgevoerd! Let er wel op dat u een **komma** gebruikt om de twee variabelen te scheiden en geen punt-komma!

3.19.3 Foreach

Een `for` lus werkt uitstekend op numeriek geïndexeerde arrays waarbij de elementen opeenvolgend doorgenummerd zijn. Maar wat als er elementen ontbreken? Of als u een associatieve array wilt doorlopen?

Vanaf PHP4 kunt u dergelijke arrays eenvoudig als een *collection* uitlezen met een `foreach` lus. Deze is niet alleen makkelijker in het gebruik maar ook nog eens sneller! De standaard opbouw is als volgt:

```
foreach (array as $value) {
    // statements die $value gebruiken...
}
```

Hoe u de variabele **\$value**, waarmee u de waarde van elk element kunt uitlezen, wilt noemen, mag u zelf weten, maar het is handig om daar een betekenisvolle naam voor te gebruiken, bijvoorbeeld:

```
$weekend = ['za' => 'zaterdag', 'zo' => 'zondag'];

foreach ($weekend as $dag) {
    echo $dag . "<br>\n";
}
```

Deze code zet de dagen onder elkaar:

```
zaterdag
zondag
```

Wanneer u in de lus ook de waarde van de key wilt uitlezen, dan maakt u gebruik van de volgende notatie:

```
foreach (array as $key => $value) {
    // statements die zowel $key als $value gebruiken...
}
```

De lus hierboven zouden we dan bijvoorbeeld zo kunnen herschrijven:

```
$weekend = ['za' => 'zaterdag', 'zo' => 'zondag'];

foreach ($weekend as $afkorting => $dag) {
    echo $afkorting . ':' . $dag . "<br>\n";
}
```

Deze lus drukt zowel de waarden (de dagen van de week) als de indicie (de afkortingen) af:

```
za: zaterdag
zo: zondag
```

De naamgeving van de variabelen kunt u uiteraard weer helemaal zelf bepalen...

3.19.3.1 Foreach voor numeriek geïndexeerde arrays

Merk op dat u een `foreach` ook gewoon op een numeriek geïndexeerde array kunt gebruiken:

```
$weekend = ['zaterdag', 'zondag'];

foreach ($weekend as $dag) {
    echo $dag . "<br>\n";
}
```

Deze lus print gewoon (en heel leesbaar) de waarden (de dagen van de week):

```
zaterdag
zondag
```

Als u per se zou willen zou u ook de numerieke keys kunnen printen met de eerder besproken `as` notatie:

```
$weekend = ['zaterdag', 'zondag'];

foreach ($weekend as $i => $dag) {
    echo $i . ':' . $dag . "<br>\n";
}
```

Deze lus drukt zowel de waarden (de dagen van de week) als de indicie (de numerieke keys) af:

```
0: zaterdag
1: zondag
```

Maar dat is meestal niet zo heel nuttig... Die numerieke keys waren bij de `for()` lus noodzakelijk om te gebruiken maar bij een `foreach()` heeft u er meestal niet zoveel aan...

Als u de items wilt nummeren voor weergave wilt u ze meestal ook bij 1 laten beginnen. Dat zou zo kunnen:

```
foreach ($weekend as $i => $dag) {
    echo $i + 1 . ':' . $dag . "<br>\n";
}
```

of eventueel zo:

```
$i = 1;
foreach ($weekend as $dag) {
    echo $i . ':' . $dag . "<br>\n";
    $i++;
}
```

Maar in HTML kunt u het eenvoudig met een ordered list oplossen:

```
echo "<ol>\n";
foreach ($weekend as $dag) {
    echo "    <li>$dag</li>\n";
}
echo "</ol>\n";
```

Mooier in template syntax:

```
<ol>
<?php foreach ($weekend as $dag): ?>
    <li><?= $dag ?></li>
<?php endforeach; ?>
</ol>
```

In beide gevallen krijgt u:

1. zaterdag
2. zondag

Oefening

Maak in de map oefeningen/ een pagina `array.php` aan waarin u een associatieve array `$medewerker` aanmaakt die de gegevens (voornaam, achternaam, afdeling en toestelnummer) van één medewerker bevat.

Loop vervolgens met een `foreach` lus door de array heen om de gegevens als volgt in een tabel te tonen:

Medewerker Array

voornaam:	Arno
achternaam:	Mol
afdeling:	Opleidingen
toestelnummer:	1234

3.19.3.2 Alleen keys uitlezen

Als u **alleen** de **keys** zou willen **uitlezen** dan kunt u dat op bovenstaande manier doen maar dan in de lus de value niet gebruiken:

```
foreach ($weekend as $afkorting => $dag) {
    echo $afkorting . "<br>\n";
}
```

3.19.3.3 `array_keys()`

Een andere oplossing is om met de functie `array_keys()` de keys van een associatieve array in een (numerieke) array te zetten en daar doorheen te lopen:

```
foreach (array_keys($weekend) as $afkorting => $dag) {
    echo $afkorting . "<br>\n";
}
```

3.19.3.4 `While/List/each`

Voor PHP4 bestond de `foreach` lus nog niet. De gebruikelijke manier om de waarden uit een associatieve array uit te lezen was in die tijd een combinatie van een `while` lus met de `list()` en `each()` functies:

```
while (list($key, $value) = each($week)) {  
    echo "$key: $value<br>\n";  
}
```

De `each()` functie retourneert de huidige index en de bijbehorende waarde in de array en zet de interne pointer daarna een element verder. Door de `while` lus blijft dit zo lang doorgaan totdat het eind van de array bereikt is, want dan retourneert `each()` namelijk de waarde `false`.

De huidige index en de bijbehorende waarde worden door `each()` als een array aangeleverd. Met de `list()` functie zorgt u ervoor dat deze waarden als gewone variabelen in de lus beschikbaar worden gesteld. Vergelijkbare functionaliteit is ook mogelijk met de `extract()` functie die we later nog zullen tegenkomen...

Dit alles geeft uiteindelijk dezelfde uitvoer als de `foreach` lus hierboven:

```
zo: zondag  
ma: maandag  
...
```

De code is alleen iets lastiger te noteren en bij grote arrays ook wat trager. We laten deze notatie dan ook vooral zien zodat u hem kunt duiden wanneer u hem in de (erg oude) code van anderen tegenkomt...

Als u overigens alleen de waarde en niet de index wilt hebben dan kunt u de `$key` variabele gewoon weg laten:

```
while (list(, $value) = each($week)) {  
    echo "$value<br>\n";  
}
```

Met als resultaat:

```
zondag  
maandag  
...
```

En omgekeerd kunt u ook wanneer u alleen de keys zou willen zien, de waarden weg laten:

```
while (list($key, ) = each($week)) {  
    echo "$key<br>\n";  
}
```

Dit geeft:

```
zo  
ma  
...
```

Een ander nadeel van deze constructie is dat u dit niet twee keer achter elkaar kunt doen (wat bij een `foreach` lus geen probleem is...). Dat komt omdat de interne pointer door `each` aan het einde van de array is gezet.

Het kan wel als u de `reset()` functie gebruikt om de interne pointer weer aan het begin van de array te zetten:

```
while(list($key, $value) = each($week)) {
    echo "$key: $value<br>\n";
}

reset($week);

while(list($key, $value) = each($week)) {
    echo "$key: $value<br>\n";
}
```

3.19.3.5 foreach with range()

U kunt als u zou willen een `foreach` lus als een `for` lus inzetten en deze dus een bepaald aantal keer laten lopen. Een reden zou (micro) optimalisatie kunnen zijn, maar dat heeft voor kleine aantallen eigenlijk geen zin. Sommige programmeurs vinden het makkelijker leesbaar/aanpasbaar, maar daar valt ook over te discussiëren want `for` lussen zijn voor de meesten toch herkenbaarder...

Anyways, dergelijke lussen zien er zo uit en maken gebruik van de standaard PHP `range()` functie die een numeriek geïndexeerde array met opeenvolgende waarden tussen het begin en eindpunt retourneert:

```
foreach (range(1, 3) as $i) {
    echo $i, '<br>';
}
```

Wat dus overeenkomt met een `for` lus die u zo schrijft:

```
for ($i=1; $i<=3; $i++){
    echo $i, '<br>';
}
```

Het is waarschijnlijk een kwestie van gewenning dan wel smaak welke u makkelijker vindt lezen en wellicht dat u de `range()` variant nooit gaat gebruiken, maar dan is het handig om te herkennen als u het in code van anderen tegenkomt.

3.19.4 Lussen combineren met if ... else

Uiteraard kunt u binnen een lus controle structuren als `if else` gebruiken om bepaalde condities af te vangen:

```
for ($i=1; $i<=3; $i++) {
    if ($i == 2) {
        echo "<strong>" . $i . "</strong><br>\n";
    } else {
        echo $i . "<br>\n";
    }
}
```

Dit geeft:

```
1<br>
<strong>2</strong><br>
3<br>
```

En dit kan natuurlijk ook weer met de template syntax...

3.19.5 comma seperated values

Wanneer u de elementen uit een array komma gescheiden in een string wilt zetten, zou u zoiets kunnen doen:

```
$dagen = '';
foreach ($week as $dag) {
    $dagen .= $dag . ', ';
}
echo $dagen;
```

Dit geeft echter:

Zondag, Maandag, Dinsdag, Woensdag, Donderdag, Vrijdag, Zaterdag,

U kunt de spatie en komma er dan simpelweg 'afknabbelen':

```
$dagen = '';
foreach ($week as $dag) {
    $dagen .= $dag . ', ';
}
$dagen = substr($dagen, 0, -strlen(', '));
echo $dagen;
```

Of ervoor zorgen dat het sowieso niet werd toegevoegd:

```
$dagen = '';
$first = true;

foreach ($week as $dag) {
    if ($first) {
        $dagen .= $dag;
        $first = false;
    } else {
        $dagen .= ', ' . $dag;
    }
}
```

Of wat compacter:

```
$dagen = '';
$first = true;

foreach ($week as $dag) {
    $dagen .= ($first) ? $dag : ', ' . $dag;
    $first = false;
}
```

Oefening 3.5c ***Maak een for lus die van 1 t/m 5 loopt en 1, 2, 3, 4, 5! toont. Hint: Gebruik een if/else statement om te bepalen of een komma en een spatie of het uitroepteken moet worden getoond.***

3.19.6 Break en continue

U kunt het verloop van een lus beïnvloeden met de twee speciale keywords `break` en `continue`. U bent `break` al eerder tegengekomen bij de bespreking van de `switch` waar het diende om de `switch` na een gematchde `case` te verlaten. U kunt `break` ook gebruiken om gelijk na een gematchde conditie een lus te verlaten:

```
for ($i=1; $i<=5; $i++) {
    if ($i == 3) {
        break;
    }
    echo $i . "<br>\n";
}
```


Dit geeft enkel:

```
1
2
```

Met `continue` verlaat u de huidige lusdoorgang om verder te gaan met de volgende:

```
for ($i=1; $i<=5; $i++) {
    if ($i == 3) {
        continue;
    }
    echo $i . "<br>\n";
}
```

De lus wordt dus niet gelijk beëindigd maar loopt door tot aan de eindconditie is voldaan. Bovenstaande code geeft:

```
1
2
4
5
```

3.19.7 Nesten

Ook lussen kunnen genest worden. U neemt dan binnen een lus (de buitenste lus) een nieuwe lus op (de binnenste lus). U doet dit bijvoorbeeld wanneer u een raster of tabel wilt genereren. Denk bijvoorbeeld aan een vermenigvuldigingstafel:

```
<?php
// begin buitenste lus
for ($i=1; $i<=3; $i++) {
    // begin binnenste lus
    for ($j=1; $j<=3; $j++) {
        // schrijf resultaten weg op een regel
        echo $i * $j . " ";
    }
    // sluit de regel af met een HTML break
    echo "<br>\n";
}
?>
```

Dit geeft in de browser:

```
1 2 3
2 4 6
3 6 9
```

Bij elke doorgang van de buitenste lus wordt de binnenste lus geheel doorlopen. Dus bij de eerste doorgang heeft `$i` een waarde van 1 terwijl `$j` van 1 tot 3 loopt. Het resultaat van hun vermenigvuldiging wordt weggeschreven, gevolgd door een spatie om te voorkomen dat ze tegen elkaar aan zouden komen te staan.

Nadat de binnenste lus in zijn geheel is doorlopen (3 doorgangen) wordt een *HTML break element* weggeschreven die ervoor zorgt dat bij de volgende doorgang van de buitenste lus (`$i` is nu 2) de uitvoer op de volgende regel begint.

Een geneste lus kan overigens uit meer dan 2 lussen bestaan. Maar meer dan 3 wordt al snel lastig te volgen. Zeker als u ze combineert met break statements. Onthoud in ieder geval dat een `break` alleen effect heeft op de dichtstbijzijnde lus waarin het gebruikt wordt.

Verdieping	<i>Schrijf een script <code>tabel.php</code> dat met behulp van twee geneste lussen een HTML tabel met 3 (kolommen) bij 4 (rijen) schrijft. De uitvoer zou zo iets kunnen zijn:</i>

3.20 Gestructureerd programmeren

Met de tot nu toe behandelde controlestructuren kunnen ook de meest complexe problemen worden opgelost. De kunst bestaat er dan wel uit het probleem te **analyseren** en op te delen in kleine **deelproblemen**. Deze zijn dan altijd weer op te delen in **sequenties** (opeenvolgingen), **selecties** (vertakkingen) en **iteraties** (herhalingen). Dit is één van de belangrijkste stappen op weg naar gestructureerd programmeren.

Zeker bij complexe problemen kan het handig zijn om het verloop van een programma eerst door middel van **pseudo-code**, **flowcharts** of **programmastructuurdiagrammen (PSD's)** in kaart te brengen. Het is een veelgemaakte fout om zonder eerst na te denken gelijk aan de slag te gaan. U kunt zich beter eerst op de logica concentreren. Pas als u het probleem opgelost hebt en correct in kaart hebt gebracht gaat u daadwerkelijk programmeren.

Een dergelijke werkwijze heeft als bijkomend voordeel dat het **taalonafhankelijk** is. U kunt met de oplossing naast uw toetsenbord het programma in elke taal die u beheerst gaan coderen.

We hebben hier helaas niet de mogelijkheid om op deze technieken in te gaan, maar de volgende tip willen we u niet onthouden:

3.20.1 Pseudo-code

Door simpelweg in gewone bewoordingen de instructies op te sommen die volgens u nodig zijn voor de oplossing van een probleem, kunt u al een heel eind komen bij het oplossen van de problemen die u bij het programmeren tegen komt.

Deze manier van werken is ook goed toe te passen door mensen die het liefst zo snel mogelijk willen gaan typen. Door de instructies als commentaar op te nemen en tussendoor de benodigde code te typen heeft u snel een (gedocumenteerd) programma.

3.21 Wiskundige operatoren

Regelmatig zullen we in PHP pagina's wiskundige operatoren nodig hebben. Denk daarbij maar eens aan de volgende toepassingen:

- BTW berekeningen
- Staffelprijzen
- Leeftijdbepaling
- Hypotheekberekeningen
- etc...

Als eerste een overzicht met de notatiewijze van de verschillende operatoren zoals ze in PHP toegepast kunnen worden.

Rekenfunctie	Operator	Voorbeeld
Optellen	+	3 + 2 // geeft 5
Aftrekken	-	3 - 2 // geeft 1
Vermenigvuldigen	*	3 * 2 // geeft 6
Delen	/	3 / 2 // geeft 1.5 4 / 2 // geeft 2
Modulus (restwaarde na deling)	%	3 % 2 // geeft 1 4 % 2 // geeft 0
Negatief	-	-3

Met de meeste hiervan zult u wel vertrouwd zijn. Iets lastiger is de **integer modulus** oftewel de **modulo operator**. Deze geeft de restwaarde na deling

Een bewerking als `3 % 2` gaat als volgt: Als u 3 deelt door 2, gaat dat precies 1x, wat u dan overhoudt is 1. Dit wordt de restwaarde na deling genoemd. Als u `4 % 2` doet dan gaat het precies 2x en houdt u niets over. De restwaarde is 0. De modulus operator wordt vaak op deze manier gebruikt om te kijken of een getal even of oneven is:

```
if ($recordnumber % 2 == 0) {
    // dit is een even rij, doe bijvoorbeeld iets
    // leuks met de opmaak van de rij...
}
```

De modulo operator werkt alleen op integers. Als u floating point values zou willen gebruiken dan worden deze voordat de berekening wordt uitgevoerd naar integers door de waarde achter de komma weg te laten. Iets als `5.7 % 2` levert dus nog steeds 1 op omdat `5 % 2` wordt uitgevoerd.

Als u met floating point values wilt werken dan kunt u de functie `fmod()` gebruiken:

```
echo fmod(5.7, 2); // 1.7
echo fmod(5.7, 1.3); // 0.5, 4 * 1.3 + 0.5 = 5.7
```

3.21.1 Operator precedence

Bij het werken met meerdere operatoren in een statement is de zogenaamde *operator precedence*, zeg maar de volgorde waarin de berekeningen worden uitgevoerd, van belang.

In PHP gelden eigenlijk dezelfde regels als bij wiskunde (**Meneer van Dalen wacht op antwoord**: eerst machtsverheffen, dan vermenigvuldigen, delen, worteltrekken, optellen en als laatste aftrekken).

Met haakjes kunt u die volgorde eventueel anders afdwingen, omdat altijd eerst de bewerkingen tussen de haakjes worden uitgevoerd:

```
$a = 10;
$b = 5;
$c;

$c = $a * $b + 25/5; // totaal = 55
$c = $a * (($b + 25)/5); // totaal = 60
```

In de bovenste berekening worden de normale regels gevolgd: eerst de vermenigvuldiging `10 * 5` wat 50 oplevert, daarna de deling `25/5` wat 5 oplevert, waarna beiden bij elkaar worden opgeteld: `50 + 5`.

In de tweede berekening wordt eerst het deel tussen de binnenste haakjes berekend: `5 + 25`. Dat is 30, daarna wordt dit door 5 gedeeld vanwege het tweede haakjespaar. Tenslotte wordt deze uitkomst 6 met 10 vermenigvuldigd.

Opmerking

Net als bij concatenatie bestaan er ook verkorte notaties voor *optellen*, *afrekken*, *delen* en *vermenigvuldigen*:

```
+=  
-=  
/=   
*=
```

Zo wordt met deze code de waarde van een variabele `$i` met 1 opgehoogd:

```
$i += 1 // is hetzelfde als $i = $i + 1
```

In lussen ziet u dat vaak nog compacter:

```
$i++ // is hetzelfde als $i = $i + 1
```

Verdieping	<i>Schrijf een PHP script dat voor ons uitrekent hoeveel rente er in totaal betaald is aan het einde van de looptijd van een lineaire hypotheek met een waarde van € 300.000. De looptijd is 30 jaar en de rente is 7 %.</i>
Verdieping	<i>Laat de waarden voor elk jaar in een HTML tabel zien.</i>
Verdieping	<i>Wissel de achtergrondkleur van de tabelrijen om. Hint: maak gebruik van de modulus operator om te testen of u te maken heeft met een even of een oneven rij.</i>

Een script als hierboven heeft natuurlijk maar beperkt nut. Veel fraaier zou het zijn wanneer een eindgebruiker zelf de waarden kan opgeven waarop de berekening moet worden uitgevoerd. Dat kan natuurlijk ook, maar daarvoor moeten we eerst weten hoe u in PHP met formulieren werkt. En laat dat nou net het onderwerp zijn van het volgende hoofdstuk...

3.22 PHP Template Syntax

U kunt de meeste controle structuren ook op een alternatieve manier noteren. Deze manier wordt de **PHP Template Syntax** genoemd omdat hij voor het gebruik in templates is bedoeld. Sommigen hebben het ook over de **alternative colon syntax** omdat er een dubbele punt (colon) wordt gebruikt in plaats van de openingsaccolade. De PHP manual heeft het simpleweg over een **alternative syntax for control structures...**

U ziet deze manier van noteren vaak in combinatie met een **minimale PHP stijl** waarbij **zoveel mogelijk standaard HTML** gecombineerd wordt met **zo weinig mogelijk PHP**.

Het is ook de manier waarop standaard in **Drupal**, **Joomla!** en **WordPress** templates gewerkt wordt.

In plaats van de beginaccolade gebruikt u dan een **dubbele punt** en in plaats van de eindaccolade herhaalt u de controlestructuur met `end` ervoor en sluit u (in principe) af met een **puntkomma** `;`.

NB.: Zoals eerder besproken is de **puntkomma** in deze situatie **optioneel**. De eind delimiter sluit het statement immers sowieso al af... Maar het is goed gebruik om hem wel te noteren!

Het **voordeel** is dat u (of diegene die de HTML moet onderhouden) dan **duidelijker** kan zien **waar** een **controlestructuur begint** en **ophoudt**:

```
<table border="1">  
<?php for ($i=1; $i <= 5; $i++) ?>
```

```

<tr>
  <?php if ($i == 5): ?>
    <td><strong>cel <?php echo $i; ?></strong></td>
  <?php else: ?>
    <td>cel <?php echo $i; ?></td>
  <?php endif; ?>
</tr>
<?php endfor; ?>
</table>

```

In de normale notatie voor `if` `()` zet u een **spatie** achter de conditie, voor de openingsaccolade:

```

<?php
...
if (...) {
  ...
}
...
?>

```

In de template notatie is het gebruikelijk om dit **niet** te doen voor de dubbele punt:

```

<?php if (...) : ?>
  ...
<?php endif; ?>

```

Dit is in ieder geval hoe het in de PHP manual wordt gedaan op <https://www.php.net/manual/en/control-structures.alternative-syntax.php> en het sluit wel mooi aan bij hoe de puntkomma gelijk achter `endif;` wordt gezet (ook al is dat niet echt nodig...). Coding Conventions als <https://www.php-fig.org/psr/psr-12/> gaan niet in op deze manier van noteren maar in het geval van **typed return values** (zie het onderdeel over **functies**) zetten ze de **dubbele punt ook gelijk achter de ronde haken** van de parameter list in de function definition...

Drupal noemt het wel expliciet in hun coding conventions:

In templates, the alternate control statement syntax using `:` instead of brackets is allowed. Note that there should **not be a space** between the closing parenthesis after the control keyword, and the colon, and HTML/PHP inside the control structure should be **indented**. For example:

```

<?php foreach ($items as $item): ?>
  <p><?php print $item; ?></p>
<?php endforeach; ?>

```

Zie: <https://www.drupal.org/docs/develop/standards/coding-standards#s-alternate-control-statement-syntax-for-templates>

Een **mogelijk nadeel** is dat in het vorige voorbeeld de **HTML uitvoer** erg **rommelig** wordt vanwege de extra witruimte. Als u dat belangrijk vindt kunt u de code eventueel ook zo noteren:

```

<table border="1">
  <?php for ($i=1; $i <= 5; $i++): ?>
    <tr>
      <?php if ($i == 5): ?>
        <td><strong>cel <?php echo $i; ?></strong></td>
      <?php else: ?>
        <td>cel <?php echo $i; ?></td>
      <?php endif; ?>
    </tr>
  <?php endfor; ?>
</table>

```

Dus dan springen HTML en PHP op hun eigen niveau in. En PHP doet dat in script delimiters die tegen de kantlijn aan beginnen.

Een ander nadeel kan zijn dat een (wat oudere) IDE de instructies misschien niet goed begrijpt. U kunt dan gewoon op accolades terugvallen. Accolades worden altijd begrepen en u kunt er precies hetzelfde mee doen:

```
<table border="1">
<?php for ($i=1; $i <= 5; $i++) { ?>
  <tr>
<?php if ($i == 5) { ?>
  <td><strong>cel <?php echo $i; ?></strong></td>
<?php } else { ?>
  <td>cel <?php echo $i; ?></td>
<?php } ?>
  </tr>
</tr>
<?php } ?>
</table>
```

Maar het is dus duidelijk minder leesbaar! Sommige richtlijnen verbieden daarom zelfs om accolades in templates te gebruiken. Probeer het in ieder geval zoveel mogelijk te voorkomen!

In beide gevallen is het resultaat natuurlijk wel exact hetzelfde:

cel 1
cel 2
cel 3
cel 4
cel 5

Welke notatie u wilt gebruiken, mag u natuurlijk geheel zelf beslissen, maar tijdens de cursus zullen we in principe binnen PHP blokken de **accolades** gebruiken **voor controlestructuren** en **binnen HTML templates** de **Template Syntax**. Het is hoogst ongebruikelijk om de alternatieve syntax in gewone PHP codeblokken te gebruiken (maar het zou wel gewoon werken).

Let op: Denk eraan dat u een `foreach:` lus afsluit met `endforeach;` Een `while:` lus sluit u met `endwhile;` etc...

3.22.1 Ternary Control operator

U kunt de ternary control operator ook gebruiken in templates. Dit is vooral handig voor korte stukken tekst. In plaats van het wollige:

```
huwelijkse staat:

<?php if ($getrouwd): ?>
  gehuwd
<?php else: ?>
  ongehuwd
<?php endif; ?>
```

Of slecht leesbare:

```

huwelijkse staat:
<?php if ($getrouwd): ?> gehuwd <?php else: ?> ongehuwd <?php endif; ?>

```

Kunt u ook heel compact noteren met de ternary control operator:

```

huwelijkse staat: <?= $getrouwd ? 'gehuwd' : 'ongehuwd' ?>

```

Als u echter conditioneel HTML wilt tonen, bijvoorbeeld voor afbeeldingen e.d. dan gebruikt u uiteraard wel de eerste notatie!

3.22.2 Switch

U kunt de template syntax in principe ook voor een switch gebruiken, maar het gaat helaas **niet** zo:

```

<!-- LET OP: Dit geeft een error -->
<?php switch ($beroep): ?>

    <?php case 'manager': ?>
        relevante productinfo voor managers
    <?php break; ?>

    <?php case 'developer': ?>
        relevante productinfo voor developers
    <?php break; ?>

    <?php case 'trainer': ?>
        relevante productinfo voor trainers
    <?php break; ?>

    <?php default: ?>
        standaard productinfo
    <?php break; ?>

<?php endswitch; ?>

```

Dit geeft:

Parse error: syntax error, unexpected ' ', expecting endswitch (T_ENDSWITCH) or case (T_CASE) or default (T_DEFAULT) in **demo.php**

Er mag namelijk **geen wit ruimte tussen** `switch()` : **en de eerste case** staan. Zo kan het wel:

```

<?php switch ($beroep): ?>
    <?php case 'manager': ?>
        relevante productinfo voor managers
    <?php break; ?>

    <?php case 'developer': ?>
        relevante productinfo voor developers
    <?php break; ?>

    <?php case 'trainer': ?>
        relevante productinfo voor trainers
    <?php break; ?>

    <?php default: ?>
        standaard productinfo
    <?php break; ?>
<?php endswitch; ?>

```

U kunt hier mogelijk nog wel wat aan verfraaien door einde en begin van opvolgende codeblokken samen te nemen:

```
<?php switch ($beroep):  
    case 'manager': ?>  
        relevante productinfo voor managers  
<?php        break;  
  
    case 'developer': ?>  
        relevante productinfo voor developers  
<?php        break;  
  
    case 'trainer': ?>  
        relevante productinfo voor trainers  
<?php        break;  
  
    default: ?>  
        standaard productinfo  
<?php        break;  
  
endswitch; ?>
```

Bovendien stellen sommige richtlijnen dat u alleen maar inline PHP met maximaal 1 statement mag gebruiken in templates en dus niet meerdere statements in een blok. Bovendien is het inspringen van de HTML wat minder fraai dan bij een `if ... elseif ... else`:

```
<?php if ($beroep == 'manager'): ?>  
    relevante productinfo voor managers  
<?php elseif ($beroep == 'developer'): ?>  
    relevante productinfo voor developers  
<?php elseif ($beroep == 'trainer'): ?>  
    relevante productinfo voor trainers  
<?php else: ?>  
    standaard productinfo  
<?php endif; ?>
```

Dit is dan ook de aanbevolen manier! Probeer switches in templates te voorkomen!

U vindt meer informatie over de alternatieve syntax in de manual op <http://php.net/manual/en/control-structures.alternative-syntax.php>.

3.22.3 Voorkom extra witruimte

Wanneer u de alternatieve syntax gebruikt dan kunt u overwegen om deze gelijk aan het begin van de regel te laten beginnen en niet in te laten springen met de HTML. Zo voorkomt u extra witruimte in de broncode. Als u er ook voor zorgt dat er geen witruimte na het einde van het PHP blok komt dan neemt de hele instructie zelfs helemaal geen ruimte in omdat de end of line dan door PHP wordt weggehaald... Uw broncode blijft zo veel netter en leesbaarder. Handig bij debuggen! Ook al wordt dit met de huidige Developer Tools in de browser wel minder belangrijk...

3.23 Andere uitvoer dan HTML

Standaard zal de uitvoer van een PHP script door een browser als HTML worden gezien en als zodanig worden geïnterpreteerd. Dit geldt ook als er helemaal geen HTML uitvoer wordt gegenereerd. U kunt dit snel controleren door een volkomen lege pagina met een `.php` extensie op te vragen. Als u de HTTP Response headers bekijkt met de Developer Tools dan ziet u dat het Content-type `text/html` is:

Response Headers

```

Connection Keep-Alive
Content-Length 0
Content-Type text/html; charset=UTF-8
Date Tue, 03 Sep 2013 21:02:44 GMT
Keep-Alive timeout=5, max=100
Server Apache/2.4.2 (Win32)

```

Dit gedrag wordt geregeld door de `default_mimetype` setting in `php.ini`:

```

; PHP's built-in default is text/html
; http://php.net/default-mimetype
default_mimetype = "text/html"

```

U kunt met PHP echter ook XML genereren. Of CSS, JavaScript, of JSON. Het probleem is echter dat dit als u geen maatregelen treft door de browser nog steeds als HTML wordt gezien.

3.23.1 XML

Als voorbeeld nemen we XML. Stel u heeft een array met cursussen en u wilt deze als XML wegschrijven:

```

<?php
    $cursussen = array('HTML', 'CSS', 'JavaScript', 'PHP');
?>
<?xml version="1.0" encoding="utf-8" ?>
<cursussen>
<?php foreach ($cursussen as $cursus): ?>
    <cursus><?php echo $cursus; ?></cursus>
<?php endforeach; ?>
</cursussen>

```

NB: Merk op dat `short_open_tag` in `php.ini` **uit** moet staan om zo te kunnen coderen. Als het aan zou staan struikelt de parser over `<?xml`:

Parse error: syntax error, unexpected 'version' (T_STRING) in **demo.php**

De oplossing is om dan de XML declaratie simpelweg te printen met een `echo` statement:

```
echo '<?xml version="1.0" encoding="utf-8" ?>' . PHP_EOL;
```

of in template syntax:

```
<?='<?xml version="1.0" encoding="utf-8" ?>'?>
```

Maar hoe u het ook aanpakt, de browser behandelt de gegenereerde XML nog steeds als HTML. De `<cursussen>` en `<cursus>` elementen worden niet herkend en daarom genegeerd. De uitvoer ziet er zo uit:

```
HTML CSS JavaScript PHP
```

Wat u moet doen is de browser vertellen dat er XML aan komt. U doet dat door met de `header()` functie het Content-type voor de HTTP header aan te passen:

```

<?php
    $cursussen = array('HTML', 'CSS', 'JavaScript', 'PHP');

```

```

header('Content-type: application/xml');
?>
<?xml version="1.0" encoding="utf-8" ?>
<cursussen>
<?php foreach ($cursussen as $cursus): ?>
    <cursus><?php echo $cursus; ?></cursus>
<?php endforeach; ?>
</cursussen>

```

Nu snapt de browser het wel:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

- <cursussen>
  <cursus>HTML</cursus>
  <cursus>CSS</cursus>
  <cursus>JavaScript</cursus>
  <cursus>PHP</cursus>
</cursussen>

```

Let op: Pas er voor op dat u **geen typefouten** maakt in de notatie voor het `Content-type`! Als deze niet exact correct is dan zal de browser het **bestandstype niet herkennen** als XML en het bestand vervolgens niet tonen maar **downloaden**!

3.23.2 JSON

Als u de uitvoer in JavaScript Object Notation (JSON) wilt tonen dan kunt u gebruik maken van de speciaal daarvoor bedoelde `json_encode()` functie. Denk eraan dat u nu een correct MIME type voor JSON opgeeft en dat u de uitvoer moet echoën:

```

<?php
$cursussen = [
    ['cursus' => 'HTML'],
    ['cursus' => 'CSS'],
    ['cursus' => 'JavaScript'],
    ['cursus' => 'PHP']
];

header('Content-type: application/json');
echo json_encode(['cursussen' => $cursussen]);
?>

```

Merk op dat we nu een 2-dimensionale array aanmaken met een beschrijvende key `'cursus'`. Bovendien plaatsen we die in de functie in een associatieve nieuwe array met de key `'cursussen'`, dit is gebruikelijke manier om de JSON data zichzelf te laten beschrijven. De uitvoer wordt dan:

```

{"cursussen": [{"cursus": "HTML"}, {"cursus": "CSS"}, {"cursus": "JavaScript"}, {"cursus": "PHP"}]}

```

De `json_encode()` functie heeft een optionele parameter `JSON_PRETTY_PRINT` waarmee u de uitvoer netter kunt printen:

```

...
echo json_encode(['cursussen' => $cursussen], JSON_PRETTY_PRINT);
?>

```

Dit geeft:

```
{
  "cursussen": [
    {
      "cursus": "HTML"
    },
    {
      "cursus": "CSS"
    },
    {
      "cursus": "JavaScript"
    },
    {
      "cursus": "PHP"
    }
  ]
}
```

Voor de aannemende code natuurlijk van geen belang, maar voor ons om het te controleren wel! U ziet nu wat duidelijker dat u een object met de key cursussen terugkrijgt en dat deze een array van cursus objecten bevat. Misschien nu niet zo spectaculair, maar als het gaat om een array van objecten met meer data (bijvoorbeeld medewerker records uit een database) dan helpt deze weergave zeker!

Firefox heeft sowieso een nette manier om JSON te tonen:



De XML en JSON uitvoeren zijn zeer handig in combinatie met de database toegang om zo AJAX applicaties te kunnen bouwen. De data kan dan vanuit een HTML pagina met behulp van JavaScript en door het XMLHttpRequest object worden opgehaald, maar even goed door een in C# of Java geschreven applicatie. Dit is ook de manier waarop veel mobiele apps met het back-end communiceren.

U leert meer over het werken met AJAX in de **JavaScript** en **jQuery cursussen**.

3.23.3 Plain Text

Wat soms ook best handig kan zijn is om voor kleine experimenten als uitvoer platte tekst te gebruiken. Dan kunt u zonder HTML te hoven gebruiken tosch snel leesbare uitvoer krijgen omdat geregeleinden en witruimte als tabs en spaties dan wel werken zoals u verwacht.

Een geneste lus als deze

```
<?php
$rows = 3;
$cols = 5;
```

```
for ($i = 1; $i <= $rows; $i++) {
    for ($j = 1; $j <= $cols; $j++) {
        echo "x";
    }
    echo "\n";
}
```

Wordt in een browser standaard zo getoond:

```
XXXXX XXXXX XXXXX
```

Voor de tabel opbouw zou de page source moeten bekijken. Maar als u er simpelweg dit boven zet:

```
<?php
header('Content-type: text/plain');

$rows = 3;
$cols = 5;

for ($i = 1; $i <= $rows; $i++) {
    for ($j = 1; $j <= $cols; $j++) {
        echo "x";
    }
    echo "\n";
}
```

Dan krijgt u gelijk het gewenste resultaat te zien:

```
XXXXX
XXXXX
XXXXX
```

Een alternatief is de **commandline** te gebruiken. Deze negeert Content-type volledig:

```
$ php for-nested.php
XXXXX
XXXXX
XXXXX
```

3.24 Coding Conventions

Met coding conventions worden de afspraken bedoeld die in een team gemaakt worden over het noteren van de code. En niet alleen over het noteren van de code, maar ook over de organisatie van bestanden en de naamgeving van bestanden en variabelen, het gebruik van witruimte, etc.

Dergelijke conventies zijn belangrijk vanwege de

- Leesbaarheid
- Aanpasbaarheid
- Uitwisselbaarheid

De meeste kosten en tijd van een applicatie zitten namelijk niet in de ontwikkeling, maar in het onderhoud. Netjes ingedeelde code en consistente code leest een stuk prettiger, en is daardoor makkelijker aan te passen, ook door andere programmeurs. **Code wordt veel vaker gelezen dan geschreven! Leesbaarheid is echt belangrijk!**

Het is daarom van groot belang duidelijke afspraken te maken over de wijze waarop u uw code noteert! Als u met meerdere mensen aan een project werkt, is het duidelijk dat een eenduidige notatie een stuk prettiger werkt en er voor zorgt dat niet iedereen steeds de code van de anderen aan het aanpassen is naar de eigen indeling. Maar ook als u alleen werkt zorgt een eenduidige notatie voor duidelijkheid in de code.

Het is handig om die afspraken in een **stijlgids** op te nemen, compleet met code voorbeelden. U kunt zo'n gids helemaal vanaf nul opzetten, maar u kunt ook een bestaande nemen en deze eventueel naar eigen inzicht aanpassen. Dat is wel zo slim want het scheelt een hoop tijd en wellicht bestaan er zelfs tools om de code te controleren. U zou zelfs een C of Java stijlgids kunnen nemen vanwege de grote gelijkenissen met de PHP syntaxis.



Een bekende PHP stijlgids is die van **PEAR**, de **PHP Extension and Application Repository**:

<http://pear.php.net/manual/en/standards.php>

De code in dit cursusboek is er op gebaseerd.

3.24.1 Controle structuren

Een van de afspraken in PEAR is om codeblokken **K&R style** te noteren (naar Kernighan en Ritchie, de auteurs van het boek *The C Programming Language*):

```
$x = 1;
while ($x <= 10) {
    if ($x % 2 == 0) {
        echo "$x is even<br>\n";
    } else {
        echo "$x is odd<br>\n";
    }
    $x++;
}
```

De begin accolades starten dus op dezelfde regel als het if, else, while, etc.. commando en dat scheelt een hoop regels...

De K&R style wordt door de aanhangers ook wel de *One True Bracing Style* genoemd maar veel programmeurs noteren toch liever in de **Allman/BSD style**:

```
$x = 1;
while ($x <= 10)
{
    if ($x % 2 == 0)
    {
        echo "$x is even<br>\n";
    }
    else
    {
        echo "$x is odd<br>\n";
    }
    $x++;
}
```

Hier staan de beginaccolades altijd op een nieuwe regel. Deze notatie kost wat meer ruimte. Bij een eenvoudige `if ... else` structuur heeft u bij K&R maar 5 regels nodig en met Allman/BSD zijn dat er 8. Dat is bijna 2 x zo lang, al wordt het verschil bij langere stukken code natuurlijk minder groot (in bovenstaande voorbeelden is de verhouding 9 tegen 12). Maar de voorstanders vinden de uitlijning van de accolades in deze notatie veel duidelijker...

Er zijn overigens nog veel meer variaties mogelijk: http://en.wikipedia.org/wiki/Indent_style, waaronder de **Stroustrup style** die een beetje het midden houdt tussen de twee eerder genoemde varianten door else ook zijn eigen regel te geven:

```
$x = 1;
while ($x <= 10) {
    if ($x % 2 == 0) {
        echo "$x is even<br>\n";
    }
    else {
        echo "$x is odd<br>\n";
    }
    $x++;
}
```

Deze 3 varianten zijn veruit de meest gebruikte in van C afgeleide talen. Alle andere varianten die u kunt kiezen of zelf bedenken zijn dan ook een slecht idee. Maar mocht u toch willen afwijken wees dan in ieder geval consistent!

Weet u van u zelf dat u **slordig** bent met dit soort dingen? Gebruik dan de **code formatting** opties van uw **editor** of **IDE**. Met een commando schoont u dan een heel script in 1x op. Bij goede editors kunt u hoe code precies geformatteerd moet worden heel nauwkeurig instellen! Is de editor toch voortvarender dan u wilde dan kunt u de wijzigingen met Ctrl+Z meestal snel ongedaan maken...

Tools

Er bestaan handige tools waarmee u uw PHP code (of die van anderen ;-)) in 1x kan opschonen en omzetten naar een andere codeerstijl. Een gratis online oplossing vindt u op

<http://www.phpformatter.com/> met de (betere) beta versie op <http://beta.phpformatter.com/>

en op:

<http://www.waterproof.fr/products/phpCodeBeautifier>

kunt u een commandline tool aanschaffen om in een keer een hele verzameling scripts aan te passen!

Waar de meeste stijlgidsen het overigens wel over eens zijn, is dat het weglaten van de accolades (wat mag als er maar 1 statement gemaakt wordt in de branch of loop) een slecht idee is:

```
if ($condition_one)
    echo "do something";
else
    if ($condition_two)
        echo "something else";
    else
        echo "don't do anything";
        echo "do always";
```

Wanneer u even niet oplet dan lijkt het laatste statement bij de `else` branch te horen, maar dat is dus niet zo, het is **onconditioneel** en wordt **altijd uitgevoerd**! U kunt het veel duidelijker maken door de accolades te gebruiken (en een extra blanco regel):

```
if ($condition_one) {
    echo "do something";
} else {
    if ($condition_two) {
        echo "something else";
    } else {
        echo "don't do anything";
    }
}

echo "do always";
```

Dus gebruik altijd accolades ook als er maar 1 statement wordt uitgevoerd. Dat is misschien iets meer werk, maar dit maakt het veel duidelijker wat er wanneer gebeurt! En is bovendien veel makkelijk uitbreidbaar als er later toch ergens een tweede statement bij moet... (want dan moet u als nog de accolades gaan zetten!)

Verder maakt PEAR gebruik van een **indent** van **4 spaties in plaats** van **tabs**.

UPDATE

Sinds kort is er ook een initiatief van de **PHP Framework Interoperability Group** (PHP FIG) om een PHP standaard voor de hele industrie neer te leggen. Voor nu voert het meeste daaruit te ver maar voor de basisdingen komt het overeen met wat u tijdens de cursus zult leren! Meer informatie vindt u op de officiële site:

<http://www.php-fig.org/>

De volgende PSR's (**PSR** staat voor **P**HP **S**tandards **R**ecommendation) zijn voor nu interessant:

PSR-1: Basic Coding Standard

<https://www.php-fig.org/psr/psr-1/>

PSR-12: Extended Coding Style

<https://www.php-fig.org/psr/psr-12/>

3.24.2 Conclusie

Over al deze onderwerpen zijn heel wat verhitte discussies gevoerd, maar op zich maakt het niet eens zo heel veel uit wat precies de afspraken zijn, zolang iedereen zich er maar aan houdt! Sommige teams gaan zelfs zo ver om afwijkende code te weigeren...

Nog enkele andere stijlgidsen ter inspiratie:

- <http://framework.zend.com/manual/en/coding-standard.html>
- https://mitsloan.mit.edu/shared/content/PHP_Code_Style_Guide.php
- <http://dev.kohanaphp.com/wiki/CodingStyle>
- <http://expressionengine.com/docs/development/guidelines/general.html>
- http://wiki.developer.mindtouch.com/Contributing/PHP_Coding_Style
- <http://www.python.org/dev/peps/pep-0008/>

En tot slot voor de liefhebber de coding conventions voor de PHP Core code zelf:

- http://cvs.php.net/viewvc.cgi/php-src/CODING_STANDARDS?view=markup

Werken met formulieren

4.1 Inleiding

In de scripts die u tot nu toe geschreven heeft, gaf u zelf de waarden van de variabelen op. Maar onze scripts worden eigenlijk pas echt interessant op het moment dat de waarden van gebruikers afkomstig zijn. Zo kunnen verschillende gebruikers hetzelfde script gebruiken om vervolgens op maat gesneden uitvoer te krijgen.

Een script kan op verschillende manieren invoer binnen krijgen. In dit hoofdstuk behandelen we twee van de meest voorkomende:

1. Invoer via een formulier (via een GET of POST request)
2. Invoer via de URL (via een GET request)

In PHP Vervolg behandelen we ook nog twee andere manieren waarop een script input kan krijgen, namelijk via **cookies** en **sessies**.

De standaardmanier om aan gebruikersinvoer te komen op het web is door middel van HTML formulieren. We gebruiken dan de verzonden informatie om de variabelen in onze scripts te vullen. De hypotheekoefening uit het vorige hoofdstuk is daarvan een prima voorbeeld:

Geef de gewenste waarden op:

Hypotheekvorm:	Lineair
Hypotheekwaarde:	, -
Looptijd:	jaar
Rente:	%
	verzend

Een gebruiker kan nu zelf voor zijn situatie het totaal aan rente te betalen bedrag uit laten rekenen. Voordat we iets vergelijkbaars kunnen maken moeten we echter wel weten hoe we de variabelen in ons PHP script met de invoer uit het formulier kunnen vullen.

Een andere manier is om achter de URL variabelen met hun waarden te plaatsen.

```
<a href="products.php?cat=3">DVD's</a>
```

Wanneer een gebruiker op een link klikt die op deze manier is opgebouwd, dan worden de gegevens naar de aangeklikte pagina verzonden. Ook deze manier zullen we in dit hoofdstuk bespreken.

Het verzenden en verwerken van data bestaat dus uit minimaal twee pagina's: een pagina met daarop een HTML formulier of hyperlink en een PHP pagina die de code bevat om de verzonden data uit te lezen en te verwerken.

In de PHP Vervolg cursus zullen we overigens leren dat deze taken ook door één script uitgevoerd kunnen worden: het formulier verzendt de data dan naar zichzelf (self-submitting form). Dit heeft grote voordelen maar is voor nu nog teveel van het goede...

Het aanmaken van HTML formulieren wordt uitgebreid behandeld in de **HTML cursus**. We kunnen hier natuurlijk niet alles herhalen, maar om het geheugen op te frissen, herhalen we in de volgende paragraaf eerst even kort de belangrijkste zaken rond het maken en gebruiken van formulieren. Door de veelvuldige inzet van WYSIWYG editors is die kennis immers niet (meer) bij iedereen paraat. Daarna gaan we in op het inlezen en verwerken van de data door PHP en bespreken nog enkele PHP-specifieke zaken.

HTML Formulieren

U kunt overal binnen de body van een HTML document een formulier aanmaken. U begint een formulier met `<form>` en u eindigt het met `</form>`. Hierbinnen kunt u de meeste HTML opmaak gebruiken (bijvoorbeeld tabellen of divisions voor de uitlijning). U kunt er alleen niet een ander formulier in opnemen. Een pagina kan één of meer formulieren bevatten, denk aan een zoekformulier en een inschrijfformulier, maar ze mogen niet genest zijn!

Het `form` element kent enkele belangrijke attributen. Allereerst het `action` attribuut dat bepaalt waar de door de gebruiker ingevulde gegevens naar toe worden gezonden. Dat kan een mailadres zijn of een CGI-script maar wij zullen daar straks uiteraard een PHP pagina voor gaan gebruiken! Dit aannemende script wordt de **form handler** genoemd.

Verder heeft het `form` element een `method` attribuut dat bepaalt op welke wijze de gegevens worden verzonden. De default is `method="get"`, de andere mogelijkheid `method="post"`. We gaan straks uitgebreid op het `method` attribuut in en de verschillen tussen deze twee manieren.

Het `enctype` attribuut bepaalt hoe de verzonden gegevens gecodeerd zijn. Als u niets opgeeft is dat de standaardmanier: `enctype="application/x-www-form-urlencoded"` maar deze wordt gewoonlijk weggelaten. In paragraaf 4.4 leest u meer over **URL Encoding**.

We hebben tot zover het volgende staan:

```
<form action="process.php" method="get">
...
</form>
```

Binnen het formulier kunt u **form controls**, oftewel formulier invoerelementen opnemen. Eén van de meest gebruikte invoerelementen is het tekstvak dat u op de volgende manier aanmaakt:

```
<form action="process.php" method="get">
  <input type="text">
</form>
```

Omdat dit het *default* invoer element is zou u hetzelfde resultaat krijgen als u `type="text"` weg zou laten. Het is echter netjes om zo volledig mogelijk aan te geven wat u wilt.

Verder zult u een gebruiker ook duidelijk moeten maken wat hier ingevuld dient te worden dus dat zet u er dan voor:

```
<form action="process.php" method="get">
  Voornaam: <input type="text" />
</form>
```

Maar wat de gebruiker invult, de waarde (*value*), moet gekoppeld worden aan een naam anders kan er niets verstuurd worden! **Elk invoerelement moet een unieke naam hebben:**

```
<form action="process.php" method="get">
  Voornaam: <input type="text" name="voornaam" />
</form>
```

Let op: de naam mag geen spaties bevatten. Gebruik indien nodig **underscores**. Verder mag de naam ook **niet met een cijfer beginnen!** `name="1"` mag niet, dat geeft problemen bij het verwerken met JavaScript en Server-side scripting talen zoals PHP. Iets als `name="nr1"` kan dan weer wel.

Maar het is natuurlijk sowieso slim om **betekenisvolle namen** aan de velden te geven! De namen worden, zoals we zo zullen zien, bij de verwerking aan de server kant gebruikt om de waarden uit de formulervelden uit te lezen, en het is dan heel prettig als de naam duidelijk maakt om welke data het gaat! Sommige WYSIWYG editors geven automatisch namen als `TextBox1` of `Field1` en nummeren daarbij door voor de volgende velden. Dit soort namen zeggen niets over de inhoud en maken het dus lastiger de inhoud later te verwerken...

Bij formulieren gaat het vaak mis doordat namen vergeten worden of niet uniek zijn. Het is daarom handig om net als bij de naamgeving voor variabelen **consequent** te zijn in de namen die u aan de verschillende typen invoervelden geeft.

U kunt het veld verder uitbreiden door attributen op te geven voor de grootte (*size*), deze staat standaard op 20 karakters en het aantal karakters dat maximaal ingevoerd kan worden (*maxlength*). Zie voor de overige attributen een goede HTML handleiding. Bijvoorbeeld de HTML Handleiding van Hans de Jong (<http://www.handleidinghtml.nl/>) die ook in zijn geheel te downloaden is.

Het formulier zal minimaal over een verzendknop moeten beschikken voordat er iets verstuurd kan worden:

```
<form action="process.php" method="get">
  Voornaam: <input type="text" name="voornaam">
             <input type="submit">
</form>
```

De tekst op de knop is afhankelijk van de browser(versie). Als u hem wilt aanpassen dan doet u dat door hem een andere *value* mee te geven:

```
<form action="process.php" method="get">
  Voornaam: <input type="text" name="voornaam">
             <input type="submit" value="Verzenden">
</form>
```

Een naam is in de meeste gevallen niet nodig omdat u meestal niet geïnteresseerd bent in het versturen van de *value* van deze knop.

NB.: In dit verband is het misschien ook nuttig er op te wijzen dat een formulier meer dan één submit button kan hebben. Er kan er namelijk toch altijd maar één aangeklikt worden. U kunt ze dan dezelfde naam geven maar een verschillende waarde. Zo kunt u Server-side onderscheid maken op welke knop is geklikt en welke actie moet worden uitgevoerd. Denk bijvoorbeeld aan een 'delete' knop en een 'help' knop. U kunt dan bijvoorbeeld voor beide knoppen de naam `name="action"` gebruiken.

Ons formulier wordt in een browser als volgt getoond:

Voornaam:

Zie hier, een functionerend formulier!

Er zijn naast het tekstvak natuurlijk ook nog tal van andere formulierelementen die u kunt gebruiken. Als u niet helemaal zeker bent van uw HTML kennis, adviseren wij u dringend deze kennis bij te spijkeren. Een goed begrip van formulieren en hun mogelijkheden is namelijk onontbeerlijk voor een webdeveloper!

4.2 Data verzenden

Maar hoe wordt data nu feitelijk verstuurd en wat hebben die twee verschillende methods daar nu eigenlijk mee te maken? Zoals u zich hopelijk herinnert uit de inleiding, worden webpagina's op de webserver aangevraagd door middel van een **HTTP Request**.

Wanneer een gebruiker op de submit knop van een formulier klikt, verzamelt de browser de formulier-invoer als een lijst van naam-waarde paren en verstuurt deze in een HTTP Request naar de server. Dat kan op één van twee mogelijke manieren: een **GET Request** of een **POST Request**, afhankelijk van de in het `method` attribuut van het formulier opgegeven methode.

In de **bijlagen** wordt dieper ingegaan op de precieze opbouw van HTTP Requests en Responses. In de volgende paragrafen gaan we vooral in op de werking en wanneer u welke methode gebruikt.

4.2.1 Method GET

Wanneer een formulier met `method="get"` wordt verzonden, zijn de naam-waarde paren zichtbaar in de URL in de browser.

Wanneer we ervan uitgaan dat de pagina `process.php` uit ons voorbeeld op een server `www.example.com` staat dan wordt de informatie in het voorbeeldformulier uit de vorige paragraaf als volgt verzonden wanneer iemand als voornaam `Arno` invult:

```
http://www.example.com/process.php?voornaam=Arno
```

U mag dit lezen als: *"geef me de pagina `process.php`, ik stuur het naam-waarde paar `voornaam=Arno` mee zodat het script in `process.php` ermee kan doen waarvoor het geprogrammeerd is"*.

De URL en de naam-waarde paren worden van elkaar gescheiden door het `?`-teken. Het deel achter de URL, inclusief `?`, wordt de **querystring** genoemd.

Meerdere naam-waarde paren worden gescheiden met een `&`-teken (ampersand). Zouden we aan het formulier een tekstvak met `name="achternaam"` toevoegen dan wordt de querystring:

```
?voornaam=Arno&achternaam=Mol
```

4.2.1.1 Voordelen

Versturen via GET is handig in een testsituatie omdat u zo makkelijk even **handmatig** de invoer in de adresbalk van uw browser kunt **aanpassen** en zien wat de uitvoer wordt. Dat kan in een productie omgeving echter als een flink nadeel uitpakken als u niet wilt dat een gebruiker op deze manier uw script van (foutieve) invoer kan voorzien! Vergeet nooit de invoer Server-side te valideren voordat u er in het script mee aan de slag gaat!

Doordat de naam-waarde paren in de URL staan kan een gebruiker het request eenvoudig **opnemen** in zijn lijst met **favorieten** en makkelijk weer opnieuw opvragen. Denk daarbij aan zoekopdrachten bij een zoekmachine of makelaar.

U kunt ook van de mogelijkheid om informatie over de querystring te verzenden gebruik maken door **hyperlinks** (dynamisch) van een querystring te voorzien:

```
<a href="cursusdata.php?cursus=PHP">
  cursusdata PHP trainingen
</a>
```

4.2.1.2 Nadelen

Het belangrijkste nadeel van de GET methode is echter de **beperkte hoeveelheid informatie** die u aan een script kunt doorgeven. De beperking zit hem in de maximale lengte van de URL die zowel de browser als de server kunnen verwerken. In Internet Explorer kunt u bijvoorbeeld maximaal 2048 karakters gebruiken. Dit is dus inclusief de karakters die nodig zijn voor de URL zelf. Als veilige marge wordt echter meestal **255 karakters** aangehouden in verband met oudere browsers en (proxy-)servers. Op zich nog best een respectabel aantal maar bijvoorbeeld niet voldoende voor een tekstvenster waarmee een gebruiker bijvoorbeeld een opmerking, e-mail of een ander zelf geschreven artikel kan verzenden.

Een bijkomend nadeel van het verzenden in de URL, is dat requests standaard gelogd worden. Hierdoor komt ook de verzonden data in de **logfiles** terecht, waardoor het wellicht ook mensen die het niets aangaat onder ogen kan komen.

4.2.2 Method POST

Bij een formulier dat met `method="post"` wordt verzonden zijn de naam-waarde paren **niet zichtbaar** in de URL omdat ze in de body van het HTTP Request staan. Zie voor details de bijlage.

4.2.2.1 Voordelen

Doordat de informatie niet zichtbaar verstuurd wordt vervallen een aantal nadelen van GET. Maar dat wil **niet** meteen zeggen dat de informatie nu **veilig** verstuurd wordt. Iemand die gebruik maakt van een *packet sniffer* op het netwerk kan nog steeds alle verstuurde gegevens onversleuteld voorbij zien komen. Maar voor een leek is in ieder geval niet meteen inzichtelijk wat naar het script verstuurd wordt en is dat wat minder eenvoudig aan te passen. Het is daarom een goed gebruik om formulieren die gegevens in een database muteren met POST te verzenden.

Voor een beveiligde verbinding zult u de gegevensuitwisseling tussen browser en server moeten *encrypten*, bijvoorbeeld met behulp van SSL (Secure Sockets Layer). Dit valt echter buiten het bestek van deze cursus.

Het grootste voordeel van POST is dat het **geen beperkingen** kent wat betreft de **maximale hoeveelheid data** die het kan verzenden.

4.2.2.2 Nadelen

Een nadeel van POST is dat het verwerkende script alleen maar door het (opnieuw) verzenden van een formulier aangesproken kan worden. Bovendien leidt een page refresh (F5) van de formhandler tot een melding van de browser dat om dit te kunnen doen de POST data opnieuw verzonden zal moeten worden, met mogelijk ongewenste resultaten (dubbele bestellingen!).

De eindgebruiker krijgt bij het verversen na een POST dan in Chrome een melding als deze te zien:



Misschien niet zo'n heel erg duidelijke omschrijving, FireFox maakt het iets duidelijker:



Maar in beide gevallen zal het voor de eindgebruiker waarschijnlijk wel alarmerend werken.

Let op als u als ontwikkelaar het gedrag probeert te repliceren dat u de Browser Developer Tools niet open hebt staan anders verschijnt het dialoogvenster niet!

De oplossing om uit deze situatie te ontsnappen is een redirect naar een bevestigingspagina die dan door de browser met GET wordt opgevraagd en vervolgens zonder problemen ververst kan worden.

U leest er meer over op <http://en.wikipedia.org/wiki/Post/Redirect/Get>. Later in dit hoofdstuk leert u hoe u redirects kan maken in PHP met de `header()` functie. In het database hoofdstuk gaat u bovenstaande oplossing ook zelf toepassen!

4.2.3 Conclusie

Wanneer u een pagina opvraagt door middel van een hyperlink of door een URL in te typen dan is dit altijd een GET request. U zegt eigenlijk tegen de server: "geef me wat je hebt". Eventueel kunt u naam-waarde paren meegeven waar de server op kan reageren door bijvoorbeeld een deel van de data of een gepersonaliseerde pagina te retourneren. GET is ook te gebruiken in formulieren met de `method="get"`.

Een POST request wordt in principe altijd met een formulier gedaan*. U zegt dan in feite tegen de server: "hier heb je data die iets gaat veranderen".

Gebruik **GET** voor **selecties** (ook voor **zoeken** en **filteren**). U kunt dan hyperlinks met parameters gebruiken en de gebruiker kan het resultaat makkelijk bookmarken. Gebruik voor **mutaties** en **logins** altijd **POST**. In Hoofdstuk 6 krijgt u hier mee te maken.

*) Het is met JavaScript en het XMLHttpRequest (XHR) dan wel Fetch ook mogelijk om POST requests te versturen!

4.3 URL encoding

Een probleem dat zowel voor GET als POST geldt is dat (onder andere) de volgende karakters absoluut niet mogen voorkomen in de naam-waarde paren omdat ze daarin een speciale betekenis hebben:

: / ? = &

Een browser als Internet Explorer zorgt er automatisch voor dat deze URL encoded worden: probleemkarakters worden daarbij vervangen door hun hexadecimale equivalent, voorafgegaan door het %-teken:

Karakter	Karakter URL encoded
=	%3D
&	%26
?	%3F
spatie	%20
@	%40

Voor mensen die het naadje van de kous willen weten:

<http://www.blooberry.com/indexdot/html/topics/urlencoding.htm>

4.4 Data verwerken

Als de data, via GET of POST, bij de formhandler terechtgekomen is, moet deze beschikbaar gemaakt worden voor het verwerkende script. PHP doet dat door de binnengekomen naam-waarde paren beschikbaar te maken als variabelen met bijbehorende waarden.

4.4.1 Register_Globals

Let op: voor PHP 4.2.0 stond de `php.ini` instelling `register_globals` default aan en werden naam-waarde paren die via een formulier werden verzonden **automatisch** als variabelen aangemaakt. Wanneer er een naam-waarde paar als `voornaam=Arno` binnenkwam, ongeacht of dat via GET of POST gebeurde, dan werd automatisch een globale variabele `$voornaam="Arno"` aangemaakt.

Om veiligheidsredenen is dat niet langer het geval en staat `register_globals` standaard **uit**. De hele optie is sinds PHP 5.4 zelfs compleet verdwenen. De verzonden waarden worden nu enkel in zogenaamde **superglobal arrays** `$_GET` of `$_POST` beschikbaar gesteld.

Let op: In scripts die u van het internet afhaalt wordt lang niet altijd van deze instelling uitgegaan!

4.4.2 \$_GET

In PHP gebruikt u de `$_GET` superglobal array om informatie die via een GET REQUEST werd meegezonden uit te kunnen lezen.

U kunt de inhoud van een naam-waarde paar dat over de querystring werd verstuurd als volgt uitlezen:

```
<?php echo $_GET['voornaam']; ?>
```

Maar vaak is het handiger om de waarde in een variabele te plaatsen:

```
$voornaam = $_GET['voornaam'];
```

U kunt dan binnen het aannemende script verder werken met deze variabele. Dat is efficiënter dan steeds de array te doorzoeken naar de waarde van een item met de index "voornaam" (en het schrijft ook wat makkelijker...).

Let op: `$_GET['voornaam']` moet exact zo geschreven worden, dus met **underscore** en **hoofdletters**. Daarna volgen **rechte haken**. De naam van het naam-waarde paar, de **index** in de array, is een **string** en staat dus **tussen quotes**. Het maakt echter niet uit of dit enkele of dubbele quotes zijn...

Merk op dat het uiteindelijk zonder quotes ook wel goed zou komen door het vergevingsgezinde karakter van PHP:

Notice: Use of undefined constant voornaam - assumed 'voornaam' in **test.php** on line 2

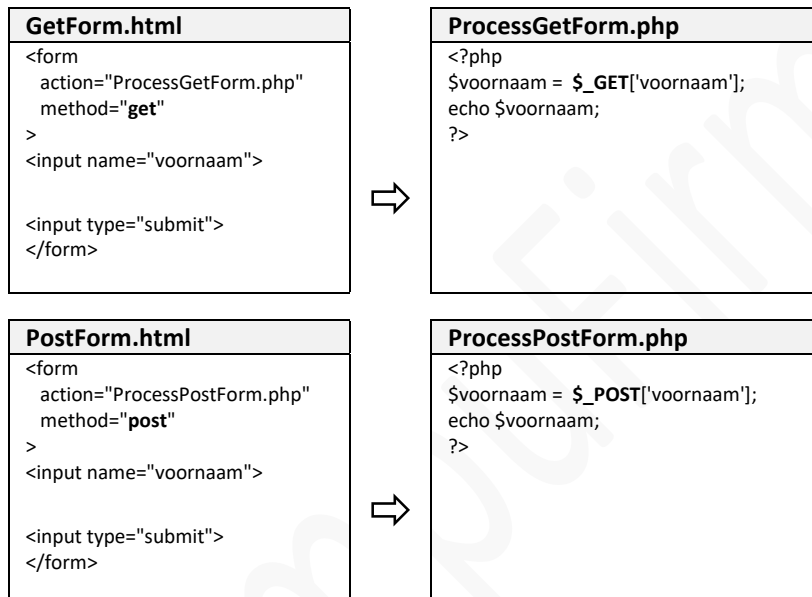
De index is **hoofdlettergevoelig**, dus als u het veld Voornaam (met een hoofdletter V) had genoemd dan had u `$_GET['Voornaam']` moeten gebruiken (ook met een hoofdletter V).

4.4.3 \$_POST

Voor een POST request gebruikt u de `$_POST` superglobal array. De opdracht uit de vorige paragraaf wordt dan:

```
<?php echo $_POST['voornaam']; ?>
```

Voor de duidelijkheid hebben we de manieren waarop u GET en POST binnen PHP gebruikt in tabelvorm gezet:



4.4.4 \$_REQUEST

Er bestaat ook een superglobal `$_REQUEST` die in feite een samensmelting is van de informatie uit de `$_GET`, `$_POST` en `$_COOKIE` arrays. Default ook precies in deze volgorde die aangegeven staat in `php.ini`:

```
variables_order = "EGPCS"
```

Dit kan problemen geven. Stel, dat het formulier er zo uit ziet:

```
<?php setcookie('test', 'cookie'); ?>
...
<form action="test.php?test=get" method="post">
  <input type="text" name="test" value="post" />
  <input type="submit" value="OK" />
</form>
```

In bovenstaand formulier staat 3x een naam-waarde paar met de naam `test`. Na verzenden is dit dan de inhoud van de superglobals:


```

GET:
Array
(
    [test] => get
)

POST:
Array
(
    [test] => post
)

COOKIE:
Array
(
    [test] => cookie
)

REQUEST:
Array
(
    [test] => cookie
)

```

Oftewel de cookie waarde overschrijft in `$_REQUEST` de andere waarden! Dit voorbeeld maakt wel duidelijk waarom u beter niet op deze superglobal kunt vertrouwen! Cookies worden in de vervolgcursus behandeld.

Om veiligheidsredenen wordt het dan ook **afgeraden** om dan maar altijd `$_REQUEST` te gebruiken. **Geef altijd expliciet aan waar u de invoer vandaan verwacht!**

Het verschijnsel dat reeds bestaande elementen in de array worden overschreven komt ook voor wanneer u velden met dezelfde naam heeft in een formulier, we bespreken dit bij het onderdeel **multi-dimensionale arrays**.

4.4.5 print_r()

Met de functie `print_r()` kunt u tijdens het programmeren snel de inhoud van een array bekijken, dus ook van `$_GET` of `$_POST`:

```
print_r($_POST);
```

geeft bijvoorbeeld:

```
Array ([voornaam] => Arno [achternaam] => Mol)
```

Alles staat hierbij op één regel. In de broncode staan de elementen van de array echter wel keurig onder elkaar. U kunt deze, beter te lezen, uitvoer ook in het browser venster zien wanneer u de functieaanroep tussen `<pre>` tags zet:

```

echo '<pre>';
print_r($_POST);
echo '</pre>';

```

geeft:

```

Array
(
    [voornaam] => Arno

```

```
[achternaam] => Mol
)
```

Handig om in de testfase snel te zien wat een formulier verzendt!

Tip: Wanneer u de tweede, optionele, parameter op `true` zet dan geeft de functie niet gelijk uitvoer maar retourneert de informatie als een string. U kunt deze dan opslaan in een variabele en er verder mee werken voordat u het met een `echo` toont.

4.5 HTML Escape

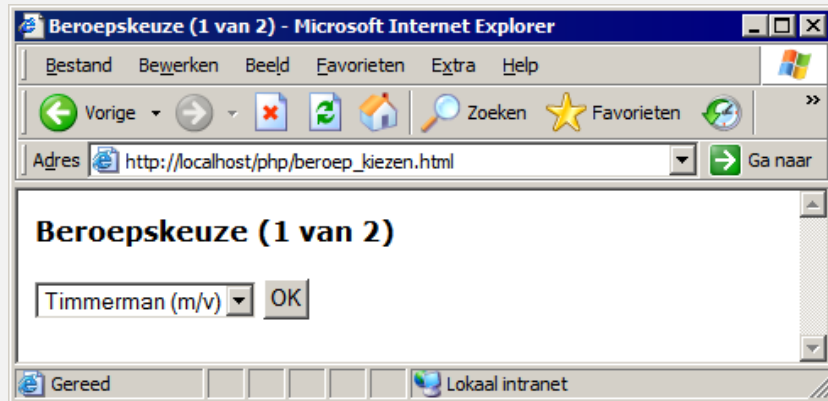
Zeker in het geval van `$_GET` is heel duidelijk te zien dat de naam-waarde paren zeer eenvoudig door gebruikers te manipuleren zijn. Voor `$_POST` is dat overigens ook het geval en u zult om de waarden veilig weg te kunnen schrijven dan ook weer gebruik moeten maken van de eerder bij `$_SERVER` besproken functie `htmlspecialchars()`:

```
<?php
// gebruik de naam uit de querystring of geef default waarde
$naam = isset($_GET['naam']) ? $_GET['naam'] : 'gast';
?>
...
<!-- schrijf veilig weg in HTML -->
<p>Hoi <?php echo htmlspecialchars($naam); ?></p>
```

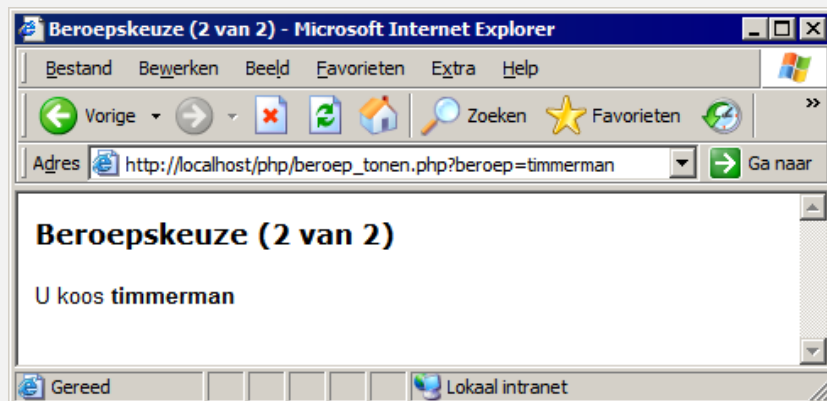
Oefening 4.1

Maak een formulier `beroep_kiezen.html` met daarin een keuzelijst waaruit een gebruiker een beroep kan kiezen. De beroepen zijn:

*timmerman (m/v)
architect (m/v)
piloot (m/v)
serveerster
kok (m/v)*



Het gekozen beroep moet vervolgens in een nieuwe pagina `beroep_tonen.php` getoond worden:



Oefening 4.2

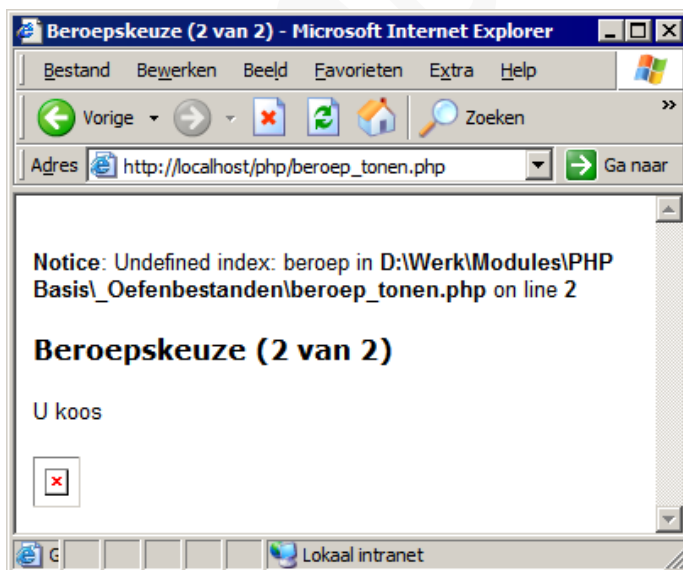
Als bovenstaande goed werkt, gaan we het tweede script (`beroep_tonen.php`) uitbreiden zodat een afbeelding van het gekozen beroep als volgt wordt getoond:



De afbeeldingen staan al op de server in de map `images` en heten: `timmerman.gif`, `piloot.gif`, `kok.gif`, `architect.gif` en `serveerster.gif`

4.6 Los opvragen formhandler

Het nadeel van een los formulier en een losse formhandler is dat een eindgebruiker (per ongeluk?) de formhandler geïsoleerd kan opvragen met alle foutmeldingen en problemen van dien. Probeer het maar eens door `beroep_tonen.php` los op te vragen:



In de cursus **PHP Vervolg** leert u hoe u dit probleem fraai kunt oplossen met zogenaamde **self-submitting forms**: formulieren die naar zichzelf versturen en dus form en formhandler in een zijn!

U zou het probleem vooralsnog kunnen oplossen door te checken of de benodigde variabelen wel gezet zijn, en als dat niet het geval is de browser het formulier te laten opvragen. U gebruikt daarvoor twee nog niet eerder besproken functies:

4.6.1 isset()

Met de functie `isset()` kunt u controleren of een variabele wel of niet gezet is:

```
if (!isset($_POST['id'])) {  
    // variabele is niet gezet  
}
```

U kunt de functie mooi gebruiken om te controleren of een bepaalde naam in de querystring aanwezig is. Als dat het geval is kent u de bijbehorende waarde toe aan een variabele, is het niet het geval dan geeft u de variabele een standaard waarde:

```
if (isset($_POST['id'])) {  
    $id = $_POST['id'];  
} else {  
    $id = 1;  
}
```

Dit kunt u met de in 3.12.6 besproken ternary control operator een stuk compacter noteren:

```
$id = isset($_POST['id']) ? $_POST['id'] : 1;
```

Doet u dit vooral pas wanneer u de notatie prettig vindt lezen...

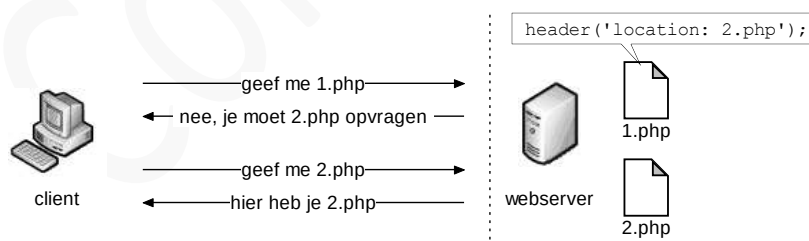
U kunt de controle op de aanwezigheid van een naam-waarde paar in de querystring ook gebruiken om een gebruiker terug te sturen naar een beginpagina:

4.6.2 redirect met header('location: ...')

Met de functie `header()` kunt u de HTTP Response headers manipuleren. Hier gebruiken we hem om met de setting Location aan te geven dat de browser een ander bestand moet opvragen:

```
header('location: formulier.html');
```

De browser zal vervolgens een GET request voor deze nieuwe URL doen:



Omdat we hierbij van de browser afhankelijk zijn of de gewenste pagina wel wordt opgevraagd, is het verstandig om na een dergelijke redirect ervoor te zorgen dat de rest van de PHP code in ieder geval niet meer kan worden uitgevoerd. Dit doen we door een `exit()` (of het alias `die()`) commando te geven. Hiermee wordt het uitvoeren van het script meteen beëindigd:

```
header('location: formulier.html');  
exit();
```

Zie voor het waarom van `exit` ook: <https://thedailywtf.com/articles/WellIntentioned-Destruction>

U kunt tussen de haakjes normaal gesproken een string plaatsen die dan als melding bij het afsluiten getoond kan worden, maar in dit geval moet u dat dus juist niet doen! Omdat `exit` en `die` language constructs zijn mag u de haakjes ook gewoon weglaten... We komen in **Hoofdstuk 6** nog op deze constructies terug bij het maken van de database verbinding.

Het is dus heel belangrijk dat u zich realiseert dat u de `header()` functie alleen maar kunt gebruiken **voordat** de HTTP headers verzonden zijn. Dat kan allereerst door de functie aan te roepen voordat er iets naar de client verzonden is. Er mag dus nog geen `echo` of HTML-code geschreven zijn. Bovenaan uw script werkt een dergelijke opdracht dus prima.

Als er al wel uitvoer is verzonden (en de HTTP headers dus al weg zijn en niet meer aangepast kunnen worden) dan krijgt u een foutmelding als deze:

Warning: Cannot add header information - headers already sent .

Verder is het belangrijk dat u zich realiseert dat u uit veiligheidsoogpunt **geen open redirects** moet maken. Een open redirect is een redirect waarbij de locatie door een **variabele** wordt bepaald. Als een kwaadwillende de waarde voor deze variabele kan manipuleren dan kan uw website daardoor misbruikt worden voor *spamming*. Gebruik bij voorkeur dus altijd letterlijke waarden die niet door een derde kunnen worden gemanipuleerd! Als het niet anders kan maak dan een *whitelist* aan van toegestane waarden en controleer daar tegen. Sta bij voorkeur ook geen redirects naar buiten uw site toe. Als dat wel nodig is dan is het slim om de gebruiker daarvan op de hoogte te stellen.

Zie voor meer informatie:

- https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet
- <https://webmasters.googleblog.com/2009/01/open-redirect-urls-is-your-site-being.html>

4.6.3 `ob_start()`

Er is wel een uitzondering op bovenstaande regel: namelijk als u ervoor gezorgd hebt dat de response *gebufferd* wordt. Output kan namelijk op twee manieren verzonden worden: ongebufferd en gebufferd. Ongebufferde output wordt onmiddellijk na verwerking naar de browser verzonden terwijl gebufferde output wordt opgeslagen in een buffer op de server. De buffer wordt in één keer automatisch verzonden als het eind van het script bereikt is.

De output buffering staat in `php.ini` mogelijk al aan, maar u kunt het ook op pagina niveau aanzetten met de functie `ob_start()`. Dit dient u uiteraard wel aan te geven voordat u informatie verzonden hebt, dus zet het bovenaan in uw script!

4.6.4 Controle

Door al de functies te combineren kan uw uiteindelijke controle er bijvoorbeeld zo uit zien:

```
if (! isset($_POST['voornaam']) &&
    ! isset($_POST['achternaam'])) {
    // variabelen zijn niet gezet
    // redirect naar het formulier
    header('location: formulier.html');
    exit;
}
```

Oefening 4.3

Maak in `beroep_tonen.php` gebruik van een switch om per beroep een andere begeleidende tekst te kunnen tonen en waarbij het niet uitmaakt hoe de bestandsnaam van de afbeelding luidt. Maak gebruik van de default case om manipulaties van de querystring af te vangen. Een mooie manier is om dan met behulp van de opdracht:

```
header('location: beroep_kiezen.html');  
exit;
```

opnieuw het keuzeformulier te laden.

Oefening 4.4

Zet de method van het formulier in `beroep_kiezen.html` op POST en plaats er een link Toon Kok in die `beroep=kok` naar `beroep_tonen.php` stuurt:

```
<a href="beroep_tonen.php?beroep=kok">Toon Kok</a>
```

Zorg ervoor dat `beroep_tonen.php` nu zowel de GET als POST requests kan verwerken. Hint: gebruik `$_SERVER['REQUEST_METHOD']`.

4.7 Leeg gelaten velden

Wanneer een formulier met een leeg tekstveld wordt verzonden dan is dat voor PHP op zich geen probleem, de entry in de bijbehorende superglobal krijgt dan als waarde een lege string toegewezen. Met andere woorden de variabele bestaat wel.

Stel dat u een formulier heeft met method POST en een tekstveld `voornaam` dan zou bij leeg verzenden gewoon een variabele `$_POST['voornaam']` aangemaakt worden. Deze heeft nu weliswaar de waarde `''` maar u kunt de variabele zonder probleem gebruiken in vergelijkingen en `echo` statements.

4.7.1 Checkboxes en radiobuttons

Bij niet aangevinkte checkboxes of niet geselecteerde radiobuttons ontstaat wel een probleem wanneer u deze in de formhandler probeert uit te lezen. Wanneer deze formcontrols namelijk **niet gebruikt** worden in het formulier dan worden ze **niet verzonden** en zijn dus ook **niet beschikbaar** in de bijbehorende superglobal array! Dit ligt overigens niet zozeer aan PHP als wel aan de wijze waarop deze controls in de HTML standaard zijn ontworpen.

Stel dat u een formulier heeft met method POST en een checkbox nieuwsbrief:

```
<input type="checkbox"  
       name="news"  
       value="ja">
```

Ja, ik wil de nieuwsbrief ontvangen

Wanneer deze **aangevinkt** wordt verzonden dan levert in de formhandler code als deze

```
Nieuwsbrief: <?php echo $_POST['news']; ?>
```

geen probleem op:

Nieuwsbrief: ja

Maar wordt de checkbox **onaangevinkt** verzonden, dan levert dit het volgende op (bij een maximaal `error_reporting` niveau):

Undefined index: news in **checkbox.php** on line 2

Dit komt omdat de variabele `$_POST['news']` niet bestaat. Bij een lager waarschuwningsniveau zou u simpelweg niets zien:

Nieuwsbrief:

Geen defaultwaarde, dus... U zou daarom code als deze moeten gebruiken:

```
<?php
$news = isset($_POST['news']) ? $_POST['news'] : 'nee';
?>
Nieuwsbrief: <?php echo $news; ?>
```

Wat dan keurig het gewenste resultaat laat zien:

Nieuwsbrief: nee

Maar eigenlijk zou u altijd op deze manier uw GET en POST variabelen moeten initialiseren! In de cursus PHP Vervolg gaan we uitgebreider op dit soort veiligheidsmechanismen in.

Oefening 4.5

Voeg aan `beroep_kiezen.php` een checkbox toe waarmee de gebruiker kan aangeven of hij/zij wel of niet de afbeelding van het gekozen beroep wil zien:



Uiteraard zorgt u ervoor dat deze keuze ook effect heeft in de formhandler `beroep_tonen.php`.

4.8 (Simpele) Validatie

U zou alle gebruikersinvoer die uw scripts binnenkomt altijd moeten valideren. Dat kan op twee manieren:

- Client-side validatie
- Server-side validatie

Client-side validatie houdt in dat u met JavaScript de formdata voor het verzenden controleert en pas als alle data OK is het formulier verzendt. Hoe u dat kunt doen leert u in onze **cursus JavaScript**.

Client-side validatie is echter heel eenvoudig te omzeilen door kwaadwillende gebruikers, dus u moet in ieder geval ook altijd Server-side valideren! Dat houdt in dat u na verzenden van de formdata met PHP gaat valideren. Als er nu fouten worden gevonden zult u de gebruikers daar op een nette manier op moeten wijzen.

Validatie wordt uitgebreid behandeld in onze **cursus PHP Vervolg**, het is een te uitgebreid onderwerp voor deze basiscursus. Een simpele manier van Server-side validatie willen we u echter niet onthouden:

```
<?php
// haal waarden uit formulier
$voornaam  = $_POST['voornaam'];
$achternaam = $_POST['achternaam'];

// begin errorstring
$errors = "";

// vul string aan indien van toepassing
// (Simpele validatie ter illustratie)
if ($voornaam == '') {
    $errors .= "* Voornaam is een verplicht veld!<br>";
}
if ($achternaam == '') {
    $errors .= "* Achternaam is een verplicht veld!<br>";
}
?>
<!DOCTYPE html>
<html>

<head>
<title>Formhandler</title>
</head>

<body>

<?php if ($errors): ?>

    <p>Er zijn fouten opgetreden:<p>

    <?php echo $errors; ?>

    <p><a href="javascript: history.go(-1);">Terug</a></p>

<?php else: // Alles OK. Verwerk formulier ?>

    <p>
        Hoi
        <strong>
            <?php echo htmlspecialchars($voornaam); ?>
            <?php echo htmlspecialchars($achternaam); ?>
        </strong>,
        bedankt voor het invullen!
    </p>

<?php endif; ?>

</body>
</html>
```

U begint met de waarden uit het formulier te halen en een lege errorstring op te bouwen. Vervolgens voert u de gewenste controles op de formuliervariabelen uit. Als er iets niet in orde is dan vult u de errorstring aan met een toepasselijke foutmelding. Na de controles kijkt u naar de lengte van de errorstring. Als deze langer dan 0 karakters is dan zijn er fouten opgetreden en laat u de fouten zien. Met een JavaScript link kan de gebruiker vervolgens terug naar het formulier om de fouten te herstellen:

Er zijn fouten opgetreden:

- * Voornaam is een verplicht veld!
- * Achternaam is een verplicht veld!

[Terug](#)

Let op: als u hier een **gewone link** had geplaatst dan zou de gebruiker het formulier **opnieuw opvragen** en waren alle eerder **ingevulde waarden verdwenen!**

Als de velden wel zijn ingevuld wordt het volgende getoond:

Hoi **Arno Mol**, bedankt voor het invullen!

Maar we zijn er nog lang niet. Spaties worden nu nog als invoer gezien en kijk eens wat er gebeurt als u een achternaam als O'Hara invult...

In de cursus **PHP Vervolg** gaan we dan ook veel dieper op deze belangrijke materie in. U leert ondermeer een veel mooiere oplossing kennen waarbij een formulier steeds naar zichzelf blijft posten met behoud van waarden en foutmeldingen in het formulier totdat alle velden gevalideerd zijn:

voornaam: *

achternaam: *

vul een achternaam in!

Bovendien leert u hoe u geavanceerde validaties met **reguliere expressies** kunt oplossen...

4.9 Hidden fields

Een hidden field is een bijzondere form control. Aan de ene kant gedraagt hij zich net als de andere form controls: u kunt er een naam-waarde paar mee versturen. Het bijzondere zit hem in het feit dat de gebruiker de waarde van het veld "in principe" niet kan aanpassen.

Let op: We hebben "in principe" tussen aanhalingstekens gezet want de gemiddelde gebruiker heeft wellicht geen flauw idee wat een hidden field is, maar elke gebruiker die ook maar iets van de werking van webapplicaties weet, is in staat de waarde van de hidden fields uit te lezen en aan te passen! Daarom zouden deze waarden eigenlijk ook liever gehashed of encrypted moeten worden zodat het minder logisch is om ze aan te passen. En na verzending zou u de waarde moeten controleren of ze niet zijn aangepast. Later in de cursus leert u hoe u dit kunt voorkomen met **sessie variabelen** en in de **PHP vervolg cursus** hoe u dat met nette **server-side validatie** kan oplossen.

Met een hidden field kunt u zo informatie doorgeven vanuit een webpagina zonder interventie van de gebruiker. Ze worden vaak gebruikt om eerder verkregen informatie naar een volgende pagina mee te sturen. De values van de hidden fields worden dan met behulp van PHP ingevuld en aan het oog onttrokken meegestuurd om in het aannemende script verder verwerkt te worden.

In de volgende oefening, en bij een latere oefening waar u een record in een database aanpast, gaat u deze techniek ook gebruiken om informatie van een pagina via een tweede pagina naar een derde pagina te sturen. Op de tweede pagina gebruikt u dan code als deze:

```
<!-- Dit is pagina 2.php -->
<form method="get" action="3.php">

<!-- geef id uit 1.php verborgen door aan 3.php-->
<input type="hidden" name="id"
      value="<?php echo htmlspecialchars($_GET["id"]); ?>">
...

```

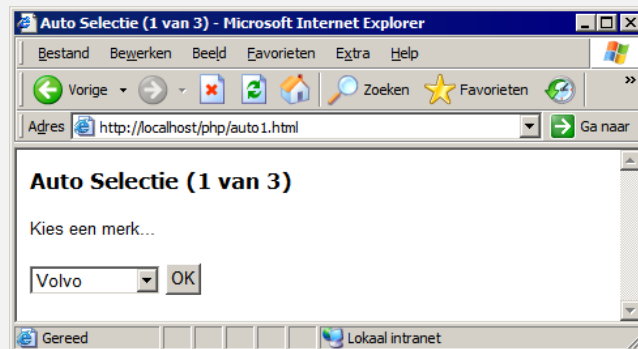
Tip: Het is voor het overzicht handig om hidden fields bovenaan uw form te plaatsen! En op de manier wordt de querystring vaak ook het meest logisch opgebouwd.

Op deze manier kunt u multi-page forms maken, bijvoorbeeld een wizard-achtige interface om een reis te boeken: kies eerst een bestemming, dan de wijze van vervoer etc... Zoals gezegd bestaan er andere, handiger en veiliger manieren om dit voor elkaar te krijgen, namelijk met behulp van **cookies** en **sessies** maar daar gaan we tijdens de basiscursus niet op in. We bespreken deze manieren uitgebreid in de cursus **PHP Vervolg**.

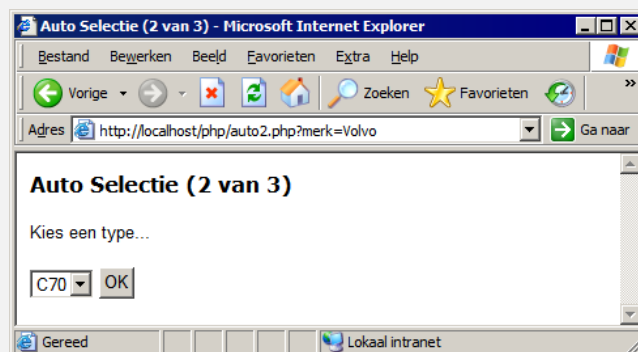
Oefening 4.6

Mak een formulier `auto1.php` waarin de bezoeker kan kiezen uit een aantal automerken, bijvoorbeeld:

- Audi
- Peugeot
- Volvo



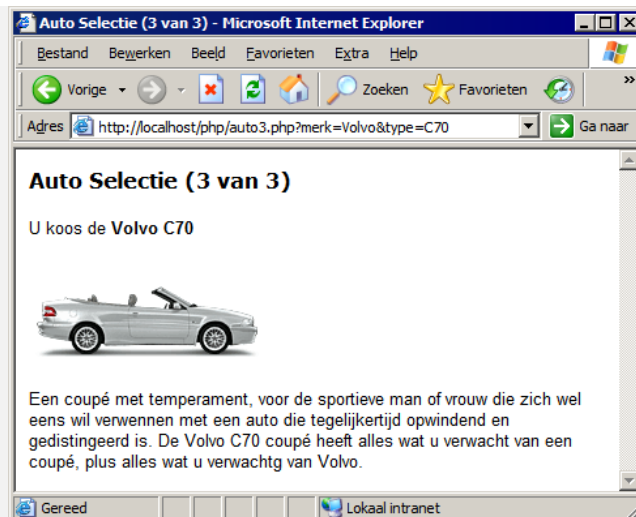
Indien gekozen wordt voor Volvo verschijnt een 2^e formulier `auto2.php` waaruit we kunnen kiezen uit een aantal Volvo typen:



De keuze voor een ander merk geeft de melding: "Dit merk zit helaas nog niet in de database – Andere keuze a.u.b." met een link terug naar het 1^e formulier:

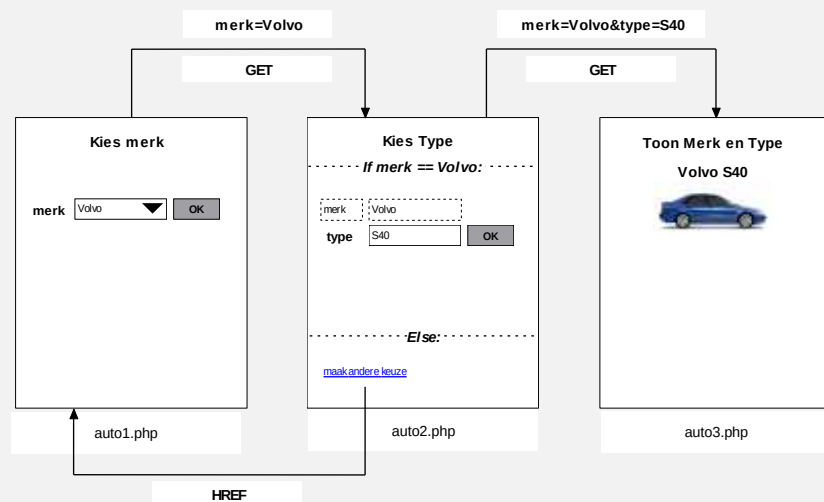


Na het kiezen van een Volvo model wordt de bijbehorende afbeelding met een modelbeschrijving eronder getoond in `auto3.php`:



De afbeeldingen en omschrijvingen staan al in de map /images.

Voor de duidelijkheid staat hieronder alles nog een keer in een overzicht:



Verdieping Maak op `auto3.php` een link [kies een ander type Volvo](#) terug naar `auto2.php` om een ander Volvo type te kunnen kiezen.

Verdieping Maak op `auto1.php` een directe link [Volvo C70](#) die verwijst naar `auto3.php` en daar de details van deze auto laat zien.

Verdieping Maak op `auto2.php` ook de types voor de andere merken zichtbaar. Om het helemaal af te maken kunt u de *bijbehorende afbeeldingen op het web opzoeken*.

Werken met databases

5.1 Inleiding

In de voorgaande oefening heeft u data opgeslagen in uw scripts. De gebruikers konden, door keuzes te maken, bepalen welk deel ze hiervan wilden zien. Hoewel dit al handiger in onderhoud is dan voor elk product een aparte pagina te hebben, is het duidelijk dat deze methode van werken niet erg schaalbaar is. U zou er als ontwikkelaar toch niet aan moeten denken om alle gangbare automerken en typen op deze manier toegankelijk te moeten maken.

Voor applicaties waarin met grote hoeveelheden data gewerkt wordt is het gebruik van een database inmiddels standaard geworden. Hetzelfde geldt voor webapplicaties. De database is de plek waar we de data hebben staan die onze webpagina's dynamisch gaat vullen.

Het opslaan van data in een database is uiteraard veel efficiënter dan deze domweg statisch in de pagina's te plaatsen. Met een database hoeft u het maar één keer op te slaan, ook al komt het op meerdere plekken in de website voor. U verkleint de kans op typefouten en updates kunnen centraal worden doorgevoerd, ook door mensen die geen verstand hebben van HTML of PHP. Zo is uw site altijd up-to-date. Denk maar eens aan het doorvoeren van een prijsverhoging van alle producten op een site. Met een database is dat een simpele opdracht. Met de hand is het een ramp. De kans op vergissingen en vergeten pagina's is enorm.

Maar wat is nu eigenlijk een database en hoe werkt een database? De meeste mensen hebben daar wel een vaag idee van, maar om soepel met een database te kunnen werken vanuit een programmeertaal als PHP moeten we wel wat meer in details treden.

5.2 Geschiedenis

In elke organisatie komen informatiestromen voor en soms komt die informatie zelfs in overweldigende hoeveelheden. Het is altijd al een uitdaging geweest hoe daar mee om te gaan en de informatie op te slaan en goed toegankelijk te maken. Tot nog niet zo heel lang geleden werden daarvoor vooral papieren systemen bedacht. Maar nu gebeurt dit uiteraard vooral digitaal.

In eerste instantie maakten programmeurs gebruik van zogenaamde **flat files**. Dit waren in feite tekstbestanden waarin data werd weggeschreven. Vergelijk het maar met een spreadsheet, of nog beter een *character separated file* waarin de data door middel van een afgesproken karakter (een komma of puntkomma bijvoorbeeld) van elkaar gescheiden wordt. Deze flat files zijn om een tweetal redenen niet zo erg efficiënt.

Voor elke applicatie moest steeds opnieuw code geschreven worden voor het wegschrijven en weer ophalen van data. Code voor het zoeken en sorteren van data kon daarbij behoorlijk complex worden. Maar het ergste is natuurlijk dat bij grote hoeveelheden data de tekstbestanden enorm lang worden en daardoor te traag worden om snel te kunnen open, doorzoeken en manipuleren...

Daarom werd er speciale software voor ontwikkeld: het zogenaamde **Database Management System (DBMS)** die dit kon. Het DBMS is in feite een stand-alone applicatie die het door reeds gedefinieerde code makkelijk maakt voor mensen of applicaties om data in de database op te slaan en te beheren. Programmeurs hoeven zo niet steeds zelf de code daarvoor te schrijven.

DBMS systemen komen grofweg gezegd in 2 soorten:

1. File-based
2. Client-Server

Voorbeelden van een File-based Database Management Systemen zijn Access en Filemaker. Deze zijn vooral bedoeld voor gebruik op de desktop en niet om door meerdere gebruikers tegelijkertijd benaderd te worden.

Database Management Systemen als SQL Server, Oracle en MySQL zijn dat duidelijk wel. Het zijn Client-Server oplossingen. Ze zijn opgesplitst in twee delen. Het Server deel is verantwoordelijk voor de toegang tot de data en het beheer van deze data. De benodigde software draait op een computer die de database server wordt genoemd. Alle verzoeken om data lopen via de server.

De client is een software programma dat verzoeken aan de server kan doen. Bijvoorbeeld om een alfabetisch gesorteerd productoverzicht op te vragen. De server voert de opdracht uit en geeft het resultaat aan de client die het vervolgens aan de eindgebruiker kan laten zien.

De client kan een door de fabrikant van het DBMS geleverd programma zijn zoals de Management Studio bij SQL Server of de MySQL Workbench voor MySQL. Maar het kan ook een programma zijn dat in een of andere programmeertaal geschreven is. Denk aan VB.NET, C# of Java voor Windows programma's of ASP.NET of PHP voor weboplossingen.

Vaak zijn de client en server op aparte computers geïnstalleerd maar dat hoeft niet. Tijdens de cursus draaien wij de client en server op een en dezelfde cursusmachine.

5.3 Wat is een database?

Een database is een methode om gegevens op een gestructureerde manier te bewaren zodat later de gegevens eenvoudig zijn terug te vinden.

5.3.1 Structuur

Databases zijn er in verschillende soorten en maten maar de bekendste en meest gebruikte is de **relationele database**. In een relationele database wordt alle data voorgesteld door de rijen en kolommen in **tabellen**. Elke tabel beschrijft een verzameling gerelateerde entiteiten zoals personen, objecten of gebeurtenissen. Elke rij beschrijft één entiteit en elke kolom een eigenschap, bijvoorbeeld de achternaam van een persoon, de kleur van een product of de datum van een gebeurtenis.

Een tabel moet een **unieke naam** hebben in een database. Hetzelfde geldt voor de kolommen in de tabel. De kolomnamen mogen wel hergebruikt worden in andere tabellen in de database, in combinatie met de unieke naam van de tabel kan er nooit verwarring ontstaan over welke kolom bedoeld wordt...

Het basiselement in een database is een **record**. Een record is een verzameling van data die als een entiteit, zeg maar als een geheel wordt gezien. Neem bijvoorbeeld een product als een (muziek) CD. Dat heeft een aantal eigenschappen aan de hand waarvan hij kan worden beschreven. Bijvoorbeeld zijn titel, maar uiteraard ook de artiest(en) door wie hij gemaakt is, de platenmaatschappij die het heeft uitgebracht. Jaartal van verschijning, speelduur, etc.

U kunt heel veel informatie opslaan, bijvoorbeeld de exacte samenstelling van het materiaal, maar dat is voor een potentiële klant niet echt van belang. Beslissingen over wat wel en wat niet op te nemen worden tijdens het ontwerpen van de database genomen. Dit is een erg belangrijke stap zoals u zult begrijpen!

Wij zullen voor de tabel `medewerkers` die we straks gaan aanmaken voor het gemak alleen de volgende eigenschappen van elke medewerker opslaan:

- Voornaam
- Achternaam
- Afdeling
- Toestelnummer

Uiteraard zal in een praktijkscenario veel meer data worden opgeslagen. Denk daarbij aan eigenschappen als:

- Adres
- Geboortedatum
- Datum in dienst
- Email
- Etc..

Maar dat zullen we voor ons voorbeeld nu niet doen.

De waarden voor deze afzonderlijke elementen vormen samen een unieke medewerker en worden vastgelegd in de **velden** van het record:

voornaam	achternaam	afdeling	toestelnummer
Jan	Jansen	Receptie	123

De velden worden bij het ontwerpen van de database bepaald en alle records in een tabel hebben altijd allemaal dezelfde velden en bevatten daardoor ook vergelijkbare gegevens

Om ervoor te zorgen dat de gegevens die in een record verzameld zijn ook weer teruggevonden kunnen worden is het noodzakelijk dat elk record **uniek** is.

In een tabel mogen dus niet twee exact dezelfde records voorkomen. Vaak wordt dit afgedwongen door een veld op te nemen waarin een unieke waarde staat voor elk record. Dat kan een bestaand gegeven zijn zoals een artikelnummer, ISBN-nummer of een andere unieke code. Maar wat ook heel veel voorkomt is een veld id dat voor elk nieuw toegevoegd record wordt opgehoogd.

Het veld of de velden die het record uniek identificeren worden met **primary key** aangeduid. Door het gebruik van een primary key kunnen twee records nooit exact hetzelfde zijn. Ook als alle andere velden dezelfde waarden hebben dan is het ID in ieder geval nog verschillend.

id	voornaam	achternaam	afdeling	toestelnummer
1	Jan	Jansen	Receptie	1234
2	Tanja	Vlucht	Development	1235

Een verzameling gelijksoortige records vormen samen een tabel. De **records** komen overeen met de **rijen** en de **velden** met de **kolommen**. Deze termen worden als synoniemen gebruikt.

Fields (columns)					Records (rows)
id	voornaam	achternaam	afdeling	toestelnummer	

Waar en hoe de gegevens exact in het bestandssysteem van de database server worden weggeschreven is voor ons niet zo van belang. Daar zorgt het DBMS voor. Wij communiceren met het RDBMS via een standaardtaal die door vrijwel elke database wordt begrepen: de **Structured Query Language (SQL)**.

5.4 Structured Query Language (SQL)

SQL is een afkorting van **Structured Query Language** (www.sql.org). De naam wordt overigens officieel uitgesproken door de letters los te noemen ("es-kjoe-el") maar veel gebruikers spreken het uit als *sequel* ("siekwel").

Deze speciale databasetaal, ontwikkeld door IBM halverwege de jaren '70, wordt gebruikt om data in relationele databases te structureren, op te halen en te manipuleren.

Query is een samentrekking van Question en Reply!

SQL werd **gestandaardiseerd** door **ANSI** (American National Standards Institute) in 1986 en een jaar later door **ISO** (International Organization for Standardization). Het is daarmee dé standaard taal voor relationele database management systemen geworden.

Er bestaan inmiddels **verschillende versies** van SQL. De **eerste** versie heette **ANSI SQL-86**. Deze kreeg een flinke update in 1992. Deze versie wordt ANSI SQL-92 genoemd. We zijn inmiddels bij de **zesde** en voorlopig **laatste** revisie en deze heet **ISO SQL:2008**. De dubbele punt is in de plaats van het streepje gekomen omdat dat bij andere ISO standaarden ook gebruikelijk is. Elke versie kent weer meer mogelijkheden, maar het moge duidelijk zijn dat wij in deze basiscursus alleen eenvoudige opdrachten zullen geven en deze zijn door alle versies heen hetzelfde gebleven.

Alle bekende relationele databasesystemen zijn compatibel met standaard SQL. Maar vele fabrikanten hebben ook hun eigen **merkspecifieke toevoegingen**. Er wordt in dit verband ook wel gesproken over **SQL dialecten**. Deze werken alleen in hun eigen databasesysteem. Denk aan **JET SQL** in Access, **T-SQL** in SQL Server en **PL/SQL** in Oracle. Ze hebben echter gelukkig allemaal standaard ANSI/ISO SQL als basis!

De standaard SQL commando's die wij gaan gebruiken om de data in de MySQL database te manipuleren, **SELECT**, **INSERT**, **UPDATE** en **DELETE**, worden door elk database systeem ondersteund.

SQL bestaat uit **drie delen**:

1. De Data Definition Language (DDL)
2. De Data Manipulation Language (DML)
3. De Data Control Language (DCL)

5.4.1 De Data Definition Language (DDL)

Met de Data Definition Language definieert en beheert u die onderdelen van een database die de data bevatten. Met DDL definieert u zo de **structuur** van de database. U kunt er bijvoorbeeld de tabellen mee aanmaken. Bij het opgeven van de veldnamen moet u aangeven wat voor type gegevens er in het veld moeten worden opgeslagen: u dient het **datatype** op te geven en welk veld de **primary key** bevat voor het uniek kunnen identificeren van de records in de tabel. In de cursus **PHP Vervolg** leert u ook hoe u **relaties** tussen tabellen kunt specificeren.

U kunt dergelijke opdrachten direct in SQL typen en eventueel ook door PHP laten uitvoeren maar meestal gaat dit echter makkelijker door een GUI tool te gebruiken die bij de DBMS wordt geleverd. Wij gaan tijdens de cursus dan ook niet rechtstreeks DDL opdrachten inkloppen, we laten die door **phpMyAdmin** voor ons genereren (zie paragraaf 5.5).

5.4.2 De Data Manipulation Language (DML)

Met de Data Manipulation Language kunt u de data uit de database ophalen of manipuleren, zonder dat hierbij de structuur wordt aangepast. U kunt zo gegevens **uitlezen**, **toevoegen**, **wijzigen** en **verwijderen**. Er kan natuurlijk veel meer, maar wij beperken ons in deze basiscursus tot deze 4 basishandelingen.

CRUD

Men spreekt in dit verband ook wel van **CRUD**. Het acroniem CRUD verwijst naar alle belangrijke functies die in een database applicatie aanwezig zouden moeten zijn om van een complete applicatie te kunnen spreken. Elke letter in het acroniem komt overeen met een standaard SQL statement:

Create	INSERT
Retrieve/Read	SELECT
Update	UPDATE
Delete/Destroy	DELETE

Als alternatieve term komt u ook wel **ABCD** tegen:
Add, Browse, Change, Delete.

We zullen de syntax en werking van deze opdrachten in het volgende hoofdstuk bespreken op het moment dat we ze nodig hebben.

Een SQL DML opdracht (commando) wordt een **query** genoemd (query is een afkorting van *question and reply*). In webapplicaties is de meest gebruikte query de **select query** waarmee u de gegevens selecteert die u nodig heeft. De andere queries worden **action queries** genoemd. Ze selecteren namelijk geen gegevens, ze voeren alleen een actie uit op de data in de database.

5.4.3 Data Control Language (DCL)

Met de Data Control Language regelt u het beheer van gebruikers en rechten. Hier vallen commando's als GRANT en REVOKE onder. We zullen er in deze cursus geen gebruik van maken.

5.5 phpMyAdmin

Het aanmaken van databases, tabellen en kolommen kunnen we handmatig doen in het geval van MySQL. We moeten dan een *shell* openen op de Linux server waar MySQL op geïnstalleerd is. Omdat deze werkwijze nogal bewerkelijk is, en niets toevoegt aan het leren PHP programmeren, maken we gebruik van een programma dat ons veel van dit werk uit handen neemt. Vrijwel elke, op Linux of Unix geïnstalleerde database server, heeft wel een dergelijk database administratie pakket. Voor MySQL is de meest populaire het gratis **phpMyAdmin**.



Omdat phpMyAdmin in PHP is geschreven is het beschikbaar voor Windows en Linux. Nadat u het configuratiebestand voor uw situatie heeft ingesteld dient u de files naar uw webserver te uploaden. Hierna kunt u uw MySQL databases eenvoudig online beheren. Veel **Internet Service Providers** maken dan ook gebruik van phpMyAdmin voor hun klanten.

Tip: Door de broncode van de pagina's te bestuderen kunt u veel over PHP en MySQL leren!

phpMyAdmin is te downloaden op <http://www.phpmyadmin.net> als een zip file van een kleine 5Mb. Dat hoeft u tijdens de cursus niet te doen, phpMyAdmin is onderdeel van XAMPP en reeds geïnstalleerd.

We zullen tijdens de cursus lang niet alle mogelijkheden van PMA bespreken maar er is een uitgebreide handleiding op http://wiki.phpmyadmin.net/pma/Welcome_to_phpMyAdmin_Wiki.

Tip: Heeft u behoefte aan iets kleineres? **Adminer** (<http://www.adminer.org/>) en **SQL Buddy** (<http://sqlbuddy.com/>) zijn volwaardige alternatieven die u simpelweg naar een webserver kunt uploaden en meteen gebruiken!

5.6 Database aanmaken

Bij het installeren van XAMPP werd ook phpMyAdmin geïnstalleerd. U kunt phpMyAdmin benaderen met een via <http://localhost/phpMyAdmin/>.

Normaal gesproken moet u bij een provider inloggen op phpMyAdmin en meestal is er dan al een database voor u aangemaakt. Tijdens de cursus hoeft u niet in te loggen en kan u gewoon zelf databases aanmaken:

Nieuwe database aanmaken

cursus Collatie Aanmaken

Let op: u kunt **geen spaties**, punten of andere vreemde tekens in de naam van de database gebruiken!

5.7 Tabellen aanmaken

U zit nu meteen in de beheeromgeving voor uw eigen database. Hier kunt u nieuwe tabellen aanmaken. Dat gaat als volgt:

1. Ga naar *Nieuwe tabel aanmaken in database cursus*: en geef een naam op voor uw nieuwe tabel en het aantal kolommen waaruit hij gaat bestaan:



Klik op Start.

2. Geef de kolommen een naam en stel het gewenste datatype in:

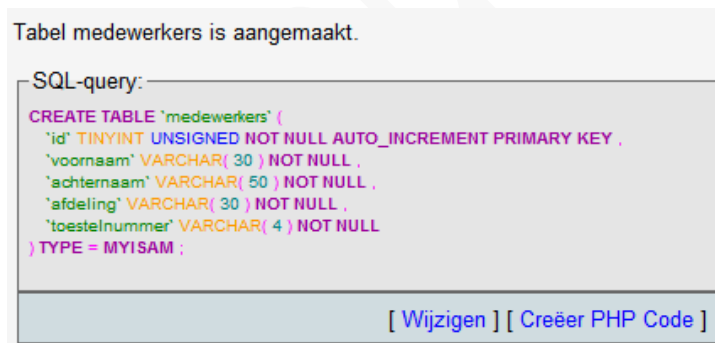
Server: localhost Database: cursist1 Tabel: medewerkers

Veld	Type ?	Lengte/Waardes*1	Collatie
id	TINYINT		
voornaam	VARCHAR	30	
achternaam	VARCHAR	50	
afdeling	VARCHAR	30	
toestelnummer	VARCHAR	4	

Het id veld is in deze tabel een speciaal veld. Het is de zogenaamde **primary key**, waarmee elk record in de tabel uniek geïdentificeerd kan worden. Een primary key moet **uniek** zijn en kan daarom **niet NULL** zijn. En er kan maar **één primary key per tabel** zijn.

Het is vrij gebruikelijk om voor de primary key een numeriek veld te gebruiken dat door de database automatisch opgehoogd wordt (**auto increment**) wanneer een nieuw record wordt toegevoegd.

3. Klik op Opslaan. Uw tabel is aangemaakt! In phpMyAdmin ziet u dat hiertoe deze SQL DDL query werd uitgevoerd:



Het grote voordeel van een interface als phpMyAdmin is dat u dergelijke statements voor u kunt laten genereren. U hoeft niet alle syntax uit uw hoofd te kennen en er is geen kans op typefouten!

Maar als u het wil kunt u dergelijke opdrachten altijd met de hand geven. Dat kan via phpMyAdmin, maar ook via PHP of shell commands. Wij zullen dat tijdens de cursus echter niet doen...

Oefening 5.1 *Mak in uw eigen database een tabel met de naam `medewerkers` en in die tabel de volgende kolommen:*

- `id` [TINYINT, auto_increment, primary key]
- `voornaam` [VARCHAR, 30]
- `achternaam` [VARCHAR, 50]
- `afdeling` [VARCHAR, 30]
- `toestelnummer` [VARCHAR, 4]

Oefening 5.2 *Vul de `medewerkers` tabel via de knop `Invoegen` met een aantal records, bijvoorbeeld:*

Veld	Type	Functie	Null	Waarde
id	tinyint(3) unsigned			
voornaam	varchar(30)			Karin
achternaam	varchar(50)			Boogaard
afdeling	varchar(30)			Directie
toestelnummer	varchar(4)			1234

☐ Negeer

Veld	Type	Functie	Null	Waarde
id	tinyint(3) unsigned			
voornaam	varchar(30)			Peter
achternaam	varchar(50)			Spanje
afdeling	varchar(30)			Sales
toestelnummer	varchar(4)			1235

Voeg toe als nieuwe rij and then Terug

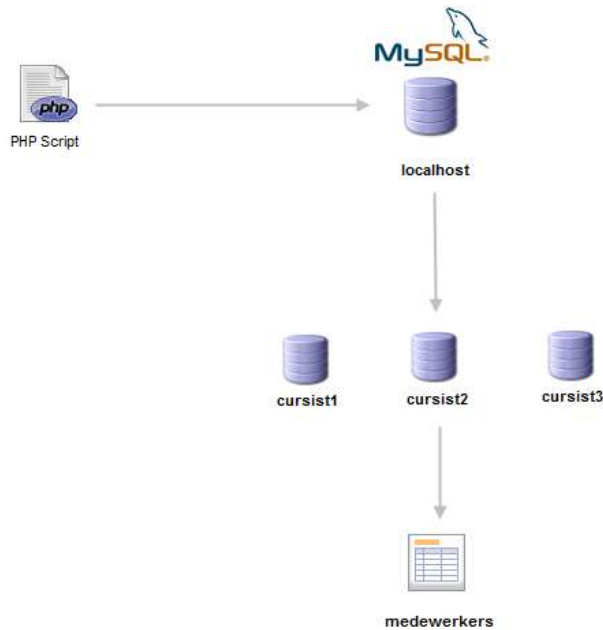


etc... Voor het veld `id` vult u geen waarden in, dat doet MySQL dankzij het attribuut `auto_increment` zelf! Zoals u ziet is het in phpMyAdmin mogelijk om twee records tegelijk toe te voegen. Zet minimaal een zestal medewerkers in de tabel, dan heeft u straks ook echt wat om in uw pagina te tonen.

5.8 Afsluiting

U heeft nu een database met daarin een tabel met medewerker gegevens. Een logische volgende stap is deze gegevens uit te lezen. Voordat we echter opdrachten aan onze database kunnen geven moeten we eerst weten hoe we er verbinding mee maken.

In vogelvlucht ziet dat proces er zo uit:



Vanuit uw PHP script maakt u straks verbinding met MySQL Server. MySQL is op dezelfde machine geïnstalleerd als waar de webserver op draait en is daardoor benaderbaar onder de naam `localhost`. Op de database server zijn al meerdere databases aangemaakt, voor elke cursist één. Binnen uw eigen database heeft u zojuist de tabel `medewerkers` aangemaakt waar we in het volgende hoofdstuk de gegevens uit zullen gaan halen om deze in de pagina te kunnen tonen.

Voor het kunnen toevoegen, aanpassen en verwijderen van medewerkers hebben we **formulieren** nodig. Denk daarbij aan wat in het vorige hoofdstuk werd gezegd over de verschillen tussen GET en POST:

HTTP	CRUD
GET	Read
POST	Create, Update, Delete

Voor het uitlezen en navigeren kunnen we gewone pagina's opvragen en hyperlinks gebruiken middels GET requests. Voor de action queries zullen we echter formulieren moeten gebruiken die data via POST requests verzenden!

In het volgende hoofdstuk gaan we uitgebreid op deze zaken in...

Let op: Tijdens de cursus maakt u gebruik van een MySQL database. Deze database bevat slechts **één tabel**: de tabel `medewerkers`. Relationele databases bestaan **in de praktijk** natuurlijk altijd uit **meerdere gerelateerde tabellen**. (MySQL is dan ook een RDBMS: een *Relationeel* DataBase Management Systeem).

Wij gebruiken tijdens de cursus echter voor het gemak slechts één tabel. Zo kunt u zich concentreren op het werken met PHP en de database. Bij meerdere tabellen verandert deze werkwijze namelijk niet wezenlijk: de SQL SELECT queries worden alleen wat complexer door het gebruik van JOINS. Maar er gaat vooral meer werk zitten in het realiseren van de beheerschermen en helaas hebben we die tijd niet tijdens deze basiscursus. Het werken met gerelateerde tabellen wordt uitgebreid behandeld in de cursus **PHP Vervolg**.

Database gebruiken

6.1 Inleiding

Nu wordt het echt interessant! Want in dit deel van de cursus leert u hoe u PHP kunt gebruiken in combinatie met een database. We gebruiken daarvoor de zojuist aangemaakte MySQL cursus database.

U leert niet alleen hoe u gegevens uit een database in uw pagina's kunt tonen maar ook hoe u via een webinterface nieuwe gegevens kunt toevoegen en bestaande gegevens kunt aanpassen of verwijderen. Het zwaartepunt zal echter liggen op het tonen van gegevens omdat bij de meeste webapplicaties het uitlezen van een database ook het meest frequent voorkomt.

Een overzicht van de door u te bouwen 'mini-applicatie' vindt u in **bijlage 8.4**.

PHP biedt **native support** voor tal van databases. Het voordeel is dat de database hierdoor op de **meest optimale manier** wordt aangesproken en dat u alle features van de specifieke database server kunt aanspreken. Een nadeel is dat de beschikbare functies per database verschillen waardoor uw code **minder portabel** is. Als het voor uw project van belang is dat er later makkelijk van database gewisseld moet kunnen worden, dan kunt u zelf een **database abstractielaag** bouwen of een bestaande gebruiken zoals **PEAR DB** of **ADODB**. Een andere universele oplossing is het gebruik van **PHP Data Objects (PDO)**. Dit zijn echter allemaal geavanceerdere technieken die helaas niet in deze basis cursus behandeld kunnen worden...

PHP kan weliswaar gebruik maken van de **ODBC** (Open DataBase Connectivity) standaard, maar de gemiddelde hosting provider ondersteunt dit helaas niet. Voor een intranet applicatie kan het echter een prima keuze zijn! Zie <http://php.net/odbc> voor meer informatie.

In deze cursus gaan we, zoals al in hoofdstuk 2 werd aangegeven, uit van het gebruik van de populaire **MySQL** database (<http://www.php.net/mysql>). Voor andere databases verwijzen wij u naar de online documentatie op <http://php.net/manual/en/refs.database.php>.

Alle in dit hoofdstuk besproken MySQLi functies (en nog veel meer) vindt u terug via <http://php.net/manual/en/mysql.summary.php>.

6.2 MySQL API kiezen

PHP kent (als we ODBC niet meetellen) maar liefst drie verschillende manieren om met een MySQL database server te verbinden:

- ~~1. MySQL API~~
2. **MySQLi API**
 - a. Procedureel
 - b. Object georiënteerd
3. **PDO_MySQL API** (enkel object georiënteerd)

API staat voor Application Programming Interface, en definieert de manier waarop u de ter beschikking gestelde functionaliteit kunt aanspreken.

Afhankelijk van uw PHP versie kunt u of alle drie, of enkel de laatste twee van de hierboven genoemde API's gebruiken om met een MySQL database te werken, want:

In PHP7 is de verouderde MySQL API niet meer beschikbaar!

6.2.1 MySQL API

De MySQL API is jarenlang dé manier geweest om binnen PHP met MySQL te werken. PHP en MySQL zijn als het ware twee handen op een buik. Je zou kunnen zeggen dat ze samen zijn opgegroeid. Inmiddels zijn ze beiden behoorlijk volwassen geworden maar de functiebibliotheek `/etc/mysql/` die velen gebruiken om vanuit PHP MySQL te benaderen is nogal verouderd. Deze bibliotheek is zelfs nog ouder dan PHP3, de versie waarin er voor het eerst gebruik van gemaakt kon worden. En PHP3 verscheen in 1997!

De MySQL extensie wordt sinds PHP 4.3.1 enkel nog onderhouden, dus de laatste keer dat er nieuwe functionaliteit aan werd toegevoegd was ergens aan het eind van 2002! De geïntegreerde MySQL API is die van MySQL 4.0.0 en alle nieuwe features die sindsdien door MySQL aan de API zijn toegevoegd worden dus niet ondersteund! Tegenwoordig wordt het gebruik ervan voor nieuwe projecten afgeraden en het gebruik van MySQLi of PDO aanbevolen:

mysql_connect

(PHP 4, PHP 5)

mysql_connect — Open a connection to a MySQL Server



Warning

This extension is deprecated as of PHP 5.5.0, and will be removed in the future. Instead, the [MySQLi](#) or [PDO_MySQL](#) extension should be used. See also [MySQL: choosing an API](#) guide and [related FAQ](#) for more information. Alternatives to this function include:

- [mysqli_connect\(\)](#)
- [PDO::__construct\(\)](#)

Als u binnen PHP 5.5 de functie `mysql_connect()` gebruikt dan krijgt u dat, in Strict Mode, ook te zien:

Deprecated: mysql_connect(): The mysql extension is deprecated and will be removed in the future: use mysqli or PDO instead

In **PHP7** zijn de `mysql_*` functies **sowieso niet meer beschikbaar**:

Fatal error: Uncaught Error: Call to undefined function mysql_connect() in demo.php

Bestaande applicaties kunnen in PHP5 in principe gewoon met de oude MySQL API blijven werken. In PHP7 is die alsnog achteraf toe te voegen, maar echt aan te raden is het niet...

Upgraden naar `mysqli` is helaas niet een kwestie van een simpele zoek-en-vervang opdracht. Omdat het een erg populaire extensie is zal het overigens best even duren voordat hij echt uit beeld verdwijnt. Zie de **bijlagen** voor een voorbeeld van een script dat de MySQL API gebruikt en waar u bij omzetten naar MySQLi tegenaan kunt lopen. Meer informatie over het afstoot proces vindt u op https://wiki.php.net/rfc/mysql_deprecation.

Als u nog behoefte mocht hebben aan meer redenen om de MySQL API niet te gebruiken:
<http://stackoverflow.com/questions/12859942/why-shouldnt-i-use-mysql-functions-in-php>

PDO heeft het voordeel dat de syntax hetzelfde blijft als u besluit om een andere database te gaan gebruiken, bijvoorbeeld SQLite of PostgreSQL. Bovendien heeft het een elegante syntax. Het is echter volledig object georiënteerd en we gaan er daarom pas na een goede introductie in OOP principes mee aan de slag in een latere cursus.

MySQLi kunt u zowel **object georiënteerd** als **procedureel** gebruiken. Wij kiezen tijdens de basiscursus voor de procedurele manier van werken. Dit is voor beginners de makkelijkste manier van coderen en heel herkenbaar als u al eens met de mysql API in aanraking bent geweest. In een vervolg cursus leert u de OOP manier gebruiken.

Let op: De procedurele manier en de OOP manier kunnen **niet door elkaar** heen gebruikt worden in één script! Met andere woorden: als u op een procedurele manier verbinding maakt dan moet u daarna ook procedureel met de openstaande verbinding werken.

Meer informatie over de API's en hun verschillen vindt u in de PHP manual op:
<http://www.php.net/manual/en/mysqlinfo.api.choosing.php>.

6.2.2 mysqlnd

Een term die u wellicht ook bent tegengekomen is **mysqlnd** waarin "nd" staat voor **Native Driver**. Dit is echter geen API waar u als programmeur direct mee zult werken. Het is een PHP extensie die dient als vervanging van de MySQL Client Library (`libmysql`). Toen MySQL door Oracle werd overgenomen werd die om licentie technische redenen uit PHP gehaald. `mysqlnd` is vervolgens door het PHP team ontworpen als vervanging. Het heeft een vergelijkbare tot betere performance en biedt extra functionaliteit. Een voorbeeld is de functie `mysqli_fetch_all()` die we in **6.15** bespreken. **Vanaf PHP 5.3** wordt `mysqlnd` **standaard** door de MySQL API en PDO_MYSQL API gebruikt.

U leest meer over de native driver op <http://www.php.net/manual/en/book.mysqlnd.php>.

6.3 MySQLi

Om de nieuwe features van PHP5 en MySQL 4.1 te kunnen gebruiken heeft de ontwikkelaar van de oude MySQL functiebibliotheek, Georg Richter, een verbeterde versie uitgebracht die luistert naar de naam **mysqli**. De *i* staat hier voor *Improved*. De bibliotheek is ontworpen om samen te werken met MySQL versies > 4.1.3.

MySQLi is sinds PHP 5.0 beschikbaar en maakt sinds 5.3.0 gebruik van `mysqlnd`.

`mysqli` heeft meer dan 100 ingebouwde functies. `mysql` heeft er daarentegen maar 48... We kunnen ze hier helaas niet allemaal bespreken, maar alle `mysqli` functies staan handig bij elkaar op <http://php.net/manual/en/mysqli.summary.php>

6.3.1 Installatie

Het kan zijn dat de **mysqli** functionaliteit standaard nog niet geactiveerd is, u moet dat dan nog in `php.ini` aangeven. Onder Windows doet u dat simpelweg door bij de dynamic extensions voor deze regel het commentaartekens ; weg te halen:

```
extension=php_mysqli.dll
```

Denk er aan dat daarna de webserver opnieuw opgestart moet worden!

Het hoeft overigens geen keuze te zijn tussen de MySQL API en de MySQLi API. U kunt zowel de `mysql` extensie als de `mysqli` extensie tegelijkertijd activeren, ze werken dan naast elkaar. Uiteraard gebruikt u ze binnen een project niet door elkaar!

Als u XAMPP gebruikt dan zijn zowel MySQL als MySQLi gewoon gelijk beschikbaar...

6.3.2 Voordelen

MySQLi maakt de volgende **nieuwe features** van MySQL beschikbaar:

- multiple statements
- prepared statements
- bound parameters
- bound results
- stored procedures
- subselects
- transactions
- unicode support

Iets wat in de praktijk veel voorkomt is dat een query meerdere malen moet worden uitgevoerd met wisselende parameters. Dit kan met het oude `mysql_query()` alleen door middel van een lus. Dit vergt de nodige overhead en is codetechnisch ook minder prettig. In `mysqli` wordt dit probleem van herhaalde queries fraai opgelost met prepared statements.

Voor veel programmeurs is alleen al de mogelijkheid van **prepared statements** de hoofdreden om op `mysqli` over te stappen. Want deze bieden ook nog eens een hele goede beveiliging tegen SQL Injection Attacks! Wij gaan hier in de **vervolg cursus** mee aan de slag.

Verder is de **snelheid** verbeterd, men claimt dat sommige acties tot 40x (!) sneller kunnen worden uitgevoerd! En ook de **veiligheid** is verbeterd want de meegezonden passwords zijn nu op het niveau van SSH versleuteld en na onderscheppen van het netwerkverkeer dus minder makkelijk te ontcijferen.

Bovendien maakt `mysqli` gebruik van het nieuwe binaire protocol waarbij niet langer meer alle data eerst naar strings moet worden omgezet maar gewoon hun binaire vorm blijven behouden.

6.3.3 Nadelen

Nadelen zijn er niet echt maar `mysqli` is **niet geheel backwards compatible** met `mysql` code. Als u in uw applicatie niet de een of andere vorm van een database abstraction layer geïmplementeerd heeft dan vergt de conversie van `mysql` naar `mysqli` de nodige **aanpassingen** en kan daardoor best **tijdrovend** zijn...

Voor nieuw te bouwen applicaties is dit natuurlijk geen issue!

6.3.4 Migratie MySQL naar MySQLi

Hoewel de `mysqli_*` functies vaak sterk lijken op hun oudere `mysql_*` naamgenoten kunt u niet simpelweg een zoek en vervang opdracht geven. De volgorde van de argumenten is bijvoorbeeld anders en sommige functies bestaan niet meer in `mysqli`.

Een overzicht van de verschillen vindt u in de **bijlagen**.

Het **grootste verschil** is dat u in de procedurele `mysqli` API de **link identifier** steeds als **verplicht eerste argument** moet opgeven, nadat u de verbinding hebt gemaakt. In de `mysql` API was dat optioneel en dan was het de laatste parameter...

U kunt de code herschrijven of gebruik maken van het **migratiescript** `mysql2i` van John Coggeshall (<http://code.google.com/p/mysql2i/>). Een andere handige is <https://sourceforge.net/projects/mysqlwithmysqli/> waarmee u toch mysql notaties kunt gebruiken in een omgeving waar alleen mysqli beschikbaar is (PHP7!).

6.3.5 Meer informatie

Meer informatie over MySQLi vindt u natuurlijk in de online PHP manual: <http://php.net/mysqli> en deze artikelen bij ZEND:

<http://www.zend.com/php5/articles/php5-mysqli.php>
<http://www.zend.com/php5/articles/php5-mysqli2.php>

6.4 Stappenplan

Het eerste wat we willen doen is de data tonen die we in het vorige hoofdstuk in de MySQL database hebben opgeslagen. De stappen die we daarvoor moeten doorlopen zijn altijd dezelfde (ook met andere databases of andere Server-side technologieën):

1. Verbinding maken met de database server
2. De database selecteren waar u mee wilt werken
3. Een SQL Select statement opstellen
4. Het SQL Select statement laten uitvoeren
5. Door de resultset heen lopen om de gegevens te tonen
6. De resultset vrij geven
7. De verbinding sluiten

We zullen de stappen hierna gedetailleerd doorlopen. Aan het einde van dit onderdeel vindt u een compleet voorbeeld script.

In de **bijlagen** kunt u een uitwerking van het voorbeeldscript voor de verouderde MySQL API vinden.

6.5 Stap 1: Verbinden met MySQL Server

De eerste stap in het werken met een database bestaat zoals gezegd altijd uit het maken van een verbinding met de database server.

Met de functie `mysqli_connect()` maakt u de verbinding met een MySQL Server. De functie heeft maar liefst **zes optionele argumenten** waarmee u achtereenvolgens de *hostname*, de *database username* en het bijbehorende *password*, de *database*, het *poortnummer* en de *socket* kunt opgeven. Die laatste twee zijn bij een standaard installatie sowieso niet van belang, u kunt ze simpelweg weglaten (voor de liefhebber: de standaardwaarden zijn respectievelijk '3306' en 'MySQL').

Ook de eerste vier argumenten kunt u in principe weglaten. Dan wordt aangenomen dat de eigenaar van de webserver een verbinding wil leggen met een lokaal draaiende MySQL Server zonder dat daar een password voor nodig is en zonder dat er een database wordt geselecteerd.

Tijdens de cursus maken wij meestal gebruik van XAMPP en dan is in feite bovenstaande situatie van toepassing. De MySQL Server is op dezelfde machine geïnstalleerd als de webserver en de servername is dus `localhost`. Tijdens de installatie werd enkel de beheerder `root` als user aangemaakt en deze heeft standaard nog geen password (!)...

U kunt dus op de volgende manier verbinding maken:

```
$conn = mysqli_connect();  
  
// is hetzelfde als:  
$conn = mysqli_connect('localhost', 'root', '');
```

Andere opties voor de servernaam zijn bijvoorbeeld de **netwerknaam** van een MySQL server in het netwerk van uw organisatie of het **IP-adres** van een MySQL Server op het internet. Informeer bij uw provider naar de situatie.

Bij succesvol verbinden retourneert de `mysqli_connect()` functie een zogenaamde **link identifier**. Deze kunt u opslaan in een variabele, vaak `$conn`, `$link` of `$dbh` (*database handler*) genoemd. Wij zullen tijdens de cursus consequent `$conn` gebruiken.

Deze variabele kunt u later in uw script gebruiken om aan te geven dat u met deze specifieke verbinding wilt werken. Het is namelijk mogelijk om in één PHP pagina **meerdere verbindingen** te maken naar verschillende MySQL Servers. Als u de variabele weglaat dan wordt automatisch aangenomen dat hierna de **laatst geopende verbinding** moet worden gebruikt.

Als het maken van de verbinding **niet lukt** (bijvoorbeeld door een typefout in één van de argumenten, of het password is gewijzigd, of de database server is down) dan retourneert de `mysqli_connect()` functie `FALSE`. Standaard gaat dit gepaard met een foutmelding op het scherm die u de eindgebruikers liever niet laat zien:

Warning: mysqli_connect(): (HY000/1045): Access denied for user 'root'@'localhost' (using password: YES) in **test.php** on line 2

Dit geeft helaas een hoop informatie prijs. Een kwaadwillende gebruiker weet nu dat er gebruik gemaakt wordt van het root account om met mysqli verbinding te maken met een lokale MySQL database. Het enige wat nog ontbreekt is het wachtwoord... En een normale gebruiker zit niet op dergelijke informatie te wachten... Het is geen fraai gezicht en deze gebruikers kunnen er helemaal niets mee.

Door voor de functie de **error suppression operator** `@` (ook wel error control of *shut up* operator genoemd) te plaatsen, zou u deze foutmeldingen kunnen onderdrukken:

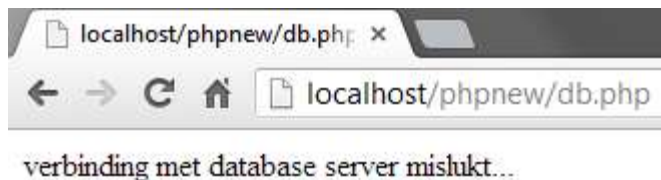
```
$conn = @mysqli_connect('localhost', 'root', '');
```

Maar dit is een paardenmiddel wat bij voorkeur niet gebruikt moet worden. Een **betere oplossing** is om het tonen van **fouten** te **onderdrukken** (standaard bij veel providers of anders met een `.htaccess` bestand) en de fouten te loggen.

In beide gevallen krijgt de eindgebruiker echter helemaal geen feedback, slechts een lege pagina. U kunt er daarom voor kiezen om een gebruikersvriendelijke boodschap op het scherm te tonen. Meestal wordt hiervoor de functie `die()` gebruikt:

```
$conn = mysqli_connect('localhost', 'root', '')  
or die('verbinding met database server mislukt...');
```

Wanneer het niet lukt om een verbinding te maken dan zal de string `'verbinding met database server mislukt...'` in een soort laatste ademtocht naar de eindgebruiker gezonden worden, waarna het script stopt:



Code hierna wordt dus niet meer uitgevoerd, wat in principe prima is omdat dit door het ontbreken van de verbinding toch maar tot allerlei vervolgfouten zou leiden. Maar deze werkwijze heeft echter wel tot gevolg dat de eindgebruiker met een **onafgemaakte pagina** blijft zitten. Later leert u hoe u dit door het **insluiten** van **error templates** kunt oplossen!

6.6 Stap 1a: Charset specificeren

Hoewel niet als een aparte stap in het overzicht genoemd, is het van wezenlijk belang om er voor te zorgen dat het duidelijk is welke karakterset moet worden gebruikt tijdens het werken met MySQL. Dit heeft niet alleen invloed op de weergave van speciale karakters in de HTML pagina, maar het is ook van wezenlijk belang voor de veiligheid omdat de werking van functies als het later in 6.22.5 te bespreken `mysqli_real_escape_string()` er vanaf hangt.

Zoals in de HTML cursus besproken wilt u uw pagina's in utf-8 tonen. Daarvoor moet de hele ketting van database tot pagina op utf-8 zijn ingesteld. Helaas valt MySQL standaard terug op Latin 1 bij het maken van de verbinding als u geen andere default charset opgeeft. Dit gaat ook op als u bij het aanmaken van de database en de tabellen overal voor utf-8 gekozen heeft.

Wanneer het misgaat krijgt u overigens uitvoer als deze:

Fabi  ne

En dat bent u vast al eens live tegengekomen op het web... Hoe gaan we dit nu oplossen?

Met de functie `mysqli_character_set_name()` kunt u uitlezen welke charset gebruikt wordt nadat u verbinding gemaakt heeft. U moet als verplicht argument de link identifier `$conn` meegeven.

```
echo mysqli_character_set_name($conn);
```

Dit geeft bijvoorbeeld:

latin1

Of als u geluk heeft en alles al goed is ingesteld:

utf8

Let op: In lijn met de de verouderde MySQL API was er ook een functie `mysqli_client_encoding()` maar die bestaat niet meer vanaf PHP 5.4. Er is tevens een functie met de logische naam `mysqli_get_charset()` maar deze retourneert een *object* en dit kunt u *niet* zomaar wegschrijven...

Als u `latin1` te zien krijgt dan moet u maatregelen treffen. U kunt het zetten van de charset het beste regelen op de server, als u daar rechten toe heeft. Maar bij een provider kan dat natuurlijk niet... We laten zo zien hoe u het dan kunt oplossen.

Database gebruiken

Het aanpassen op de server doet u door de juiste karakterset op te geven in `my.ini`, het **configuratiebestand van MySQL**, aan te passen, als dat niet al gebeurd is bij installatie:

```
[mysql]  
default-character-set=utf8
```

U kunt het vinden in `C:\ProgramData\MySQL\MySQL Server 5.5\my.ini` bij een standaard Windows installatie.

Tip: u kunt dit ook eenvoudig regelen via de **MySQL Workbench** via Server Administration -> Configuration -> Options File

Als u XAMPP gebruikt dan kunt u het vinden in `C:\xampp\mysql\bin\my.ini`. Maar een makkelijke manier is om dit via het XAMPP control Panel te regelen:



En dan in `my.ini` naar de sectie UTF8 Settings te gaan:

```
...  
  
## UTF 8 Settings  
#init-connect='SET NAMES utf8\  
#collation_server=utf8_unicode_ci  
  
# UNCOMMENTED BY ARNO  
character_set_server=utf8  
  
#skip-character-set-client-handshake  
#character_sets_dir="C:/xampp/mysql/share/charsets"  
  
...
```

Denk er wel aan dat u **MySQL** vervolgens moet **herstarten**! U doet dit snel via de knop Stop en vervolgens Start.

Maar wat nu als u geen rechten op de server heeft?

Bij het werken met de oude MySQL API kon u de charset gelijk na het openen van de verbinding instellen met een query. Dat kan in de MySQL API nog steeds:

```
// verbinding openen en dan gelijk daarna charset instellen met query  
mysqli_query($conn, "SET NAMES 'utf8'");
```


Als u dit doet denk er dan aan dat `utf8` tussen single quotes moet staan!

Maar vanaf PHP 5.0.5 is voor het instellen van de karakterset de speciale `mysqli_set_charset()` functie bedoeld. Deze neemt als eerste verplichte argument de link identifier `$conn` en als tweede argument de naam van de charset als string:

```
// stap 1a: set charset specificeren
mysqli_set_charset($conn, 'utf8')
or die('Character set kon worden niet ingesteld. ' .
      'De huidige character set is: ' . mysqli_character_set_name($conn));
```

Normaal gesproken zou dit natuurlijk altijd moeten lukken tenzij u een typefout maakt in de naam van de charset. De `or die()` is dan ook voornamelijk bedoeld omdat nog even te controleren! Let op de onlogische naam, er bestaat vreemd genoeg geen `mysqli_get_charset()`.

Let op: U moet voor MySQL `utf8` gebruiken en **niet** `utf-8` zoals in de HTTP headers! `utf-8` wordt niet herkend en dan wordt automatisch terug gevallen op `latin1`.

6.7 Stap 2: Database Selecteren

Als de verbinding met MySQL geslaagd is, dan is het tijd voor de volgende stap: de te gebruiken database selecteren. Op één MySQL server kunnen namelijk meerdere databases draaien. Dat selecteren doet u met de functie `mysqli_select_db()` waarbij u als tweede argument de naam van de te selecteren database opgeeft.

Tijdens de cursus heeft iedere cursist zijn eigen database, genaamd `cursus`, aangemaakt. De benodigde code is dan:

```
mysqli_select_db($conn, 'cursus')
or die('database is niet beschikbaar...');
```

6.7.1 Verbinden en selecteren combineren

Met de (procedurele) MySQLi API kunt u bij het maken van de verbinding met een MySQL server ook gelijk de database selecteren waarmee u wilt werken:

```
$conn = mysqli_connect('localhost', 'root', '', 'cursus');

// dit is hetzelfde als
$conn = mysqli_connect('localhost', 'root', '');
mysqli_select_db($conn, 'cursus');
```

Deze manier van noteren is net even compacter maar minder flexibel. U kunt tijdens het script minder makkelijk van database switchen en ook uw foutmeldingen zullen wat generieker moeten zijn...

6.8 Stap 3: Query opstellen

Nu u weet hoe u een verbinding met een database kunt openen wordt het tijd voor de volgende stap. We willen uiteraard de records die we in de database `cursus` in de tabel `medewerkers` hebben staan in een pagina kunnen tonen.

Het ophalen van informatie uit een database tabel, oftewel het selecteren van data, doen we met behulp van een SQL SELECT statement.

6.8.1 Gegevens selecteren: SELECT

Met een SQL SELECT statement bepaalt u welke records en welke velden uit welke tabel(len) getoond moeten worden. SQL SELECT statements kunnen daarom behoorlijk complex worden omdat u er alle data uit een database mee boven water moet kunnen halen. Bovendien kent elk database systeem nog eigen toevoegingen aan de standaardnotatie. Dat geldt ook voor MySQL, maar wij zullen zoals gezegd tijdens deze cursus alleen standaard ANSI notaties gebruiken en alleen data ophalen uit één tabel.

In eerste instantie gaan we echter voor een eenvoudig statement, we willen alle velden en alle records uit een tabel. We nemen als voorbeeld de tabel `telnrs` met twee velden: `naam` en `telefoon`. Tijdens de oefeningen gebruikt u uiteraard de door u aangemaakte tabel `medewerkers`.

U gebruikt het SQL SELECT statement om de database te doorzoeken en de gegevens op te halen die voldoen aan de door u opgegeven criteria. In zijn meest simpele vorm ziet een SELECT statement er zo uit:

```
SELECT veldna(a)m(en)
FROM tabelnaam;
```

Na het woord SELECT geeft u met behulp van veldnamen in de zogenaamde *fieldlist* aan welke kolommen in het resultaat zullen zitten. De tabelnaam achter het woord FROM geeft aan welke tabel er doorzocht wordt om de gewenste gegevens op te halen.

Zoals eerder gezegd zullen we in de basiscursus uitgaan van data die afkomstig is uit één tabel. In de praktijk zult u met FROM ook meerdere (joined) tables opgeven (of een view, of subqueries, of derived tables...).

Wanneer u meer dan 1 veldnaam gebruikt dan worden de veldnamen met komma's gescheiden:

```
SELECT veldnaam1, veldnaam2
FROM tabelnaam;
```

Let op: Denk er aan dat **na de laatste veldnaam geen komma** mag staan!

Wanneer u alle namen en telefoonnummers uit de tabel `telnrs` zou willen selecteren dan ziet het benodigde SQL SELECT statement er dus zo uit:

```
SELECT naam, telefoon
FROM telnrs
```

6.8.2 SELECT statement in PHP String

Om een dergelijk statement in combinatie met PHP te gebruiken, kunt u het SQL statement het best als string opslaan in een variabele die u bijvoorbeeld `$query` of `$sql` noemt. Wij gebruiken tijdens de cursus consequent `$query`:

```
$query = "SELECT naam, telefoon FROM telnrs;"
```

SQL statements eindigen met een puntkomma

Het statement hoort volgens de ANSI standaard met een `;` te worden afgesloten. Mocht u hem vergeten dan gaat het echter wel goed in PHP, dat toch slechts maximaal 1 query tegelijk kan uitvoeren...

Werkt u via een telnet sessie op de commandline met MySQL dan is het gebruik van de puntkomma

echt noodzakelijk omdat het statement anders niet wordt uitgevoerd... U gaat dan na een [enter] gewoon naar de volgende regel!

In het PHP statement hierboven ziet u dus **twee** puntkomma's staan, **één voor het einde van het SQL statement** en **één voor het einde van het PHP statement**.

U slaat het statement op in een variabele voor een betere leesbaarheid en omdat u het dan makkelijk even weg kunt schrijven in de pagina om het op juistheid te controleren. Dat is nu nog niet zo spannend omdat in bovenstaand voorbeeld nog geen variabelen worden gebruikt, maar later in dit hoofdstuk zullen we dat wel doen en dan is het erg prettig om snel te kunnen controleren of alles wel klopt voordat u de query gaat uitvoeren. U kunt dat snel doen met de volgende code:

```
// toon query en stop uitvoer script  
die($query);
```

6.8.3 Case Sensitivity

Keywords als SELECT en FROM in een SQL statement zijn **niet hoofdlettergevoelig**. Het is echter een goed gebruik om ze in **kapitalen** te noteren omdat het uw statements veel leesbaarder maakt doordat het **contrast** met veld- en tabelnamen groter wordt.

Of veld- en tabelnamen hoofdlettergevoelig zijn hangt af van de database en het onderliggende besturingssysteem. Wanneer MySQL op een Linux of Unix machine draait dan zijn de tabelnamen **case-sensitive**! Dat komt omdat ze als bestanden worden weggeschreven in het filesystem. Veldnamen binnen een tabel zijn in principe niet hoofdlettergevoelig! Maar let op: als u een veldnaam meer dan eens in een query gebruikt dient u er wel voor te zorgen dat ze beide keren exact hetzelfde genoteerd zijn!

Tijdens de cursus is dit ook het geval. Als u de tabel `medewerkers` genoemd heeft dan moet u dus niet `Medewerkers` proberen uit te lezen! Maar het is natuurlijk sowieso aan te raden om altijd exact dezelfde notatie te gebruiken als in de database gedefinieerd is...

6.8.4 Selecteer alleen de benodigde velden

Wanneer u gegevens uit de database wil ophalen dan is een gouden regel dat u niet meer moet ophalen dan u daadwerkelijk nodig heeft. Selecteer alleen de velden die u ook daadwerkelijk gaat gebruiken: *Always specify an explicit field list!*

U mag namelijk op zich als u alle velden wilt selecteren ook de **wildcard** * (asterisk) gebruiken:

```
$query = "SELECT * FROM telnrs;";
```

Deze query selecteert **alle velden** in de `telnrs` tabel. Maar het is dus erg inefficiënt om dit altijd maar te doen! Soms heeft u misschien maar 2 van de 20 aanwezige velden nodig. Door de velden specifiek op te geven, haalt u alleen op wat u echt nodig heeft. Bovendien bestaat altijd de mogelijkheid dat **later** velden worden **toegevoegd** aan de database en deze neemt u met een wildcard dan automatisch (ongewild) mee... Sommige ontwikkelaars beschouwen dit overigens ook als een voordeel...

Een andere reden waarom het sowieso handig is om in uw SELECT statements altijd **alle veldnamen uit te schrijven**, ook wanneer u alle velden wilt gebruiken, is dat u dan in uw code **overzicht** over de opbouw van de tabel heeft. U hoeft niet naar de database te gaan om te zien hoe de velden ook al weer heten. De beschikbare velden staan immers gewoon in de SELECT query!

Maar het belangrijkste voordeel is misschien wel het feit dat uw query met een expliciete fieldlist in de meeste databases (iets) **sneller** verwerkt kan worden door de database! Het extra typewerk is eenmalig maar van een snellere verwerking blijft u steeds profijt hebben!

```
// in plaats van de wildcard waarmee u vaak
// teveel selecteert en
// niet snel kunt zien hoe de velden heten
$query = "SELECT * FROM telnrs;";

// kunt u beter de velden uitschrijven
// dan haalt u alleen op wat u echt nodig heeft
// bovendien kan uw query zo (iets) sneller
// door de database verwerkt worden!
$query = "SELECT naam, telefoon FROM telnrs;";
```

6.8.5 Witruimte in SQL Statement

U kunt zoveel witruimte in uw SQL statement gebruiken als u nodig vindt om het statement zo duidelijk mogelijk **leesbaar** te maken:

```
$query = "SELECT
        naam,
        telefoon
FROM
    telnrs;";
```

De witruimte wordt volledig genegeerd door de database. Hier blijkt maar weer hoe handig het is dat u in PHP strings over meerdere regels kunt 'uitsmeren'!

Bijkomend voordeel is dat u op deze manier ook **duidelijkere foutmeldingen** kunt krijgen wanneer er onverhoopt iets mis is met de SQL syntax. De database kan dan melden dat er een fout is op regel 4 of 5 in plaats van altijd regel 1... Zie verder **6.9.1. mysqli_error()**.

Veel ontwikkelaars hebben overigens duidelijk uitgesproken meningen over de wijze waarop ze SQL statements willen noteren. Een ander voorbeeld:

```
$query = "SELECT naam,
        telefoon
FROM telnrs;";
```

Of:

```
$query = "SELECT naam, telefoon
FROM telnrs;";
```

Of zo:

```
$query = "SELECT naam
        , telefoon
        , email
FROM telnrs;";
```

Bovenstaande notatie voorkomt dat u problemen krijgt met een komma aan het einde van de fieldlist. Waar iedereen het wel over eens is, is dat het handig is om een regel zo af te breken dat de nieuwe regel met een SQL keyword begint...

Tip: U kunt uw SQL statements netjes formatteren met een van deze gratis online tools: **SQLinFORM** (<http://www.sqlinform.com>) en **Instant SQL Formatter**

(<http://www.dpriver.com/pp/sqlformat.htm>). De laatste kan er ook eenvoudig een PHP statement van maken. Iets wat phpMyAdmin overigens ook kan!

6.8.6 Volgorde

Merk op dat deze derde stap op zich ook de eerste stap had kunnen zijn. Het enige wat u in deze stap gedaan heeft is namelijk het SQL SELECT statement opstellen dat u wilt gaan uitvoeren en het in een variabele opslaan. Het commando is op dit moment dus nog niet meer dan een stuk tekst in een variabele.

U had deze stap ook als eerste kunnen maken en pas daarna de verbinding kunnen openen en de database selecteren... De volgorde maakt niet zo heel veel uit. We kiezen nu voor deze volgorde vanwege didactische redenen, maar voor de hierboven geschetste volgorde valt ook veel te zeggen: eerst alles voorbereiden en dan pas naar de database gaan...

De vierde stap is echter in beide gevallen hetzelfde: u wilt de door u opgestelde query ook daadwerkelijk gaan uitvoeren!

6.9 Stap 4: Query uitvoeren

We gebruiken de functie `mysqli_query()` om queries naar de database te sturen. Als argumenten geeft u na de verplichte link identifier `$conn` voor de openstaande verbinding het in de variabele `$query` opgeslagen SQL statement mee:

```
$result = mysqli_query($conn, $query);
```

Wanneer de opgegeven query succesvol is uitgevoerd dan levert dat een `mysqli_result` object op, in feite een **pointer** naar de geretourneerde **recordset** (ook wel **resultset** genoemd). Deze waarde slaat u op in een variabele die u bijvoorbeeld `$result` noemt, maar u had natuurlijk ook een wat specifiekere naam als `$rs_telnrs` kunnen gebruiken... In de volgende stap leert u hoe u met het result verder dient te werken.

Merk op dat het statement zonder gebruik van de variabele `$query` er ook zo uit had kunnen zien:

```
$result = mysqli_query($conn, "SELECT naam, telefoon FROM telnrs;");
```

U geeft het SQL statement dan gelijk als string op. Maar deze werkwijze is, zoals eerder gezegd, meestal niet echt handig. SQL SELECT statements kunnen behoorlijk lang worden en dan leest bovenstaande notatie niet echt makkelijk meer. Maar wat vervelender is: u kunt een statement dat is opgebouwd met behulp van variabelen niet even snel naar het scherm schrijven om te zien of het wel correct is opgebouwd! Met een variabele kan dat wel heel makkelijk.

6.9.1 `mysqli_error()`

Wanneer de query mislukt dan retourneert de functie `mysqli_query()` de waarde `false`. U kunt hier op de inmiddels bekende manier gebruik van maken:

```
$result = mysqli_query($conn, $query)
or die ('query mislukt...');
```

Maar dit is natuurlijk niet zo heel erg informatief. We willen wel graag weten wat er niet goed was aan de query. Tijdens de ontwikkelfase is het heel handig om bij eventuele fouten de door MySQL Server gegenereerde foutmelding te tonen omdat u dan een beter idee krijgt waarom de query niet gelukt is. U kunt dat doen met behulp van de `mysqli_error()` functie die de laatste foutmelding van de MySQL Server geeft. Als verplicht argument moet u de link identifier meegeven:

Database gebruiken

```
$result = mysqli_query($conn, $query)
        or die (mysqli_error($conn));
```

Stel dat u nu een typefout maakt of u vergist in de naam van een kolom dat wordt dat in één klap duidelijk:

Unknown column 'telefon' in 'field list'

U kunt uiteindelijk ook een eigen functie schrijven die in de productiefase alleen net opgemaakte foutmeldingen laat zien, eventueel met aanvullende informatie. Maar dat is iets wat we later zullen doen...

Overigens kunt u `mysqli_error($conn)` ook gebruiken om fouten die optraden bij het **selecteren** van de **database** te tonen. Voor fouten bij het **maken** van de **verbinding** moet u echter `mysqli_connect_error()` gebruiken.

Let op: Regelnummering in de foutmelding slaat op de query!

Stel u heeft een PHP pagina waarvan het begin er zo uitziet:

```
1 <?php
2 ini_set('display_errors', 'off');
3
4 // stap 1: verbinding met MySQL server maken
5 $conn = mysqli_connect('localhost', 'root', '')
6         or die('kan geen verbinding met de server maken...');
7
8 // stap 2: database selecteren
9 $db = mysqli_select_db($conn, 'cursus')
10        or die('database niet beschikbaar...');
11
12 // stap 3: query opstellen
13 $query = "SELECT
14             naam,
15             telefoon,
16         FROM
17             telnrs";
18
19 // stap 4: query uitvoeren
20 $result = mysqli_query($conn, $query)
21         or die(mysqli_error($conn));
22 ?>
```

Dan zou u een foutmelding van MySQL kunnen krijgen die er bijvoorbeeld zo uit ziet:

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'FROM telnrs' at line 4

U bent dan geneigd om in de broncode op regel 4 te gaan kijken, maar daar staat alleen een commentaar regel! Bedoeld wordt echter regel 4 van het SQL SELECT statement!

Op regel 4 staat het Keyword FROM en daar is natuurlijk niks mis mee. Op regel 3 is na telefoon echter nog een **komma** blijven staan:

```
$query = "SELECT
            naam,
            telefoon,
        FROM
            telnrs";
```

(waarschijnlijk omdat op een gegeven moment de veldnaam erachter is weggehaald). Nu verwacht

MySQL op deze plek een veldnaam en niet het keyword FROM. Vandaar de foutmelding. Dit is een veel voorkomende fout!

6.9.2 Query tonen bij error

`mysqli_error()` geeft alleen de foutmelding. Het laat niet de query zien die u aan de database gaf om uit te voeren. U kunt de query ook zichtbaar maken op de volgende manier:

```
$result = mysqli_query($conn, $query)
        or die(mysqli_error($conn) . '<hr>' . $query);
```

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'FROM telns' at line 4
SELECT naam, telefoon, FROM telns
```

U kunt dit ook combineren met het HTML `pre` element om de regels in het SQL statement te kunnen zien. Maar dan zult u voor de leesbaarheid ook de notatie in uw code moeten aanpassen. Bijvoorbeeld zo:

```
$query = "
SELECT
    naam,
    telefoon,
FROM telns;
";
```

Dat geeft in combinatie met

```
$result = mysqli_query($conn, $query)
        or die(mysqli_error($conn) . '<hr><pre>' . $query . '</pre>');
```

het volgende resultaat:

```
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'FROM telns' at line 4

SELECT
    naam,
    telefoon,
FROM
    telns
```

Dit is vooral handig wanneer de query variabelen bevat zoals we ze later in **dynamische SQL statements** zullen gebruiken.

6.9.3 `mysqli_errno()`

Er is ook een functie `mysqli_errno()` die als alles goed gaat de waarde 0 retourneert en anders een foutcode in de vorm van een getal (geen melding). Sommige programmeurs gebruiken dan ook de controle `if (mysqli_errno($conn) > 0)` om te controleren of er iets mis is gegaan bij het uitvoeren van de query:

```
$result = mysqli_query($conn, $query)
        or die('error: ' . mysqli_errno($conn));
```

De foutmelding die u nu ziet kan er bijvoorbeeld zo uitzien:

```
error: 1054
```

U kunt de betekenis van de code opzoeken op het web, bijvoorbeeld via:

<http://dev.mysql.com/doc/refman/5.0/en/error-messages-server.html>

De melding die u daar terugvindt is echter dezelfde die `mysqli_error()` zou geven, namelijk dat het om een onbekende kolomnaam gaat, maar het kan toch net handig zijn bij het zoeken naar meer informatie op het web...

6.9.4 Eén query

Met `mysqli_query()` kunt u, om veiligheidsredenen, maar 1 query tegelijkertijd uitvoeren. Als u meerdere SQL statements gescheiden met een `;` in `$query` hebt staan dan krijgt u een foutmelding en wordt geen van de statements uitgevoerd.

Als u perse meerdere queries achter elkaar wilt kunnen uitvoeren (denk aan een installer script dat de database aanmaakt, tabellen creëert en records toevoegt) dan moet u `mysqli_multi_query()` gebruiken. Deze retourneert `true` als alle queries konden worden uitgevoerd. Als u meerdere SELECT queries heeft uitgevoerd dan moet u gebruik maken van functies als `mysqli_store_result()`, `mysqli_more_results()` en `mysqli_next_result()` om het resultaat te kunnen tonen. Wij zullen dat tijdens deze cursus niet doen.

Zie verder: <https://www.php.net/manual/en/mysqli.multi-query.php>

6.10 Stap 5: Gegevens tonen

Als u de pagina op dit moment zou opvragen dan is er, als alles goed is gegaan, nog helemaal niets te zien! We hebben weliswaar gegevens in de database geselecteerd, maar nu moeten we de gegevens nog ophalen en tonen in onze PHP pagina. Dat doen we in deze vijfde stap.

Wanneer uw SELECT statement foutloos wordt uitgevoerd resulteert dit zoals gezegd in een **resultset**. U mag dit zien als een tijdelijke tabel die in MySQL wordt aangemaakt. Wanneer u een query als

```
$query = "SELECT * FROM telnrs;";
```

zou hebben opgegeven dan heeft u zelfs een exacte kopie van de tabel omdat u geen enkele beperking oplegt aan het aantal geretourneerde kolommen of rijen. Vaak zal een resultset echter een subset zijn van de gegevens in de database.

Stel dat dit de inhoud van de **tabel** `telnrs` is:

id	naam	afdeling	telefoon
1	Karin	Sales	123
2	Peter	Sales	124
3	Anneke	Receptie	100
4	Dirk	Sales	126
5	Marie	Directie	101
6	Michelle	Sales	125

en u voert deze **query** uit

```
// query opstellen
$query = "
SELECT naam, telefoon FROM telnrs
WHERE afdeling = 'Sales'
ORDER BY naam ASC;";

// query uitvoeren
$result = mysqli_query($conn, $query) or die(mysqli_error($conn));
```


dan ziet de **resultset** in de **database** er zo uit:

Dirk	126
Karin	123
Michelle	125
Peter	124

Let op: dit staat op dit moment dus nog **niet** op uw scherm! De resultset bestaat alleen in (het geheugen van) de database. De WHERE en ORDER BY clauses worden later in dit hoofdstuk uitgebreid besproken. Ze dienen hier enkel ter demonstratie van hoe een resultset tot stand komt...

Wat we nu graag willen is de inhoud van deze resultset op het scherm tonen. Helaas is het **niet** zo dat u simpelweg `$result` kunt echoën:

```
// dit werkt helaas NIET:  
echo $result;
```

Dit levert iets op als:

Catchable fatal error: Object of class mysqli_result could not be converted to string in **test.php** on line 35

Duidelijk niet wat we willen... Een `var_dump($result)` laat zien waar we mee te maken hebben:

```
object(mysqli_result)#2 (5) {  
    ["current_field"]=>  
        int(0)  
    ["field_count"]=>  
        int(2)  
    ["lengths"]=>  
        NULL  
    ["num_rows"]=>  
        int(4)  
    ["type"]=>  
        int(0)  
}
```

U heeft in principe dus al wel wat informatie over de resultset (het aantal rijen en het aantal kolommen) maar nog geen data.

We kunnen echter wel met behulp van `$result` door de resultset heen navigeren. U doet dat met behulp van een `while` lus en een van de `mysqli_fetch_*` functies. Deze functies verwachten `$result` als argument en geven dan steeds het record als een array terug, en zetten de pointer naar het volgende record net zolang tot het einde van de resultset is bereikt. Dan retourneren ze `false` en stopt de lus.

6.10.1 `mysqli_fetch_*` functies

U kunt gebruik maken van een drietal fetch functies te weten:

1. `mysqli_fetch_row()`
levert elk record als een **numeriek array** aan
2. `mysqli_fetch_assoc()`
levert elk records als een **associatief array** aan
3. `mysqli_fetch_array()`
levert elk records zowel als een **numeriek array** en een **associatief array** aan. U kunt de functie met de constanten `MYSQLI_NUM` en `MYSQLI_ASSOC` laten functioneren als

respectievelijk `mysqli_fetch_row()` en `mysqli_fetch_assoc()`. De standaardwaarde is `MYSQLI_BOTH`.

De arrays zijn **eendimensionaal**. Er wordt per index precies één waarde opgeslagen. De array bevat één record uit de resultset en bij elke index hoort één veldwaarde.

Het **verschil** tussen de functies zit dus in de **notatie van de index** (de veldnaam). Deze is **numeriek** (bij `mysqli_fetch_row()`), **associatief** (bij `mysqli_fetch_assoc()`) of **beide** (bij `mysqli_fetch_array()`).

NB.: Een array kan ook **meerdimensionaal** zijn en een hele resultset (dus meerdere records) bevatten. Deze situatie wordt later in dit hoofdstuk behandeld.

6.10.2 `mysqli_fetch_row()`

We beginnen met de snelste functie. De functie `mysqli_fetch_row()` retourneert de rijen uit de recordset steeds per rij als een numerieke array met de veldwaarden:

0	1	2
Dirk	Sales	126

U kunt dat met `print_r($row)` goed zichtbaar maken:

```
Array
(
    [0] => Dirk
    [1] => Sales
    [2] => 126
)
```

Let op: de waarde van de index wordt bepaald door de volgorde van de velden in het SQL SELECT statement. Als u de volgorde aanpast kan dat gevolgen hebben voor het uitlezen!

6.10.3 `mysqli_fetch_assoc()`

De `mysqli_fetch_assoc()` functie retourneert de records uit de resultset steeds per rij als een **associatieve** array:

naam	afdeling	telefoon
Dirk	Sales	126

U kunt dat met `print_r($row)` ook goed zichtbaar maken:

```
Array
(
    [naam] => Dirk
    [afdeling] => Sales
    [telefoon] => 126
)
```

Het is natuurlijk heel handig dat `mysqli_fetch_assoc()` de veldnamen als index meelevert en dat het aanduiden van de veldwaarden **niet afhankelijk** is van de **volgorde** die de velden in het SQL SELECT statement hebben... We zullen in principe dan ook deze functie werken tijdens de rest van de cursus!

6.10.4 mysqli_fetch_array()

De functie `mysqli_fetch_array()` combineert in feite de mogelijkheden van `mysqli_fetch_row()` en `mysqli_fetch_assoc()`. Ook deze functie retourneert een rij uit de recordset als array. Het verschil is echter dat u nu met behulp van een optionele constante kunt aangeven wat voor een type array dit moet zijn.

De algemene notatie is:

`mysqli_fetch_array(resultsetpointer, [constante])`

De optionele constante kan de volgende waarden hebben:

constante	beschrijving
<code>MYSQLI_ASSOC</code>	Hiermee worden de kolommen in de rij geretourneerd als een associatieve array met de veldnaam als index.
<code>MYSQLI_NUM</code>	Hiermee worden de kolommen in de rij geretourneerd als een array met een numerieke index. Deze index start op 0.
<code>MYSQLI_BOTH</code>	Combinatie van bovenstaande varianten. De geretourneerde array is geïndexeerd op nummer en veldnaam (default).

Geeft u geen constante mee dan wordt de geretourneerde array automatisch voorzien van zowel een index op nummer als op veldnaam:

0	naam	1	afdeling	2	telefoon
Dirk	Dirk	Sales	Sales	126	126

oftewel

```
Array
(
    [0] => Dirk
    [naam] => Dirk
    [1] => Sales
    [afdeling] => Sales
    [2] => 126
    [telefoon] => 126
)
```

Met deze functie kunt u de werking van de andere functies simuleren:

```
mysqli_fetch_array($result, MYSQLI_NUM);
// werkt exact hetzelfde als
mysqli_fetch_row($result);
```

En

```
mysqli_fetch_array($result, MYSQLI_ASSOC);
// werkt hetzelfde als
mysqli_fetch_assoc($result);
```

6.10.5 Tonen met while lus

Met een `while` lus kunt u vervolgens door alle records in de recordset heen 'lopen' en de waarden opslaan in een array die u bijvoorbeeld `$row` noemt. In de lus kunt u dan werken met de waarden van deze array. De loop wordt beëindigd wanneer er geen rijen meer in de recordset zijn. De array `$row` zal namelijk de waarde `null` krijgen wanneer de fetch functies klaar zijn met het 'fetchen' van

Database gebruiken

de rijen. In de meeste gevallen zal `mysqli_fetch_assoc()` de handigste functie zijn omdat deze ook de veldnamen bevat:

```
while ($row = mysqli_fetch_assoc($result)) {  
    echo $row['naam'];  
}
```

Bovenstaande code geeft de volgende uitvoer:

```
DirkKarinMichellePeter
```

Let op: Denk er aan dat u nu data aan het wegschrijven bent. De data moet dan natuurlijk wel ergens binnen de body van uw webpagina terechtkomen!

Om de namen onder elkaar te zetten zult u wat HTML code in de lus moeten gebruiken:

```
while ($row = mysqli_fetch_assoc($result)) {  
    echo $row['naam'] . "<br>\n";  
}
```

Dit geeft het gewenste resultaat:

```
Dirk  
Karin  
Michelle  
Peter
```

De `\n` zorgt ervoor dat de namen ook in de HTML broncode onder elkaar komen te staan.

Als u meerdere veldwaarden wilt uitlezen dan plaatst u deze ook in de lus:

```
while ($row = mysqli_fetch_assoc($result)) {  
    echo $row['naam'] . " (" . $row['telefoon'] . ")<br>\n";  
}
```

Dit geeft:

```
Dirk (126)  
Karin (123)  
Michelle (125)  
Peter (124)
```

Door een beetje met de HTML te spelen, kunt u elk gewenst resultaat verkrijgen, bijvoorbeeld een ordered list:

```
echo "<ol>\n";  
while ($row = mysqli_fetch_assoc($result)) {  
    echo "<li>" . $row['naam'] . " (" . $row['telefoon'] . ")</li>\n";  
}  
echo "</ol>\n";
```

Dit geeft:

- Dirk (126)
- Karin (123)

- Michelle (125)
- Peter (124)

In bovenstaand voorbeeld wordt bewust gebruik gemaakt van **concatenatie**. Het **rechtstreeks** opnemen van de array notatie in een double-quoted string op deze manier, gaat namelijk **niet werken**:

```
// LET OP: dit geeft een parse error!  
echo " <li>$row['naam'] ($row['telefoon'])</li>\n";
```

U kunt het eventueel oplossen door de array notatie tussen accolades te zetten:

```
// LET OP: dit werkt wel maar oogt niet zo lekker...  
echo " <li>{$row['naam']} ({$row['telefoon']})</li>\n";
```

U kunt er ook voor kiezen om met **variabelen** te werken:

```
echo "<ol>\n";  
while ($row = mysqli_fetch_assoc($result)) {  
  
    // zet de veldwaarden in variabelen  
    $naam = $row['naam'];  
    $telefoon = $row['telefoon'];  
  
    // schrijf de variabelen weg met HTML opmaak  
    echo " <li>$naam ($telefoon)</li>\n";  
}  
echo "</ol>\n";
```

Uiteraard kunt u de data zo in tal van indelingen tonen, bijvoorbeeld in een HTML tabel. Sterker nog, dat gaat u in de eerstvolgende oefening ook doen!

Let op: De te tonen velden moeten wel geselecteerd zijn! Alle velden die u in de lus opgeeft moeten in de *fieldlist* van uw SQL SELECT statement staan! Zou u bijvoorbeeld het afdeling veld uit proberen te lezen terwijl het niet in uw query is opgenomen dan krijgt u meteen bij *elke* lusdoorgang een melding als deze (ervan uitgaande dat error reporting maximaal is en fouten worden getoond):

Notice: Undefined index: afdeling in **test.php** on line 39

Iets anders waar u op moet letten is het volgende: wanneer er een fout in de query staat dan is `$result` na het uitvoeren van de query `false`. We tonen dan de foutmelding (in een error template) en stoppen het script. Niets aan de hand. Maar als u de fout alleen zou printen en het script zou laten doorlopen dan wordt `false` aan de fetch functie gegeven die daar niets mee kan:

Warning: mysqli_fetch_assoc() expects parameter 1 to be mysqli_result, boolean given in **test.php**

6.10.6 In de lus naar HTML mode

U kunt er ook voor kiezen om gelijk na het begin van de lus naar HTML mode te gaan. Hier noteert u dan uw HTML code op de gebruikelijke manier om daarna met PHP codeblokken de veldwaarden op de gewenste plaats weg te schrijven. Het is dan ook niet nodig om met "tussen variabelen" te werken:

```
<ul>  
<?php  
// begin lus in PHP mode  
while ($row = mysqli_fetch_assoc($result)) {
```

Database gebruiken

```
// ga naar HTML mode en echo variabelen:
?>
<li><?php echo $row['naam']; ?> (<?php echo $row['telefoon']; ?>)</li>
<?php
} // sluit lus in PHP mode
?>
</ul>
```

Of wat compacter genoteerd met de template syntax:

```
<ul>
<?php while ($row = mysqli_fetch_assoc($result)): ?>
    <li><?=$row['naam']?> (<?=$row['telefoon']?>)</li>
<?php endwhile; ?>
</ul>
```

Let op dat er **geen extra spaties achter** de `<?php ... ?>` blokjes komen te staan. De enters moeten er gelijk achter komen want alleen dan worden deze door PHP genegeerd en komen ze niet in de uitvoer terecht:

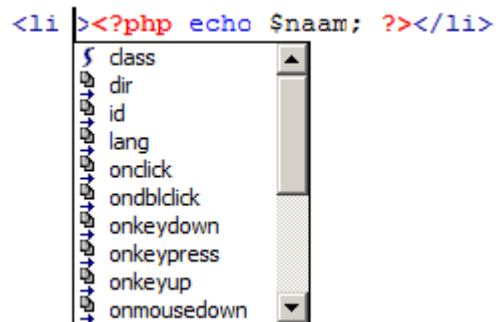
Code	Output
<pre> <?php while (\$row = mysqli_fetch_assoc(\$result)): ?> <?=\$row['naam']?> (<?=\$row['telefoon']?>) <?php endwhile; ?> </pre>	<pre> Dirk (126) Karin (123) Michelle (125) Peter (124) </pre>
<pre> <?php while (\$row = mysqli_fetch_assoc(\$result)): ?> <?=\$row['naam']?> (<?=\$row['telefoon']?>) <?php endwhile; ?> </pre>	<pre> Dirk (126) Karin (123) Michelle (125) Peter (124) </pre>

Het wisselen van PHP en HTML mode is, in tegenstelling tot bijvoorbeeld bij Classic ASP, even snel als met PHP de HTML code wegschrijven.

Het voordeel van bovenstaande werkwijze zit hem vooral in combinatie met uw (WYSIWYG) editor. In Dreamweaver bijvoorbeeld heeft u in een string met HTML code geen ondersteuning:

```
<?php
echo '<li |>'. $naam. '</li>';
```

Maar in HTML modus wel:



Bovendien heeft u in Design Mode een beter zicht op wat er gebeurt. Neem het voorbeeld van een tabel met gegevens, in plaats van:



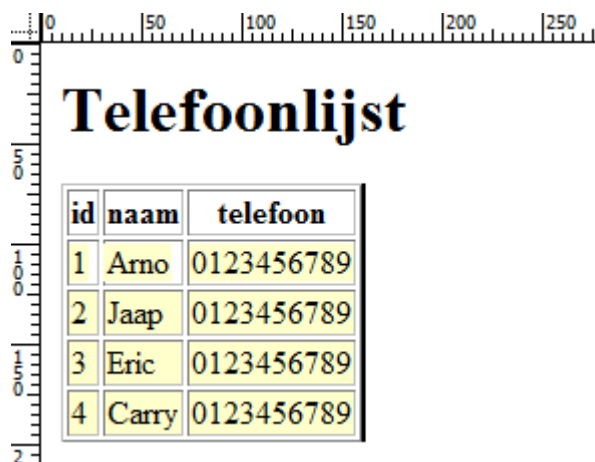
Ziet u dan bijvoorbeeld dit:



In combinatie met Live Data View:



geeft dit een heel goed idee over hoe de pagina er in de browser uit komt te zien:



Telefoonlijst

id	naam	telefoon
1	Arno	0123456789
2	Jaap	0123456789
3	Eric	0123456789
4	Carry	0123456789

De ene manier is niet beter dan de andere: u kunt zelf kiezen welke werkwijze het best bij u past!

Later in dit hoofdstuk zullen we echter een **beter alternatief** presenteren waardoor u de `mysqli_fetch_assoc()` functie helemaal **uit de HTML** template kunt laten!

6.11 Stap 6: Geheugen vrijgeven (Optioneel)

Na het uitlezen van de resultset kunt u het geheugen dat MySQL gebruikt voor de resultset vrijgeven met de functie `mysqli_free_result()`:

```
mysqli_free_result($result); // optioneel
```

Dit wordt echter bijna nooit gedaan omdat dit na het beëindigen van het script automatisch door PHP wordt gedaan. Het is eigenlijk alleen interessant om te gebruiken als u meerdere, grote, queries achter elkaar wilt uitvoeren. Dat gaan we tijdens deze cursus zeker niet doen, vandaar dat we het vanaf nu gewoon weglaten.

6.12 Stap 7: Verbinding verbreken (Optioneel)

Hetzelfde geldt voor het expliciet sluiten van de verbinding. Dit wordt bij het einde van het script eveneens automatisch geregeld. Wanneer de verbinding met een MySQL server niet langer nodig is dan kunt u die verbinding expliciet sluiten met de functie `mysqli_close()`:

```
mysqli_close($conn); // optioneel
```

In de praktijk zult u `mysqli_close()` echter maar zelden tegenkomen; het wordt door de meeste programmeurs, lui als ze zijn, gewoon weggelaten.

De opdracht is, net als `mysqli_free_result()`, met name van belang als u met grote resultsets werkt en na de databasehandelingen nog andere intensieve operaties gaat uitvoeren. Wij zullen ze in deze cursus verder niet gebruiken.

Let op: Wanneer u de verbinding gesloten heeft, kunt u uiteraard geen `mysqli_*` functies meer uitvoeren die een verbinding nodig hebben. Zou het toch doen dan krijgt u een vrij onduidelijke melding als deze:

Warning: mysqli_query(): Couldn't fetch mysqli in **demo.php**

Waarbij het wellicht niet meteen duidelijk is dat de oorzaak in de reeds gesloten verbinding gezocht moet worden...

6.13 Het hele script

Als u alle hierboven gemaakte stappen gemaakt heeft dan zou uw script er in de **compacte vorm** als volgt uit kunnen zien:

```
<?php
// Stap 1: Verbinding met MySQL Server maken
$conn = mysqli_connect('localhost', 'root', '')
    or die('kan geen verbinding met database maken...');

// Stap 1a: set charset specificeren
mysqli_set_charset($conn, 'utf8')
or die('charset kan niet correct worden ingesteld...');

// Stap 2: Database selecteren
mysqli_select_db($conn, 'cursus')
or die('database niet beschikbaar...');

// Stap 3: Query opstellen
$query = "SELECT naam, telefoon FROM telnrs;";

// Stap 4: Query uitvoeren
$result = mysqli_query($conn, $query) or die (mysqli_error($conn));
?>
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>Telefoonlijst</title>
</head>

<body>

    <!-- Stap 5: Gegevens uit resultset tonen -->
    <?php while ($row = mysqli_fetch_assoc($result)): ?>
        <?=$row['naam']?> (<?=$row['telefoon']?>) <br>
    <?php endwhile; ?>

<?php
// Stap 6: Geef resultset vrij (Optioneel)
mysqli_free_result($result);

// Stap 7: Verbinding sluiten (Optioneel)
mysqli_close($conn);
?>

</body>
</html>
```

Dit is de manier die we voorlopig zullen volgen. In een later stadium zult u deze basis opzet steeds verder verbeteren!

Zie voor een uitwerking met de inmiddels *deprecated* **MySQL API** de **Bijlagen**.

Oefening 6.1

Maak een nieuwe pagina `medewerkers_tonen.php`. Hierin maakt u verbinding met de MySQL database `cursus`. Vervolgens stelt u een SQL `SELECT` statement op dat alle gegevens uit de tabel `medewerkers` selecteert. Verstuur het als string opgeslagen statement vervolgens naar de database en sla het resultaat op in een variabele `$result`. Zorg er vervolgens voor dat de records worden getoond in een eenvoudige HTML tabel. Zet in de eerste rij de veldnamen in tableheaders:



ID	Voornaam	Achternaam	Afdeling	Toestelnummer
1	Sheila	Lavanter	Verkoop	1000
2	Peter	Spanje	Verkoop	1000
3	Jaap	Soetendaal	Directie	9999
4	Kees	Mol	Opleidingen	1245
5	Kees	Prins	Opleidingen	1234

Verdieping

Zet het aantal medewerkers boven de tabel. Hint: gebruik hiervoor de functie `mysqli_num_rows()`. Als alternatief kunt u ook `$result->num_rows` gebruiken.

6.14 VERDIEPING: Generiek tonen

In bovenstaande oefening worden de tableheaders en de veldwaarden hard in de pagina gecodeerd. Wanneer u de fieldlist in het `SELECT` statement aanpast zodat er meer of minder kolommen getoond moeten worden dan moet u uw code op verschillende plekken aanpassen. Het zou mooi zijn als u enkel uw query hoefde aan te passen.

U kunt allereerst beginnen om in plaats van het apart benaderen van elke `$row[]` een `foreach` loop te gebruiken:

```
<table>
  <tr>
    <th>Naam</th>
    <th>Telefoon</th>
  </tr>
  <?php while ($row = mysqli_fetch_assoc($result)): ?>
    <tr>
      <?php
        foreach ($row as $value): ?>
          <td><?php echo $value; ?></td>
        <?php
          endforeach; ?>
      </tr>
    <?php endwhile; ?>
  </table>
```

De manier van inspringen lijkt misschien wat onhandig maar is een aardig compromis voor de leesbaarheid van zowel de programmacode als de gegenereerde HTML. Als u de PHP blokken ook zou laten inspringen genereert u extra witruimte in de HTML. Door op deze manier zowel de HTML als de PHP logica in te laten springen, houden ze elk hun eigen niveau. HTML springt in met 2 spaties en PHP met 4 spaties. Zie ook **3.17**.

Om ook de **headers** dynamisch te genereren kunt u gebruik maken van de `array_keys()` functie. Deze geeft de keys van een array terug als een (numerieke) array. Als u in de lus

`print_r(array_keys($row))` zou opnemen dan krijgt u (voor elke lus doorgang) het volgende resultaat:

```
Array
(
    [0] => naam
    [1] => telefoon
)
```

Hier kunt u vervolgens weer met een `foreach` door heen lopen om ze in de tableheaders te zetten. Het probleem is dat dit **binnen** de `while` lus moet (anders is er geen `$row` om te gebruiken) maar dat het **maar 1x** mag gebeuren, u wilt natuurlijk geen tableheaders boven elke rij! De oplossing bestaat uit het gebruik van een **flag**:

```
<!-- flag voor headers -->
<?php $headers = true; ?>

<table>
<!-- schrijf datarijen -->
<?php while ($row = mysqli_fetch_assoc($result)): ?>

<?php    if ($headers): ?>
    <!-- met eenmalig de headers -->
    <tr>
    <?php        foreach (array_keys($row) as $name): ?>
        <th><?php echo $name; ?></th>
    <?php        endforeach; ?>
    </tr>
    <!-- vanaf nu niet meer -->
<?php $headers = false; ?>
<?php    endif; ?>

    <tr>
    <?php        foreach ($row as $value): ?>
        <td><?php echo $value; ?></td>
    <?php        endforeach; ?>
    </tr>
<?php endwhile; ?>
</table>
```

Helaas leest dit in template code niet zo heel prettig, dit is misschien ook meer iets voor een **template hulpfunctie**... U leert in de **PHP vervolg** cursus hoe u zelf functies kunt schrijven.

Maar we hebben in ieder geval wel het gewenste resultaat. De query is nu bepalend voor de opbouw van de tabel. We nemen hier de tabel medewerkers omdat deze wat meer kolommen vat om mee te spelen:

```
SELECT
    voornaam,
    achternaam
FROM medewerkers
```

voornaam	achternaam
Sheila	Lavanter
Peter	Spanje
Jaap	Soetendaal
Kees	Mol
Kees	Prins

```
SELECT
    id,
    achternaam,
    afdeling
FROM medewerkers
```

id	achternaam	afdeling
1	Lavanter	Verkoop
2	Spanje	Verkoop
3	Soetendaal	Directie
4	Mol	Opleidingen
5	Prins	Opleidingen

```
SELECT
    achternaam,
    toestelnummer AS tel
FROM medewerkers
```

achternaam	tel
Lavanter	1000
Spanje	1000
Soetendaal	9999
Mol	1245
Prins	1234

Database gebruiken

Wanneer u een veldwaarde anders wilt benaderen dan de andere in de `foreach` lus dan zult u iets moeten doen als:

```
...
<?php foreach ($row as $value): ?>
<?php     if ($value == $row['id']): ?>
    <!-- doe iets speciaals met id -->
<?php     else: ?>
    <!-- standaard actie -->
<?php     endif; ?>
<?php endforeach; ?>
...
```

Als u dergelijke code ook nog opneemt in de template wordt het echt feest! Maar niet heus... We zullen tijdens de cursus dan ook met de "harde notatie" blijven werken. Deze is beter leesbaar en beter aanpasbaar. Maar probeer bovenstaande code vooral eens uit!

Let op: In een later stadium zullen we onze code veel handiger gaan organiseren. Zo zullen we bijvoorbeeld **losse templates insluiten** en daarin de data tonen. Bovendien zullen we de **data** dan aanleveren **in** de vorm van een **2-dimensionale array** voor maximale portabiliteit. Voorlopig zullen we echter alles in 1 bestand houden en ons concentreren op het werken met SQL...

6.15 Selectie ordenen: ORDER BY

De volgorde van de records in de recordset is tot nu toe ongedefinieerd. De recordset is een verzameling records en een verzameling heeft in principe geen volgorde.

De op de voorgaande wijze geselecteerde records zullen dan ook in de volgorde getoond worden waarmee ze in de database werden ingevoerd. Ze worden dus in feite op **oplopend ID** gerangschikt.

Als u de gegevens met een andere sortering wilt weergeven dan moet u dat vooral niet met PHP logica proberen op te lossen. U kunt het namelijk heel eenvoudig met SQL op database niveau regelen:

Met het `ORDER BY` statement kunt u een ander veld (of velden) opgeven waarop gesorteerd moet worden.

De algemene notatie is:

```
SELECT veldna(a)m(en)
FROM tabelnaam
ORDER BY veldna(a)m(en) [ASC|DESC];
```

De sortering kan oplopend (*ascending*) of aflopend (*descending*) zijn. Voor een oplopende sortering gebruikt u het keyword `ASC` en voor een aflopende sortering `DESC`. Als u het weglaat dan wordt default oplopend geordend:

```
SELECT * FROM medewerkers
ORDER BY achternaam ASC;
```

is dus hetzelfde als:

```
SELECT * FROM medewerkers
ORDER BY achternaam;
```

6.15.1 Meerdere velden

Wanneer twee gebruikers dezelfde achternaam hebben dan worden ze alsnog in willekeurige volgorde getoond. U zou er bij een gelijke achternaam voor kunnen kiezen om deze medewerkers op voornaam te ordenen. Dat doet u door `voornaam` na `achternaam` te plaatsen in de `ORDER BY` clausule:

```
SELECT * FROM medewerkers
ORDER BY achternaam, voornaam ASC;
```

6.15.2 Positie in plaats van veldnaam

In plaats van veldnamen mag u ook met getallen de positie van de velden in de fieldlist aangeven waarop gesorteerd moet worden.

```
SELECT voornaam, achternaam, afdeling
FROM medewerkers
ORDER BY 3, 2, 1;
```

Is dus hetzelfde als:

```
SELECT voornaam, achternaam, afdeling
FROM medewerkers
ORDER BY afdeling, achternaam, voornaam;
```

Oefening 6.2 *Pas het SQL `SELECT` statement in de pagina `medewerkers_tonen.php` dusdanig aan dat de medewerkers met oplopende voornaam worden getoond.*

In de cursus **PHP Vervolg** leert u hoe u de veldnamen klikbaar kunt maken zodat de eindgebruiker zelf kan bepalen op welke kolom gesorteerd moet worden.

6.15.3 VERDIEPING: Groeperen

Als u in uw `SELECT` statement de medewerkers gesorteerd heeft op afdeling:

```
$query = "SELECT voornaam, achternaam, afdeling
          FROM medewerkers
          ORDER BY afdeling;";
```

Dan is op zich redelijk duidelijk te zien hoe de medewerkers per afdeling verdeeld zijn, maar het kan duidelijker door de records per afdeling te **groeperen**.

Dit doet u dit keer niet met SQL. Er bestaat weliswaar een `GROUP BY` clausule in SQL maar die is bedoeld voor het aggregeren van data in combinatie met speciale SQL functies als `COUNT()` en `SUM()`, bijvoorbeeld voor het berekenen van het aantal medewerkers per afdeling.

In de PHP code kunnen we met wat eenvoudige logica de reeds gesorteerde data groeperen:

```
// initialiseer variabele voor vorige afdeling
$previous_afdeling = '';

while ($row = mysqli_fetch_assoc($result)) {

    // bij wisseling van de afdeling
    if ($row['afdeling'] != $previous_afdeling) {
        // naam van de afdeling als kop wegschrijven
        echo '<h2>'. $row['afdeling'] . '</h2>'.PHP_EOL;
        // en opgeven als nieuwe waarde voor vorige
```

```

        $previous_afdeling = $row['afdeling'];
    }
    // schrijf medewerkers voor de afdeling weg
    echo $row['voornaam'] . ' ' . $row['achternaam'] . '<br />'.PHP_EOL;
}

```

Het resultaat is dan iets als:

Medewerkers Groeperen	
Administratie	
Karin Soetendaal	
Michel Terlouw	
Kees Kwaks	
Development	
Pieter Bergman	
Directie	

Uiteraard kunt u het zo mooi maken als u zelf wilt. U kunt de groepering, met wat meer werk, bijvoorbeeld ook zo tonen:

Medewerkers Groeperen				
Administratie				
id	voornaam	achternaam	afdeling	toestelnummer
50	Karin	Soetendaal	Administratie	4041
8	Michel	Terlouw	Administratie	9999
55	Kees	Kwaks	Administratie	9999
				aantal: 3
Development				
id	voornaam	achternaam	afdeling	toestelnummer
15	Pieter	Bergman	Development	3456
				aantal: 1
Directie				

6.16 Selectie beperken: fieldlist en WHERE

Het SQL Statement uit de vorige oefening gebruikte alle velden en liet alle records zien. U kreeg in feite een exacte kopie van de database tabel terug in uw resultset.

6.16.1 fieldlist

U kunt allereerst de selectie beperken door minder veldnamen in de **fieldlist** te plaatsen.

Oefening 6.3 *Beperk nu de selectie door alleen de voornaam en het toestelnummer van de medewerkers in het SQL statement te gebruiken. Wat is nu het probleem?*

In voorgaande oefening kwam u tot de ontdekking dat de wijze waarop we tot nu toe records in een tabel plaatsen niet echt flexibel is. U kunt uiteraard alleen die velden tonen die u ook in de fieldlist heeft genoemd. Haalt u ze in de fieldlist weg dan moet u alle verwijzingen ernaar in uw code ook weghalen. Afhankelijk van de manier waarop u gecodeerd heeft kan dat inhouden dat u soms wel top op 3, 4 plaatsen uw code moet aanpassen voor elk veld! Dat is niet echt handig. Het zou fijn zijn

als we een HTML tabel, compleet met table headers konden laten genereren, enkel op basis van de query. In de cursus **PHP Vervolg** leert u hoe u dit kunt coderen.

6.16.2 WHERE Clause

Wanneer u het aantal geretourneerde records wenst te beperken dan doet u dat met een WHERE clause. De algemene notatie is:

```
SELECT veldna(a)m(en)
FROM tabelnaam
WHERE conditie(s);
```

Let op: De WHERE clause en de eerder besproken ORDER BY clause zijn niet met elkaar verbonden. U mag ze allebei in dezelfde query gebruiken, maar ook alleen de een of de ander. Als u ze echter samen gebruikt dan is de **volgorde** van **groot belang**: U moet **eerst** de **WHERE** clause noteren en **pas op het einde** de **ORDER BY**. De volgorde wordt dan dus:

```
SELECT veldna(a)m(en)
FROM tabelnaam
WHERE conditie(s)
ORDER BY veldna(a)m(en) [DESC|ASC];
```

Bij de condities achter het woord WHERE geeft u de voorwaarden op waaraan de gegevens moeten voldoen om opgehaald te worden uit de database. Zo kunt u specifieke records benaderen. De query selecteert dan alleen die records waarvoor de vergelijking in de WHERE vergelijking TRUE oplevert. Alle andere records worden uit het resultaat gefilterd.

De vergelijking in de WHERE clause kan vele vormen aannemen. U kunt een conditie combineren met extra condities (AND) of alternatieve condities (OR). Bovendien kunt u er variabelen in opnemen zoals we straks zullen zien.

Een conditie bestaat meestal uit een veldnaam, een operator en een waarde:

```
SELECT veldna(a)m(en)
FROM tabelnaam
WHERE veldnaam operator waarde;
```

Een voorbeeld:

```
SELECT tel
FROM telnrs
WHERE naam = 'Karin';
```

Nu worden alleen die records uit de tabel met telefoonnummers geselecteerd waarvan de waarde van het veld `naam` gelijk is aan `Karin`.

6.16.3 Quotes

Let op: Waarden voor alle **tekstvelden** in de database (CHAR en VARCHAR) moeten tussen **quotes** staan! Dat moet omdat binnen deze waarden woorden en punctuatie voor kan komen die anders als SQL geïnterpreteerd zou worden wat onverwachte gevolgen kan hebben.

Het soort quotes verschilt per database server. In MySQL mag u zowel single als double quotes gebruiken, maar in SQL Server moeten het single quotes zijn. Het is dan ook verstandig om maar altijd **single quotes** te gebruiken. In dat geval kunt u het best het SQL statement in een **double quoted** PHP String opnemen:

```
$query = "SELECT tel
        FROM telnrs
        WHERE naam = 'Karin';";
```

6.16.4 Case insensitivity

Het is overigens zo dat MySQL, net als een aantal andere bekende databases, standaard **hoofdletterongevoelig** is voor de strings waarop een match wordt gezocht. Medewerkers die *karin* (met een kleine k) heten worden dus ook gewoon geselecteerd!

Dit komt in MySQL doordat de standaard **collatie** (de regels die bepalen hoe karakters in een karakterset geordend en vergeleken moeten worden) in MySQL *latin1_swedish_ci* is. *Latin1* is de karakterset waarin de meest voorkomende karakters uit de West-Europese landen zitten, *swedish* omdat de firma MySQL AB oorspronkelijk in Zweden gevestigd was (MySQL is nu van Sun) en de *ci* op het eind staat voor *case insensitive*. U kunt de collatie per veld veranderen, bijvoorbeeld naar *latin1_general_cs* of *utf8_bin* die wel hoofdlettergevoelig zijn, *cs* staat hier voor *case sensitive* en *bin* voor *binary*.

Als u de structuur van de database niet wilt wijzigen maar een query wel hoofdlettergevoelig wilt maken dan kunt u de collatie ook in de query meegeven:

```
$query = "SELECT tel
        FROM telnrs
        WHERE naam = 'Karin' COLLATE utf8_bin;";

// of met de BINARY operator
$query = "SELECT tel
        FROM telnrs
        WHERE naam = BINARY 'Karin';";
```

In beide gevallen worden nu alleen maar Karins met een hoofdletter K gevonden. Als u andere condities ook hoofdlettergevoelig wilt maken dan zult u dit per conditie moeten toevoegen!

In de MySQL documentatie vindt u zeer uitgebreide informatie over het werken met collaties en character sets.

6.16.5 Operatoren

U kunt de condities waaraan de selectie moet voldoen in het **WHERE** deel van het **SELECT** statement opbouwen met de volgende operatoren:

operator	omschrijving	Opmerkingen
=	Gelijk aan	
!=	Niet gelijk aan	Let op: MySQL specifiek!
<>	Niet gelijk aan	
>	Groter dan	
<	Kleiner dan	
>=	Groter dan of gelijk aan	Let op: => is niet juist!
<=	Kleiner dan of gelijk aan	Let op: =< is niet juist!

Meerdere condities kunnen gecombineerd worden met de logische operatoren **AND** en **OR**. U kunt ook de ontkenning zoeken met het keyword **NOT**:

operator	omschrijving	Opmerkingen
AND	Beide vergelijkingen zijn waar	U kunt ook && gebruiken
OR	Een van de twee vergelijkingen is waar	U kunt ook gebruiken
NOT	De vergelijking is niet waar	U kunt ook ! gebruiken

Bijvoorbeeld:

```
SELECT tel
FROM telnrs
WHERE naam = 'Karin'
AND afdeling = 'Sales';
```

Nu worden alleen de telefoonnummers geselecteerd van de Karins die op de Sales afdeling werken. U wordt zo dus steeds specifieker.

Had u daarentegen OR gebruikt dan had u alle medewerkers die Karin heten geselecteerd én alle medewerkers die op de afdeling Sales werken:

```
SELECT tel
FROM telnrs
WHERE naam = 'Karin'
OR afdeling = 'Sales';
```

Had u juist alle Karins die niet op de afdeling Sales werken willen selecteren dan had u NOT moeten gebruiken:

```
SELECT tel
FROM telnrs
WHERE naam = 'Karin'
AND NOT afdeling = 'Sales';
```

Of in alternatieve syntax:

```
SELECT tel
FROM telnrs
WHERE naam = 'Karin'
&& afdeling != 'Sales';
```

U kunt zich voorstellen dat u door het combineren van de operatoren, eventueel met haakjes voor de volgorde, oneindig veel mogelijkheden heeft om precies die data te selecteren die u nodig heeft. U kunt in dergelijke **compound conditions** overigens net zoveel condities combineren als u maar nodig heeft, er zit geen maximum aan.

Het gebruik van haakjes is overigens essentieel bij combinaties van AND en OR om ervoor te zorgen dat de juiste keuzes gemaakt worden. **AND heeft** namelijk een **hogere prioriteit** dan OR en **wordt** altijd **eerst geëvalueerd**. Een voorbeeld maakt dit snel duidelijk. Stel, u wilt de telefoonnummers tonen van zowel Karin als Pieter die beiden op de afdeling Sales werken. Als u de query zo zou opstellen:

```
SELECT tel
FROM telnrs
WHERE afdeling = 'Sales'
AND naam = 'Karin'
OR naam = 'Pieter';
```

Dan krijgt u inderdaad Karin en Pieter van Sales maar ook Pieter van de ICT afdeling! In bovenstaande query wordt AND namelijk als eerste uitgevoerd (Karin en Sales) en daarna pas komt OR aan de beurt (of Pieter). Hierdoor worden dus alle Pieters geselecteerd, ongeacht de afdeling waarop ze werken. U kunt dit zo oplossen:

```
SELECT tel
FROM telnrs
```

Database gebruiken

```
WHERE afdeling = 'Sales' AND naam = 'Karin'  
OR afdeling = 'Sales' AND naam = 'Pieter';
```

Of door haakjes te gebruiken:

```
SELECT tel  
FROM telnrs  
WHERE afdeling = 'Sales'  
AND (naam = 'Karin' OR naam = 'Pieter');
```

Een andere toepassing van de haakjes:

```
SELECT tel  
FROM telnrs  
WHERE id > 10 OR (id > 1 AND id < 5)
```

U kunt ook een aantal speciale relationele operatoren gebruiken in uw condities. Deze behoeven wellicht wat meer uitleg:

operator	omschrijving	Opmerkingen
BETWEEN	Tussen	
NOT BETWEEN	Niet tussen	
IS NOT NULL	Heeft een waarde	
IS NULL	Heeft geen waarde	
LIKE	Gedeeltelijk gelijk aan	In combinatie met % of _ als wildcards

BETWEEN gebruikt u om records binnen een bepaalde numerieke range te selecteren, u heeft daarvoor ook altijd het keyword AND nodig:

```
SELECT naam, tel  
FROM telnrs  
WHERE tel  
BETWEEN 1000 AND 2000
```

U krijgt nu alle namen en telefoonnummers van mensen met een telefoonnummer tussen de 1000 en de 2000. De grenswaarden maken ook deel uit van de selectie.

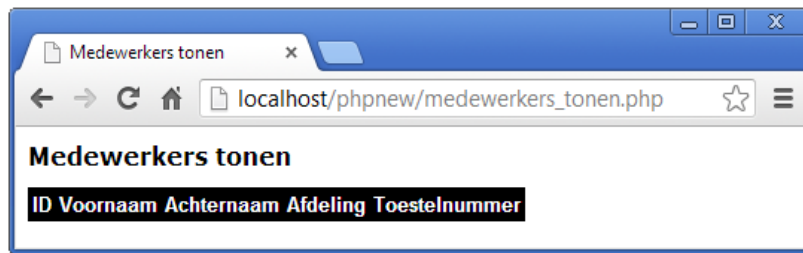
Met IS (NOT) NULL kunt u records selecteren waarvan een bepaald veld juist wel of niet een waarde heeft. Dit is handig wanneer velden in een tabel gedefinieerd zijn die NULL mogen zijn, oftewel die niet per se een waarde bevatten, maar ook leeg kunnen zijn.

Met LIKE kunt u **zoekopdrachten** construeren. We bespreken dit in de cursus **PHP Vervolg**.

Oefening 6.4	<i>Beperk nu de selectie door alleen de medewerkers van de afdeling Opleidingen te tonen.</i>

6.17 Controle op aanwezigheid records

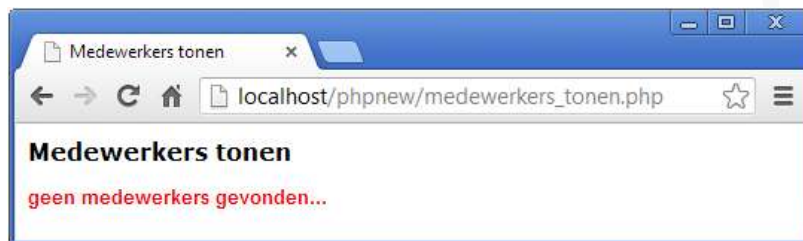
In de code die u in de vorige oefeningen geschreven hebt, mist nog een belangrijke controle, namelijk of het SELECT statement wel records oplevert. Als er niet minimaal een record in de resultset zit, kan dat tot een vreemde schermopbouw leiden, variërend van een compleet lege pagina tot een tabel die alleen tableheaders bevat:



Het is daarom verstandig om altijd te controleren op de aanwezigheid van records in de resultset, oftewel het aantal records moet groter zijn dan nul. Is dat niet het geval dan geeft u dat met een duidelijke melding door aan de eindgebruiker, bijvoorbeeld zo:

```
<?php if (mysqli_num_rows($result) > 0): ?>
    zet resultaat in html tabel ...
<?php else: ?>
    <p class="warning">geen medewerkers gevonden...</p>
<?php endif; ?>
```

Wat een veel logischere pagina geeft:



Oefening 6.5 Voeg bovenstaande controle toe aan `medewerkers_tonen.php` en test dit met een *WHERE* clause in uw *SELECT* statement die geen records oplevert.

6.18 Afbeeldingen en hyperlinks tonen

Met een beetje extra opmaak kunt u bijvoorbeeld ook een 'smoelenboek' met foto's en klikbare e-mail adressen uit de database laten rollen. In feite beschikt u nu al over de hiervoor noodzakelijke kennis.

Een foto toevoegen is simpel wanneer u een veld `foto` aan de database heeft toegevoegd met daarin het (relatieve) pad naar de foto. Iets vergelijkbaars heeft u al eerder in de cursus gedaan:

```
">
```

Het is ook mogelijk om de foto's in de database op te slaan als een BLOB (Binary Large Object) maar dat valt buiten het bestek van deze cursus. Bovenstaande oplossing waarbij verwezen wordt naar een locatie op het bestandssysteem van de server is overigens veel efficiënter.

Smoelenboek



Arjen van den Achterberg
manager
arjen@compufirm.nl



Bettine van Berends
directeur
bettine@compufirm.nl



Cora Custers
secretaresse
cora@compufirm.nl

id	voornaam	tussenvoegsel	achternaam	functie	afdeling	toestelnummer	email	foto
1	Arjen	van den	Achterberg	manager	Opleidingen	1234	arjen@compufirm.nl	arjen.jpg
2	Bettine	van	Berends	directeur	Sales	2345	bettine@compufirm.nl	bettine.png
3	Cora	NULL	Custers	secretaresse	Directie	3456	cora@compufirm.nl	cora.gif

Een klikbaar e-mail adres maakt u als volgt wanneer u een veld `email` in de databasetabel heeft staan:

```
<a href="mailto:<?php echo $row['email']?>"><?php echo $row['email']?></a>
```

Let heel erg goed op het correct afsluiten van de attribuutwaarden in HTML. Een quote is zo vergeten! Valideer daarom steeds uw HTML uitvoer!

6.19 Gegevens in een formulier tonen

U bent natuurlijk niet gebonden aan het tonen van database gegevens in tabellen. U kunt ze overall binnen de pagina tonen. Wat in de praktijk veel voorkomt is het tonen van gegevens in formulieren:

Om waarden in de opties van een selectbox (keuzelijst) te zetten kunt u de volgende code gebruiken. Als value kunt u het ID van het record gebruiken:

```
<form action="toon_medewerkers.php" method="get">
  <select name="id">
    <?php while ($row = mysqli_fetch_assoc($result)): ?>
      <option value="<?=$row['id']?>"><?=$row['naam']?></option>
    <?php endwhile; ?>
  </select>
  <input type="submit" value="toon record">
</form>
```

6.20 Gebruik van dynamische SQL statements

We hebben nu een uit de database gevuld formulier waarmee we een persoon kunnen selecteren. Na selectie wordt het *name-value pair*, bijvoorbeeld `id=3` naar de pagina verzonden die we bij het

`action` attribuut hebben gespecificeerd. We willen uiteraard op deze pagina de waarde uitlezen en opslaan in een variabele. Dit doen we op de bekende manier:

```
$id = $_GET["id"];
```

Maar wat gaan we er dan mee doen? We gaan de variabele gebruiken in wat we een **dynamisch SQL statement** noemen. Tot nu toe hebben we **statische** SQL statements gebruikt. In onze code gaven we van tevoren precies aan waaruit het SELECT statement in `$query` moest bestaan en vervolgens werd het exact zo uitgevoerd.

We gaan onze SQL statements nu een stuk flexibeler maken door ze samen te stellen met informatie die afkomstig is uit variabelen. U gebruikt het verzonden ID om in het aannemende script een SELECT statement mee op te bouwen waarmee verdere gegevens van het betreffende record worden opgehaald. Let wel goed op de samenstelling van de totale string:

```
$query = "SELECT * FROM telnrs WHERE id = " . $id . ";;";
```

Eindgebruikers kunnen zo elk record selecteren dat ze maar willen. Vandaar dat in dit verband ook wel van **user-driven querying** wordt gesproken.

Let op: U zou nooit zomaar gebruikersinvoer zonder enige vorm van controle mogen gebruiken voor de opbouw van een dynamisch SQL statement! Dit vanwege het risico op zogenaamde **SQL Injection Attacks**. We gaan hier in de cursus **PHP Vervolg** uitvoerig op in.

Wat u in dit geval al kunt doen is met de functie `intval()` er voor zorgen dat niet numerieke waarden worden omgezet naar 0. Zie ook **6.20.2**.

Omdat er nu maar één record in de resultset zit, kunnen we bij het tonen van de gegevens de lus achterwege laten.

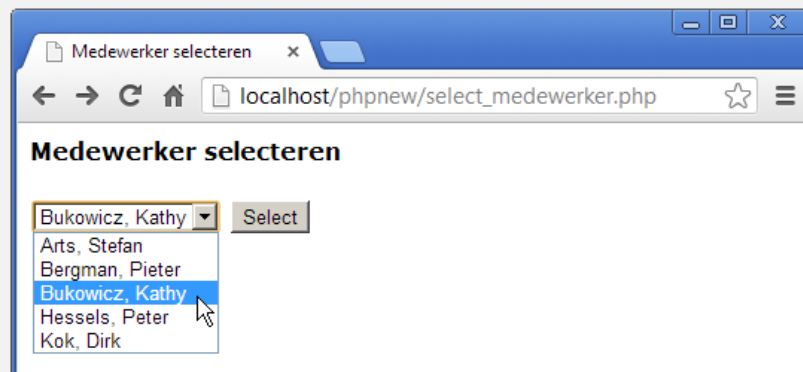
Tip: Schrijf altijd eerst uw dynamisch opgebouwde SQL statement naar de pagina toe voordat u het laat uitvoeren. Zo kunt u het eerst goed op de juiste notatie controleren! U kunt het wegschrijven en niet laten uitvoeren combineren door de query met `die()` weg te schrijven voordat u de query uitvoert:

```
die($query);  
$result = mysqli_query($query, $conn);
```

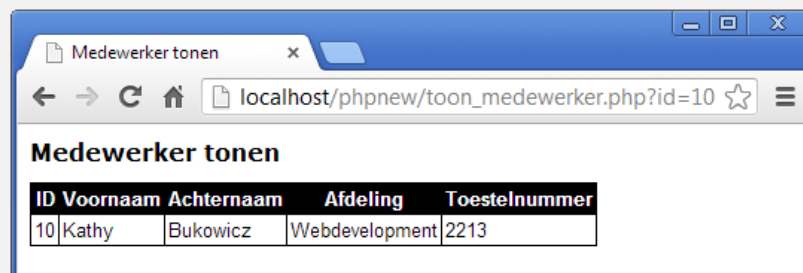
De query wordt nu naar het scherm geschreven en **niet uitgevoerd**.

Oefening 6.6a

Maak een pagina `select_medewerker.php` waarin u een formulier opneemt met een selectbox die de voornaam en achternaam van de medewerkers laat zien. Toon eerst de achternaam, dan een komma en dan de voornaam. Orden de medewerkers oplopend op achternaam.



Bij selectie wordt als value het bijbehorend id verstuurt via de GET method naar een pagina `toon_medewerker.php`:



U mag ook de pagina `medewerkers_tonen.php` hergebruiken zoals beschreven in een volgende paragraaf.

Verdieping

Toon de uitgevoerde query in beeld.

Verdieping

Bouw beveiliging tegen het los opvragen van de pagina in.

Verdieping

Zorg er met de functie `intval()` voor dat manipulaties van de querystring geen kwaad kunnen en toon een nette melding wanneer er geen record gevonden kan worden.

Op dit moment is de detailpagina niet zo spannend, maar u kunt zich voorstellen dat bij meer velden in de database dit een **compleet overzicht** van een bepaalde medewerker zou kunnen geven (met foto, datum in dienst, geboortedatum, GSM nummers, e-mailadressen etc.).

Dergelijke functionaliteit komt veel in applicaties voor en wordt vaak **master-detail** pagina's of **drill-down** genoemd.

6.20.1 beveiliging tegen geïsoleerd opvragen

Let op: het geïsoleerd opvragen van `toon_medewerker.php` (dus zonder eerst het formulier op te vragen) geeft nu een lelijke foutmelding omdat het id nu ontbreekt:

Notice: Undefined index: id in **toon_medewerker.php** on line 3

You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near " at line 1

De oplossing bestaat er uit om eerst bovenaan uw script te controleren of er wel een variabele `$_GET['id']` aanwezig is, en als dit niet het geval is, de browser te instrueren om het formulier op te vragen:

```
if (!isset($_GET['id'])) {  
    header('location: select_medewerker.php');  
    exit();  
}
```

In de cursus **PHP Vervolg** gaan we dieper in op dergelijke problematiek en hoe dit op te lossen.

Oefening 6.6b

Maak van de pagina `select_medewerker.php` een variant `select_medewerker_links.php` waarin u de medewerkers als hyperlinks toont:



Deze versturen eveneens het bijbehorende id naar de pagina `toon_medewerker.php`.

We zullen later in de cursus, bij het aanpassen van gegevens, ook nog een andere variant tegenkomen waarbij gegevens van een record in de tekstvelden van een formulier worden getoond.

6.20.2 Hergebruik medewerkers_tonen.php

U kunt ook uw eerder gemaakte pagina `medewerkers_tonen.php` hergebruiken door de query uit te breiden met een WHERE clause indien er een id wordt meegestuurd:

```
// query selecteert alle medewerkers  
$query = "SELECT * FROM medewerkers";  
  
// maar als er een id is meegestuurd dan selecteert  
// de query alleen de gegevens van die medewerker  
if (isset($_GET['id'])) {  
    $query .= " WHERE id=" . intval($_GET['id']) . " ";  
}  
  
// voer de query uit  
$result = mysqli_query($conn, $query);
```

Database gebruiken

Hierdoor wordt, wanneer de pagina `medewerkers_tonen.php` gewoon wordt opgevraagd, de volgende query uitgevoerd:

```
SELECT * FROM medewerkers
```

Wordt de pagina echter opgevraagd met een id in de querystring bijvoorbeeld `medewerkers_tonen.php?id=7`, dan wordt die query door middel van de shorthand notatie `.=` voor concatenatie uitgebreid met een WHERE clause:

```
SELECT * FROM medewerkers WHERE id=7
```

De functie `intval()` zorgt ervoor dat de waarde van `$id` naar een integer wordt geconverteerd. Zou een gebruiker het numerieke id veranderen in een string die niet geconverteerd kan worden naar een integer, bijvoorbeeld `id=bla` dan zorgt `intval()` ervoor dat dit wordt omgezet naar de waarde 0. De query zal vervolgens geen resultaten opleveren maar ook geen foutmeldingen of erger.

Fraaier is het dan ook om de controle zo te doen:

```
$id = isset($_GET['id']) ? intval($_GET['id']) : 0;
```

En dan alleen de query uit te breiden als het id groter is dan 0:

```
$query = "SELECT * FROM medewerkers";  
  
if ($id > 0) {  
    $query .= " WHERE id = " . $id;  
}  
  
$query .= " ORDER BY id DESC";
```

Merk op dat als u de queries met een ORDER BY clause (die we later in dit hoofdstuk zullen bespreken) dat u dan de query niet in de WHERE clause met een `;` kan afsluiten!

Beide voorbeelden zijn eenvoudige manieren om **SQL Injection Attacks** te voorkomen op plekkern waar u numerieke invoer verwacht. In de **PHP Vervolg cursus** leert u meer over deze en andere *security issues* en zult u ook leren dat u in dit soort gevallen beter met **prepared statements** kunt werken.

6.20.3 Sprintf()

Dynamisch SQL statements kunt u overigens ook mooi opbouwen met `sprintf()`. U maakt dan gebruik van **placeholders**:

```
$query = sprintf("SELECT * FROM medewerkers WHERE id = %d", $id);
```

Of wanneer het om strings gaat in een INSERT statement (dat we in het volgende hoofdstuk leren maken):

```
$query = sprintf("INSERT INTO medewerkers  
    (voornaam, achternaam, afdeling, toestelnummer)  
VALUES  
    ('%s', '%s', '%s', '%s');",  
    $voornaam, $achternaam, $afdeling, $toestelnummer);
```


Dit lijkt al een beetje op de geparametriseerde queries die we later zullen leren maken, maar hier moet u nog wel zelf quotes om de stringwaarden zetten en ze niet vergeten te escapen!

Want deze manier van noteren is namelijk nog steeds **niet veilig** wanneer de variabele een string is! U leest hier meer over in [Escape SQL](#).

6.21 Gegevens toevoegen: INSERT

Voor het toevoegen van records aan een tabel in een database gebruiken we het SQL `INSERT` statement. Het voegt altijd een hele nieuwe rij toe. Het standaard formaat is:

```
INSERT INTO tabelnaam
(kolom1, kolom2)
VALUES
(waarde1, waarde2);
```

Er is geen `WHERE` clause voor `INSERT` statements omdat er niet wordt gewerkt met bestaande data...

Een insert statement wordt ook wel een *append query* genoemd. Om een dergelijke query juist te kunnen uitvoeren heeft u een drietal gegevens nodig:

1. de **naam** van de **tabel**
2. de **namen** van de **velden** in de tabel in een kommagescheiden lijst (*fieldlist*). De velden worden tussen haakjes gezet.
3. de **waarden** die in de **velden** moet komen tabel in een kommagescheiden lijst (*valuelist*). Dit deel wordt ook wel als de *row constructor* aangeduid. De veldwaarden worden eveneens tussen haakjes gezet.

Een `INSERT` statement zou er bijvoorbeeld als volgt uit kunnen zien:

```
INSERT INTO telnrs
(naam, telefoon)
VALUES
('Karin', 612345);
```

Let op: Het **aantal** en de **volgorde** van de velden in de *fieldlist* moet **precies overeen komen** met de values in de *valuelist*. U moet bovendien een waarde opgeven voor alle velden waarvoor geen default value is opgegeven en waar `NULL` niet is toegestaan.

In bovenstaand voorbeeld werd **geen id** in de *fieldlist* en geen waarde voor `id` in de *valuelist* opgegeven. Dat kan omdat dit veld gedefinieerd is als **primary key met een auto_increment**. MySQL zal bij het invoegen van een nieuw record **automatisch** de veldwaarde **toevoegen**, namelijk de waarde van het vorige record opgehoogd met 1.

Let op: Waarden voor alle **tekstvelden** in de database (`CHAR` en `VARCHAR`) moeten tussen **quotes** staan! Dat moet omdat binnen deze waarden woorden en punctuatie voor kan komen die anders als SQL geïnterpreteerd zou worden en onverwachte gevolgen kan hebben. Als u het SQL statement tussen dubbele quotes heeft staan dan is het handig om single quotes te gebruiken...

6.21.1 Andere formaten

In principe is het ook mogelijk om het `INSERT` statement iets compacter te noteren door de *fieldlist* **weg te laten** maar dan moet u waarden voor **alle velden** in de tabel opgeven (ook `NULL` waarden) en u exact houden aan de **juiste volgorde** die de velden in de tabel hebben:

```
INSERT INTO telnrs
VALUES
    (NULL, 'Karin', 612345);
```

De fieldlist is nu *assumed*. Als u te weinig veldwaarden opgeeft dan is de lijst met velden langer en krijgt u een syntax error. Als u de velden wel correct heeft opgegeven dan loopt u nog altijd het risico dat het mis zal gaan als de tabel waar de rijen ingezet moeten worden in een later stadium wordt uitgebreid, want ook dan komt het aantal aangenomen velden niet overeen met de opgegeven waarden!

Advies: gebruik dus altijd de eerder genoemde notatie!

Sommige databases zoals MySQL en PostgreSQL staan toe dat met één INSERT statement meerdere rijen toegevoegd kunnen worden:

```
INSERT INTO telnrs
    (naam, telefoon)
VALUES
    ('Karin', 612345),
    ('Jaap', 789012),
    ('Arno', 456789);
```

De row constructors worden nu met **komma's** gescheiden. Hoewel dit valide SQL is wordt het niet door alle database systemen ondersteund. SQL Server staat het bijvoorbeeld niet toe. In dergelijke gevallen (en wanneer u uw code portabel wilt houden) geeft u 3 aparte INSERT queries op:

```
INSERT INTO telnrs
    (naam, telefoon)
VALUES
    ('Karin', 612345);

INSERT INTO telnrs
    (naam, telefoon)
VALUES
    ('Jaap', 789012);

INSERT INTO telnrs
    (naam, telefoon)
VALUES
    ('Arno', 456789);
```

De INSERT statements worden nu, zoals gebruikelijk, met puntkomma's gescheiden.

6.21.2 Insert in PHP

Als u een INSERT opdracht wilt geven vanuit een PHP pagina dan zet u het statement eerst weer in een variabele `$query`:

```
$query = "INSERT INTO telnrs
    (naam, telefoon)
VALUES
    ('Karin', 612345);";
```

In bovenstaande situatie zijn de gegevens die moeten worden toegevoegd hard in de query gecodeerd. Dat komt in de praktijk natuurlijk zelden voor omdat dan iedere keer als dit statement wordt uitgevoerd het telefoonnummer en de naam van Karin opnieuw aan de lijst met telefoonnummers zullen worden toegevoegd!

6.21.3 Insert data uit formulier

Normaal gesproken zullen de waarden die we gaan invoegen ergens vandaan komen; ze komen bijvoorbeeld uit een door een gebruiker ingevuld formulier.

We kunnen de ingevoerde waarden dan opslaan in variabelen en daarmee het SQL statement opbouwen. We maken dan een zogeheten **dynamisch SQL statement**. Op die manier kunnen we het INSERT statement gebruiken voor het invoeren van vele verschillende namen en telefoonnummers en niet alleen die van Karin!

Stel dat de in te voegen gegevens afkomstig zijn van een formulier, dan kunnen we deze uitlezen en in variabelen plaatsen:

```
$naam      = $_POST["naam"];
$telefoon  = $_POST["telefoon"];
```

We lezen hier de `$_POST` superglobal uit omdat formulieren die mutaties in een database veroorzaken **altijd** met de method POST verstuurd dienen te worden!

Let op: Dergelijke code zou meteen **alarmbellen** moeten doen **rinkelen**! **Gebruikersinvoer** uit **superglobals** moet u **niet direct gebruiken** maar **eerst filteren** en **daarna escaperen**. We gaan daar in een later stadium nog uitgebreid op in... Vooralsnog geven we alvast aan dat het escaperen vroeger automatisch ging met een instelling `magic_quotes_gpc`. Dit is inmiddels achterhaald en u moet er nu zelf voor zorgen. U doet dat met de functie `mysqli_real_escape_string()`:

```
$naam      = mysqli_real_escape_string($conn, $_POST["naam"]);
$telefoon  = mysqli_real_escape_string($conn, $_POST["telefoon"]);
```

Door de escape functie te gebruiken zorgt u er allereerst voor dat invoer waarin een single quote voorkomt geen error oplevert. Bij een naam als O'Brian zou de single quote anders het SQL statement breken. Dit wordt na escaperen echter O'Brian wat keurig als O'Brian in de database wordt gezet. U voorkomt hiermee tevens dat kwaadwillende het gebruik van single quotes kunnen exploiteren in zogenaamde **SQL Injection Attacks**. Dit zijn aanvallen waarbij een kwaadwillende gebruiker het SQL statement dat u naar de database server stuurt probeert aan te passen, waardoor een andere query wordt uitgevoerd dan die u voor ogen had. Samen met XSS Injection Attacks zijn SQL Injection Attacks de meest voorkomende aanvallen op websites.

In de cursus **PHP Vervolg** leert u hier meer over dit alles en hoe problemen te voorkomen, o.a. door gebruik te maken van **prepared statements**.

We maken vervolgens bij het opbouwen van het statement in de variabele `$query` gebruik van de eerder behandelde methodes om stukken tekst en variabelen aan elkaar te plakken m.b.v. de concatenatie operator (`.`). De complete opdracht voor het samenstellen van het (dynamische) SQL statement komt er dan uiteindelijk zo uit te zien:

```
$naam      = mysqli_real_escape_string($conn, $_POST["naam"]);
$telefoon  = mysqli_real_escape_string($conn, $_POST["telefoon"]);

$query = "INSERT INTO telnrs
        (naam, telefoon)
        VALUES
        ('" . $naam . "', '" . $telefoon . "')";
```

6.21.4 'Testen' query

Het opbouwen van dergelijke statements is best lastig. U moet dan ook heel goed opletten of u de strings, variabelen en operatoren op de juiste manier gebruikt. Juist omdat het zo lastig is om deze

statements op te bouwen is het prettig om ze in een string te hebben staan. U kunt het SQL statement dan makkelijk even naar scherm schrijven voordat u het uit laat voeren:

```
die($query);
```

Zo kunt u eerst controleren of het resultaat wel een valide opdracht oplevert voordat u de opdracht door de database laat uitvoeren (en de onvermijdelijke foutmelding volgt en u mogelijk allerlei onvolledige of zelfs lege records uit de database moet gaan halen...).

Een handige manier is om de velden met eenvoudige opvolgende data in de vorm van letters te vullen en te controleren of die op de juiste volgorde in het INSERT statement terecht komen, of ze allemaal komma gescheiden zijn en als het om strings gaat dat ze tussen quotes staan:



Dit zou de volgende uitvoer moeten geven:

```
INSERT INTO telnrs (naam, telefoon) VALUES ('A', 'B');
```

Als de controle geslaagd is kunt u de `die($query)` opdracht uitcommentariëren of weghalen.

Let op: bij het invoegen van een string waarin een *single quote* voorkomt zou u wellicht een foutmelding verwachten, bijvoorbeeld bij een achternaam als `O'Brian`. De single quote in de invoer zou normaal gesproken het SQL statement breken.

U krijgt echter géén foutmelding als in `php.ini` de optie `magic_quotes_gpc` aan staat, iets wat in veel installaties het geval is. Die instelling zorgt ervoor dat quotes in strings die binnenkomen via `$_GET`, `$_POST` of `$_COOKIES` automatisch worden escaped met backslashes.

De variabele `$_POST['achternaam']` krijgt daardoor de waarde `"O\'Brian"` wat keurig als `O'Brian` in de database wordt gezet.

In de cursus **PHP Vervolg** leert u meer over het **veilig escaperen** van **data** in SQL statements, onder andere met de functie `mysql_real_escape_string()`.

6.21.5 Action query: geen resultset

Nadat de INSERT query correct is opgesteld, geven we hem, net als bij de SELECT opdracht uit de vorige oefening, als argument aan de functie `mysqli_query()` zodat het INSERT commando naar de database verzonden kan worden.

Let op: Omdat een INSERT statement een zogenaamde *action query* is krijgt u geen records terug. Het heeft dan in principe ook geen zin om de `$result` identifier te gebruiken, maar `mysqli_query()` retourneert nu `true` als de query geslaagd is (en uiteraard nog steeds `false` wanneer het mis gaat). U kunt dat gebruiken om een foutmelding te tonen:

```
// Voer INSERT query uit
// omdat het een action query is,
// is er geen resultset...
if (!mysqli_query($conn, $query)) {
    echo 'De opdracht kon niet worden uitgevoerd';
}
```

Het is overigens standaard niet mogelijk om met `mysqli_query()` meer dan één query tegelijk uit te voeren. Als u meerdere records wilt toevoegen als op de bovenstaande manier dan moet u steeds een nieuwe query uitvoeren!

6.21.6 Veel gemaakte fouten

Enkele veel gemaakte fouten zijn:

1. Variabelen zijn niet juist geïnitieerd
2. Vergeten van de spaties die tussen de delen van de SQL opdracht horen te staan
3. Onjuist gebruik van de single quotes. Deze moeten de waarden insluiten wanneer het om strings gaat
4. Vergeten van het laatste haakje
5. Na de laatste veldnaam of veldwaarde toch nog een komma plaatsen

Het even naar het scherm schrijven van de samengestelde query, zoals hier boven beschreven, maakt echter vaak meteen duidelijk waar het probleem zit!

Voor het toevoegen van een record zullen we dus 2 pagina's gebruiken: een formulier om de waarden in te voeren en een aannemende pagina die de waarden aanneemt, in een insert query zet, database verbinding maakt en de query uitvoert.

In de oefening met de beroepen in hoofdstuk 4 hebben we gezien hoe we informatie uit een vorig formulier kunnen gebruiken in een volgende pagina. Van deze wetenschap gaan we weer gebruik maken in de volgende oefening:

Oefening 6.7

Maak een nieuwe pagina `medewerker_toevoegen.html` die een formulier bevat waarmee u een nieuwe medewerk(st)er kunt invoeren:



The screenshot shows a web browser window with the title 'Medewerker toevoegen'. The address bar displays 'localhost/phpnew/medewerker_toevoegen.html'. The main content area contains a form with the title 'Medewerker toevoegen'. The form consists of four labeled input fields: 'Voornaam:', 'Achternaam:', 'Afdeling:', and 'Toestelnummer:'. Below these fields is a button labeled 'Toevoegen'.

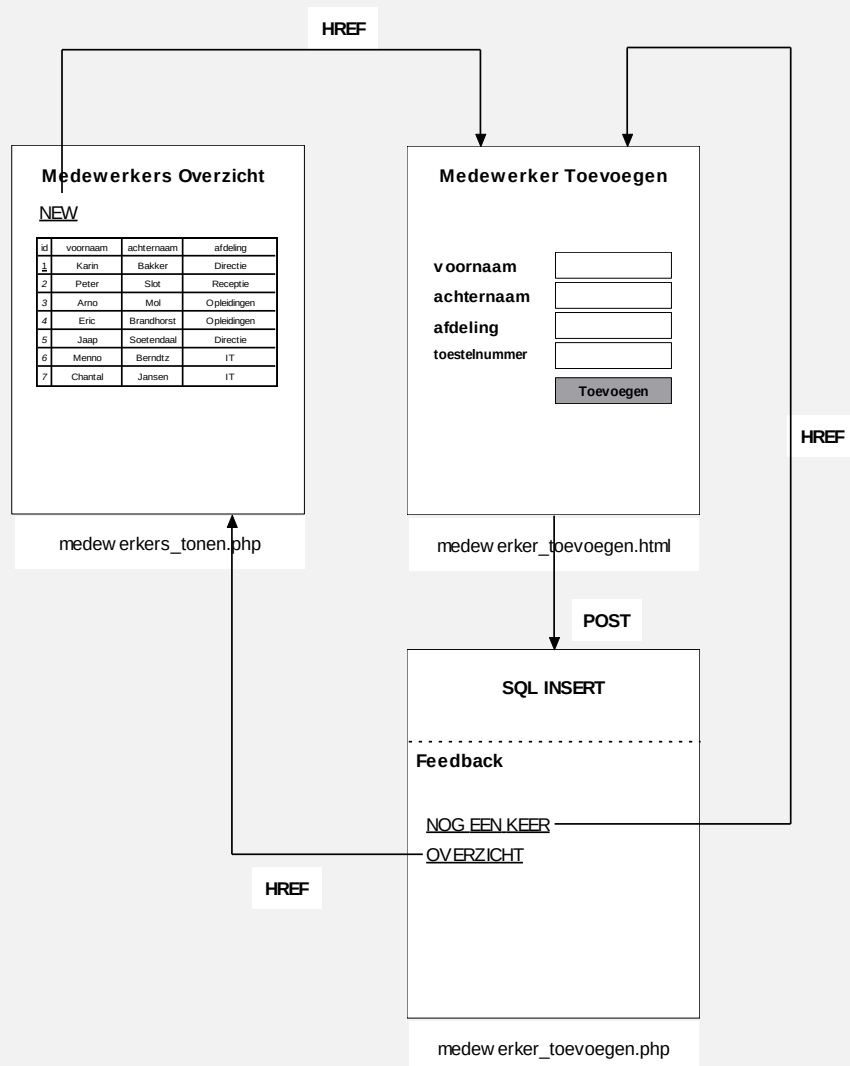
Na een klik op de submit knop, die we de toepasselijke naam Toevoegen hebben gegeven, wordt deze informatie in een tweede pagina `medewerker_toevoegen.php` aangenomen. In deze pagina worden de verzonden gegevens in variabelen gezet. Hiermee wordt een dynamisch SQL INSERT statement opgebouwd waarna de gegevens als een nieuw record worden toegevoegd aan de database.

Let op: Het ID hoeft niet in dit SQL statement opgenomen te worden omdat MySQL hier zelf voor zorgt. U heeft bij het aanmaken van de tabel `medewerkers` immers aangegeven dat het veld `ID` `auto_increment` is. Hierdoor zal het veld `ID` bij het toevoegen van een nieuw record automatisch met 1 worden opgehoogd.

Toon na het uitvoeren van de opdracht een boodschap op scherm die aangeeft dat het record is toegevoegd. Maak een hyperlink terug naar `medewerker_toevoegen.html` zodat makkelijk opnieuw een medewerker kan worden toegevoegd. Maak ook een hyperlink naar `medewerkers_tonen.php` uit de vorige oefening en controleer of het record ook daadwerkelijk is toegevoegd.

Maak tenslotte op de pagina `medewerkers_tonen.php` een hyperlink naar `medewerker_toevoegen.html`.

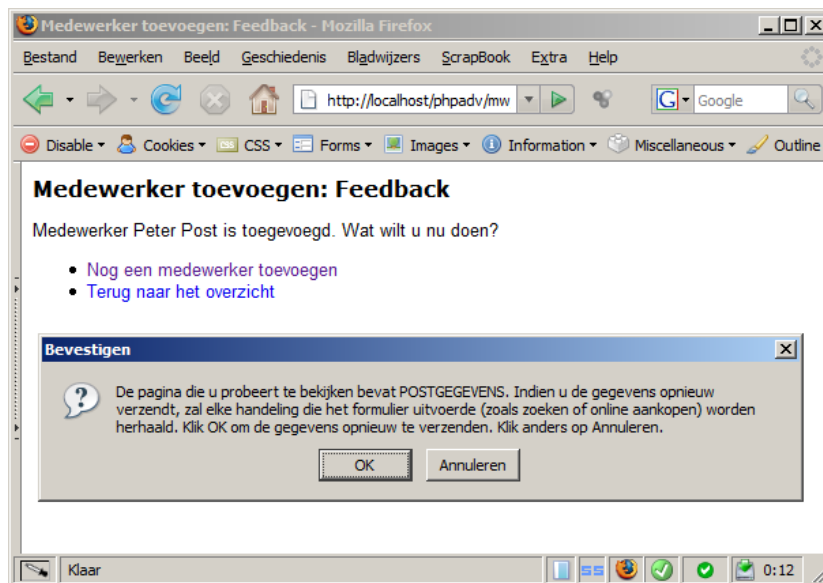
Uw mini-applicatie ziet er in schema nu zo uit:



LET OP: In bovenstaande oefeningen wordt **geen** enkele **controle** uitgevoerd om te controleren of de invoer van een gebruiker wel in het formaat is dat wij verwachten. In de praktijk zult u dat **altijd** moeten **controleren** om problemen te voorkomen! **Data validatie** is echter een te uitgebreid onderwerp voor deze basiscursus. In de **PHP Vervolg cursus** gaan we er wel diep op in.

6.21.7 VERDIEPING: duplicate records voorkomen

Als een gebruiker de pagina waar de feedback op staat (per ongeluk) zou verversen, dan zou hij een melding als deze te zien krijgen:



Deze melding wordt door de browser gegenereerd en informeert de gebruiker dat de pagina die hij wil verversen tot stand is gekomen door een POST request. Als hij deze pagina wil verversen dan kan dat alleen door de pagina opnieuw aan te maken door het formulier nog eens te verzenden.

Als de gebruiker op Annuleren klikt is er niets aan de hand maar als hij op OK klikt wordt de zojuist ingevoerde medewerker nog een keer toegevoegd, alleen nu met een ander id:

53	Peter	Post	ICT	5678
54	Peter	Post	ICT	5678

Dat is op zich correct want de records verschillen op hun primary key, ook al is de rest van de velden identiek. Maar toch is dit iets wat u waarschijnlijk niet wil laten gebeuren.

6.21.7.1 Redirect naar feedback pagina

U kunt het eenvoudig oplossen door na het toevoegen geen feedback te geven en met een `header('location: medewerkers_tonen.php')` de gebruiker weer naar het overzicht te dirigeren. Denk eraan deze opdracht te laten volgen door `exit()`. We zullen deze werkwijze straks bij de UPDATE toepassen.

Wilt u toch feedback geven dan kunt u de gebruiker op een vergelijkbare manier met de `header` functie doorsturen naar een pagina `feedback.php`. Die kan de gebruiker naar hartenlust verversen zonder dat er het risico bestaat dat de medewerker steeds weer opnieuw wordt toegevoegd.

U kunt in dit geval echter niet zo makkelijk de ingevoerde gegevens laten zien. De enige manier die u op dit moment heeft is om de gegevens van de medewerker in de querystring naar de feedbackpagina mee te sturen:

```
header('location: feedback.php?voornaam=Peter&achternaam=Post');  
exit();
```

U kunt ze vervolgens in `feedback.php` inlezen en tonen. In de **PHP Vervolg cursus** zult u leren dat dit doorgeven van waarden fraaier opgelost kan worden door zogenaamde **sessie variabelen** te gebruiken. Deze kunt u ook gebruiken om een algemene succes boodschap te tonen.

De hierboven beschreven oplossing is een bekend patroon op het web en dit **redirect-after-post pattern** wordt ook wel het **PRG Pattern** genoemd waarbij PRG staat voor **Post/Redirect/Get**. U leest er meer over op <http://en.wikipedia.org/wiki/Post/Redirect/Get>.

6.21.7.2 Controle op duplicate records

Ook als u redirect blijft het mogelijk eenzelfde medewerker toe te voegen. Het probleem met de duplicate data bestaat eigenlijk ook al bij de eerste keer invoeren van een nieuwe medewerker. Het is daarom verstandig om sowieso altijd te controleren of er niet al een medewerker met dezelfde gegevens in de database zit.

U kunt dat doen door voor het uitvoeren van de INSERT query eerst een SELECT query uit te voeren met in de WHERE clause de gegevens uit het formulier:

```
// Invoer uit formulier halen en opslaan in variabelen
$voornaam      = $_POST["voornaam"];
$achternaam    = $_POST["achternaam"];
$afdeling       = $_POST["afdeling"];
$toestelnummer = $_POST["toestelnummer"];

// voer eerst SELECT query uit met de nieuwe waarden om te
// controleren of er niet al zo'n record is toegevoegd...
$duplicate_query = "SELECT COUNT(*) AS aantal FROM medewerkers
                    WHERE voornaam      = '$voornaam'
                    AND   achternaam    = '$achternaam'
                    AND   afdeling      = '$afdeling'
                    AND   toestelnummer = '$toestelnummer'";

// INSERT statement opbouwen met de variabelen
$query = "INSERT INTO medewerkers
        (voornaam,
         achternaam,
         afdeling,
         toestelnummer)
        VALUES
        ('$voornaam',
         '$achternaam',
         '$afdeling',
         '$toestelnummer')";

// Verbinding met de database maken
$conn = mysqli_connect("localhost", "gebruiker", "geheim")
        or die('Kan geen verbinding maken...');

// Database selecteren
mysqli_select_db($conn, "cursus")
        or die('database niet beschikbaar...');

// Duplicate query uitvoeren
$result = mysqli_query($conn, $duplicate_query)
        or die(mysqli_error($conn));
$duplicate_count = mysqli_result($result, 0, 'aantal');

if ($duplicate_count == 0) {
    // Query uitvoeren
    // (dit is een action query dus er is geen resultset)
    mysqli_query($conn, $query) or die(mysqli_error($conn));
}
```

Na het op deze manier uitvoeren van de duplicate query heeft u een resultset die, afhankelijk van het feit of er wel of niet een duplicate record in de database zit, de waarde 0 of 1 bevat, bijvoorbeeld:

aantal

Door deze waarde op te slaan in een variabele, kunt u er vervolgens uw feedback op aanpassen:

```
<h1>Feedback</h1>

<?php if ($duplicate_count == 0): ?>

<p class="ok">
  Medewerker <?php echo $voornaam . ' ' . $achternaam; ?>
  is toegevoegd. Wat wilt u nu doen?
</p>

<ul>
  <li><a href="medewerker_toevoegen.html">
    Nog een medewerker toevoegen</a></li>
  <li><a href="medewerkers_tonen.php">
    Terug naar het overzicht</a></li>
</ul>

<?php else: ?>

<p class="warning">
  Er is al een medewerker
  <?php echo $voornaam . ' ' . $achternaam; ?>
  op de afdeling <?php echo $afdeling ?>
  met toestelnummer <?php echo $toestelnummer ?>!
</p>

<ul>
  <li><a href="javascript:history.go(-1)">
    Terug naar het formulier</a></li>
  <li><a href="medewerkers_tonen.php">
    Terug naar het overzicht</a></li>
</ul>

<?php endif; ?>
```

Dit zorgt ervoor dat wanneer de ingevoerde medewerker al in de database zit deze niet wordt toegevoegd:

Feedback

Er is al een medewerker Peter Post op de afdeling ICT met toestelnummer 5678!

- [Terug naar het formulier](#)
- [Terug naar het overzicht](#)

Zit de medewerker echter nog niet in de database dan wordt deze gewoon toegevoegd:

Feedback

Medewerker Petra Plezier is toegevoegd. Wat wilt u nu doen?

- [Nog een medewerker toevoegen](#)
- [Terug naar het overzicht](#)

6.21.7.3 Duplicate records voorkomen in database

Het is ook mogelijk om het voorkomen van duplicate records te voorkomen op database niveau. Wanneer er al een gelijk record bestaat, wordt dit record niet toegevoegd, maar wordt een error gegenereerd. De wijze waarop dit dient te gebeuren, verschilt echter per database. Bovendien moet u de error op een nette wijze zien af te handelen. We bespreken deze methode daarom verder niet in deze basis cursus.

6.21.7.4 `mysqli_affected_rows()`

De feedback die we tot nu toe geven gaat er van uit dat het toevoegen inderdaad gelukt is. Als er iets fout gaat in het uitvoeren van de query dan geven we met `or die()` een foutmelding. U kunt deze ook achterwege laten en met de functie `mysqli_affected_rows()` controleren of het toevoegen echt gelukt is.

Deze functie retourneert een integer die aangeeft hoeveel records door de laatst uitgevoerde query zijn beïnvloed. Wanneer de query mislukt is dan wordt -1 geretourneerd. U kunt de functie gebruiken om feedback te geven aan de eindgebruiker of de uitgevoerde action query (INSERT, UPDATE of DELETE) succesvol is geweest. U moet daarbij de link identifier `$conn` als verplicht argument meegeven.

Bij een INSERT query zou de feedback er dan als volgt uit kunnen zien:

```
if (mysqli_affected_rows($conn) == 1) {  
    echo '<p class="ok">Record is succesvol toegevoegd</p>';  
} else {  
    echo '<p class="warning">'.mysqli_error($conn).'</p>';  
}
```

Bij een SELECT query is de waarde gelijk aan het aantal geselecteerde records, en dus gelijk aan `mysqli_num_rows($result)`.

6.21.7.5 Nieuwe ID achterhalen

MySQL zorgt er door middel van de `auto_increment` op het ID veld voor dat elke nieuwe record automatisch een uniek id krijgt dat 1 hoger is dan de vorige in de tabel. Als u wilt weten wat het id van het zojuist toegevoegde record is dan kunt u dat uitlezen met de functie `mysqli_insert_id()`. Het neemt als verplicht argument de link identifier:

```
// voeg nieuw record toe  
...  
  
// lees id van zojuist toegevoegde record uit  
$new_id = mysqli_insert_id($conn);
```

6.21.7.6 Toevoegen lege records voorkomen

Wat in de praktijk nogal eens gebeurt, is dat cursisten het bestand `medewerker_toevoegen.php` rechtstreeks opvragen, bijvoorbeeld omdat ze willen testen of hun laatste aanpassingen werken. Wanneer een cursist op F12 drukt in Dreamweaver dan wordt de pagina rechtstreeks opgevraagd en als er geen fouten in de code staan dan wordt nu automatisch een leeg record toegevoegd! U kunt dit eenvoudig voorkomen door code als deze bovenaan uw pagina op te nemen:

```
if ($_SERVER['REQUEST_METHOD'] == 'GET') {  
    header('location: medewerker_toevoegen.html');  
    exit();  
}
```

Wanneer de pagina dan per ongeluk via GET wordt opgevraagd in plaats van via POST vanuit het invoerformulier dan wordt er geen leeg record toegevoegd maar wordt u automatisch naar het invoerformulier gebracht want natuurlijk het juiste vertrekpunt is voor het werken met het invoerscript!

In de **vervolgcursus** zullen we dit oplossen met **selfsubmitting forms**.

6.22 Gegevens aanpassen: UPDATE

Met het UPDATE statement kunt u records in de database aanpassen:

```
UPDATE tabel  
SET kolom1 = waarde1,  
    kolom2 = waarde2  
WHERE conditie(s);
```

We overschrijven als het ware bestaande informatie in de database met nieuwe informatie.

Het UPDATE statement wordt meestal gebruikt om alleen records die aan bepaalde criteria voldoen aan te passen. Dit kunnen we bereiken door het zorgvuldig samenstellen van de WHERE clause:

```
UPDATE
    telnrs
SET
    tel = 1234
WHERE
    afdeling = 'Sales';
```

De nieuwe waarde hoeft niet persé een vaste waarde te zijn maar kan ook een expressie zijn, bijvoorbeeld:

```
UPDATE
    cds
SET
    catID = 9,
    Prijs = Prijs * 1.1
WHERE
    catID = 5;
```

Dit statement verplaatst alle producten uit categorie 5 naar 9 en verhoogt de prijs met 10%.

In de praktijk wordt UPDATE vaak toegepast om één, reeds bestaand, record te kunnen aanpassen. De werkwijze is dan als volgt:

1. Een gebruiker selecteert in het overzicht het record dat moet worden aangepast, meestal door te klikken op een knop of hyperlink.
2. Het id van het betreffende record wordt gebruikt om de huidige gegevens op te halen.
3. De opgehaalde gegevens worden in een formulier getoond.

Het tonen van de opgehaalde gegevens in een formulier gaat als volgt:

```
<?php
// haal id op en gebruik deze om een dynamisch SELECT statement te maken
$id = isset($_GET["id"]) ? intval($_GET["id"]) : 0;
$query = "SELECT naam, telefoon FROM telnrs WHERE id = " . $id . " ";

// maak verbinding en voer query uit
```

```

...
// het resultaat bevat als het goed is precies een record
// dus de while lus is niet nodig:
$contact = mysqli_fetch_assoc($result);
?>
<!-- begin formulier -->
...
<!-- zet id in hidden field! (zie uitleg hierna) -->
<input type="hidden" name="id" value="<?php echo $id; ?>">
<!-- zet veldwaarden in het formulier -->
<input type="text" name="naam" value="<?php echo $contact['naam']; ?>">
...

```

Let op: u schrijft een veldwaarde weg binnen de aanhalingstekens van het HTML value attribuut. Denk eraan dat u het input element correct afsluit!

Nu de gegevens in het formulier staan zijn dit de volgende stappen:

4. De gebruiker past de gegevens aan.
5. Een druk op de knop "Aanpassen" zorgt er voor dat de gewijzigde gegevens middels een POST naar het aannemende script worden gestuurd.
6. Er wordt een (dynamisch) UPDATE statement opgebouwd en naar de database verzonden waarna de oude gegevens in de database ook daadwerkelijk met de nieuwe waarden worden overschreven.
7. De gebruiker krijgt feedback of wordt doorgestuurd naar het overzicht waar het aangepaste record te zien is.

Een voorbeeld van een dergelijk dynamisch opgebouwd UPDATE statement:

```

$id      = intval($_POST["id"]);
$naam    = mysqli_real_escape_string($conn, $_POST["naam"]);
$telefoon = mysqli_real_escape_string($conn, $_POST["telefoon"]);

$query = "UPDATE
        telnrs
        SET naam      = '" . $naam . "',
            telefoon = '" . $telefoon . "'
        WHERE
            id = " . $id . " ";

```

Bij het uitvoeren van deze action query is, net als bij INSERT, ook **geen resultset** beschikbaar, dus u kunt voor het uitvoeren van de query weer volstaan met:

```

if (!mysqli_query($conn, $query)) {
    $error = mysqli_error($conn);
}

```

6.22.1 WHERE clause

Let op: U moet bij het updaten van records altijd heel goed op de aanwezigheid van de WHERE clause letten! Als u die vergeet krijgen **alle records** in de tabel **dezelfde** (nieuwe) **waarde**!

Stel u heeft de volgende telefoonlijst:

Telefoonlijst

[Nieuw contact toevoegen](#)

id	naam	telefoon
1	Arno	1123456789
2	Jaap	0234567891
3	Eric	0345678912
4	Carry	0456789123

En in het aanpass scherm wilt u het eerste telefoonnummer wijzigen:

Contact bewerken

Naam:

Telefoon:

En u zou dat doen met deze query:

```
$query = "UPDATE
        telnrs
        SET naam      = '" . $naam . "',
            telefoon = '" . $telefoon . "' ;";
```

dan zullen **alle** records in de database dezelfde waarden krijgen:

Telefoonlijst

[Nieuw contact toevoegen](#)

id	naam	telefoon
1	Arno	0123456789
2	Arno	0123456789
3	Arno	0123456789
4	Arno	0123456789

En dat is natuurlijk niet de bedoeling! Het is ook **niet terug te draaien** als u geen back-up van de data had! Let hier dus heel goed op!

De **oplossing** is hetzelfde als in de auto oefening die u eerder maakte: U moet in het invoerscherm het **id** van het aan te passen record in een **hidden field** zetten en in het script dat de feitelijke update verzorgt moet u dit id dan **gebruiken** in de **WHERE clause**.

TIP: Sommige programmeurs bouwen nog een extra beveiliging in door aan het eind van de UPDATE query LIMIT 1 op te geven:

```
UPDATE ... SET ... WHERE ... LIMIT 1;
```

Dit is een **MySQL specifieke syntax** die ervoor zorgt dat er maximaal 1 record wordt aangepast. U kunt het ook bij DELETE gebruiken. Misschien had u de constructie al eens in phpMyAdmin voorbij zien komen...

In de cursus **PHP Vervolg** zullen we LIMIT ook gebruiken in combinatie met SELECT om **gepagineerde resultsets** te maken.

We zullen de hierboven beschreven situatie uitwerken in de volgende oefening.

Oefening 6.8

U begint met het aanpassen van de pagina `medewerkers_tonen.php`. In de tabel moet het ID van elk record een hyperlink worden:



The screenshot shows a web browser window titled "Medewerkers tonen - Microsoft Internet Explorer". The address bar shows "http://localhost/php/medewerkers_tonen.php". The page content includes a link "Nieuwe medewerker toevoegen" and a table with employee data. The table has columns: ID, Voornaam, Achternaam, Afdeling, and Toestelnummer. The IDs are hyperlinks. The status bar at the bottom shows "http://localhost/php/medewerker_bewerken.php?id=4".

ID	Voornaam	Achternaam	Afdeling	Toestelnummer
1	Sheila	Lavanter	Verkoop	1000
2	Peter	Spanje	Verkoop	1000
3	Jaap	Soetendaal	Directie	9999
4	Arno	Mol	Opleidingen	1234
5	Kees	Prins	Opleidingen	1234

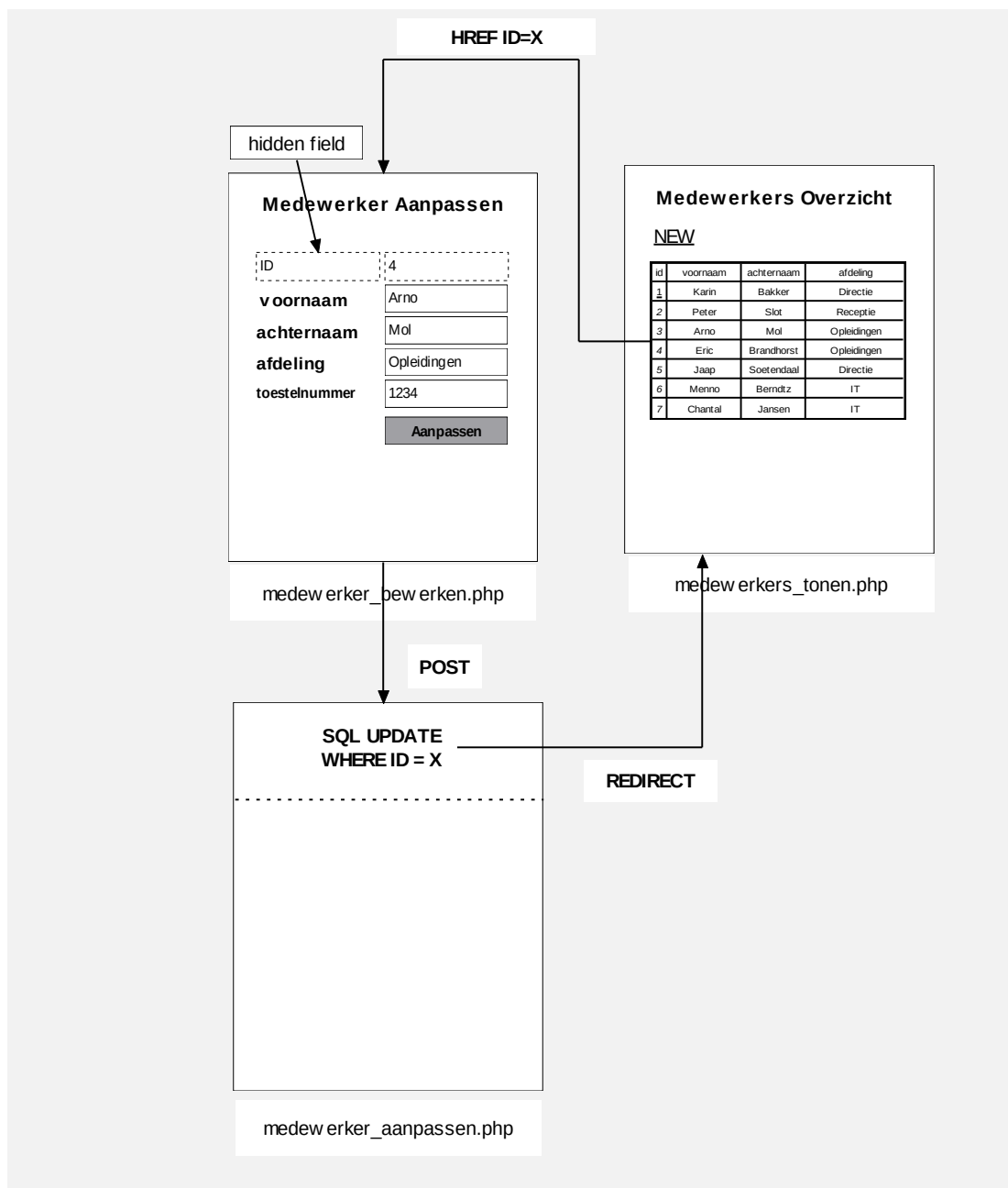
De hyperlinks verwijzen allen naar een tweede pagina: `medewerker_bewerken.php`. Door te klikken op het id veld van een bepaald record wordt het ID van het betreffende record meegestuurd over de `QueryString`. In de pagina `medewerker_bewerken.php` wordt dit ID uitgelezen en gebruikt in de `WHERE` clause van een `SELECT` statement om de veldwaarden van het record in een formulier in te vullen:

Voornaam:	Arno
Achternaam:	Mol
Afdeling:	Opleidingen
Toestelnummer:	1234
Bewerken	

Hierna kunt u de gegevens wijzigen (let op: het ID zelf dus NIET). Na klikken op de `submit` knop die we `Aanpassen` noemen, zal de gewijzigde informatie verzonden worden naar een derde pagina `medewerker_aanpassen.php` waar het `UPDATE` statement wordt opgebouwd en verzonden naar de database. Denk eraan dat u het ID als hidden field meestuurt, u heeft het nodig in de `WHERE` clause van de `UPDATE` query!

Zorg er door middel van de functie `header('location: medewerkers_tonen.php')` voor dat na het uitvoeren van de `UPDATE` opdracht automatisch de pagina `medewerkers_tonen.php` wordt getoond.

In schema ziet dit er zo uit:



6.23 Gegevens verwijderen: DELETE

U verwijdert gegevens uit een tabel met het DELETE statement:

```
DELETE FROM tabel
WHERE conditie(s)
```

Omdat u geen losse kolommen kan verwijderen (alleen hele rijen) hoeft u geen column list te specificeren. Met andere woorden: u hoeft geen veldnamen op te geven. De condities zijn optioneel, maar als u ze weg laat dan worden alle records verwijderd! De tabelstructuur blijft daarbij overigens wel intact... Men heeft het in dit verband ook wel over *truncating the table* (de tabel inkorten).

Let op: U moet bij het verwijderen van records altijd heel goed op de aanwezigheid van de WHERE clause letten! Als u die vergeet verwijdert u **alle** records uit de tabel!

Hieronder een voorbeeld hoe een DELETE statement er uit zou kunnen zien wanneer het binnen een PHP pagina wordt toegepast:

```
$query = "DELETE FROM telnrs WHERE id=" . $id . " ";
```

Met DELETE verwijdert u zoals eerder gezegd altijd hele rijen. Het is niet mogelijk om velden te verwijderen. Het enige wat u kan doen is met een UPDATE statement de waarde van een veld op NULL zetten, maar dit kan natuurlijk alleen bij nullable fields...

6.23.1 Soft Delete

Het is vrij gebruikelijk om records niet echt te verwijderen maar deze door middel van een UPDATE inactief te maken. U kunt daarvoor een veld *tonen* of *actief* gebruiken en deze dan naar wens aan of uitvinken. Een dergelijke manier van werken haalt het record dus niet echt weg en wordt ook wel met **soft delete** aangeduid.

Wanneer u dan een SELECT query uitvoert moet deze de als verwijderd gemarkeerde records natuurlijk niet alsnog in de actieve selectie meenemen. Dat betekent het toevoegen van een WHERE clause die op de actieve records filtert (of de inactieve eruit filtert).

Maar dat doet u natuurlijk alleen niet doen in uw CRUD overzicht waar u de records toont anders heeft u hier geen mogelijkheid meer om de inactieve users weer te activeren! Tenzij u natuurlijk weer rechtstreeks naar de database gaat via phpMyAdmin maar dat was nu juist niet de bedoeling van het CRUD idee.

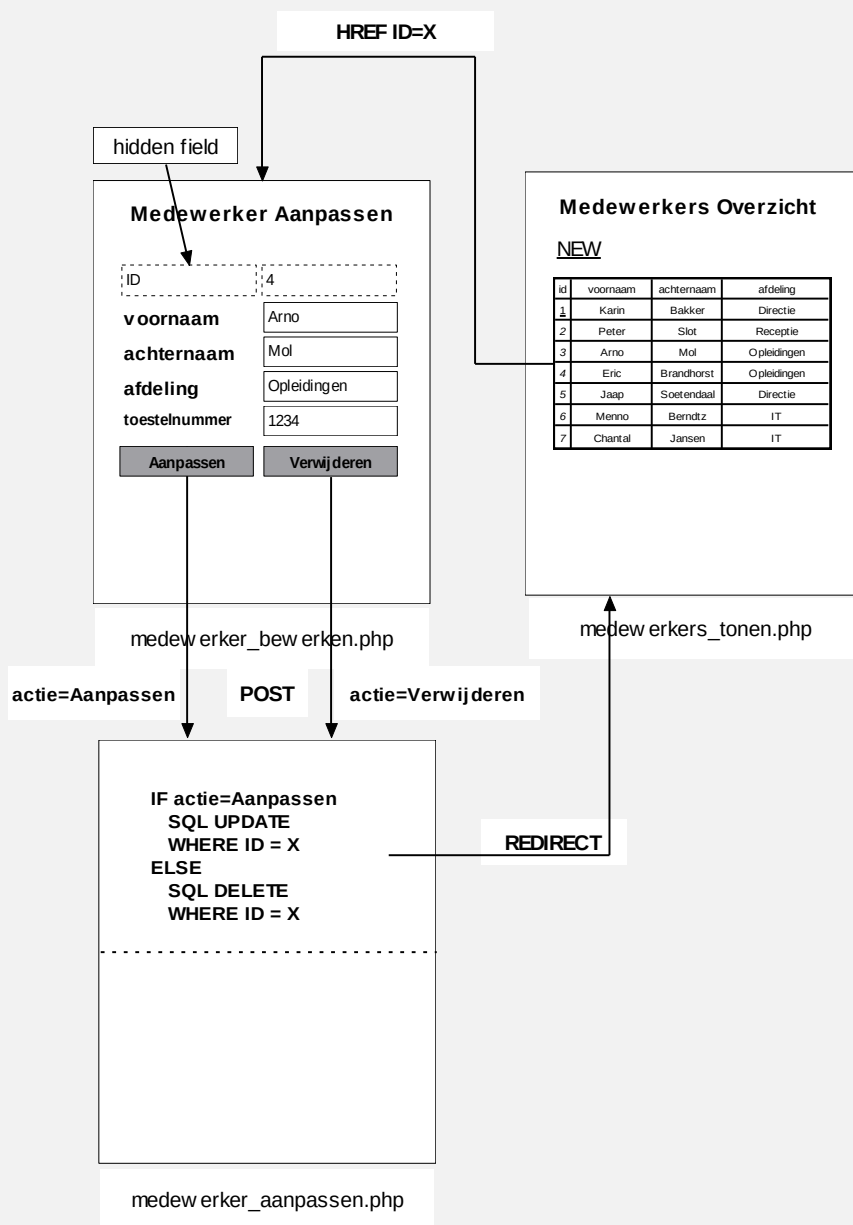
Wanneer u later in de cursus verschillende userlevels heeft geïmplementeerd dan zal in dit soort situaties een gebruiker met admin rechten bijvoorbeeld mogen (de)activeren terwijl een gewone user alleen de actieve records ziet...

Oefening 6.9 Voeg aan het formulier op de pagina `medewerker_bewerken.php` een tweede submit button toe die u Verwijderen noemt. Geef de button dezelfde naam als de button met de tekst Bewerken, bijvoorbeeld actie, u heeft nu 2 submit buttons:

```
<input type="submit" name="actie" value="Bewerken">
<input type="submit" name="actie" value="Verwijderen">
```

Controleer in de derde pagina van de vorige oefening, `medewerker_aanpassen.php` welke waarde `$_POST["actie"]` heeft om vervolgens te bepalen of een UPDATE of DELETE statement moet worden uitgevoerd voor het betreffende record.

In schema:



6.23.2 Verwijderen bevestigen

Omdat het verwijderen van een record onherroepelijk is, moet u altijd eerst waarschuwen voordat u een record verwijdert!

Dat kan heel snel Client-side met een JavaScript confirm:



die heel eenvoudig aan de delete-knop of link toe te voegen is:

```
<input type="submit"
       name="actie"
       value="Verwijderen"
       onclick="return confirm('Zeker weten?');">
```

Bij een keuze voor Annuleren wordt het formulier dan niet verzonden... tenminste als de browser JavaScript ondersteunt...

Beter is het om in plaats van het JavaScript Server-side een tussenpagina `delete_confirm.php` toe te voegen. Hoe dat zou kunnen werken, laten we zien bij de alternatieve presentatie:

6.24 Alternatieve presentatie

In plaats van het id klikbaar te maken en de update en delete acties te combineren in de pagina's `medewerker_bewerken.php` en `medewerker_aanpassen.php` kunt u de pagina `medewerkers_tonen.php` ook zo opzetten:

Overzicht Medewerkers				
Nieuwe medewerker toevoegen				
Voornaam	Achternaam	Afdeling	Toestelnummer	Acties
Sheila	Lavanter	Verkoop	1201	update delete
Peter	Spanje	Verkoop	1201	update delete
Jaap	Soetendaal	Directie	1299	update delete
Arno	Mol	Opleidingen	1234	update delete
Kees	Prins	Opleidingen	1234	update delete
Denise	Smit	Verkoop	1201	update delete

De verschillende handelingen zijn nu rechtstreeks toegankelijk. Nog fraaier maakt u het met icoontjes voor de insert, update en delete acties en een afwisselende rij kleur:

Overzicht Medewerkers

 [Nieuwe medewerker](#)

Voornaam	Achternaam	Afdeling	Toestelnummer	Acties	
Sheila	Lavanter	Verkoop	1201		
Peter	Spanje	Verkoop	1201		
Jaap	Soetendaal	Directie	1299		
Arno	Mol	Opleidingen	1234		
Kees	Prins	Opleidingen	1234		
Denise	Smit	Verkoop	1201		

Verdieping

Voer bovenstaande presentatie door in `medewerkers_tonen.php`. U kunt gebruik maken van de afbeeldingen `insert.png`, `update.png` en `delete.png` in de `/images` map op de server.

Online kunt u ook diverse gratis icon sets vinden, bijvoorbeeld:

<http://www.famfamfam.com/lab/icons/silk/>

Een mooi alternatief is een icon font gebruiken, zoals bijvoorbeeld

<http://fontawesome.io>

In de **PHP Vervolg cursus** leert u hoe u de database acties met formulieren kunt terugbrengen van een los formulier met een aparte formhandler naar **één script per actie** in een zogenaamd **self-submitting form**. Uiteindelijk kunt u zelfs alle acties door middel van een switch in één script afhandelen en het formulier hergebruiken voor zowel de insert als de update acties.

6.24.1 Verwijderen bevestigen

Wanneer u de update en delete acties als hyperlinks opneemt naast de tabel dan ziet dit er weliswaar fraai uit, maar het is dan ook nogal verleidelijk om de delete links simpelweg zo uit te voeren:

```
medewerker_verwijderen.php?id=3
```

Als de delete pagina echter simpelweg op deze manier zou worden aangeroepen en meteen het record met behulp van het id uit de querystring zou verwijderen, dan zondigt u tegen het principe dat een **action query altijd door** middel van een **POST** moet worden uitgevoerd.

In theorie zou het nu dan ook mogelijk zijn dat een spider van een zoekmachine die alle links binnen uw site volgt alle records zou kunnen deleten. Zo'n vaart zal het niet snel lopen, want een pagina als deze zal normaal in een afgeschermd gedeelte van een site staan en alleen na inloggen toegankelijk zijn...

NB: Hoe u een afgeschermd gedeelte met een login maakt leert u later in de cursus.

Merk overigens op dat dit voor de update link niet geldt! De link ziet er weliswaar vergelijkbaar uit:

```
medewerker_aanpassen.php?id=3
```

Maar hier is de id die via GET wordt verzonden wordt alleen bedoeld om een selectie te maken! Het haalt een record op wat vervolgens in een formulier geladen kan worden. Dit formulier, dat via POST wordt verzonden, regelt de daadwerkelijke update.

U kunt het GET DELETE probleem op verschillende manieren ombuigen naar een POST DELETE oplossing.

Merk op dat gebruik van het JavaScript confirm dialoogvenster op de link weliswaar bescherming biedt tegen per ongeluk verwijderen van records door een onoplettende gebruiker of een spider, maar dat de werking afhankelijk is van JavaScript en dat na bevestiging nog steeds via GET verwijderd wordt.

We moeten dus wat anders verzinnen...

6.24.1.1 Confirmatie pagina met POST form

De makkelijkste manier is om de Delete link eerst naar een confirmatie pagina te laten verwijzen en dan pas vanuit daar naar de daadwerkelijke verwijder actie over te gaan:

```
<a href="medewerker_delete_confirm.php?id=?=$id?">delete</a>
```

Die pagina zou er bijvoorbeeld zo uit kunnen zien:

```
...  
<h1>Medewerker verwijderen</h1>  
  
<p>U staat op het punt de medewerker met het id  
<strong><?php echo $id ?></strong> definitief  
uit de database te verwijderen. U kunt dit <strong>  
niet ongedaan</strong> maken!</p>  
  
<p>Weet u zeker dat u door wilt gaan?</p>  
  
<form method="post" action="medewerker_delete.php">  
    <!-- cancel hyperlink, via GET terug naar overzicht -->  
    <a href="medewerkers_tonen.php">Cancel</a>  
  
    <!-- delete button, id gaat als hidden field via POST  
    naar de delete pagina -->  
    <input type="hidden" name="id" value="<?php echo $id; ?>">  
    <input type="submit" value="Delete">  
  
</form>  
...
```

Wanneer iemand op de knop Delete klikt wordt het id via POST naar de daadwerkelijke delete pagina gestuurd. We hebben ook een Cancel hyperlink toegevoegd die de gebruiker via GET weer terugvoert naar de overzichtspagina:

Veel mooier en duidelijker is het echter om de gegevens van het record erbij te tonen, dit laat goed zien wat er eventueel wordt verwijderd. U moet dan wel de medewerker gegevens uit de database ophalen met behulp van het id uit de querystring. Dit kan op precies dezelfde manier als u deed bij het ophalen van de gegevens om ze in het update formulier te kunnen tonen. Alleen toont u de gegevens nu in een details tabel. Dat gaat heel makkelijk met een foreach loop, die u ook al eerder heeft gemaakt:

```
<?php
    // haal medewerker gegevens uit database op met id uit querystring...
    ...
?>
...

<h1>Medewerker verwijderen</h1>

<p>U staat op het punt de medewerker met het id
<strong><?php echo $id ?></strong> definitief
uit de database te verwijderen.</p>

<table class="details">
    <?php foreach ($medewerker as $key => $value): ?>
        <tr>
            <th scope="row"><?=$key?>:</th>
            <td><?=$value?></td>
        </tr>
    <?php endforeach; ?>
</table>

<p>U kunt dit <strong>niet ongedaan</strong> maken!</p>

<p>Weet u zeker dat u door wilt gaan?</p>

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

    <!-- cancel hyperlink, via GET terug naar overzicht -->
    <a href="medewerkers_tonen.php">Cancel</a>

    <!-- delete button, id gaat als hidden field via POST
    naar de delete pagina -->
    <input type="hidden" name="id" value="<?php echo $id; ?>">
    <input type="submit" value="Delete">

</form>

...
```

Dan is dit het resultaat:

Medewerker verwijderen

U staat op het punt de medewerker met het id **1** definitief uit de database te verwijderen.

id:	1
voornaam:	Sheila
achternaam:	Devenster
afdeling:	Development
toestelnummer:	1234

U kunt dit **niet ongedaan** maken!

Weet u zeker dat u door wilt gaan?

[Cancel](#)

6.24.1.2 Hyperlink vs. Button

Wat echter niet zo fraai is, is dat er een hyperlink gebruikt wordt voor het annuleren en een knop voor het deleten. Mooier zou het zijn om twee knoppen te hebben...

U kunt dit ook weer op verschillende manieren oplossen:

6.24.1.2.1 Gebruik twee submit buttons

Gebruik voor de Cancel ook een `<input type="submit">`. Geef de Cancel en Delete buttons **dezelfde** name, bijvoorbeeld "action":

```
<input type="submit" name="action" value="Cancel">
<input type="submit" name="action" value="Delete">
```

Als u liever `button` elementen gebruikt dan kan dat natuurlijk ook:

```
<button name="action" value="Cancel">Cancel</button>
<button name="action" value="Delete">Delete</button>
```

Na verzenden kijkt u welke waarde voor action is meegegeven. Alleen als het Delete is dan voert u de delete uit en anders doet u een redirect terug naar het overzicht:

```
<?php
// in delete.php:
if ($_POST['action'] === 'Delete') {
    // voer delete actie uit...
} else {
    // anders terug naar overzicht
    header('location: medewerkers_tonen.php');
}
```

Dit voelt echter wat tricky bij een delete actie en het kost sowieso een extra roundtrip...

6.24.1.2.2 Gebruik `input type="button"` en JavaScript

Maak van de Cancel link een `<input type="button">`. Deze verzendt het formulier niet. Met JavaScript kunt u nu 1 stap terug gaan in de browser history of het overzicht laden:

```
<!-- button die standaard niet submit maar met simpele inline
      JavaScript onclick handler werkt -->
<input type="button" value="Cancel"
      onclick="document.location='medewerkers_tonen.php'">
```

Hierdoor wordt het formulier niet verzonden en geen roundtrip gemaakt, maar het werkt natuurlijk alleen als JavaScript aanstaat. Als u het toch gebruikt dan natuurlijk liever zo:

```
<!-- beter zo en dan eventlistener in externe JS zetten -->
<input type="button" value="Cancel" id="annuleren">
```

In het externe JavaScript kunt u dan zoiets doen:

```
(function () {
    document.addEventListener('DOMContentLoaded', function() {
        var btn = document.getElementById('annuleren')
        btn.addEventListener('click', function() {
            document.location = 'medewerkers_tonen.php';
        });
    });
})();
```


6.24.1.2.3 Gebruik afbeeldingen en input type="image"

Gebruik voor zowel de Cancel link als de Delete button **plaatjes** van een knop. Voor de Delete button maakt u dan gebruik van een `<input type="image">`:

```
<a href="medewerkers_tonen.php">
  
</a>
<input type="image" src="img/btn-delete.png" alt="delete">
```

Maar het is nogal een gedoe om de plaatjes aan te maken en voelt wel heel erg gekunsteld aan. Overigens worden nu ook de X en Y coördinaten waar op de button geklikt is meegezonden...

6.24.1.2.4 Gebruik HTML5 attributen

U kunt voor Cancel een submit button maken maar er met de HTML5 attributen `formmethod` en `formaction` voor zorgen dat deze via GET naar het overzicht verstuurt:

```
<!-- submit button maar aangepast met andere method en action
(HTML5, LET OP: vanaf IE10) -->
<input type="submit" value="Cancel"
      formmethod="get"
      formaction="medewerkers_tonen.php">
```

Het grote probleem met deze oplossing is dat in oudere MSIE versies (alles voor IE10) deze attributen niet worden herkend en de button simpelweg als een submit button werkt. Met als gevolg dat het record dan dus ook wordt verwijderd als op Cancel wordt geklikt!

Dit is natuurlijk op te lossen door een Server-side controle op de name-value van de button te verrichten zoals getoond bij de twee submit buttons. Voelt nu al wel iets veiliger omdat de controle en de roundtrip alleen in (hele) oude browsers plaats vindt...

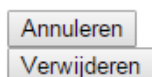
6.24.1.2.5 Gebruik twee formulieren

Een hele simpele, maar ook robuuste, oplossing is om 2 formulieren te maken, een GET met als action het overzicht voor annuleren en een POST voor verwijderen:

```
<!-- NEE: terug naar overzicht met GET formulier -->
<form method="get" action="medewerkers_tonen.php">
  <button>Cancel</button>
</form>

<!-- JA: verzend het id in een hidden field via form met method POST
waarna het record kan worden verwijderd -->
<form method="post" action="medewerker_delete.php">
  <input type="hidden" name="id" value="<?=$id?>">
  <button name="action" value="delete">Delete</button>
</form>
```

Probleem hier is alleen dat forms standaard blok elementen zijn en dat de buttons daardoor onder elkaar komen:



Dit is simpel op te lossen door ze in een div te zetten

```
<div class="forms">
  <form><button>Annuleren</button></form>
  <form><button>Verwijderen</button></form>
</div>
```

Database gebruiken

en dan in CSS iets als:

```
.forms form {display: inline-block;}
```

6.24.1.2.6 Maak de hyperlink met CSS op als een knop

Een andere oplossing is om de hyperlink gewoon te laten staan maar er met CSS voor te zorgen dat de link er als een knop uit ziet! Dit is echter relatief veel werk als u het vanaf scratch moet doen. Bovendien is het heel lastig om voor elk OS de link en de button er in iedere state (denk aan :hover en :active) er precies hetzelfde uit te laten zien als de default button...

Een opzetje:

```
a.btn {
  text-decoration: none;
  font: 0.85em Arial;
  background-color: buttonface;
  color: buttontext;
  padding: 1px 6px;
  box-sizing: border-box;
  border: 2px outset buttonshadow;

  ...

  -webkit-appearance: button;
  -moz-appearance: button;
  appearance: button;
}
```

6.24.1.2.7 Gebruik Bootstrap (of een ander framework)

Het wordt echter heel makkelijk als u een framework als **Bootstrap** gebruikt. Hiermee kunt u simpelweg door het toekennen van classes een link er hetzelfde uit laten zien als een knop:

```
<a href="medewerkers_tonen.php" class="btn btn-secondary">Cancel</a>
<input type="submit" value="Delete" class="btn btn-primary">
```

De hyperlink en button zien er dan zo uit:



U kunt ook kiezen voor andere classes maar deze geven de bedoeling wel helder weer. Classes als `btn-success` en `btn-danger` zien er wel leuk uit met respectievelijk groen en rood als achtergrondkleur maar zijn ook best verwarrend:



Want op een verwijder pagina is de delete button juist een succes actie toch? Bovendien betekenen de kleuren al helemaal niets voor (kleuren)blinden...

Over kleuren en volgorde (Welke knop moet eerst?) worden sowieso interessante discussies gevoerd. Zie bijvoorbeeld:

- <http://ux.stackexchange.com/questions/49991/should-yes-delete-it-be-red-or-green>
- <http://www.nngroup.com/articles/ok-cancel-or-cancel-ok/>

6.25 Afsluiting

SQL kent nog veel meer statements en de hier besproken statements kennen ook meer mogelijkheden dan hier aan de orde zijn gekomen. Binnen het bestek van deze cursus hebben we echter geen tijd om daar dieper op in te gaan. Dat doen we wel in de **PHP Vervolg cursus**.

U doet zichzelf echter een heel groot plezier door u nu alvast verder te verdiepen in de (on)mogelijkheden van SQL. De volgende links vormen een goed startpunt:

- <http://www.sqlcourse.com>
- <http://www.sqlcourse2.com>

Bovengenoemde sites zijn deels interactief: u kunt online opdrachten geven en het resultaat bekijken!

Naast deze fundamentele kennis dient u zich ook zeker op de hoogte te stellen van de toevoegingen die in het SQL dialect van de database van uw keuze mogelijk zijn. Voor MySQL kunt u daarvoor de online documentatie doornemen:

<http://dev.mysql.com/doc/>

Zorg er wel voor dat u de documentatie voor de **juiste versie** bekijkt!

Modulariseren

De cursus **PHP Vervolg** is geschreven als het logische vervolg op de basiscursus **PHP Basis**. Voor cursisten die de basiscursus niet gevolgd hebben volgt hieronder een overzicht van de in de basiscursus behandelde stof.

7.1 Inleiding

U heeft na het doorlopen van de cursus **PHP Basis** een aardig beeld gekregen van de werking van PHP. Tijdens de eerste dag heeft u geleerd hoe Server-side scripting werkt en hoe u binnen HTML pagina's PHP code kunt opnemen. Wanneer u nog geen programmeerervaring had dan fungeerde deze eerste dag als een inleiding in programmeren.

U heeft vervolgens geleerd hoe u (gebruikers)invoer aan kunt nemen die via de URL (GET) of via een formulier (GET of POST) binnenkomt.

In de tweede en laatste dag van de cursus heeft u geleerd hoe u vanuit een PHP pagina verbinding kunt maken met een MySQL database. U heeft daarna geleerd hoe u de data in een tabel in de database kunt uitlezen (met een SELECT QUERY) en gezien hoe u andere basishandelingen kon verrichten zoals het invoeren (INSERT), bijwerken (UPDATE) en verwijderen (DELETE) van data. Dit resulteerde uiteindelijk in een kleine applicatie waarmee u een database met één tabel online kunt beheren.

Deze "mini-applicatie" laat echter nog wel wat te wensen over. Zo is om te beginnen de code behoorlijk rommelig. De applicatie bestaat uit vele pagina's waarin steeds vergelijkbare stukken code worden herhaald. Er wordt in dit verband ook wel gesproken over *spaghetti code*, omdat een en ander enigszins ad hoc aan elkaar geknoopt is. En dat kan uiteraard handiger.

In dit hoofdstuk besteden we aandacht aan de manieren waarop u uw code beter kunt organiseren en hergebruiken.

Het sleutelwoord is daarbij **modulariseren**: u gaat uw code opdelen in **kleinere stukken** die niet alleen **beter hanteerbaar**, **beter onderhoudbaar** en **beter uitbreidbaar** zijn maar die ook nog eens **herbruikbaar** zijn. Hierdoor bespaart u uiteindelijk tijd en geld. Een bijkomend voordeel van het hergebruik van code is dat deze, juist door het vele gebruik, uiteindelijk vrijwel geen bugs meer zal bevatten en dus **robuustere code** oplevert.

We zullen voor de oefeningen in dit hoofdstuk (en de rest van de cursus) zoveel mogelijk **gebruik maken van de bestanden** die u in de **basiscursus** heeft gemaakt. Door deze bestanden als uitgangspunt te gebruiken worden de voordelen van de nieuw aangeleerde technieken hopelijk nog duidelijker!

De code die u tot nu toe geschreven heeft is namelijk wel functioneel maar niet erg onderhoudsvriendelijk. U heeft waarschijnlijk met behulp van *copy & paste* veel stukken code herhaald op verschillende pagina's en deze waar nodig licht aangepast.

Het verbinden met de database server en het selecteren van de database zijn bijvoorbeeld stappen die in elke pagina steeds hetzelfde zijn. Het is dan handig om dit vanaf een centrale plek te kunnen insluiten of anderszins te regelen. Als dan bijvoorbeeld het wachtwoord gewijzigd moet worden dan

hoeft u dat maar één keer aan te passen en niet op elke pagina afzonderlijk! En dat is natuurlijk altijd het streven van programmeurs, 'lui' als we zijn...

In dit hoofdstuk gaan we uitgebreid in op de diverse mogelijkheden om uw code beter te organiseren. Aan bod komen ondermeer **includes**, **templates**, **functies** en de al eerder genoemde **self-submitting forms**.

7.2 Includes

Includes worden gebruikt om functies, templates en/of andere stukken code die u meer dan eens nodig heeft, makkelijk binnen meerdere scripts te kunnen toepassen. U importeert dan als het ware code. Die code kan direct voor uitvoer in de pagina zorgen of alleen functionaliteit beschikbaar stellen. Bij het eerdere onderdeel werken met databases heeft u al gezien dat we op deze manier bijvoorbeeld een error template kunnen insluiten. In dit onderdeel krijgt u meer inzicht in het insluiten van templates en zullen we dat ook gaan toepassen op het tonen van data.

NB.: Er zijn overigens nog andere manieren om de inhoud van een bestand in te sluiten, onder andere `file()` en `fopen()`, maar hiermee kunt u geen PHP code insluiten en uit laten voeren. We komen hier in een later hoofdstuk nog op terug.

7.2.1 Notatie

Met behulp van het `include` statement kunt u de inhoud van een bestand toevoegen aan een PHP script. Om een bestand op te nemen in een PHP bestand gebruikt u de volgende code, die uiteraard in een PHP codeblok moet staan:

```
<?php
include('bestandsnaam.extensie');
?>
```

Opmerking: Omdat `include()` geen functie is maar een language construct mag u de haakjes eventueel ook weglaten (net als bij bijvoorbeeld `echo`):

```
<?php
// dit mag ook zonder haakjes:
include 'bestandsnaam.extensie';
?>
```

Omdat de laatste notatie net wat prettiger leest (en makkelijker typt) zullen we in de voorbeelden zonder haakjes werken.

Op de plek van `bestandsnaam.extensie` komt uiteraard een pad naar een bestaand bestand te staan. Dit kan een `.php` bestand zijn, waarin weer uit te voeren PHP code staat, maar dat hoeft niet. U kunt ook een `.txt` of `.html` bestand gebruiken om respectievelijk een stuk tekst of statische HTML in te sluiten. Sterker nog: u kunt zelfs uitvoerbare PHP code in een `.txt` bestand plaatsen, alleen is dat niet zo handig zoals we straks zullen zien...

Door een `include` te gebruiken wordt de inhoud van het bestand letterlijk ingesloten in de PHP pagina. Dit gebeurt precies op de plek waar u het `include` statement plaatst. PHP zal dat eerst voor alle includes doen en pas daarna het volledige script uitvoeren.

7.2.2 Paden

Het pad in de `include` kan zowel een relatieve als een absolute URL bevatten. Stel dat u een `script.php` wilt insluiten in een andere pagina. Wanneer beide webpagina's in de webroot staan dan kunt u ze als volgt insluiten:

```
// relatief
include 'script.php';
include './script.php'; // <-- BETER, zie uitleg na code

// absoluut fysiek pad Windows
include 'C:\xampp\htdocs\php\script.php';

// absoluut fysiek pad Linux
include '/home/cursist/htdocs/php/script.php';

// volledige lokale URL <-- LET OP DE EXTENSIE!
include 'http://localhost/php/script.txt';

// volledige URL
include 'http://www.provider.nl/php/script.txt';
```

Tip: Het is beter om de `./` notatie voor een relatief pad te zetten omdat dan gelijk de huidige working directory als uitgangspunt wordt genomen. Er hoeft dan niet door de paden die in `include_path` genoemd staan gezocht te worden (zie 1.2.3).

7.2.2.1 URL paden (Remote Includes)

Het is **in principe mogelijk** om tekst dan wel code die op een andere webserver staat in te sluiten, als dit tenminste niet beperkt wordt door de `allow_url_include` en `open_basedir` instellingen in `php.ini`. Voor de (lokale) URL paden moet het `allow_url_include` directive in `php.ini` de waarde `On` hebben, anders krijgt u foutmeldingen als deze:

Warning: include(): URL file-access is disabled in the server configuration in test.php

We willen u echter **sterk ontraden** om externe bronnen te gebruiken indien u dat ook maar enigszins kunt voorkomen!

- Het is traag!
- Als een hacker toegang kan krijgen tot de bronserver of de doelserver kan spoofen dan kan hij zijn eigen scripts laten insluiten en is de ellende niet te overzien!
- De bronbestanden die u vanaf een php-enabled server wilt laden moeten de PHP code als tekst doorgeven, u kunt de bestanden dus **geen** `.php` extensie geven, omdat de bestanden dan eerst verwerkt zullen worden en het resultaat zal worden ingesloten. U zult dus een andere extensie moeten gebruiken, bijvoorbeeld `.txt`, en omdat de bestanden in ieder geval in de webroot moeten staan (anders zijn ze ook niet voor u opvraagbaar) is de kans dat onbevoegden de code kunnen lezen dus niet denkbeeldig.
- Als de bronserver down is, dan is uw website dat ook...

7.2.2.2 Zet includes bij elkaar in een map

Zet alle includes bij elkaar in een directory die u `/includes` of iets dergelijks noemt. Dat maakt het voor uzelf en voor andere ontwikkelaars makkelijk om ze terug te vinden.

Het beste is om deze map **buiten de webroot** te plaatsen zodat onbevoegden er nooit vanaf het web bij kunnen komen! Zie ook hieronder en 1.2.12.1.

7.2.3 Include_path

In `php.ini` kunt u met het `include_path` directive de paden opgeven waarin standaard gezocht wordt naar een include. Bijvoorbeeld:

```
; Windows: "\path1;\path2"
include_path = ".;C:\php\includes;D:\Sites\_Includes"
```

Dit geeft aan dat in dezelfde map (.), in C:\php\includes en in D:\Sites_Includes gezocht wordt.

Houd de lijst zo kort mogelijk en zet altijd het meest gebruikte pad vooraan!

Tip: Als u nu in de map D:\Sites_Includes\ een subfolder `includes` aanmaakt en daarin uw common includes zet, dan kunt u op deze manier een standaardbibliotheek met includes aanleggen, die u op één plaats kunt onderhouden en in elk project in de map D:\Sites\ kunt insluiten. Dus in zowel D:\Sites\site.nl\htdocs als in D:\Sites\andresite.nl\htdocs kunt u nu simpelweg volstaan met iets als:

```
require_once('includes/common.php');
```

Als u online gaat zorgt u er simpelweg voor dat de bestanden uit de standaardbibliotheek in een map `includes` in de webroot op de webserver komen te staan.

Let op: Nog even voor de duidelijkheid, het is dus **NIET** zo dat u door het zetten van het `include_path` ervoor zorgt dat PHP **automatisch elk bestand in die directory insluit!** Het voegt alleen de naam van de directory toe aan de lijst die PHP doorzoekt wanneer een bestand moet worden ingesloten...

Let op: Het kan zijn dat er in `php.ini` beperkingen zijn opgelegd in het `open_basedir` directive. U kunt dan alleen bestanden insluiten die in deze map (of in mappen eronder) staan. Dit is bij providers nogal eens het geval. Of zo'n beperking voor u geldt kunt u snel controleren met `phpinfo()`.

7.2.3.1 `set_include_path()` en `get_include_path()`

Als u in een project een ander include path wenst maar dat niet in `php.ini` kunt (of wilt) wijzigen dan kunt u dat ook met de functie `set_include_path()` doen. Hiermee kunt u het pad (tijdelijk) overschrijven. Met zijn tegenhanger `get_include_path()` kunt u altijd uitlezen wat de huidige include paden zijn.

7.2.4 HTML insluiten

Met includes kunt u makkelijk standaard (navigatie) elementen als een menu of een footer aan pagina's toevoegen.

Stel u heeft het volgende menu in een bestand `menu.txt`:

```
<div id="nav">
  <a href="index.php">Home</a> |
  <a href="products.php">Products</a> |
  <a href="services.php">Services</a> |
  <a href="contact.php">Contact</a>
</div>
```

Dan kunt u dit eenvoudig op deze manier in elke pagina insluiten:

```
<html>
...
<body>
...
<!-- sluit menu in -->
<?php include 'menu.txt'; ?>
...
</body>
</html>
```

Modulariseren

De HTML code wordt exact daar geplaatst waar u de include opdracht geeft.

We werken deze methodiek aan het einde van dit onderdeel op een andere manier uit, namelijk met (nested) templates, maar eerst kijken we naar de situatie waarin het in te sluiten bestand PHP code bevat.

7.2.5 PHP code insluiten

U kunt op dezelfde manier als hierboven ook PHP code insluiten, bijvoorbeeld de code die nodig is om verbinding te maken met de database. Deze is immers voor elke pagina in de applicatie hetzelfde. U kunt de benodigde code eenvoudig in een eigen bestand `db_connect.inc.php` zetten en dan insluiten:

U moet nu echter wel **goed op** de volgende twee zaken **letten**:

1. De in te sluiten **PHP code** moet **tussen script delimiters** staan.
2. De **extensie** van het in te sluiten bestand moet vanwege de veiligheid bij voorkeur **.php** zijn. Voor de werking op zich is het echter niet noodzakelijk en zou u ook andere extensies kunnen gebruiken.

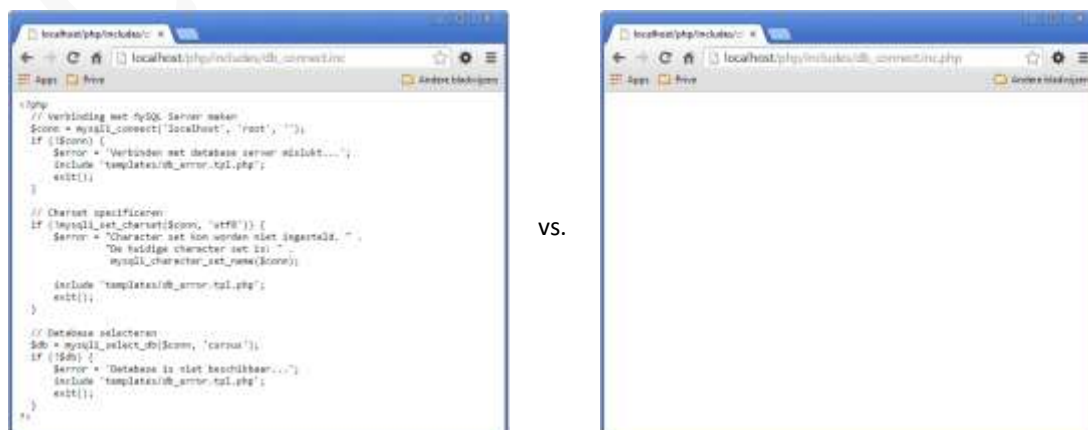
7.2.5.1 Script delimiters

De reden dat PHP code tussen script delimiters moet staan is omdat wanneer de PHP scripting engine een include voor een bestand uit een lokale directory tegenkomt, de engine zal handelen of het script werd afgesloten. Het verlaat dan de PHP modus om over te gaan naar HTML modus. De inhoud van het bestand wordt vervolgens letterlijk in de pagina gekopieerd en vanaf het begin geëvalueerd. Tekst en HTML worden ongewijzigd doorgegeven, maar eventuele **PHP code** in het include bestand **moet** dus **in een eigen PHP codeblok staan**, dus tussen `<?php` en `?>`. Alleen dan wordt het geïnterpreteerd. **Zou u de script delimiters vergeten dan wordt uw code zichtbaar in de browser getoond!**

7.2.5.2 Extensie

U kunt uw includes **in principe elke extensie** geven die u wilt. Voor includes wordt nogal eens de extensie `.inc` gebruikt. Ook voor includes die PHP code bevatten, zou u die extensie kunnen gebruiken (of zelfs `.txt`) zolang de code maar tussen script delimiters staat zal ze worden ingesloten en uitgevoerd.

U kunt includes die PHP script bevatten echter beter van de extensie `.php` voorzien, bijvoorbeeld `common_inc.php` of `common.inc.php`. Mocht iemand namelijk de naam en locatie van de include kunnen achterhalen dan zou hij of zij de inhoud van de include rechtstreeks op kunnen vragen in de browser waarbij dan de inhoud alsnog zichtbaar getoond wordt! Met een `.php` extensie zal de code echter niet zichtbaar zijn, omdat de code dan eerst door de webserver verwerkt wordt. Neem het voorbeeld van een include die database connectie maakt (zoals we dit straks ook zullen maken):



db_connect.inc

db_connect.inc.php

Als het bestand geen .php extensie heeft en iemand raadt de locatie dan wordt alle code (inclusief wachtwoord) zichtbaar!

Ook met .php extensie is het dan natuurlijk wel zo prettig als het include bestand geen directe uitvoer geeft, anders wordt dat alsnog zichtbaar. Maar wanneer u er straks enkel functies in heeft staan, is dat dus sowieso geen probleem. Dan worden de functiedefinities weliswaar even in het geheugen geladen maar omdat ze niet worden aangeroepen wordt er geen code uitgevoerd. Ook kan het uitvoeren van code die niet bedoeld is om stand-alone te worden uitgevoerd onvoorspelbare resultaten opleveren. U kunt de in te sluiten documenten daarom maar beter altijd **buiten de webroot** plaatsen zodat ze **nooit rechtstreeks** vanuit een browser kunnen **worden opgevraagd**. Zie **1.12.2.1**.

Een ander alternatief is natuurlijk om de webserver simpelweg te verbieden de bestanden met een .inc extensie publiekelijk te serveren.

7.2.5.3 Exit

Merk op dat wanneer u in de PHP code van het in te sluiten bestand `exit` of `die` aanroept PHP ook op die plek met de uitvoer van het script stopt en niet meer terugkeert naar het aanroepende script!

7.2.6 PHP functies insluiten

De PHP code die we in de vorige sectie insloten, wordt meteen uitgevoerd. Verderop in dit hoofdstuk zult u leren dat u uw code ook in zogenaamde functies kunt gieten. De code wordt dan pas uitgevoerd wanneer de functie wordt aangeroepen. Functiedefinities zijn bij uitstek geschikt om in een include te zetten en in te sluiten op de pagina's waarop u de gewenste functionaliteit nodig heeft.

Wanneer u functiedefinities wilt insluiten moet u dat uiteraard doen **voordat** u een dergelijke functie aanroept. Het is goed gebruik om dergelijke includes meteen **bovenaan** uw pagina op te nemen en niet pas wanneer u ze nodig heeft. Dat maakt het voor u en voor andere programmeurs duidelijker hoe uw applicaties zijn opgebouwd:

```
<?php
// sluit bibliotheken met functiedefinities in
include 'includes/functions.common.inc.php';
include 'includes/functions.db.inc.php';
?>
<!DOCTYPE html>
<html>
...
<?php
// code waarin de nu beschikbare functies
// aangeroepen worden...
?>

</html>
```

Alle andere includes kunt u uiteraard gewoon plaatsen op die plekken waar u ze nodig heeft.

De in de ingesloten bestanden gedefinieerde functies (variabelen en constanten) zijn meteen voor de pagina beschikbaar! Dit is ontzettend makkelijk en handig.

Stel, u wilt een functie binnen meerdere pagina's gebruiken. In plaats van de code te kopiëren en in te plakken kunt u nu de functie in één bestand plaatsen en deze importeren wanneer u hem wilt gebruiken. Dit wordt heel veel toegepast om applicaties te modulariseren en zo beter te kunnen onderhouden. Denk maar even hoe handig het is om op deze manier een functie te hebben die voor

Modulariseren

een gegeven prijs van een product de prijs inclusief BTW uitrekent. Als nu het BTW tarief verandert hoeft u dat maar op één plek aan te passen...

Functies worden besproken in 1.3, maar het BTW voorbeeld kan ook met alleen een constante duidelijk worden gemaakt:

In een `config.inc.php` bestand heeft u het BTW tarief in een constante gedefinieerd:

```
<?php
// dit is config.inc.php
define('BTW', 1.19);
...
?>
```

Let op: Let er heel goed op dat na de `?>` in het in te sluiten script **geen karakters** meer staan. Ook **geen witruimte** als spaties, returns e.d. want deze worden ook ingesloten! En deze gelden dan als uitvoer die bij alle aanpassing van de HTTP headers (header, cookies, sessies) problemen kunnen geven! U kunt er overigens in PHP only scripts ook voor kiezen om de end delimiter helemaal weg te laten...

U kunt de BTW constante gebruiken op elke pagina waarop u het configuratiebestand heeft ingesloten:

```
<?php
// sluit config.inc.php in
include 'includes/config.inc.php';
...
$bedrag_excl = 100;
$bedrag_incl = $bedrag_excl * BTW;
?>
<!DOCTYPE html>
<html>
...
De prijs is
<?php echo $bedrag_excl; ?> (excl. BTW) en
<?php echo $bedrag_incl; ?> (incl. BTW).
...
</html>
```

Het is overigens gebruikelijk om dergelijke configuratiebestanden en functiebibliotheken op elke pagina in te sluiten, ook als u misschien niet op elke pagina alles nodig heeft.

7.2.7 Variabele includes

U kunt voor de opbouw van de bestandsnaam ook variabelen gebruiken. Zolang het resultaat maar een geldige bestandsnaam oplevert.

Stel u wilt op basis van de waarde van `$_GET['lang']` een bestand met vertalingen voor teksten insluiten. Is de gekozen taal Nederlands (`?lang=nl`) dan wilt u `nl.inc.php` insluiten:

```
// dit is nl.inc.php
define('PRODUCTS', 'producten');
define('SERVICES', 'diensten');
...
```

Maar is er gekozen voor Engels (`?lang=en`) dan moet `en.inc.php` worden ingesloten:

```
// dit is en.inc.php
define('PRODUCTS', 'products');
```

```
define('SERVICES', 'services');
...
```

Het idee is dat u op de pagina de constanten gebruikt en dat ze afhankelijk van de gekozen taal de juiste vertaling bevatten...

Let op: U moet in dergelijke situaties wel erg goed oppassen dat een eindgebruiker niet ongewenste bestandsnamen kan opgeven! Als een hacker weet dat de waarde `nl` gebruikt wordt om een bestandsnaam op te bouwen dan kan hij die waarde wellicht zodanig aanpassen dat ook heel andere bestanden kunnen worden ingelezen...

De oplossing bestaat uit het aanmaken van een **whitelist** en controleren of de gekozen waarde daarin zit:

```
$lang = isset($_GET['lang']) ? $_GET['lang'] : 'nl';
$languages = array('nl', 'en', 'de', 'fr', 'it', 'es');

if (in_array($lang, $languages)) {
    include $lang . 'inc.php';
} else {
    include 'nl.inc.php';
}
```

U leert meer over het belang van validatie in Hoofdstuk 4. In hoofdstuk 5 Cookies leert u hoe u een dergelijke instelling kan vasthouden.

7.2.8 (Geen) Include Parameters

Het is **niet mogelijk** om bij een `include` parameters mee te geven en die uit te lezen in het ingesloten bestand om er vervolgens keuzes op te baseren:

```
// Let op: dit werkt niet!
include './bestand.php?mode=expert';
```

En dan in `bestand.php` iets proberen als dit:

```
// Dit is bestand.php
if ($_GET['mode'] == 'expert') {
    // Doe iets geavanceerds
}
```

Dit werkt om twee redenen niet. Allereerst lukt het al niet om het bestand in te sluiten. PHP gaat nu namelijk op zoek naar een bestand dat letterlijk `bestand.php?mode=expert` heet en dat wordt niet gevonden. Ten tweede is de wijze waarop het bestand wordt opgevraagd geen GET request. Het bestand wordt aan de serverkant ingesloten door PHP. Dus ook als het al zou lukken om een parameter mee te sturen, dan had u hem nog niet middels `$_GET` kunnen benaderen.

Het is wel mogelijk om met globale variabelen of constanten de werking van de in te sluiten code aan te passen:

```
// Zo kan het eventueel wel
$mode = 'expert';
include 'bestand.php';
```

U kunt nu in het `bestand.php` het volgende doen:

```
// Dit is bestand.php
if ($mode == 'expert') {
```

```
} // Doe iets geavanceerds
```

Het op deze manier doorgeven van globale variabelen is echter niet heel netjes... U kunt beter met functies en argumenten werken. Zie 1.3.

7.2.9 require()

Behalve `include()` is het sinds PHP4 ook mogelijk `require()` te gebruiken. Het grote verschil tussen de twee is dat bij `require()`, zoals de naam al aangeeft, het in te sluiten bestand **vereist** is voor de verdere verwerking van het script.

Wanneer bij `include()` een bestand niet kan worden ingesloten dan blijft het bij een waarschuwing waarna de rest van het script gewoon verder wordt afgehandeld:

```
<?php
include './niet_te_vinden.php';
echo 'dit ziet u wel omdat het script doorgaat...';
?>
```

Dit geeft de volgende waarschuwingen, maar het script gaat wel verder:

Warning: include(niet_te_vinden.php) [[function.include.html](#)]: failed to open stream: No such file or directory in **D:\test.php** on line 2

Warning: include() [[function.include.html](#)]: Failed opening 'niet_te_vinden.php' for inclusion (include_path='.;c:\php\includes;D:\Sites_Includes') in **D:\test.php** on line 2
dit ziet u wel omdat het script doorgaat...

Maar als bij `require()` het bestand niet ingesloten kan worden:

```
<?php
require './niet_te_vinden.php';
echo 'dit ziet u niet omdat het script stopt...';
?>
```

dan **stopt** de uitvoer van het script met een **fatal error**:

Warning: require(niet_te_vinden.php) [[function.require.html](#)]: failed to open stream: No such file or directory in **D:\test.php** on line 2

Fatal error: require() [[function.require.html](#)]: Failed opening required 'niet_te_vinden.php' (include_path='.;c:\php\includes;D:\Sites_Includes') in **D:\test.php** on line 2

Bij het insluiten van **functiedefinities** met een `include()` zal bij het ontbreken van het bestand de verwerking dan ook gewoon doorgaan. Omdat er in de code dan hoogstwaarschijnlijk functieaanroepen gedaan worden voor de ontbrekende functies zal dat op zijn minst vele foutmeldingen geven, maar wellicht ook tot onvoorspelbare resultaten leiden...

Daarom is het eigenlijk bij het insluiten van functies altijd verstandig om dat met `require()` te doen of met `require_once()` dat hierna wordt besproken.

Let er op dat u ook foutmeldingen zult krijgen wanneer u functies declareert in een include en deze include op zijn beurt ook weer gebruik maakt van dezelfde of een andere include waarin functies met dezelfde namen voorkomen!

Om dit te voorkomen kunt u gebruik maken van `include_once()` of `require_once()`.

7.2.10 include_once() en require_once()

Deze statements werken op een vergelijkbare manier als `include()` en `require()` maar met één belangrijk verschil: ze controleren of het bestand al een keer eerder werd ingesloten. Als dat het geval is doen ze niets.

Als u met `include_once()` en `require_once()` een bestand insluit dan weet u zeker dat het ook maar één keer ingelezen wordt, waardoor de code ook maar één keer geïnterpreteerd wordt. Dit voorkomt foutmeldingen die anders zouden optreden bij het meer dan eens insluiten van dezelfde functiedefinities.

```
<?php
require_once 'a.php'; // sluit a.php in...
require_once 'a.php'; // nu niet nog een keer...
?>
```

U kunt deze statements bijvoorbeeld goed gebruiken bij het laden van functiebibliotheken die zelf ook weer andere bestanden insluiten. De naam `require_once()` maakt daarbij ook goed duidelijk wat er moet gebeuren: verplicht eenmalig insluiten.

Aan de andere kant is het wel zo dat als er per ongeluk ergens wel dezelfde bestanden worden ingesloten (en er dus iets vreemds aan de hand is met uw code) dat u daar dan niet meer op wordt geattendeerd... Bovendien kost de controle extra tijd. Niet schokkend veel, maar als u het eigenlijk toch al niet nodig heeft omdat u uw code zelf al zorgvuldig heeft georganiseerd...

Let op: als u na `require_once()` weer `require()` gebruikt om hetzelfde bestand in te sluiten dan wordt dit bestand **wel** gewoon voor de tweede keer ingesloten:

```
<?php
require_once 'a.php'; // sluit a.php in...
require 'a.php';      // en nog een keer...
?>
```

Overigens onthouden `include_once()` en `require_once()` wel dezelfde bestandslijst, dus ook als u ze door elkaar zou gebruiken kan een bestand niet meer dan eens worden ingesloten...

7.2.11 Waarschuwingen onderdrukken

Als u niet de waarschuwingen van PHP wilt laten zien, maar uw eigen meldingen wanneer het in te sluiten bestand niet kan worden gevonden, dan kan dit op de volgende manier:

```
if (!@include('./includes/common.inc.php')) {
    die('kan verplicht bestand niet insluiten');
}
```

of iets korter (op één regel):

```
include('./includes/common.inc.php')
or die('kan verplicht bestand niet insluiten');
```

Let op: dit kan niet met `require()` omdat door het optreden van de fatal error het `die()` statement niet zal worden uitgevoerd. U kunt dit oplossen met de functie `file_exists()` die de `require()` alleen uitvoert als het bestand bestaat:

```
if (file_exists('./includes/common.inc.php')) {
    require('./includes/common.inc.php');
} else {
```

```
die('kan verplicht bestand niet insluiten');  
}
```

Dit kan ook weer iets korter genoteerd worden (op één regel):

```
file_exists('./includes/common.inc.php') ?  
    require('./includes/common.inc.php') :  
    die('kan verplicht bestand niet insluiten');
```

Maar u zult dergelijke code waarschijnlijk dan toch in een zelfgemaakte functie `require_file($file)` of iets dergelijks willen opnemen, dus dan is beknoptheid niet echt een issue. Zie 1.3 voor het zelf definiëren van functies.

7.2.12 Beveiliging tegen los opvragen

Wanneer u PHP code in includes heeft staan, dan wilt u niet dat eindgebruikers deze includes los op kunnen vragen. U kunt dit op verschillende manieren voorkomen:

- Plaats de includes buiten de webroot
- Gebruik een `.htaccess` bestand
- Beperk lees rechten op de folder
- Schrijf code die los opvragen voorkomt
- Geef de bestanden een `.php` extensie

Het mooiste is natuurlijk een combinatie van deze oplossingen. We bespreken ze hieronder wat uitgebreider.

7.2.12.1 Includes buiten de webroot

Een zeer afdoende manier om te voorkomen dat bestanden via het web kunnen worden opgevraagd, is door deze buiten de webroot te plaatsen! Eindgebruikers kunnen er dan niet bij, maar PHP wel!

Bij de meeste providers komt u via FTP binnen op een niveau hoger dan de webroot, bijvoorbeeld zo:

```
/logs  
/stats  
/www
```

Waarbij `www` dan de webroot is, oftewel de folder waarin de bestanden staan die uw eindgebruikers via uw domein kunnen opvragen. Andere veel voorkomende namen voor deze map zijn `public` of `htdocs`. Als u op dezelfde hoogte een map `includes` maakt. Dus zo:

```
/includes  
/logs  
/stats  
/www
```

Dan kunt u er vanuit uw code wel naar verwijzen, maar dan kunnen mensen er via het web vanuit de browser nooit bij...

De verwijzing vanuit een PHP bestand in de webroot, bijvoorbeeld `index.php`, naar een bestand `common.php` kan er dan zo uitzien:

```
include '../includes/common.inc.php';
```

Het is ook mogelijk het volledige pad op te geven, bijvoorbeeld iets als:

```
include '/home/example.org/includes/common.inc.php';
```

U vindt het volledige pad eenvoudig via `phpinfo()` of met de opdracht:

```
echo dirname(__FILE__);
```

Als u het beheer over de webserver heeft dan kunt u de includes ook heel ergens anders in het bestandsysteem plaatsen. Bijvoorbeeld op een Windows systeem:

```
include 'D:/sites/includes/common.inc.php';
```

Tijdens de cursus zullen we voor het gemak echter de includes map gewoon in de webroot zetten, zodat we er makkelijk vanuit onze editor bij kunnen komen.

Veel open source producten installeren overigens, noodgedwongen, hun includes ook in de webroot...

7.2.12.2 .htaccess bestand

Wanneer u de map includes toch onder de webroot moet of wilt houden dan kunt u het los opvragen van de includes voorkomen door in de map een `.htaccess` bestand te plaatsen met de volgende inhoud:

```
deny from all
```

Dit zorgt er allereerst voor dat de folder niet te zien is in de directory listing als deze onverhoopt aan mocht staan. De directory listing wilt u sowieso uit hebben staan voor alles onder de webroot... (zie verder ook het hoofdstuk over veiligheid).

Als iemand nu in de map includes zou willen kijken door dit in de URL in te typen dan levert dat deze melding op:

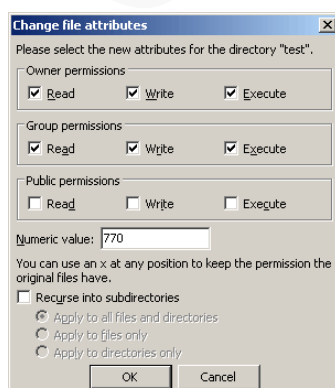
Forbidden

You don't have permission to access /includes/common.inc.php on this server.

U kunt de melding eventueel aanpassen door een eigen 403 error page te maken...

7.2.12.3 Beperk rechten

U kunt de folder ook onleesbaar maken vanaf het internet door de toegangsrechten (access permissions) te beperken tot 770. Dat komt neer op het uitzetten van alle publieke permissies. U kunt dit via de commandline doen maar ook via FTP met een programma als FileZilla:



7.2.12.4 Code tegen los opvragen

U kunt in de in te sluiten file vragen om een variabele of constante die in de parent file gedefinieerd is. Als deze niet aanwezig is beëindigt u de verwerking. Dit werkt als volgt:

In het bestand waar u de include wilt insluiten, bijvoorbeeld `index.php`, definieert u voorafgaand aan het insluiten een constante als deze:

```
define('INCLUDE', true);
include('./includes/common.inc.php');
```

In de include file `common.inc.php` controleert u dan of deze constante is gedefinieerd. Wanneer het bestand rechtstreeks wordt opgevraagd is dat niet het geval en dan beëindigt u de uitvoer:

```
if (!defined('INCLUDE')) {
    die('Deze include file kunt u niet rechtstreeks opvragen!');
}
```

De controle kan nog iets korter genoteerd worden:

```
defined('INCLUDE') or die('Geen toegang...');
```

Wanneer u niet in elke pagina die includes insluit vooraf de constante wilt definiëren dan kunt u ook iets als dit doen in de include zelf:

```
if (basename($_SERVER['PHP_SELF']) == 'common.inc.php') {
    die('Geen toegang...');
}
```

7.2.12.5 .php extensie

Hoewel niet zozeer een beveiliging tegen direct opvragen, voorkomt het gebruik van de `.php` extensie zoals al eerder gezegd wel dat de PHP code getoond wordt. De code wordt nu namelijk geïnterpreteerd en alleen de eventuele uitvoer is zichtbaar. Bij een functie bibliotheek worden dan weliswaar de functies in het geheugen van de webserver geladen, maar dit leidt verder niet tot uitvoer: de eindgebruiker krijgt een blanco pagina te zien...

7.2.13 Auto_prepend_file & auto_append_file

U zou uw (ontwikkel)leven zelfs nog eenvoudiger kunnen maken door gebruik te maken van de `auto_prepend_file` en `auto_append_file` configuratie instellingen in `php.ini`. Hiermee kunt u aangeven dat PHP altijd een bepaald bestand aan het begin / einde van uw script moet insluiten, voordat het aan de verwerking van het hoofdbestand begint. Het werkt hetzelfde als een `require()`, wat betekent dat de instellingen uit `include_path` gebruikt worden en dat als het bestand niet gevonden wordt er een fatal error optreedt.

Als u op deze manier automatisch `begin_page.php` en `end_page.php` laat insluiten dan hoeft u deze bestanden niet langer op elke pagina in te sluiten.

U moet zich echter wel realiseren dat de bestanden nu echt **altijd** bij **elk** PHP script worden ingesloten, wat uiteindelijk misschien toch minder handig is dan gedacht. Vanaf PHP 4.2.3 kunt u het gelukkig ook **per directory** instellen door een `.htaccess` bestandje met de volgende inhoud op te nemen:

```
php_value auto_prepend_file "includes/begin_page.php"
php_value auto_append_file  "includes/end_page.php"
```


Met de speciale waarde "none" kunt u het automatisch insluiten (tijdelijk) ongedaan maken, maar dan alleen in de .htaccess. Er is geen mogelijkheid om op page niveau met `ini_set` een uitzondering te maken.

Nog even over de volgorde van insluiten: eerst wordt gecontroleerd of het hoofdbestand bestaat, dan wordt het prepend bestand ingesloten en daarna wordt het hoofdbestand verwerkt. Dit houdt in dat u in de prepend file wel acties kunt uitvoeren die de werking van het hoofdbestand veranderen maar dat u er geen andere bestanden mee kunt laten uitvoeren dan het hoofdbestand...

7.2.14 PRAKTIJK

Met de nieuwe kennis over includes kunnen we de code in onze doorlopende oefening grondig reorganiseren!

7.2.14.1 include database connectie

Iets wat op vrijwel elke pagina van uw applicatie herhaald wordt is de PHP code voor het maken van de database verbinding. De uit te voeren query zal weliswaar steeds verschillen maar de stappen voor het maken van de verbinding en het selecteren van de database zijn in principe steeds precies hetzelfde. Om steeds kopiëren en plakken te voorkomen en, belangrijker nog, om in een later stadium snel op een centrale plek wijzigingen door te kunnen voeren (zoals bijvoorbeeld het veranderen van het wachtwoord) is het slim om de benodigde code in een eigen bestand `db_connect.inc.php` te zetten en dat op elke pagina waar u database verbinding nodig heeft in te sluiten.

```
<?php // dit is db_connect.inc.php

// Verbinding met MySQL Server maken
$conn = mysqli_connect('localhost', 'root', '')
    or die('kan geen verbinding met database maken...');

// Charset specificeren
mysqli_set_charset($conn, 'utf8')
or die('charset kan niet correct worden ingesteld...' .
    'De huidige character set is: ' .
    mysqli_character_set_name($conn) );

// Database selecteren
mysqli_select_db($conn, 'cursus')
or die('database niet beschikbaar...');
?>
```

Insluiten is zo eenvoudig als:

```
// dit is medewerkers_tonen.php

// sluit code in om database verbinding te realiseren
include 'includes/db_connect.inc.php';

// stel SQL SELECT statement op
...
// voer query uit
...
//etc...
```

Na insluiten heeft u verbinding. U moet natuurlijk nog wel de query opstellen, uitvoeren en door de resultset/dataset heen lopen...

Oefening	Zet de code voor de database connectie in een apart bestand <code>db_connect.inc.php</code> dat u in een map <code>includes</code> plaatst. Reorganiseer al uw pagina's zo dat ze de include gebruiken voor de database connectie.

7.2.14.2 include configuratie bestand

Als u op dit moment het wachtwoord wilt wijzigen dan hoeft dit enkel nog maar in de include, maar u (of iemand anders) moet daarvoor wel de code in. Het zou makkelijker zijn als dit soort wijzigingen in een speciaal daarvoor bedoeld configuratiebestand te maken zijn. Hierin staan dan niet alleen de instellingen voor de database, maar ook andere die u binnen uw applicatie nodig heeft. Denk aan het eerder genoemde BTW percentage, email adressen, etc. Eigenlijk alles wat u door de applicatie heen gebruikt en waarvan het makkelijk zou zijn als u ze op een centrale plek zou kunnen wijzigen.

In de meest eenvoudige vorm zou het er zo uit kunnen zien:

```
<?php // dit is config.inc.php

// database constanten
define('DB_SRV', 'localhost');
define('DB_USR', 'root');
define('DB_PWD', 'geheim');
define('DB_NAME', 'cursus');

// overige constanten
define('COMPANY_NAME', 'CompuFirm');
define('COMPANY_EMAIL', 'info@compufirm.nl');

// etc...
?>
```

Constanten hebben als voordeel dat ze **niet per ongeluk te overschrijven** zijn én dat u ze **binnen functies kunt uitlezen**, zoals we later zullen zien.

U moet nu natuurlijk wel zorgen dat de constanten gebruikt worden in `db_connect.inc.php`:

```
<?php // dit is db_connect.inc.php

// Verbinding met MySQL Server maken
$conn = mysqli_connect(DB_SRV, DB_USR, DB_PWD)
...

// Database selecteren
mysqli_select_db($conn, DB_NAME)
...
```

En u moet er uiteraard ook voor zorgen dat u nu het configuratie bestand insluit vóór dat u de database verbinding maakt:

```
// dit is medewerkers_tonen.php

// sluit configuratie bestand in
include 'includes/config.inc.php';

// sluit code in om database verbinding te realiseren
// deze maakt nu gebruik van de constanten uit config.inc.php
include 'includes/db_connect.inc.php';
...
```

Tip: U zou de foutmeldingen ook als constanten in de config op kunnen nemen.

Het is niet handig om de config in de database connectie in te sluiten omdat u ook de beschikking over de andere constanten wilt hebben als u geen database verbinding wilt maken. In een later stadium zullen we de config wel in een algemene functiebibliotheek insluiten.

U kunt het configuratie bestand uitbreiden met verschillende settings voor de development en de live settings. En een DEBUG constante introduceren om in development mode andere uitvoer te geven dan in productie. Bovendien kunt er ook mooi instellingen voor het tonen van fouten, het error reporting level, het starten van de output buffer en het starten van sessies (waarover later meer) in opnemen:

```
<?php
// config.inc.php

//*****
// CONSTANTS & VARS
//*****

define('DEBUG', true);

if ($_SERVER['SERVER_NAME'] == 'localhost') {
    define('DOMAIN', 'http://localhost');
    define('APPDIR', '/sitenaam');
} else {
    define('DOMAIN', 'http://www.sitenaam.nl');
    define('APPDIR', '');
}
define('HTDOCS', DOMAIN.APPDIR.'/');

define('SELF', $_SERVER['PHP_SELF']); // is wat korter!
define('N', "\n"); // newline character

//*****
// DATABASE
//*****

if ($_SERVER['SERVER_NAME'] == 'localhost') {
    define('HOST', 'localhost');
    define('USR', 'root');
    define('PWD', 'geheim');
    define('DB', 'naam');
} else {
    define('HOST', '100.100.100.100');
    define('USR', 'gebruikersnaam');
    define('PWD', 'geheimer');
    define('DB', 'naam');
}

//*****
// OUTPUT
//*****

ob_start(); // outputbuffering aan
error_reporting(E_ALL); // maximaal waarschuwningsniveau

if (DEBUG) {
    // tijdens debuggen andere instellingen...
    // en/of hulpfuncties insluiten
}

?>
```

Na het bespreken van de functies ziet u een voorbeeld hoe u `DEBUG` kunt gebruiken..

Oefening	<i>Maak een bestand <code>config.inc.php</code> aan dat u in de map <code>includes</code> plaatst. Definieer constanten voor de database gegevens. Herschrijf <code>db_connect.inc.php</code> zo dat deze de constanten gebruikt. Reorganiseer al uw pagina's zo dat ze eerst het configuratie bestand insluiten.</i>

7.2.14.3 include header en footer

Iets anders wat u ook graag op een centrale plek wilt organiseren is de page layout. Deze is als het goed is namelijk consistent over de pagina's heen. En u wilt niet dat u wijzigingen in de structuur

Modulariseren

steeds op elke pagina apart moet doorvoeren. Wijzigingen in de presentatie doet u al op een centrale plek, namelijk in uw externe stylesheet(s).

U zou een standaard (HTML) pagina kunnen nemen en deze splitsen in een aantal functionele delen en deze in aparte files zetten (bijvoorbeeld `header.php` en `footer.php`) om deze vervolgens met evenveel includes in te sluiten op de plek waar u ze wilt hebben. Denk eraan dat deze bestanden dus geen eigen HTML-skelet mogen hebben!

In de eenvoudige vorm zou `header.php` er zo uit kunnen zien:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title><?= $title ?></title>
</head>

<body>

  <header>
    <h1><?= COMPANY_NAME ?></h1>
  </header>

  <nav> ... </nav>

  <main>
    <article>
      <h1><?= $title ?></h1>
      <!-- begin page specific content -->
```

U ziet dat er gebruik gemaakt wordt van een variabele `$title`, dit om te voorkomen dat elke pagina dezelfde, vaste, titel zou krijgen. Bovendien is er hier voor gekozen de titel in een Heading 1 te herhalen. In de header wordt de bedrijfsnaam getoond met behulp van de constante `COMPANY_NAME` uit `config.inc.php`.

Let op: Denk er aan dat u de **variabele** voor de **titel** wel moet initialiseren **voordat** u de header insluit! Anders krijgt u zoiets te zien:

Notice: Undefined variable: title in C:\xampp\htdocs\cursus\templates\header.tpl.php on line 10

U zou ook in de header template een **check** kunnen maken **of** er wel een **titel gezet is**. Zo niet dan valt u terug op een **default waarde** voor titel, bijvoorbeeld `COMPANY_NAME`. U zou dat heel compact zo kunnen noteren:

```
<title><?= isset($title) ? $title : COMPANY_NAME ?></title>
```

De `footer.php` zou er bijvoorbeeld zo uit kunnen zien:

```
<!-- end page specific content -->
</article>
<aside>Meer informatie</aside>
</main>

<footer>&copy; 2011-<?= date('Y') ?> <?= COMPANY_NAME ?></footer>
</body>
</html>
```

Let op het automatisch ingevoegd copyright jaartal en het opnieuw inzetten van de constante voor de bedrijfsnaam!

U kunt alle PHP pagina's in uw site nu op de volgende manier opbouwen:

```
<?php
// sluit configuratie bestand in
include 'includes/config.inc.php';

// maak database verbinding
include 'includes/db_connect.inc.php';

// haal paginaspecifieke data op uit de database
...

// zet titel voordat u de header insluit
$title = 'Welkom op deze pagina!';

// sluit header in
include 'includes/header.php';
?>
...
<p>pagina specifieke content ...</p>
...
<?php include 'includes/footer.php'; ?>
```

Na opvragen van deze pagina wordt dan een complete HTML pagina neergezet in de browser. Als u vervolgens de header of footer iets aanpast wordt dat automatisch in alle pagina's op uw site die de headers en footers insluiten doorgevoerd!

Wilt u nog flexibeler zijn met pagina specifieke metatags, met CSS, met JavaScripts etc. dan zult u ook hiervoor extra variabelen moeten aanmaken. U begrijpt het al, dit kan al gauw vrij chaotisch worden. Zeker in combinatie met de include voor de database connectie (en eventueel een voor het sluiten daarvan) en de globale variabelen en constanten die binnen de includes gebruikt moeten worden.

Maar dit is in ieder geval wel een snelle manier om code beter herbruikbaar te maken. En er zijn nogal wat ontwikkelaars die al hun pagina's beginnen met een `header.php` en afsluiten met een `footer.php`. In de header wordt dan vaak ook nog de database verbinding gemaakt. In de `footer.php` wordt dan alles weer netjes afgesloten.

7.2.14.4 Templates

In plaats van de header steeds aan het begin en de footer steeds aan het einde van elke pagina in te moeten sluiten, en per script te bepalen wat de pagina specifieke content moet zijn (inclusief de benodigde markup wat natuurlijk sowieso een slecht idee is!) kunt u beter een **template** aan het einde van het **PHP script in sluiten**. Dit geeft u een veel **betere scheiding tussen logica, data en presentatie**.

En het geeft u ook de **vrijheid** om makkelijk **andere uitvoer** voor uw script te kiezen door een andere template, of view zoals u wilt, in te sluiten. Of misschien zelfs helemaal geen view maar **JSON** of XML uitvoer.

Bovendien sluit dit ook beter aan bij ons streven naar een **MVC manier van werken**. Ons script is dan de Controller die de data uit het Model (de database) aan een View (de template) geeft.

Uw code ziet er dan zo uit:

```
<?php
// sluit configuratie bestand in
```

```
include 'includes/config.inc.php';

// maak database verbinding
include 'includes/db_connect.inc.php';

// haal paginaspecifieke data op uit de database
...

// zet titel voordat u de header insluit
$title = 'Medewerkers overzicht';

// sluit template in
include 'templates/medewerkers_list.tpl.php';
```

De template `medewerkers_list.tpl.php` in de map `/templates` (en dus niet `/includes`!) zou een complete pagina kunnen zijn:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title><?= $title ?></title>
</head>

<body>

  <ol>
    <?php while ($row = mysqli_fetch_assoc($result)): ?>
      <li><?=$row['voornaam']?> (<?=$row['achternaam']?>)</li>
    <?php endwhile; ?>
  </ol>

</body>
</html>
```

Wilt u de medewerkers liever in een tabel zien? Dan sluit u de template `medewerkers_in_tabel.tpl.php` in.

7.2.14.5 Nested Templates

Als u *site-wide* wijzigingen door wilt voeren in de structuur van de site dan is het natuurlijk niet handig als u alle afzonderlijke templates moet aanpassen... Voor de benodigde consistentie in layout zal elke template dan ook een header en een footer insluiten. Met andere woorden we gaan gebruik maken van **nested templates**! Het idee is als volgt:

De template `medewerkers_list.tpl.php` in zou u dan zo op kunnen bouwen:

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

<ol>
  <?php while ($row = mysqli_fetch_assoc($result)): ?>
    <li><?=$row['voornaam']?> (<?=$row['achternaam']?>)</li>
  <?php endwhile; ?>
</ol>

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

De template sluit een header en footer template in. U kunt hiervoor exacte dezelfde code gebruiken die u al in eerdere opdracht had gerealiseerd. Tussen deze includes toont de template de pagina specifieke data, in dit geval doorloopt het de aangeboden 2-dim array `$medewerkers` in een ordered list.

Als u de data echter liever in een tabel ziet dan sluit u, zoals gezegd, de template `medewerkers_table.tpl.php` in (die op zijn beurt ook weer dezelfde header en footer insluit). Deze template toont de data in een HTML tabel. Maar u zou net zo makkelijk een template kunnen kiezen die de `$medewerkers` als XML of JSON toont. U moet er dan overigens wel aan denken om het MIME type aan te passen...

U heeft nu een duidelijke scheiding tussen de data en de presentatie gemaakt. Het maakt ons script niet uit wat voor een template gekozen wordt om de data in te tonen en het maakt de template niet uit waar de data vandaan komt. Of het nu uit MySQL, PostgreSQL, SQLite, een CSV of XML bestand komt dat boeit niet. Zolang het maar in de vorm van een 2-dimensionale array wordt aangeleverd...

Templates kunnen wel PHP code gebruiken (in de lijn van *PHP as a templating language*), maar u zou zich moeten beperken tot de volgende subset:

- Gebruik alleen **eenvoudige expressies zonder side-effect**.
Ga vooral geen variabelen in de template declareren!
- Gebruik **alleen if, else, foreach, and include statements**.
- Gebruik alleen de **template syntax** (alternatieve syntax) for controle structuren
- Gebruik in principe maar **1 statement per PHP code block**.
- Gebruik zoveel mogelijk de **short echo** `<?= $var ?>`.
Met spaties maar zonder puntkomma op het eind.
- Maak de code **zo leesbaar mogelijk** wat betreft indents. Maak u niet druk om de gegenereerd witruimte in de uiteindelijke HTML broncode.

Door deze *separation of concerns* beperken we vrijwillig de kracht van PHP in de templates. In de templates mag namelijk niets over de toepassing beslist, berekend of opgevraagd worden. Dat is de taak van de page controller om dit in samenwerking met de database te doen... De templates zijn alleen bedoeld voor de visuele presentatie, de grafische interface om de data op een logische manier te tonen.

Dit is het basis idee waar we tijdens de rest van deze cursus mee zullen werken. In het volgende onderdeel over functies zullen we echter nog wel wat verbeteringen aanbrengen!

Oefening	<i>Bouw uw applicatie om naar bovenstaand voorbeeld. Maak een map templates aan met daarin een header.tpl.php en een footer.tpl.php. Maak 2 templates aan waarmee u de medewerkers in een lijst dan wel in een tabel kan tonen.</i>
Verdieping	<i>Maak een template aan waarmee u de medewerkers in XML formaat kan aanbieden .</i>

7.2.14.6 Beveiliging

Als een gebruiker templates rechte reeks uit de template directory kan opvragen dan kan dit tot vreemde pagina's leiden. Maar mogelijk dat u er zelf ook intrapt als u een template aan het bouwen bent en deze rechtstreeks in de browser lanceert (in plaats van het control script). F12 in Dreamweaver bijvoorbeeld? Om dit te voorkomen kunt u zoals eerder beschreven een constante in uw config definiëren, bijvoorbeeld:

```
// in config.inc.php
define('TEMPLATE', true);
```

Vervolgens controleert u bovenaan elke template of deze constante inderdaad is gedefinieerd:

```
<?php
if (!defined(TEMPLATE)) die('Templates can not be accessed directly...');
?>
```

Dat zal alleen het geval zijn als de template door een controller (die op zijn beurt het configuratie bestand heeft ingesloten) werd ingesloten. Als de template rechtstreeks wordt opgevraagd verschijnt de melding. Merk op dat u de melding dus niet in het configuratiebestand kan definiëren...

Door de controle in ieder geval in `header.tpl.php` op te nemen dekt u waarschijnlijk al de meeste situaties af, omdat de meeste templates daar toch mee zullen beginnen...

Tip: een handige manier om te voorkomen dat u (of eindgebruikers) per ongeluk een template opvragen is om een `.htaccess` file met `deny from all` in de templates folder op te nemen!

7.2.14.7 Error templates

De (nested) templates lenen zich ook bij uitstek om de eventuele foutmeldingen die onze database include zou kunnen maken in een complete pagina in dezelfde look-and-feel als de rest van de site te tonen. Als er nu iets mis gaat ziet u alleen de kale melding. Er wordt geen HTML pagina opgebouwd omdat het script gelijk stopt:

```
$conn = mysqli_connect('localhost', 'root', '');
or die('verbinding met database server mislukt...');
```

Als alternatief kunt u dan ook de volgende code gebruiken:

```
$conn = mysqli_connect('localhost', 'root', '');

if (! $conn) {
    echo 'verbinding met database server mislukt...';
}
```

Of zelfs in één keer:

```
if (! $conn = mysqli_connect('localhost', 'root', '')){
    echo 'verbinding met database server mislukt...';
}
```

Het **script** gaat dan weliswaar **verder**, maar nu moet u alle **vervolgfouten** (die zonder verbinding zeker zullen plaats vinden) gaan **afhandelen**...

Een **betere oplossing** is om het tonen van **fouten** te **onderdrukken** (standaard bij veel providers of anders met een `.htaccess` bestand) en een speciale **error template** in te **sluiten**:

```
$conn = mysqli_connect('localhost', 'root', '');

if (! $conn) {
    $title = 'Database Error';
    $error = 'verbinding met database server mislukt...';
    include 'templates/db_error.tpl.php';
    exit();
}
```

De error template bevat voor de consistentie dan de volgende opbouw:

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



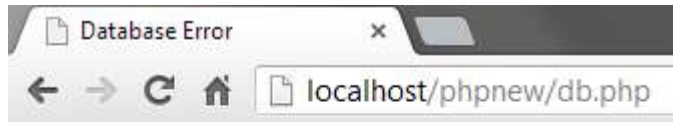
```

<p class="error">
  <?php echo $error; ?>
</p>

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

```

Het aanroepen van `exit()` na het insluiten van de template is noodzakelijk om te voorkomen dat de rest van de code in het hoofdsript alsnog wordt uitgevoerd! Uiteindelijk krijgt de eindgebruiker dan deze pagina te zien:



Database Error

verbinding met database server mislukt...

Op dezelfde manier kunt u de andere fouten afhandelen. Uw database connectie include zou er uiteindelijk zo uit kunnen zien

```

<?php // dit is db_connect.inc.php

// Verbinding met MySQL Server maken
$conn = mysqli_connect('localhost', 'root', '');
if (! $conn) {
    $title = 'Database Error';
    $error = 'Verbinden met database server mislukt...';
    include 'templates/db_error.tpl.php';
    exit();
}

// Charset specificeren
if (! mysqli_set_charset($conn, 'utf8')) {
    $title = 'Database Error';
    $error = "Character set kon worden niet ingesteld. " .
        "De huidige character set is: " .
        mysqli_character_set_name($conn);

    include 'templates/db_error.tpl.php';
    exit();
}

// Database selecteren
$db = mysqli_select_db($conn, 'cursus');
if (! $db) {
    $title = 'Database Error';
    $error = 'Database is niet beschikbaar...';
    include 'templates/db_error.tpl.php';
    exit();
}
?>

```

Er zit nog wat **herhaling** in. Dit kunnen we na het bespreken van **functies** corrigeren.

U kunt deze template uiteraard zo mooi maken als u maar wilt. Let bij het tonen van foutmeldingen in ieder geval op de volgende zaken:

Modulariseren

- Gebruik dezelfde taal als in de rest van de applicatie. Dus geen Engelse foutmeldingen op een Nederlandstalige site. Heeft u een multilingual site denk er dan aan dat de foutmeldingen ook vertaald moeten worden!
- Gebruik begrijpelijke taal zodat de gebruiker snapt wat het probleem is. Gebruik geen termen of codes die alleen u of andere programmeurs wat zeggen.

In een later hoofdstuk leert u ook hoe u de foutmeldingen aan u zelf kunt **mailen** zodat u er meteen op geattendeerd wordt wanneer het plaatsvindt en u het snel kunt verhelpen!

Oefening	<i><code>Maak een <code>db_error.tpl.php</code> aan en sluit die in op de wijze zoals hier boven werd beschreven. Test de werking!</code></i>

7.2.14.8 Twee-dimensionale data array

Na deze reorganisatie is uw code een stuk eenvoudiger te onderhouden. Er is echter nog steeds een **probleem** met de code die u tot nu toe geschreven heeft en dat is dat deze nog steeds **PHP data access logica** in de het **template** deel bevat, namelijk de `mysqli_*` functies. De template is daarom nog **sterk gekoppeld** aan de logica bovenin het script. Als u de data uit een andere database wilt halen of misschien wel uit een `.csv` of `.xml` bestand dan moet u ook de template aanpassen...

Het zou mooier zijn als u de data aan de template kon aanleveren op een standaardmanier, zodat het niet uitmaakt waar deze vandaan komt. Dit kan door middel van een **2-dimensionale array**. Eerder heeft u al geleerd dat een **array** ook **arrays** kan **bevatten**. We kunnen gebruik maken van dit principe door eerst een lege array aan te maken en deze vervolgens te vullen met de arrays die `mysqli_fetch_assoc()` aanlevert:

```
// maak een lege array aan
$rows = array();

// doorloop de resultset en vul de array met arrays
while ($row = mysqli_fetch_assoc($result)) {
    $rows[] = $row;
}
```

U kunt nu natuurlijk ook heel goed een wat **beschrijvender naamgeving** gebruiken:

```
$medewerkers = array();

while ($medewerker = mysqli_fetch_assoc($result)) {
    $medewerkers[] = $medewerker;
}
```

7.2.14.9 `mysqli_fetch_all()`

Bovenstaande manier maakt het concept goed duidelijk en dit is ook de wijze waarop het met de oude MySQL API moest worden aangepakt. De `mysqli` API maakt dit scenario wat eenvoudiger met de `mysqli_fetch_all()` functie.

Let op: `mysqli_fetch_all()` is alleen beschikbaar als de MySQLi API de Native Driver gebruikt! (standaard vanaf PHP 5.3)

U moet er wel aan denken om de `MYSQLI_ASSOC` constante mee te geven als u wilt dat de records arrays **associatief** zijn! Standaard zijn ze namelijk numeriek. En dat is lastiger uitlezen en maakt het dynamisch genereren van de tableheaders onmogelijk.

```
// zet alle records in een 2-dimensionale array
$medewerkers = mysqli_fetch_all($result, MYSQLI_ASSOC);
```

Hoe u het ook aanpakt, `$medewerkers` bevat nu alle rijen uit de resultset. Een `print_r($medewerkers)` geeft het volgende beeld:

```
Array
(
    [0] => Array
        (
            [id] => 1
            [voornaam] => Sheila
            [achternaam] => Lavanter
            [afdeling] => Verkoop
            [toestelnummer] => 1000
        )

    [1] => Array
        (
            [id] => 2
            [voornaam] => Peter
            [achternaam] => Spanje
            [afdeling] => Verkoop
            [toestelnummer] => 1000
        )

    [2] => Array
        (
            [id] => 3
            [voornaam] => Jaap
            [achternaam] => Soetendaal
            [afdeling] => Directie
            [toestelnummer] => 9999
        )

    ...
)
```

We zouden nu in principe de database verbinding kunnen sluiten en toch over de data beschikken in de template. U toont de data met een `foreach` lus. Bijvoorbeeld als een geordende lijst:

```
<ol>
    <?php foreach ($medewerkers as $medewerker) :?>
        <li><?=$medewerker['voornaam']?> <?=$medewerker['achternaam']?></li>
    <?php endforeach; ?>
</ol>
```

Maar het kan net zo makkelijk als een tabel:

```
<?php if ($medewerkers): ?>
<table>
    <tr>
        <th>ID</th>
        <th>Voornaam</th>
        <th>Achternaam</th>
        <th>Afdeling</th>
        <th>Toestelnummer</th>
    </tr>
    <?php foreach ($medewerkers as $medewerker): ?>
    <tr>
        <td><?php echo $medewerker['id']; ?></td>
        <td><?php echo $medewerker['voornaam']; ?></td>
        <td><?php echo $medewerker['achternaam']; ?></td>
        <td><?php echo $medewerker['afdeling']; ?></td>
        <td><?php echo $medewerker['toestelnummer']; ?></td>
    </tr>
    <?php endforeach; ?>
</table>
```

Modulariseren

```
<?php else: ?>
    <p class="warning">geen medewerkers gevonden...</p>
<?php endif; ?>
```

Merk op hoe logisch de naamgeving nu werkt!

In het tabel voorbeeld springt de `foreach` lus binnen de PHP codeblokken in met 4 spaties, deze zit immers binnen de `if` branch die op de aanwezigheid van medewerkers controleert. De HTML volgt zijn eigen indentatie met 2 spaties per childlevel. Er wordt geen overbodige witruimte gegenereerd, iets wat bij het list voorbeeld wel gebeurt...

U kunt ook makkelijk de array **sorteren** met `array_sort()` of **twee keer** er door heen lopen. Dat was bij de `fetch_*` in een `while` lus **niet** gelukt omdat na de eerste keer de **pointer** dan aan het **einde** van de **tabel** had gestaan. Deze had u dan eerst met `mysqli_data_seek($result,0)` weer boven aan moeten zetten...

7.2.14.10 generieke tabel

U kunt de HTML ook op deze manier weer **generiek** opbouwen. U heeft dan **geneste** `foreach` lussen nodig:

```
<?php if ($medewerkers): ?>
<table>
    <tr>
        <th>ID</th>
        <th>Voornaam</th>
        <th>Achternaam</th>
        <th>Afdeling</th>
        <th>Toestelnummer</th>
    </tr>
    <?php foreach ($medewerkers as $medewerker): ?>
        <tr>
            <?php foreach ($medewerker as $value): ?>
                <td><?php echo $value; ?></td>
            <?php endforeach; ?>
        </tr>
    <?php endforeach; ?>
</table>
<?php else: ?>
    <p class="warning">geen medewerkers gevonden...</p>
<?php endif; ?>
```

Om de **table headers dynamisch** te genereren kunt u de functie `array_keys()` loslaten op het eerste record array in de medewerkers array :

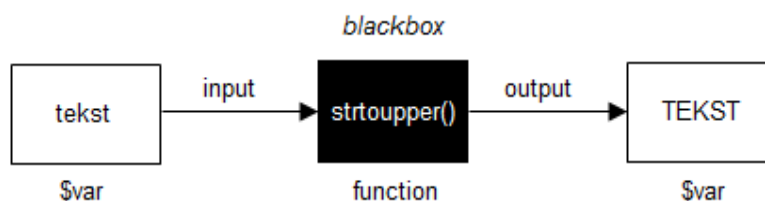
```
<?php if ($medewerkers): ?>
<table>
    <tr>
        <?php foreach (array_keys($medewerkers[0]) as $name): ?>
            <th><?php echo $name; ?></th>
        <?php endforeach; ?>
    </tr>
    <?php foreach ($medewerkers as $medewerker): ?>
        <tr>
            <?php foreach ($medewerker as $value): ?>
                <td><?php echo $value; ?></td>
            <?php endforeach; ?>
        </tr>
    <?php endforeach; ?>
</table>
<?php else: ?>
    <p class="warning">geen medewerkers gevonden...</p>
<?php endif; ?>
```

Als u eenmaal aan deze manier van werken gewend bent zult u al snel niet anders meer willen! Later in dit hoofdstuk bespreken we overigens een template helper, een hulpfunctie, die het makkelijker maakt om snel een HTML tabel van een 2-dimensionale array te maken.

7.3 Functies

U heeft inmiddels al heel wat in PHP ingebouwde functies leren kennen. Deze functies zijn onderdeel van de taal en worden ook wel *built-in*, *core* of *native functions* genoemd. Functies kunnen ook beschikbaar zijn omdat een extensie is geactiveerd, denk bijvoorbeeld aan de `mysqli_*` functies.

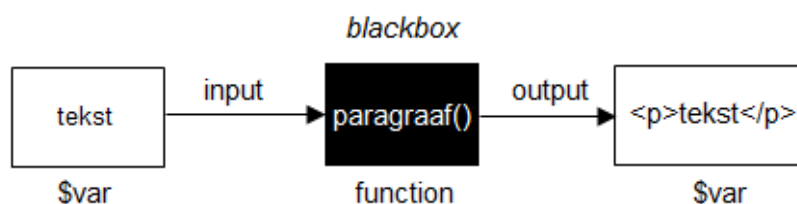
Wat al deze functies eigenlijk allemaal met elkaar gemeen hebben is dat ze direct output genereren (of een waarde retourneren) of input (in de vorm van een of meer argumenten) omzetten naar output of een (return waarde):



In code:

```
$var = 'tekst';
$var = strtoupper($var); // bevat nu 'TEKST'
```

In bovenstaand voorbeeld werd een bestaande functie gebruikt maar in dit deel van de cursus gaat u leren hoe u zelf functies kunt schrijven. Deze functies zijn dus geen vast onderdeel van PHP, maar een uitbreiding erop. U bedenkt ze zelf, vandaar dat men er soms de naam *User Defined Functions* (UDF's) voor hanteert. Neem bijvoorbeeld de zeer eenvoudige, maar zelfbedachte functie `paragraaf()` die tekst in een HTML paragraaf zet:



Dergelijke functies werken net als standaard PHP functies. U kunt nu alleen zelf bedenken wat uw functie moet gaan doen, hoeveel argumenten uw functie gaat aannemen en wat voor return waarden deze gaan teruggeven.

Wanneer u veel herhalingen in uw code heeft, of wanneer uw code te lang wordt en daardoor onoverzichtelijk, is het slim om de code op te delen in blokken. Wanneer u merkt dat u stukken code kopieert, omdat u deze op meerdere plaatsen in uw applicatie nodig heeft, en daarbij merkt dat er soms wat kleine aanpassingen voor een specifieke situatie nodig zijn, dan zondigt u tegen twee principes:

1. het DRY principe (*Don't Repeat Yourself*) en
2. het SPOC principe (*Single Point Of Control*).

Modulariseren

Op dergelijke momenten zijn functies de oplossing!

Wanneer code te lang wordt is niet duidelijk aan te geven, omdat het per programmeur verschilt. Veel programmeurs vinden echter alles wat langer is dan één schermbeeld eigenlijk al te lang omdat er dan steeds *gescrolled* moet worden om het overzicht te houden...

U kunt blokken code die functioneel gezien bij elkaar horen isoleren en van een naam voorzien die aangeeft wat het codeblok doet. Vervolgens kunt u deze blokken in het hoofdprogramma simpelweg met hun naam aanroepen waarna ze als vanouds worden uitgevoerd. Deze blokken zijn eigenlijk een soort zelfstandige miniprogrammaatjes en worden **functies** genoemd. Andere benamingen die u tegen kunt komen zijn **subroutines** en **procedures**.

Het grote voordeel van deze werkwijze is dat het verloop van het hoofdprogramma er veel beter door te volgen is en dat de codeblokken nu apart van elkaar kunnen worden onderhouden.

In feite zou het zelfs zo moeten zijn dat u de code binnen een functie aan zou moeten kunnen passen zonder dat het invloed heeft op het verloop van het hoofdprogramma; zolang ze maar dezelfde invoer accepteren en de verwachte uitvoer blijven geven. Op een vergelijkbare manier kunt u zo ook door anderen geschreven functies in uw eigen programma's toepassen zonder dat u precies weet hoe ze intern werken. Dit wordt ook wel met het **black box principe** aangeduid.

Functies vallen in feite onder het kopje procedurele logica of *Jumping Logic*, controlestructuren dus. Ze vormen immers een belangrijke manier om de *flow* van uw programma te kunnen bepalen.

Kortom, functies zijn nuttig om dezelfde code op meerdere plaatsen te kunnen gebruiken zonder het steeds te hoeven kopiëren. Functies kunnen met parameters worden aangeroepen en voeren van te voren vastgestelde acties uit. Indien gewenst kunnen ze waarden retourneren.

We gebruiken dit deel tevens om meer te kunnen vertellen over het **geldigheidsbereik** (*scope*) van variabelen.

7.3.1 Language constructs

Het is goed om het op deze plek nog even over *language constructs* te hebben. Wat zijn dat precies en wat is het verschil met functies? Dat is namelijk niet altijd even duidelijk voor iedereen omdat ze nogal op elkaar lijken qua notatie.

Voorbeelden van language constructs in PHP zijn in alfabetische volgorde:

- `array()`
- `die()`
- `each()`
- `echo()`
- `empty()`
- `exit()`
- `include()` en `include_once()`
- `isset()`
- `list()`
- `print()`
- `require()` en `require_once()`
- `unset()`

Language constructs zijn **ingebouwd in de taal**. Ze zijn er een **vast onderdeel** van. En ze kunnen **niet door gebruikers** worden **gedefinieerd** en ook **niet door een extensie** worden **toegevoegd**. De PHP scripting engine weet bijvoorbeeld dat een `if` opdracht altijd gevolgd moet worden door een

openingshaakje (. En als dat niet gebeurt dan krijgt u een parse error. Net als bij keywords als `while` en `function`. En zo weet de parser ook meteen wat er moet gebeuren als er een `echo` commando wordt gegeven. Functies daarentegen moeten eerst in het geheugen worden geladen en dan omgezet worden naar kleinere bouwstenen, waaronder weer de language constructs...

Language constructs zijn daarom over het algemeen **iets sneller** dan functies, maar dan moet u dus wel aan nano seconden denken.

Language constructs zijn ingebouwd en kunnen niet door gebruikers worden overschreven. Dat kan met de ingebouwde functies in principe wel met `override_function()` en `rename_function()` uit de Advanced PHP Debugger extension via PECL.

Verder kunnen language construct ook **niet disabled** worden via `php.ini`. Bij functions kan dat wel. En tot slot: language constructs kunnen **niet** gebruikt worden **als callback functions**.

Het verwarrende met de language constructs is dat ze (soms) zonder haakjes kunnen worden aangeroepen. Bij funties moet dat altijd, maar `die;` mag terwijl `die 'reutel';` niet mag!

```
print('One');
// is hetzelfde als
print 'One';

echo('One');
// is hetzelfde als
echo 'One';

// MAAR
echo 'One', 'Two'; // is OK
echo('One', 'Two'); // geeft ERROR!

die();
// is hetzelfde als
die;

// MAAR
die('We stoppen ermee'); // is OK
die 'We stoppen ermee'; // geeft ERROR!
```

Language constructs hebben zo dus hun eigen regels. Denk daarbij ook aan een construct als `isset()`. Als dit een gewone functie zou zijn en u zou een niet bestaande variabele opgeven ter controle dan zou alsnog een foutmelding krijgen!

Zie ook:

- <http://www.phpknowhow.com/basics/language-constructs-vs-built-in-functions/>
- <http://dreamaranis.com/blog/php-concepts-difference-between-functions-language-constructs>

7.3.2 Werking

Een functie is in zijn simpelste vorm niet meer dan een van een naam voorzien codeblok. Door de naam van de functie in uw hoofdprogramma aan te roepen zorgt u ervoor dat het codeblok wordt uitgevoerd. U kunt, zoals we later zullen zien, een functie echter uitbreiden met argumenten waardoor ze veel flexibeler en krachtiger worden.

Functies zijn standaard **public**, dat wil zeggen dat ze **zichtbaar** zijn **voor** alle **andere functies** in uw script. U kunt dus in een functie andere functies aanroepen.

7.3.3 Voorbereiding

Voordat u een functie gaat schrijven zou u eerst goed nagedacht moeten hebben wat de functie moet gaan doen. Een handig hulpmiddel daarbij is om het **doel van de functie in één zin samen te vatten**. Deze zin moet geen programmeerbegrippen bevatten, maar moet in gewoon Nederlands aangeven wat de functie doet. Het is handig om deze zin vervolgens als **commentaar** aan de functie toe te voegen. Bijvoorbeeld:

```
// deze functie voegt een nieuwe medewerker toe aan de medewerkers tabel
```

Een andere belangrijke stap is de **naam van de functie**. Deze moet ook tot uitdrukking brengen wat de functie doet en mag dus niet te algemeen zijn. Als de functie een operatie uitvoert dan is het in het Engels gebruikelijk om een sterk werkwoord te gebruiken en daarna het object. Bijvoorbeeld `add_new_employee()`. In het Nederlands is dat echter niet altijd even makkelijk te realiseren. U zou dan kunnen overwegen het object eerst te noemen: `medewerker_toevoegen()`.

Wanneer een functie een waarde retourneert (zie 1.3.11) dan is die waarde vaak een goede naam, bijvoorbeeld `$user_id`. Dit wordt vaak aangevuld met het werkwoord "get": `get_user_id()`.

Als u liever de **camelCase notatie** gebruikt dan worden de voorbeelden natuurlijk `addNewEmployee()`, `medewerkerToevoegen()` en `getUserId()`.

Wanneer u dit allemaal gedaan heeft dan heeft u een goed uitgangspunt voor het schrijven van een goede functie. Vervolgens kunt u dan de verschillende **stappen** die u in de functie nodig heeft eerst in **pseudo-code** opsommen. Daarna kunt u deze regels pseudo-code **uitcommentariëren** en pas dan gaat u deze stappen **invullen met programmacode**.

Als u op bovenstaande manier werkt bent u ervan verzekerd dat de functie precies doet wat u wilt en heeft u bovendien de code gelijk netjes van commentaar voorzien!

7.3.4 Opbouw

De opbouw van een functie ziet er als volgt uit:

```
function functie_naam()
{
    // Regels code binnen de functie
    // ...
}
```

Bovenstaande code wordt ook wel een **functie definitie** genoemd. Functie definities beginnen met het keyword `function`, gevolgd door de naam van de functie, die u dus zelf bedenkt.

Denk eraan dat de **naam** met een letter moet beginnen en geen spaties, punten of andere speciale karakters mag bevatten. Kies een duidelijke, beschrijvende naam, zoals hiervoor beschreven. De naam mag uiteraard **geen gereserveerd woord** zijn in PHP, zoals bijvoorbeeld `continue()`. U komt er natuurlijk snel genoeg achter omdat dat foutmeldingen zal opleveren, maar u kunt het simpelweg voorkomen door goed naar de kleur van uw code in uw editor te kijken. Een zelfverzonnen functienaam wordt namelijk niet herkend als een bestaande PHP opdracht. En u kunt natuurlijk altijd een kijkje nemen op: <http://php.net/manual/en/reserved.keywords.php>

Tip: sommige programmeurs vinden het handig om hun initialen of de naam van hun bedrijf in de naam van de functie te gebruiken:

```
// functie gedefinieerd door Arno Mol (am)
function am_db_connect()
```



```
{
    ...
}
```

Zo zijn de eigen functies meteen herkenbaar en herleidbaar tot de auteur(s). Bovendien kunnen ze zo niet 'botsen' met gelijknamige functies van anderen.

Achter de naam komt **altijd** een **haakjespaar**. Deze zijn bedoeld voor de argumentenlijst, maar ook als een functie zonder argumenten werkt, moet u ze plaatsen! Zie voor meer informatie over argumenten paragraaf 1.3.7.

De uit te voeren code staat tussen **accolades**. In principe kunt u de eerste accolade ook gelijk na het haakjespaar plaatsen, maar veel PHP programmeurs geven de voorkeur aan bovenstaande notatie, omdat deze schrijfwijze in veel open source projecten afgesproken is.

U zou op deze manier elk stuk code waarin meerdere statements na elkaar worden uitgevoerd in een functie kunnen zetten. Bijvoorbeeld alle code die nodig is voor het tonen van contact informatie in een functie `show_contact_info()`. Deze kunt u vervolgens zo vaak aanroepen als u maar wilt op uw pagina.

Ook code die in principe maar één keer hoeft te worden uitgevoerd kunt u om redenen van leesbaarheid in een functie zetten. Bijvoorbeeld alle code om een formulier aan te maken en te tonen in een functie `show_form()`.

7.3.5 Indeling

Het werkt erg prettig om in functiebibliotheken (lange) functiedefinities visueel duidelijk van elkaar te scheiden en de naam van de functie te herhalen als commentaar na de afsluitende accolade. Zo ziet u bij het doorlopen van een reeks functies duidelijk de scheidingen en bij welke functie de voorafgaande code hoort. U zou het bijvoorbeeld zo kunnen doen:

```
<?php
//*****
function db_connect()
{
    ...
} // end db_connect
//*****

//*****
function db_close($conn)
{
    ...
} // end db_close
//*****
?>
```

Het is handig om deze commentlines **80 karakters lang** te maken, dus na de comment slashes komen dan nog 78 sterretjes. Dan ziet u snel wanneer de code erbinnen te lang wordt en u voor de leesbaarheid zou moeten afbreken om op een volgende regel verder te gaan. Veel editors hebben de mogelijkheid een markeringslijn voor deze grens aan te zetten.

Helemaal handig wordt het wanneer u de code voor de definitie van een functie in een zogenaamde **snippet** zet. De werkwijze verschilt per editor.

NB. Het kan we zijn dat sommige **code sniffers** de commentaar lijnen als een **te groot percentage commentaar** aanduiden...

7.3.6 Aanroepen functie

Code die u in een functie gezet heeft wordt niet zonder meer uitgevoerd. U moet de functie daarvoor eerst **aanroepen**!

Het aanroepen van een functie doet u door eenvoudigweg de naam van de functie te plaatsen op die plek in de pagina waar de in de functie beschreven acties moeten worden uitgevoerd:

```
functie_naam();
```

De aanroep mag in hetzelfde PHP codeblok staan waarin de functie werd gedefinieerd maar dat hoeft niet. De aanroep kan ook heel goed een eind verderop in het script plaatsvinden.

Een heel simpel voorbeeld:

```
<?php
// de functie aanroep kan voor
// de functie definitie komen...
test();

// de functie definitie
//*****
function test()
{
    echo 'test';
}
//*****

// Of erna. De functie naam is overigens
// case-insensitive...
TEST();
?>
```

In feite maakt het niet eens zo heel veel uit waar u de functie definities plaatst. Zolang ze maar beschikbaar zijn in het huidige script. Wanneer een PHP script door de Zend Engine verwerkt wordt dan zal eerst naar de aanwezigheid van functies gezocht worden. De functies worden vervolgens in het geheugen geladen (pre-compilatie) en pas daarna wordt het script geïnterpreteerd. Dit houdt ondermeer in dat u in uw script een functie kunt aanroepen **vóórdát** u de functie gedefinieerd heeft!

De definitie moet echter **wel op dezelfde pagina** staan als de aanroep, al zullen we straks zien dat zelfs dat niet per se noodzakelijk is wanneer u ze in oefening 1.4 in combinatie met de eerder besproken **includes** leert gebruiken. Overigens moet u de **includes** dan wel **eerst insluiten voordat** u de **functies erin** kunt **aanroepen**, maar dat is best logisch...

Functienamen zijn **case-insensitive**, maar uiteraard gebruikt u bij het aanroepen dezelfde notering die gebruikt is in de functie-definitie. Sommige editors herkennen in hoofdletters geschreven functies ook niet...

Let op: de tot nu toe besproken functies **retourneren (nog) geen waarden**. U kunt dergelijke functies dan ook **niet aanroepen in een expressie**! Hoe u een functie schrijft die wel waarden retourneert leert u later in dit hoofdstuk.

Oefening

Maak in de map oefeningen/ een pagina functies.php aan en definieer daarin een functie test() die de tekst "dit is een test" wegschrijft tussen twee <hr> elementen.

Roep de functie in dezelfde pagina aan om het resultaat te bekijken. Zet de aanroep ook eens voor de functie-definitie.

7.3.7 Parameters en Argumenten

De functie die u in de oefening gemaakt heeft is erg beperkt. Als u wel eens een andere tekst zou willen weergeven dan moet u een andere functie schrijven...

U kunt een functie gelukkig flexibeler maken door het gebruik van **parameters**. Door bij de aanroep wisselende waarden voor de parameters mee te geven kan één en dezelfde functie dan toch verschillende uitvoer geven. De waarden die bij aanroep gebruikt worden noemen we de **argumenten**. Dit principe wordt in het Engels *argument passing* genoemd. Omdat de begrippen vaak door elkaar gehaald worden, hier nogmaals de omschrijving:

Een functie definitie kan **parameters** bevatten. Deze worden *eenmalig gedefinieerd*. De functie kunt u vervolgens aanroepen met *steeds wisselende argumenten*.

Wanneer u **parameters** in uw functies wilt gaan gebruiken dan plaats u deze tussen de haakjes in een zogenaamde *parameter list*. Meerdere parameters worden daarbij gescheiden door komma's:

```
function functie_naam($parameter1, $parameter2)
{
    // Regels code binnen de functie waarin $
    // parameter1 en $parameter2 als variabelen dienen...
    // ...
}
```

De namen van de parameters kunt u weer zelf kiezen, maar ze moeten met een \$-teken beginnen en exact zo terugkomen in de code binnen de functie.

Bij de aanroep van de functie dient u nu de argumenten mee te geven, precies **evenveel** en in **dezelfde volgorde** als bij de definitie van de functie:

```
functie_naam($argument1, $argument2);
```

De parameters nemen de waarden aan van de argumenten waarmee de functie wordt aangeroepen en deze worden als lokale variabelen binnen de functie gebruikt. Na de aanroep worden deze waarden vernietigd. Ze zijn op geen enkel moment voor het hoofdprogramma beschikbaar.

Bij een volgende aanroep is het overigens alsof de functie voor de eerste keer wordt aangeroepen, een functie heeft dus geen 'geheugen'. Tenzij u het keyword **static** gebruikt (zie 1.3.13)

De variabelen of constanten die u bij de aanroep gebruikt worden de **argumenten** of de **actual parameters** genoemd en de variabelen die u in de parameterlist van de functie definieert worden de **formal parameters** genoemd. Bij het aanroepen van de functie worden nu de waardes van de actuele parameters gekopieerd in de formele parameters waarmee intern wordt verder gewerkt.

Laten we dit eens aanschouwelijk maken met een voorbeeld. In de oefening hierboven zou u zich voor kunnen stellen dat u in de aanroep van de functie een argument meegeeft welke tekst getoond moet worden. Dat kan de tekst zelf zijn of een variabele waarin tekst is opgeslagen. We kiezen in eerste instantie voor de eerste manier:

```
<?php
// Functie definiëren
// *****
function paragraaf($tekst)
{
    echo "<p>" . $tekst . "</p>\n";
}
```

Modulariseren

```
//*****  
  
// Functie aanroepen  
paragraaf("voorbeeld met argument");  
?>
```

U had natuurlijk net zo makkelijk de tekst in een **variabele** kunnen zetten en deze als argument mee kunnen geven:

```
$tekst = "nog een voorbeeld met argument";  
  
// Functie aanroepen  
paragraaf($tekst);
```

Merk op dat de variabelen die voor de actuele parameters en de formele parameters gebruikt worden **dezelfde naam** kunnen hebben maar dat dit zeker **niet nodig** is:

```
$zin = "nog een voorbeeld met argument";  
  
// Functie aanroepen  
paragraaf($zin);
```

Dit geeft uiteraard precies hetzelfde resultaat...

U kunt binnen de functie ook controlestructuren als `if ... else` of een `switch` gebruiken om aan de hand van de waarde van het argument te kunnen beslissen over de door de functie te ondernemen acties.

Oefening	Breid de functie <code>test()</code> in de eerder aangemaakte pagina <code>oefeningen/functions.php</code> uit met een argument zodat u wisselende boodschappen tussen de <code><hr></code> elementen kunt wegschrijven.

7.3.8 Type Hinting for Parameters

Voor PHP5 was het niet mogelijk om aan te geven van welk datatype een parameter moest zijn. Vanaf PHP5 wel en in PHP7 is het nog beter geregeld. Het gebruik van type hinting is niet verplicht maar het maakt uw code beter leesbaar en foutmeldingen duidelijker. Bovendien zal uw IDE dan ook beter code completion kunnen doen! Dit is met name het geval wanneer u met classes werkt. Als u besluit type hinting te gaan gebruiken dan is hinting eigenlijk misleidend, want het opgeven type wordt dan eigenlijk gewoon verplicht omdat u anders een error krijgt...

7.3.8.1 PHP5

Vanaf PHP5 kunt u bij de declaratie van een parameter steeds beter aangeven van welk datatype deze moet zijn. In eerste instantie kon dit alleen voor class types, interfaces en callables. Vanaf PHP5.1 kan het echter ook voor arrays:

```
function enroll(Student $student, array $courses)  
{  
    foreach ($courses as $course) {  
        echo $student->name . ' will study ' . $course . '<br>';  
    }  
}
```

Dit gaat goed als u de functie zo aanroept:

```
require_once 'classes/Student.php';  
  
$student = new Student();
```

```
$student->name = 'Arno';
$courses = ['JavaScript', 'PHP'];

enroll($student, $courses);
```

Want de datatypen van de argumenten komen nu overeen met de vereiste datatypen voor de parameters, respectievelijk een Student datatype (een instantie van de Student class) en een array.

Als u iets anders op zou geven, bijvoorbeeld een string ipv Student:

```
enroll('Arno', $courses);
```

dan resulteert dit in een error:

Fatal error: Uncaught TypeError: Argument 1 passed to enroll() must be an instance of Student, string given

Dat gebeurt natuurlijk ook als u voor de tweede parameter geen array opgeeft:

```
enroll($student, 'PHP');
```

dan:

Fatal error: Uncaught TypeError: Argument 2 passed to enroll() must be of the type array, string given

7.3.8.2 PHP7

Vanaf PHP7 is Type Hinting ook mogelijk voor **Scalar Variables**:

- integer
- float
- boolean
- string

Standaard wordt daarbij uitgegaan van de zogenaamde *coercive mode*, waarbij conversie van de parameters wordt toegestaan. In *strict mode* is dat niet het geval.

7.3.8.2.1 Coercive mode

In *coercive mode* wordt gepoogd conversies op de parameters uit te voeren. Stel, we hebben de volgende functie:

```
function add(int $a, int $b)
{
    return $a + $b;
}
```

Wanneer u 2 integers opgeeft werkt dat natuurlijk zoals verwacht:

```
var_dump(add(1, 2)); // geeft int(3)
```

Maar als u strings opgeeft die zonder problemen geconverteerd kunnen worden, dan werkt het nog steeds:

Modulariseren

```
var_dump(add('1', '2')); // int(3)
```

Wanneer het lastiger wordt, bijvoorbeeld bij een invoer als '1e' en '2e' dan geeft PHP7 wel notices als deze:

Notice: A non well formed numeric value encountered

Maar doet het uiteindelijk wel:

```
var_dump(add('1e', '2e')); // int(3)
```

Als het vanaf het begin al niet kan, dan krijgt u bij iets als:

```
var_dump(add('een', 'twee'));
```

gelijk:

Fatal error: Uncaught TypeError: Argument 1 passed to add() must be of the type integer, string given

7.3.8.2.2 Strict mode

Met de volgende declaratie zet u PHP7+ in **strict mode**:

```
<?php  
declare(strict_types=1);
```

Deze moet **helemaal bovenaan** in het **bestand** komen dat de **functie aanroept!** Anders krijgt u een error:

Fatal error: strict_types declaration must be the very first statement in the script

Nu gaat zelfs deze, al zouden de argumenten zonder problemen geconverteerd kunnen worden, al mis:

```
var_dump(add('1', '2'));
```

Want nu krijgt u gelijk deze melding:

Fatal error: Uncaught TypeError: Argument 1 passed to add() must be of the type integer, string given

7.3.8.3 PHP 7.1

Vanaf PHP7.1 zijn ook **Nullable Types** mogelijk. kunt u de type hint ook vooraf laten gaan door een vraagteken:

```
function test(?string $name)  
{  
    var_dump($name);  
}
```

Dit houdt in dat de waarde naast een string ook NULL mag zijn:

```
var_dump(test('test')); // test
var_dump(test(null));  // null
```

Merk op dat het accepteren van NULL niet betekent dat u het argument kunt weglaten:

```
var_dump(test());
```

geeft:

Uncaught Error: Too few arguments to function test(), 0 passed in...

7.3.9 meer/minder argumenten

Wanneer u een functie met **minder argumenten** aanroept dan er parameters gedefinieerd zijn, dan zal PHP met een warning en een notice reageren, maar de functie wel uitvoeren. Er van uitgaand dat de functiedefinitie voor `paragraaf()` een parameter `$tekst` heeft (zoals hiervoor) en u roept de functie zonder argument aan:

```
paragraaf();
```

Dan krijgt u:

Warning: Missing argument 1 for paragraaf(), called in demo.php on line 7 and defined in demo.php on line 7

Notice: Undefined variable: tekst in demo.php on line 4

Maar de functie wordt wel uitgevoerd en het resultaat is een lege paragraaf:

```
<p></p>
```

Wanneer u een functie aanroept met **meer argumenten** dan er parameters gedefinieerd zijn, dan worden de argumenten die te veel zijn gewoon genegeerd:

```
paragraaf('tekst', 'meer tekst');
```

Er worden geen warning of notices getoond en het resultaat is:

```
<p>tekst</p>
```

U kunt dit in uw voordeel gebruiken door functies met een wisselend aantal argumenten aan te roepen. U kunt dat op drie verschillende manieren doen:

- Met de `func_*_arg*()` functies
- Met default arguments
- Met een array (of object) als parameter

7.3.10 flexibel aantal argumenten met `func_*_arg*()`

Om met een variabel aantal argumenten te kunnen werken kunt u er voor kiezen om gebruik te maken van de volgende functies:

- `func_get_args()`
geeft een array met alle argumenten

Modulariseren

- `func_num_args()`
geeft het aantal argumenten
- `func_get_arg($index)`
geeft de waarde van een argument op een bepaalde positie

Let op: Deze functies moeten **binnen** de user defined functie worden gebruikt!

Een voorbeeld:

```
function test()
{
    $args = func_get_args();
    foreach ($args as $arg) {
        echo $arg . "<br>";
    }

    echo "<br>";

    // met posities
    for ($i=0; $i<func_num_args(); $i++){
        echo "argument $i: " . func_get_arg($i) . "<br>";
    }
}
```

Merk op dat u door het aanroepen van `func_get_args()` in feite ook de beschikking heeft over het aantal en de positie:

```
...
    $num_args = count($args);
    for ($i=0; $i<$num_args; $i++){
        echo "argument $i: " . $args[$i] . "<br>";
    }
...
```

Wanneer we de functie nu zo aanroepen:

```
test('a', __FILE__, 3);
```

Dan ziet het resultaat er als volgt uit:

```
a
D:\Sites\test.php
3

argument 0: a
argument 1: D:\Sites\test.php
argument 2: 3
```

7.3.11 array als parameter

U kunt natuurlijk ook een array als parameter gebruiken. Hierin kunt u een wisselend aantal naam-waarde paren opnemen. U dient natuurlijk wel te controleren op aanwezigheid van de key voordat u de waarde probeert te gebruiken. Idee:

```
/** *****
function input($options)
{
    $html = '<input';
    if (is_array($options)) {
        if (isset($options['type'])) {
```



```

        $html .= ' type="' . $options['type'] . '"';
    }
    if (isset($options['name'])) {
        $html .= ' name="' . $options['name'] . '"';
    }
    if (isset($options['value'])) {
        $html .= ' value="' . $options['value'] . '"';
    }
    // andere attributen (class, maxlength etc)
    ...
}
$html .= '>';
return $html;
}
// *****

$options = array("type" => "text",
                 "name"  => "voornaam",
                 "value" => "Arno");
echo input($options);

// of in 1x, nu met een naam-waarde paar minder
echo input(array("type"=>"text", "name"=>"voornaam"));

```

Merk op dat u vanaf PHP 5.1 ook een type hint array had kunnen opgeven. Dan is de check `is_array()` in principe niet meer nodig...

7.3.12 JSON string als parameter

U kunt de notatie wat compacter houden door een JSON string als parameter te gebruiken. Hierin kunt u wederom zoveel naam-waarde paren opnemen als u wilt. Met de ingebouwde `json_decode()` functie krijgt u dan een object met de naam-waarde paren als properties. Idee:

```

// *****
function input($options)
{
    $html = '<input';
    $obj = json_decode($options);
    if ($obj) {
        if (isset($obj->type)) {
            $html .= ' type="' . $obj->type . '"';
        }
        if (isset($obj->name)) {
            $html .= ' name="' . $obj->name . '"';
        }
        if (isset($obj->value)) {
            $html .= ' value="' . $obj->value . '"';
        }
        // andere attributen (class, maxlength etc)
        ...
    }
    $html .= '>';
    return $html;
}
// *****

echo input('{"type":"text","name":"voornaam","value":"Arno"}');

```

Zie voor meer informatie over JSON de **JavaScript cursus**.

NB.: In **JavaScript** kunt u ipv associatieve arrays of JSON strings heel eenvoudig een **Object parameter** gebruiken. Zie daarvoor ook de **JavaScript cursus**.

7.3.13 Variadic functions met de ... operator

Vanaf PHP 5.6 kunt u gebruik maken van de ... operator om aan te geven dat een functie een variabel aantal argumenten accepteert:

```
function sum(...$numbers) {
    $result = 0;
    foreach ($numbers as $number) {
        $result += $number;
    }
    return $result ;
}

echo sum(1, 2, 3, 4); // 10
```

U kunt de ... operator ook gebruiken bij het aanroepen:

```
$numbers = [1, 2, 3, 4, 5];
echo sum(...$numbers); // 15
```

De array wordt dan automatisch uitgepakt!

7.3.14 Default argumenten

Een handige mogelijkheid van PHP is dat u functies kunt schrijven die gebruik maken van argumenten met een default waarde. Hierdoor maakt u het gebruik van een argument in feite **optioneel** omdat bij een functie aanroep zonder het argument de default waarde wordt gebruikt:

```
<?php
// Functie definiëren met default argument
//*****
function message($txt="Hoi")
{
    echo $txt;
}
//*****

// aanroep zonder argument
// gebruikt default argument
schrijf_tekst ();

// aanroep met argument
// overschrijft default argument
schrijf_tekst ("Hallo");
?>
```

Het resultaat van bovenstaande aanroepen is dus:

```
Hoi
Hallo
```

Als u in de functie ook **verplichte argumenten** wilt gebruiken, dan moeten deze **als eerste** in de parameter lijst staan. U kunt meerdere default argumenten gebruiken, maar als u in de aanroep minder argumenten gebruikt dan is de volgorde wel van essentieel belang en moet u eventueel voorafgaande optionele argumenten toch een waarde geven. Het is een beetje wie het eerst komt die het eerst maalt:

```
<?php
//*****
function schrijf_groet($groet='Hoi', $naam='Arno')
{
```

```

    echo $groet . ' ' . $naam;
}
//*****
?>

```

Als u Hoi Stefan weg zou willen schrijven dan kan dat **niet** zo:

```
schrijf_groet('Stefan');
```

Dat resulteert namelijk in:

```
Stefan Arno
```

U moet dan toch een waarde voor het eerste argument opgeven:

```
schrijf_groet('Hoi', 'Stefan');
```

U kunt ook met **variabele aantallen argumenten** werken.

Oefening	<i>Voeg aan de functie <code>end_page()</code> een default argument <code>\$copyright</code> toe die als defaultwaarde <code>true</code> heeft en daardoor standaard een footer als deze toevoegt:</i> © 2002-2013 <uw naam> <i>Het laatste jaartal wordt uiteraard dynamisch gezet. Controleer dat u er met <code>end_page(false)</code> voor kunt zorgen dat de footer niet getoond wordt.</i>

7.3.15 Named Arguments

Wanneer u maar een paar default waarden wilt aanpassen dan is zoals we eerder zagen de volgorde waarin u dat doet van groot belang. Vanaf PHP 8 kunt u echter, net als in bijvoorbeeld Python, **named arguments** gebruiken.

Named arguments maken het doorgeven van argumenten aan een functie dus mogelijk op basis van de **parameternaam in plaats van de parameterpositie**. Dit heeft de volgende voordelen:

- Het maakt de betekenis van het argument **zelfdocumenterend**,
- Het maakt de argumenten **orderonafhankelijk**
- Het staat het **willekeurig overslaan** van **standaardwaarden** toe!

Named arguments worden doorgegeven door de waarde vooraf te laten gaan door de **parameternaam** gevolgd door een **dubbele punt**. Het gebruik van gereserveerde trefwoorden als parameternamen is toegestaan. De parameternaam moet een identifier zijn, dynamisch specificeren is niet toegestaan, zoals we zo zullen zien.

Stel we nemen onze eerdere functie:

```

function schrijf_groet($groet='Hoi', $naam='Arno')
{
    echo $groet . ' ' . $naam;
}

```

Modulariseren

Dan kunt u hem nu we eenvoudig aanroepen met een andere naam en toch de default waarde te krijgen:

```
schrijf_groet(naam: 'Stefan');
```

En de uitkomst is dan:

```
Hoi Stefan
```

Dit werkt ook heel fijn bij standaard PHP functies, bijvoorbeeld bij `setcookie()`:

```
setcookie(  
    name: 'test',  
    expires: time() + 60 * 60 * 2,  
);
```

Waarbij hier dus geen value wordt meegegeven (normaal het tweede argument zoals u later in de cursus zult leren).

Named arguments kunnen worden gecombineerd met positionele argumenten. In dit geval moeten de genoemde argumenten na de positionele argumenten komen. Het is ook mogelijk om slechts enkele van de optionele argumenten van een functie te specificeren, ongeacht hun volgorde.

7.3.16 By Value vs. By Reference

Bij het aanroepen van een functie verandert de waarde van de bronvariabele buiten de functie normaal gesproken niet:

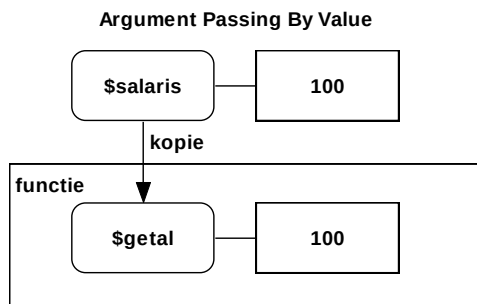
```
<?php  
//*****  
function verhogen($getal)  
{  
    $getal = $getal + ($getal * 0.05);  
    return $getal;  
}  
//*****  
  
$salaris = 100;  
  
// schrijf verhoogde salaris weg  
echo verhogen($salaris). "<br>";  
  
// schrijf salaris nogmaals weg. Wat is dit nu?  
echo $salaris;
```

Geeft:

```
105  
100
```

Het salaris is dus **niet blijvend** verhoogd! Dit **standaardgedrag** wordt met **Argument Passing by Value** aangeduid omdat er een **kopie** werd gemaakt van de **originele waarde** van de variabele. U geeft dus niet de variabele zelf door maar alleen de waarde.

Met andere woorden de globale variabele `$salaris` en de lokale variabele `$getal` maken **niet** gebruik van **dezelfde geheugenruimte**. Bewerkingen die worden losgelaten op de waarde van `$getal` hebben dus **geen invloed** op de waarde van `$salaris`:



Aangezien dit lang niet altijd gewenst is, bestaat er ook een andere manier om argumenten door te geven. Deze heet **Argument Passing by Reference**. Nu wordt er **verwezen naar de originele waarde** van de variabele en een verandering in de waarde van de parameter zal resulteren in een verandering in de waarde van de originele variabele die als argument werd gebruikt. U moet dit wel aangeven in de parameter list met het voorvoegsel `&`:

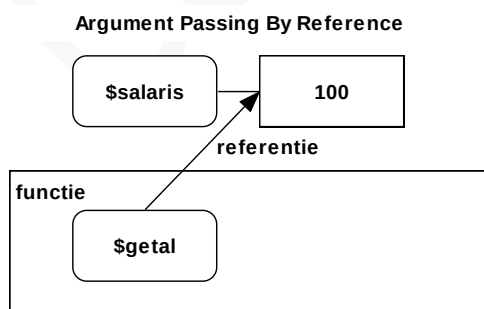
```
<?php
//*****
function verhogen(&$getal)
{
    $getal = $getal + ($getal * 0.05);
    return $getal;
}
//*****

$salaris = 100;
echo verhogen($salaris). "<br>";
echo $salaris;
?>
```

Geeft:

105
105

Het salaris is nu wel blijvend verhoogd omdat beide variabelen **wel** gebruik maken van **dezelfde geheugenruimte**:



U heeft zo effectief de waarde van de doorgegeven variabele aangepast.

Let op: als u argumenten *by reference* doorgeeft dan kan het argument alleen een variabele zijn en geen *literal* (letterlijke waarde):

```
verhogen(100); // LET OP: kan niet!
```

Fatal error: Only variables can be passed by reference

7.3.16.1 call-time by reference

U mocht in PHP vroeger het ampersand-teken ook **bij de aanroep** van de functie plaatsen, en het dan weglaten bij de functie definitie:

```
verhogen(&$salaris); // LET OP: niet meer doen!
```

Dit wordt *Call-time pass-by-reference* genoemd en het geeft u weliswaar meer flexibiliteit omdat u het argument bij dezelfde functie op twee verschillende manieren kunt doorgeven, maar wordt gezien als slecht programmeren omdat het verloop van dergelijke code hierdoor heel lastig te volgen is... Vandaar dat het sinds PHP 5.3 ook als **deprecated** wordt aangegeven en sinds PHP 5.4 is het zelfs verwijderd en geeft het een fatal error:

Fatal error: Call-time pass-by-reference has been removed; If you would like to pass argument by reference, modify the declaration of verhogen()

7.3.17 Static

Een functie als deze heeft weinig zin:

```
function ophogen()  
{  
    $a = 1;  
    echo $a;  
    $a++;  
}
```

Immers: iedere keer dat de functie wordt aangeroepen wordt `$a` op 1 gezet en wordt er 1 weggeschreven. Het ophogen van `$a` heeft geen zin omdat deze waarde bij het beëindigen van de functie toch vernietigd wordt. Met andere woorden:

```
ophogen(); // geeft: 1  
ophogen(); // geeft: 1  
ophogen(); // geeft: 1
```

U kunt echter met behulp van het keyword **static** zorgen dat `$a` een "geheugen" krijgt:

```
function ophogen()  
{  
    static $a = 1;  
    echo $a;  
    $a++;  
}
```

Nu zal wel, elke keer dat `ophogen()` wordt aangeroepen (in hetzelfde script) de waarde van `$a` worden weggeschreven en opgehoogd, zodat een volgende keer de opgehoogde waarde wordt weggeschreven:

```
ophogen(); // geeft: 1  
ophogen(); // geeft: 2  
ophogen(); // geeft: 3
```

7.3.18 Bereik (scope)

U kunt naast argumenten ook expliciet gedeclareerde variabelen gebruiken. Daarbij maken we onderscheid tussen variabelen die expliciet **binnen** de procedure zijn gedeclareerd en variabelen die **buiten** de procedure in het hoofdsript zijn gedeclareerd.

Variabelen die gedeclareerd zijn **binnen** de functie hebben altijd een **local scope**, dat wil zeggen dat ze alleen uit te lezen zijn binnen de functie. Na aanroep worden ze ‘vernietigd’. De waarde van lokale variabelen wordt dan ook niet bewaard tussen verschillende aanroepen.

Variabelen die **buiten** de procedure zijn gedeclareerd kunnen niet binnen een functie gebruikt worden. Zij hebben namelijk een **global scope** wat wil zeggen dat ze door het hele script heen zijn uit te lezen, behalve in een functie. U kunt dit gedrag aanpassen met het keyword `global`.

Onderstaande code levert problemen op:

```
$globaal = 'globaal gedeclareerde variabele';

//*****
function scope_test()
{
    $lokaal = 'lokaal gedeclareerde variabele';
    echo 'lokaal = ' . $lokaal . '<br />';
    echo 'globaal = ' . $globaal . '<br />';
}
//*****

scope_test();
```

De uitvoer is namelijk:

```
lokaal = lokaal gedeclareerde variabele
Notice: Undefined variable: globaal in test.php
globaal =
```

En als u (buiten de functie) de volgende opdrachten zou geven:

```
echo 'lokaal = ' . $lokaal . '<br />';
echo 'globaal = ' . $globaal . '<br />';
```

Dan levert de eerste regel een probleem op en gaat het bij de tweede regel juist goed:

```
Notice: Undefined variable: lokaal in test.php
lokaal =
globaal = globaal gedeclareerde variabele
```

Voor het hoofdsript bestaat de lokale variabele `$lokaal` namelijk niet!

7.3.18.1 Globale variabelen gebruiken

U kunt binnen een functie toch de waarde van een globale variabele gebruiken wanneer u er het keyword `global` voor zet:

```
$globaal = 'globaal gedeclareerde variabele';

//*****
function scope_test()
{
    global $globaal;
```

Modulariseren

```
$lokaal = 'lokaal gedeclareerde variabele';
echo 'lokaal = ' . $lokaal . '<br />';
echo 'globaal = ' . $globaal . '<br />';
}
//*****
scope_test();
```

geeft:

```
lokaal = lokaal gedeclareerde variabele
globaal = globaal gedeclareerde variabele
```

Let op: U moet wel **oppassen** met deze manier van werken omdat u zo **niet alleen** de **waarde** van globale variabelen binnen functies **uit** kunt **lezen** maar ze (per ongeluk) **ook** zou kunnen **veranderen**! Dit is overigens ook de reden waarom er in PHP überhaupt voor deze constructie gekozen is. Dat een globale variabele eerst met het keyword `global` moet worden 'geïmporteerd' voorkomt dat beginners per ongeluk deze variabelen zouden overschrijven binnen functiedefinities. Vrijwel geen enkele andere bekende programmeertaal kent een dergelijke bescherming.

Een andere reden waarom u met deze werkwijze moet oppassen is omdat het op deze manier wat onduidelijker is waar functies hun waarden vandaan halen. Het aanpassen of hergebruiken van functies wordt daardoor wellicht wat lastiger.

Aan de andere kant kan het soms juist reuze handig zijn om met functies globale waarden aan te passen! Als het maar een bewuste keuze is en goed gedocumenteerd, dan is dat natuurlijk geen enkel probleem! Wij zullen in de cursus vooral global link identifiers binnen functies gebruiken. Omdat het aanpassen van de waarde van een link identifier sowieso nooit logisch is, loopt u hiermee verder geen risico's en het werkt prettiger dan het steeds doorgeven van een link identifier als argument.

U kunt op zich het keyword `global` pas gebruiken vlak voordat u de globale variabele nodig heeft. Het is echter gebruikelijk de globals standaard **gelijk aan het begin** van uw functie te **noteren**, dat maakt het snel duidelijk of er globale variabelen in gebruik zijn en zo ja welke.

Let op: Wanneer u binnen een functie een globale variabele met het keyword `global` probeert binnen te halen terwijl deze **niet** eerder **gedefinieerd** was in het script **dan** geeft dat **geen foutmelding** maar wordt de variabele gewoon op dat moment **aangemaakt**! U kunt de variabele daarna gewoon buiten de functie waarin hij gedefinieerd werd gebruiken. Met het keyword `global` kunt u dus zowel refereren aan een bestaande variabele als een nieuwe globale variabele declareren.

U kunt **meerdere variabelen in een keer** globaal declareren door ze met komma's te scheiden:

```
function test()
{
    global $a, $b, $c;
    ...
}
```

Let op: U kunt het keyword `global` niet gebruiken in combinatie met andere functies. Ook is het niet mogelijk om gelijk een waarde toe te kennen:

```
// LET OP: Dit kan niet!
echo global $var;
```



```
// LET OP: Dit kan ook niet!
global $x = 'y';
```

7.3.18.2 \$GLOBALS

Een andere manier om globale variabelen binnen een functiedefinitie te gebruiken is met behulp van de superglobal `$GLOBALS`. Dit is een associatieve array waarin alle globals (en superglobals) die in een script beschikbaar zijn in worden opgeslagen. Let overigens op de **schrijfwijze zonder _** na het `$`-teken! Onderstaande twee statements doen exact hetzelfde, namelijk het declareren en initialiseren van een globale variabele:

```
$GLOBALS['x'] = 'y';
// is hetzelfde als:
$x = 'y';
```

U kunt deze notatie dan ook gebruiken in plaats van het keyword `global`:

```
$gloobaal = 'globaal gedeclareerde variabele';

//*****
function scope_test()
{
    $lokaal = 'lokaal gedeclareerde variabele';
    echo 'lokaal = ' . $lokaal . '<br />';
    echo 'globaal = ' . $GLOBALS['gloobaal'] . '<br />';
}
//*****

scope_test();
```

De notatie is echter wat omslachtiger vandaar dat u het in de praktijk waarschijnlijk niet zoveel zal tegenkomen...

7.3.18.3 Argumenten

U kunt buiten een functie gedefinieerde variabelen natuurlijk ook gewoon binnen de functie gebruiken door ze als **argumenten** 'binnen te halen' zoals u in 1.3.7 heeft geleerd:

```
$gloobaal = 'globaal gedeclareerde variabele';

//*****
function scope_test($var)
{
    $lokaal = 'lokaal gedeclareerde variabele';
    echo 'lokaal = ' . $lokaal . '<br />';
    echo 'globaal = ' . $var . '<br />';
}
//*****

scope_test($gloobaal);
```

Dit geeft hetzelfde resultaat als met het keyword `global`! Maar omdat het argument hier *by value* wordt doorgegeven, kan de waarde van `$gloobaal` niet binnen de functie worden aangepast. Als u de waarde *by reference* zou doorgeven zoals in `function scope_test(&$var)` dan kan dat natuurlijk wel (zie ook 1.3.9).

7.3.18.4 Superglobals

Zoals de naam al aangeeft zijn de superglobal arrays eveneens **altijd globaal**. We zetten ze hieronder even op een rij. Een aantal kent u al vanuit de basiscursus, anderen zult u later deze cursus leren kennen:

Modulariseren

Superglobal	Omschrijving
<code>\$_COOKIE</code>	Bevat cookie informatie (zie H5 van deze cursus)
<code>\$_ENV</code>	Bevat omgevingsvariabelen (waarbij de inhoud sterk afhankelijk is van het Operating System waarop PHP draait)
<code>\$_FILES</code>	Bevat geupload(e) bestand(en). Zie H8 van deze cursus.
<code>\$_GET</code>	Bevat GET request data (via hyperlink of form). Zie de basis cursus.
<code>\$GLOBALS</code>	Verzameling van alle globals, inclusief superglobals. Is zelf ook global...
<code>\$_POST</code>	Bevat POST request data (via form). Zie de basis cursus.
<code>\$_REQUEST</code>	Bevat combinatie van GET, POST en COOKIE data. Zie de basis cursus.
<code>\$_SERVER</code>	Bevat CGI variabelen waarin informatie uit het HTTP Request en de HTTP Response. Zie de basis cursus.
<code>\$_SESSION</code>	Bevat sessie data (per gebruiker). Zie H6 van deze cursus.

Een wat vreemde eend in de bijt is `$GLOBALS`. Merk op dat u deze ook als enige **zonder underscore** noteert. Stel u heeft een lokale variabele `$voornaam = 'Arno'` buiten een functie. Dan kunt u deze functie zoals u weet binnenhalen met `global $voornaam`. Maar u kunt de waarde binnen de functie ook gelijk gebruiken als u hem noteert als `$GLOBALS['voornaam']`.

Op dezelfde manier is `$_GET['id']` beschikbaar als `$GLOBALS['_GET']['id']`. Dit laatste zal echter zelden of nooit nuttig zijn omdat `$_GET` zelf al een superglobal is.

U kunt de superglobals altijd en overal in uw script aanroepen, **ook binnen functies**:

```
//*****
function browser()
{
    echo htmlspecialchars($_SERVER['HTTP_USER_AGENT']);
}
//*****

// Dit geeft het gewenste resultaat
browser();
```

Merk op dat bovenstaande functie goed werkt terwijl onderstaande code, waarbij de waarde eerst in een **lokale variabele** weg wordt geschreven, **niet** zondermeer zou werken:

```
$agent = $_SERVER['HTTP_USER_AGENT'];

//*****
function browser()
{
    echo $agent;
}
//*****

// Dit geeft GEEN resultaat
browser();
```

U krijgt dit alleen werkend als u de variabele als argument mee zou geven aan de functie of door de variabele te globaliseren (zie hiervoor).

7.3.18.5 Constanten

Constanten zijn eveneens **per definitie global**. U hoeft ze dus niet te globaliseren!

```
// Constante definiëren
define("CONSTANTE", "Constanten zijn globaal");

// Functie definiëren
```

```
// *****
function constante_test()
{
    echo CONSTATE;
}
// *****

// Functie aanroepen
constante_test ();
```

Dit geeft:

Constanten zijn globaal

Deze eigenschap maakt het gebruik van constanten **ideaal voor configuratiebestanden** die u via includes kunt insluiten (zie ook paragraaf 1.4).

7.3.19 Render Functies

Het is in PHP mogelijk om na het openen van de functie van PHP naar HTML modus te switchen en voor de eindaccolade weer terug naar PHP te gaan. Dit stelt u in staat om in de function body HTML te gebruiken:

```
<?php
// *****
function toon_medewerkers($medewerkers)
{
    ?>
    <table>
    <tr>
        <th>ID</th>
        <th>Voornaam</th>
        <th>Achternaam</th>
        <th>Afdeling</th>
        <th>Toestelnummer</th>
    </tr>
    <?php foreach ($medewerkers as $medewerker): ?>
        <tr>
            <td><?php echo $medewerker['id'] ?></td>
            <td><?php echo $medewerker['voornaam'] ?></td>
            <td><?php echo $medewerker['achternaam'] ?></td>
            <td><?php echo $medewerker['afdeling'] ?></td>
            <td><?php echo $medewerker['toestelnummer'] ?></td>
        </tr>
    <?php endforeach; ?>
    </table>
    <?php
}
// *****
?>
```

Dit soort functies worden **render functies** genoemd. Het is goed om te weten dat ze bestaan, maar u kunt ze beter **niet gebruiken**... Als u grote stukken HTML wilt wegschrijven dan kunt u beter templates gebruiken. Zo voorkomt u dat u PHP en HTML logica in de PHP scripts mixt. Als u toch HTML wilt genereren vanuit een (hulp)functie dan kunt u beter een string HTML retourneren. U heeft dan volledige controle over de opbouw, indeling en witruimte. Bovendien kunt u makkelijk delen becommentariëren en de uitvoer hergebruiken (bijvoorbeeld voor in een HTML email, of er nog een zoek-en-vervang opdracht op loslaten. We zullen zo een voorbeeld van een dergelijke functie zien.

7.3.20 Waarden retourneren

Functies worden in de praktijk heel vaak gebruikt om **waarden te berekenen** en deze **waarden** vervolgens **terug te geven** aan de functieaanroep. Dit in tegenstelling tot de **rechtstreekse output** van de functies die we tot nu toe gemaakt hebben. Die worden in sommige talen eerder met de term **(sub)routines** aangeduid.

Als u wilt dat uw functie waarden teruggeeft, dan moet er in uw functie definitie een regel staan welke waarde u terug wilt geven. Dat doet u met het keyword **return**. Hiermee wordt automatisch de functie beëindigd. Het geheel komt er dan zo uit te zien:

```
function functie_naam ($parameter1, $parameter2)
{
    // codeblok waarin de argumenten gebruikt worden als variabelen
    // ...
    // uiteindelijk:
    return bepaalde_waarde;
}
```

De hierboven gedefinieerde functie doet uit zichzelf nog niets. Er is alleen bepaalde functionaliteit voor het script beschikbaar gemaakt. Om resultaat te krijgen moet u de functie eerst aanroepen, waardoor de return waarde beschikbaar wordt gesteld. U moet dan echter wel gelijk iets met de geretourneerde waarde doen, anders heeft het aanroepen weinig zin. Er moet met andere woorden wel een **side-effect** zijn. U zou de geretourneerde waarde bijvoorbeeld naar scherm kunnen schrijven of op kunnen slaan in een variabele om daar vervolgens mee verder te werken:

```
echo functie_naam($waarde_argument1, $waarde_argument2);
// of
$resultaat = functie_naam($waarde_argument1, $waarde_argument2);
```

Laten we als voorbeeld eens een simpele functie schrijven die twee willekeurige getallen bij elkaar kan optellen en het resultaat teruggeeft:

```
function optellen($a, $b)
{
    $som = $a + $b;
    return $som;
}
```

U zou de functie als volgt aan kunnen roepen:

```
// LET OP: dit heeft geen zin:
optellen(1, 3);
```

Maar dat levert verder niets op, want er is dus geen **side-effect**. Dit is een logischere toepassing:

```
// haal getallen uit formulier
$getal1 = intval($_POST['getal1']);
$getal2 = intval($_POST['getal2']);

// side-effect: resultaat wegschrijven
echo optellen($getal1, $getal2);
```

U zou het resultaat van een functie ook eerst kunnen opslaan in een (nieuwe) globale variabele om daar vervolgens mee verder te werken:

```
// side-effect: resultaat opslaan in variabele
$result = optellen(1, 3);
```

Een functie kan meerdere return statements bevatten, maar er kan altijd maar één waarde worden teruggegeven. Dit komt vaak voor in combinatie met een controle structuur als `if else`. Het uitvoeren van het `return` statement betekent automatisch dat de functieaanroep wordt beëindigd:

```
<?php
//*****
function test($boolean)
{
    if ($boolean) {
        return true;
    } else {
        return false;
    }
}
//*****

$check = false;

if (!test($check)) {
    echo 'mislukt!';
}

?>
```

Een functie kan natuurlijk wel een **array** teruggeven die **meerdere waarden** bevat!

7.3.21 Typed return values

In PHP7 kunt u ook het datatype van de return value opgeven. Dit gebeurt niet, zoals u misschien zou verwachten, bij het `return` statement maar bovenin de functie declaratie:

```
function add(int $a, int $b) : int
{
    return $a + $b;
}
```

Dit geeft aan dat het return type van de functie een integer moet zijn. Wat natuurlijk ook logisch is gezien de type hints voor de parameters. Als u een functie als deze heeft:

```
function test() : int
{
    return true;
}
```

Dan zal in **coercive mode** (zie hierboven) geprobeerd te worden het geretourneerde datatype te converteren.

```
var_dump(test()); // geeft int(1)
```

Maar in **strict mode** geeft dat de volgende foutmelding:

Fatal error: Uncaught TypeError: Return value of test() must be of the type integer, boolean returned

7.3.22 PHPDoc Blocks

Documentatie is heel belangrijk. Als het goed is, is uw code al deels *self-documenting* vanwege de zorgvuldige gekozen, beschrijvende namen van uw mappen en bestanden en de daarin gebruikte variabelen, constanten en functies.

Modulariseren

U kunt met behulp van commentaar uitleg geven wat u doet binnen de code maar bij functiebibliotheken is het erg handig om dat in een speciaal afgesproken formaat te doen.

Binnen PHP is de standaard hiervoor **phpDoc** (<http://en.wikipedia.org/wiki/PHPDoc>). Het is een in PHP geschreven standaard, gebaseerd op JavaDoc.

Het met phpDoc becommentariëren van code heeft 3 belangrijke voordelen boven een eigen commentaar stijl:

1. Het zorgt voor leesbaar commentaar op een standaard manier die aanmoedigt om code van uitleg te voorzien
2. Met een documentation generator zoals **phpDocumentor** (<http://www.phpdoc.org/>) kunt u automatisch nette documentatie genereren, bijvoorbeeld in de vorm van een klikbare website
3. IDE's zoals Zend Studio, phpStorm, NetBeans en Aptana Studio kunnen er gebruik van maken voor code completion, type hinting en debugging! Bovendien kunnen sommige IDE's het commentaar ook voor u genereren.

Het idee is dat u uw commentaar schrijft in de vorm van PHP DocBlocks. Een **PHP DocBlock** is een speciale vorm van meerregelig commentaar en kan er voor onze functie `html_table()` bijvoorbeeld zo uit zien:

```
/**
 * return HTML table representation of 2-dimensional data array
 *
 * This function takes an 2-dimensional array of data. It assumes the
 * outer array is numerical indexed which contains associative arrays
 * as typically returned by a db_fetch_all function. It builds a HTML table
 * string with headers for all the keys
 *
 * @param array $rows
 * @return string $html representing HTML table
 */
function html_table($rows)
{
    ...
    return $html;
}
```

Merk ook de data types voor de parameter en het return type op!

U plaatst het blok gelijk boven de functie definitie. In een editor als NetBeans typt u daar simpelweg `/**` [Enter] om een basisblok neer te zetten!

Een goede uitleg van de DocBlock syntax vindt u op:

http://manual.phpdoc.org/HTMLSmartyConverter/HandS/phpDocumentor/tutorial_phpDocumentor.howto.pkg.html

7.3.23 Functies: PRAKTIJK

Met de nieuwe kennis over functies kunnen we de code in onze doorlopende oefening grondig reorganiseren!

7.3.23.1 functie `db_connect()`

Een goed voorbeeld van een functie die een waarde retourneert is een functie die een database verbinding maakt en de link identifier retourneert. Wanneer we uitgaan van onze inmiddels opgebouwde code base dan zou deze er zo uit kunnen zien:

```

//*****
function db_connect()
{
    // Verbinding met MySQL Server maken
    $conn = mysqli_connect('localhost', 'root', '');
    if (!$conn) {
        $title = 'Database Error';
        $error = 'Verbinden met database server mislukt...';
        include 'templates/db_error.tpl.php';
        exit();
    }

    // Charset specificeren
    if (!mysqli_set_charset($conn, 'utf8')) {
        $title = 'Database Error';
        $error = "Character set kon worden niet ingesteld. " .
            "De huidige character set is: " .
            mysqli_character_set_name($conn);
        include 'templates/db_error.tpl.php';
        exit();
    }

    // Database selecteren
    $db = mysqli_select_db($conn, 'cursus');
    if (!$db) {
        $title = 'Database Error';
        $error = 'Database is niet beschikbaar...';
        include 'templates/db_error.tpl.php';
        exit();
    }

    return $conn;
}
//*****

```

Denk er aan dat de functie in ieder geval de link identifier `$conn` moet retourneren! Merk op dat dit de `mysqli` identifier is en dat deze dus `false` is als het niet lukt om de verbinding te maken. De error templates hebben dan echter hun werk al gedaan!

Wanneer u een dergelijke functie in een functiebibliotheek plaatst en insluit dan kunt u vervolgens heel makkelijk verbinding maken met een database:

```
$conn = db_connect();
```

Oefening

Voeg aan `common.inc.php` de functies `db_connect()` en eventueel `db_close()` toe. `db_connect()` retourneert na het openen van de verbinding een linkidentifier en `db_close()` neemt deze als argument aan om de verbinding te kunnen sluiten. Test de functies door ze na aan te roepen in `medewerkers_tonen.php`.

7.3.23.2 Database Abstraction Layer

U heeft zo in feite al een begin gemaakt met een eenvoudige **DAL (Database Abstraction Layer)**. Als u in een later stadium gebruik wilt gaan maken van bijvoorbeeld een PostgreSQL database dan kan dit door de code binnen de functie aan te passen:

```

//*****
function db_connect()
{
    $conn = pg_connect('host      = ' . DB_SRV . '
                       port      = 5432
                       dbname    = ' . DB_NAME . '
                       user      = ' . DB_USR . '

```

Modulariseren

```
        password = ' . DB_PWD);

if (!$conn) {
    $title = 'Database Error';
    $error = 'Verbinden met database server mislukt...';
    include 'templates/db_error.tpl.php';
    exit();
}

return $conn;
}
//*****
```

Helemaal mooi is het om de ondersteuning gelijk al in de functie in te bouwen:

```
//*****
function db_connect($rdbms=RDBMS)
{
    switch (strtolower($rdbms)) {
        case 'postgresql':
            $conn = pg_connect('host      = ' . DB_SRV . '
                                port      = 5432
                                dbname    = ' . DB_NAME . '
                                user      = ' . DB_USR . '
                                password = ' . DB_PWD);

            break;
        case 'mysql':
        case 'mysqli':
        default:
            $conn = mysqli_connect(DB_SRV, DB_USR, DB_PWD, DB_NAME);
            break;
    }

    if (!$conn) {
        $title = 'Database Error';
        $error = 'Verbinden met database server mislukt...';
        include 'templates/db_error.tpl.php';
        exit();
    }

    return $conn;
}
//*****
```

Waarbij u de constanten RDBMS in uw configuratiebestand opneemt zodat u deze *at design time* kunt aanpassen...

Denk er dan wel aan dat u dit dan voor **alle** (MySQLi) **database functies** moet doen: u moet ze opnemen in **wrapper functies** die allemaal consequent met `db_` beginnen. U maakt dan ook een `db_query()` waarin u minimaal `mysqli_query()` opneemt en eventueel ook een `pg_query()`. En een `db_error()` en een `db_fetch_assoc()` etc..

Maar als u behoefte heeft aan een database abstraction layer dan kunt u misschien beter gelijk een goede nemen zoals PEAR DB, AdoDb of PDO... Maar hopelijk maakt dit het principe erachter duidelijk!

7.3.23.3 functie `get_medewerkers()`

U weet inmiddels hoe u een **resultset** in een **tweedimensionale array** kunt zetten. De database verbinding kan nu in principe worden gesloten en de array kan vervolgens als argument aan de template die de data zal tonen. U kunt uw code nog beter leesbaar maken door het ophalen van de data in een functie te gieten, zodat u hoofdprogramma er dan als volgt uit kan zien:


```
...
// haal medewerkers op en zet in 2dim array
$medewerkers = get_medewerkers();

// sluit template (of hulpfunctie) in die medewerkers toont
...
```

De functie om de gegevens op te halen kan er zo uit zien:

```

/*****
function get_medewerkers()
{
    // open de database verbinding
    $conn = db_connect();

    $query = "SELECT voornaam, achternaam, afdeling
              FROM medewerkers";

    $result = mysqli_query($conn, $query);
    if (!$result) {
        $title = 'Database Error';
        $error = mysqli_error($conn);
        include 'templates/db_error.tpl.php';
        exit();
    }

    // maak een lege array aan
    $medewerkers = array();

    while ($row = mysqli_fetch_assoc($result)) {
        // zet records (zelf ook arrays) in medewerker array
        $medewerkers[] = $row;
    }

    // database verbinding sluiten
    db_close($conn);

    return $medewerkers;
}
*****/

```

In bovenstaand voorbeeld wordt de database verbinding gemaakt en gesloten binnen de functie, maar uiteraard had u dat ook buiten de functie kunnen doen. Het gaat er echter om dat u nu, na het sluiten van de verbinding, nog steeds de beschikking hebt over de data uit de resultset.

Let op: Het is heel belangrijk dat u zich realiseert dat wanneer u de database verbinding **buiten** een functie maakt en u de openstaande verbinding wilt gebruiken **binnen een functie** dat dit **niet** (zomaar) **zal lukken**. U moet de link identifier `$conn` dan als **argument** de functie binnen te halen, of als **global** in de functie.

Stel, u heeft de volgende code:

```
$conn = db_connect();
$afdelingen = get_afdelingen();
db_close($conn);
```

Dan moet, als u binnen `get_afdelingen()` een query wilt uitvoeren, de link identifier `$conn` wel beschikbaar zijn! U kunt dat oplossen door de link identifier als argument mee te geven:

```
$conn = db_connect();
$afdelingen = get_afdelingen ($conn);
db_close($conn);
```

Modulariseren

Of u kunt de code laten zoals hij is en de link identifier als global binnen de functie definitie voor `get_afdelingen()` opnemen:

```
function get_afdelingen()
{
    global $conn;
    ...
}
```

De laatste oplossing heeft als voordeel dat uw code wat schoner blijft. In het geval van een resource als een link identifier is het per ongeluk overschrijven van een globale variabele binnen een functie ook niet echt een issue.

7.3.23.4 Generieke functies: `get_rows()`

Hopelijk bent u geïnspireerd door bovenstaande manier van werken en wilt u het ook toepassen bij andere gegevens uit de database, bijvoorbeeld voor de afdelingen:

```
...
// haal afdelingen op en zet in 2-dim array
$afdelingen = get_afdelingen();

// toon afdelingen in template (of hulpfunctie)
...
```

U zult dan bij het schrijven van de code merken dat u nogal wat stappen aan het kopiëren bent. Eigenlijk verschillen de functies `get_medewerkers()` en `get_afdelingen()` alleen in de query van elkaar.

We kunnen dus verder abstraheren:

```
/** *****
function get_rows($query)
{
    global $conn;

    $result = mysqli_query($conn, $query);
    if (!$result) {
        $title = 'Database Error';
        $error = mysqli_error($conn);
        include 'templates/db_error.tpl.php';
        exit();
    }

    $rows = array();

    while($row = mysql_fetch_assoc($result)) {
        $rows[] = $row;
    }

    return $rows;
}
/** *****
```

Deze functie neemt een query als argument en retourneert een tweedimensionale array met data. Deze variant maakt gebruik van een global `$conn`, en gaat er van uit dat er al een database verbinding buiten de functie gemaakt is.

U kunt dat echter eventueel ook in de functie zelf detecteren en de verbinding maken indien die er niet is:

```

//*****
function get_rows($query, $conn='')
{
    // is er wel een link identifier $conn?
    if (!$conn) global $conn;
    if (!$conn) $conn = db_connect();

    ...
}
//*****

```

Nu worden de `get_*` functies heel compact. Voor de medewerkers:

```

//*****
function get_medewerkers()
{
    $query = "SELECT * FROM medewerkers";
    $medewerkers = get_rows($query);
    return $medewerkers;
}
//*****

```

En voor de afdelingen:

```

//*****
function get_afdelingen()
{
    $query = "SELECT * FROM afdelingen";
    $afdelingen = get_rows($query);
    return $afdelingen;
}
//*****

```

De functies verschillen nu inderdaad alleen nog op de query! Merk ook op dat u nog compacter kunt noteren door de functies gelijk af te sluiten met `return get_rows($query);` maar dat is wellicht iets minder leesbaar...

Onze oplossing is nu alleen nog heel duidelijk gekoppeld aan een database, heel specifiek MySQL. We kunnen dat laatste mooi oplossen door al onze `mysqli_*` functies te omsluiten met algemene `db_*` **wrappers**, net zoals we bij de functie `db_connect()` hebben gedaan. `get_rows()` zou er dan uiteindelijk zo uitzien:

```

//*****
function get_rows($query)
{
    global $conn;

    $result = db_query($conn, $query);
    if (!$result) {
        $title = 'Database Error';
        $error = db_error($conn);
        include 'templates/db_error.tpl.php';
        exit();
    }

    $rows = array();

    while($row = db_fetch_assoc($result)) {
        $rows[] = $row;
    }

    return $rows;
}

```

```
}
//*****
```

De database functies zet u bij elkaar in een bestand `functions.database.inc.php` dat u insluit in uw `common.inc.php`. Als u nu een andere database wilt gaan gebruiken dan hoeft u alleen de `mysql_*` functies in dit bestand te vervangen door de benodigde functies voor uw database, bijvoorbeeld `mysqli_*`, `mssql_*` of `pg_*`, als u respectievelijk met MySQLi, Microsoft SQL Server of PostgreSQL wilt gaan werken. Sterker nog, u kunt alle functies al binnen de wrappers opnemen en dan met een configuratie constante bepalen welke u wilt gebruiken voor uw applicatie, zoals al eerder is besproken.

7.3.23.5 functie `db_error()`

Met de constante `DEBUG` in het configuratiebestand kunt u aangeven of u in *debug modus* bent of niet. U kunt dit bijvoorbeeld gebruiken in een zelfgeschreven functie `db_error()` die hierboven al ingezet werd:

```
//*****
function db_error($conn)
{
    if (DEBUG) {
        $error = mysqli_error($conn);
    } else {
        $error = "Database opdracht is helaas mislukt...";
    }
    return $error;
}
//*****
```

In debug modus zorgt een foute query ervoor dat `mysqli_error()` een zinnige foutmelding geeft, bijvoorbeeld:

```
Unknown column 'telefon' in 'field list'
```

terwijl het een "gebruikersvriendelijke" foutmelding wordt als `DEBUG` de waarde `false` heeft:

```
Query mislukt.
```

In het echt zult u natuurlijk een iets informatievere melding aan de eindgebruiker willen geven...

7.3.23.6 DATA Folder

Op een zelfde manier kunt u ook generieke functies maken voor het toevoegen van een nieuwe medewerker en het aanpassen of verwijderen van een bestaande. En idem voor een afdeling.

Nu kunt u al uw data functies voor medewerkers bij elkaar zetten in een bestand `medewerkers.php` en dat in een map zetten die u bijvoorbeeld `/data` (of `/queries`) noemt. Hetzelfde doet u dan voor afdelingen, etc. Zo heeft u dan ook een centrale plek waar u al uw queries kunt aanpassen!

7.3.23.7 Template helper: `html_table()`

Het op een generieke manier printen van een 2-dimensionale array naar een HTML tabel is in een template vrij lastig en het leest ook niet heel prettig in template syntax. In templates wilt u waarschijnlijk ook vooral de controle over de structuur zoveel mogelijk in eigen hand houden, denk maar aan verschillende (format) functies die u over verschillende velden heen wilt halen (om een datum, prijs of boolean op te maken of bijvoorbeeld een afbeelding in te sluiten. Maar met een template hulpfunctie of **template helper** als `html_table()` heeft u toch het gemak om in ieder geval snel even een tabel met tableheaders te kunnen genereren:

```

/**
 * return HTML table representation of 2-dimensional data array
 *
 * This function takes an 2-dimensional array of data. It assumes the
 * outer array is numerical indexed which contains associative arrays
 * as typically returned by a db_fetch_all function. It builds a HTML table
 * string with headers for all the keys
 *
 * @param array $rows
 * @return string $html representing HTML table
 */

function html_table($rows)
{
    // begin table string
    $html = '';

    if ($rows) {
        // begin table
        $html .= '<table>' . PHP_EOL;

        // begin tablerow for table header
        $html .= ' <tr>' . PHP_EOL;

        // create table header for each key
        foreach (array_keys($rows[0]) as $key) {
            $html .= ' <th>' . $key . '</th>' . PHP_EOL;
        }
        // end table row

        // create table row for each record in the array
        $html .= ' </tr>' . PHP_EOL;
        foreach ($rows as $row) {
            $html .= ' <tr>' . PHP_EOL;

            // create table cell for each value
            foreach ($row as $value) {
                $html .= ' <td>' . $value . '</td>' . PHP_EOL;
            }
            // end table row
            $html .= ' </tr>' . PHP_EOL;
        }
        // end table
        $html .= '</table>' . PHP_EOL;
    }

    // return table string
    return $html;
}

```

Merk op hoeveel controle u over de uitvoer heeft en hoe duidelijk het commentaar werkt...

U kunt de functie als volgt gebruiken:

```

// dit is demo.php
include 'includes/common.inc.php';

$conn = db_connect();
$medewerkers = get_medewerkers();

include 'templates/demo_print_';

```

De template roept de hulp functie aan om de medewerkers in de tabel te tonen:

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

<?php if ($medewerkers): ?>
    <?php html_table($medewerkers) ?>
<?php else: ?>
    <p class="error">Helaas, geen medewerkers gevonden...</p>
<?php endif; ?>

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

Vandaar de aanduiding **template helper**. Andere handige template helpers zijn bijvoorbeeld functies om snel een hyperlink te kunnen maken of een selectbox te selecteren!

Let op: vergeet niet dat u de uitvoer van de functie moet echoën...

Met name voor formulier elementen werkt dit heel prettig omdat u anders relatief veel markup zelf moet schrijven en deze daarbij vaak ook van de nodige controlestructuren moet worden voorzien.

7.3.23.8 Template helper: `html_select()`

Dit is niet handig:

```
<select name="afdelingid">
    <option>Kies een afdeling...</option>

    <?php foreach($afdelingen as $afdeling): ?>
        <?php if ($medewerker['afdelingid'] == $afdeling['id']) : ?>
            <option value="<?= $afdeling['id'] ?>" selected>
                <?= $afdeling['afdeling'] ?>
            </option>
        <?php else: ?>
            <option value="<?= $afdeling['id'] ?>">
                <?= $afdeling['afdeling'] ?>
            </option>
        <?php endif; ?>
    <?php endforeach; ?>

</select>
```

Het enige verschil in de `if` en de `else` is de aanwezigheid van `selected`... Kan eventueel zo:

```
<select name="afdelingid">
    <option>Kies een afdeling...</option>
    <?php foreach($afdelingen as $afdeling): ?>
        <option value="<?= $afdeling['id'] ?>"
            <?php if ($medewerker['afdelingid'] == $afdeling['id']): ?>
                selected
            <?php endif; ?><?= $afdeling['afdeling'] ?>
        </option>
    <?php endforeach; ?>
</select>
```

Of met de ternary operator:

```
<select name="afdelingid">
    <option>Kies een afdeling...</option>
    <?php foreach($afdelingen as $afdeling): ?>
        <option value="<?= $afdeling['id'] ?>"
            <?= $medewerker['afdelingid'] == $afdeling['id'] ? 'selected' : '' ?>
            <?= $afdeling['afdeling'] ?>
        </option>
    <?php endforeach; ?>
</select>
```

Maar dat leest nog steeds niet echt lekker... Een template helper functie als deze maakt het makkelijker:

```
/**
 * return HTML selectbox of 2-dimensional data array
 * (with selected option)
 *
 * @param string $name the name of the selectbox
 * @param array $rows the 2-dimensional array
 * @param string $option_field the name of the (default) option
 * @param string $value_field the value for the value attribute (defaults to
id)
 * @param string $select_value the value to compare $value_field with
 * (which makes an option selected)
 * @return string $html representing the generated selectbox
 */

function html_select($name, $rows,
                    $option_field, $value_field='id',
                    $select_value='')
{
    // begin string
    $html = '';

    // begin selectbox with name and id
    // the id is useful for labeling, styling and scripting
    $html .= '<select name="' . $name . '" id="' . $name . '">' . PHP_EOL;

    // if there is no select_value (eg in an INSERT scenario)
    if (! $select_value) {
        // add default option with choose an 'option_field' as text
        // value is an empty string
        // this makes it easier to validate when it is an required field
        // otherwise the value would be the text in the option element...
        $html .= ' <option value="">';
        $html .= 'Kies een ' . $option_field . '&hellip;';
        $html .= '</option>' . PHP_EOL;
    }

    foreach ($rows as $row) {
        $html .= ' <option value="' . $row[$value_field] . '"';
        if ($row[$value_field] == $select_value) {
            $html .= ' selected';
        }
        $html .= '>';
        $html .= $row[$option_field];
        $html .= '</option>' . PHP_EOL;
    }
    // end selectbox
    $html .= '</select>' . PHP_EOL;

    // return selectbox string
    return $html;
}
```

U kunt nu de afdelingen dropdownlist voor een INSERT template eenvoudig zo genereren:

```
<?= html_select('afdelingid', $afdelingen, 'afdeling', 'id') ?>
```

En in de UPDATE template zo:

```
<?= html_select('afdelingid', $afdelingen, 'afdeling', 'id',
$medewerker['afdelingid']) ?>
```

Modulariseren

Denk aan de (shorthand) echo!

Disclaimer: Helper functies om HTML te genereren kunnen heel nuttig zijn, maar blijven echter altijd ook een beetje tricky. Want u zult altijd wel weer situaties tegenkomen waar u behoefte heeft aan zaken die op dat moment (nog) niet in de functie zitten. Denk bij de tabel functie aan links voor insert, update en delete. En moet je daar dan de tekst en de href-waarde nog van kunnen aanpassen? En een class of id zetten is natuurlijk altijd handig...

Denk bij de selectbox functie ook aan een class, of het werken met optgroups, een blanco item toevoegen en dan met welke tekst. En wat als dat multilingual moet zijn?

U kunt uw functies natuurlijk blijven uitbreiden met meer argumenten, maar dit worden er al snel te veel. Bovendien wilt u liever de volgorde niet aanpassen om code die er reeds gebruik van maakt niet te breken.

Een alternatief kan zijn om dit met OOP op te lossen door het zetten van properties, maar dan wordt het alsnog vaak heel wollig. Gewoon uitschrijven in de template blijft vaak toch het meest flexibel!

7.3.23.9 Template variabelen in array

Op dit moment initialiseert u variabelen los in de global scope om ze daarna in de template te gebruiken. Met een paar variabelen is dat prima, maar zo gauw er meer komen wordt het al snel onoverzichtelijk.

U kunt dan ook beter alles wat u aan de template gaat geven in een array zetten, die u bijvoorbeeld `$page`, `$view` of `$data` noemt:

```
...
// haal medewerkers op
$data['medewerkers'] = get_medewerkers();
...
$data['title'] = 'Medewerkers Overzicht';
...
// etc...
```

U kunt de array dan zo veel uitbreiden als nodig is...

7.3.23.10 Functie `include_template()`:

Nadeel is dan wel dat u in de template niet meer zomaar `echo $title` kan doen maar dat als `echo $data['title']` moet doen. Maar er is een oplossing! En een die ook een ander "probleem" oplost: De manier van insluiten is nu namelijk nogal star. Als het template pad zou wijzigen of de extensie van de templates dan moet u dat in alle controllers aanpassen. Niet handig!

U kunt er voor kiezen om het insluiten met een custom functie te doen, die u in `common.inc.php` definieert. Bijvoorbeeld zoiets:

```
// *****
function include_template($name, $data)
{
    // bouw pad naar template op met constante uit config
    $template = TEMPLATE_PATH . $name . '.tpl.php';

    // als de template bestaat sluit hem dan in
    if (file_exists($template)) {
        // sluit content in
        include($template);
    } else {
        die('template ' . $template . ' is niet beschikbaar...');
    }
}
```



```

    }
}
// *****

```

Omdat u nu de template binnen een functie insluit, geldt voor de template de local scope. De globale variabele `$data` moet u daarom als argument meegeven.

7.3.23.11 *extract variabelen*

Voor het gemak kunt u met de standaard PHP functie `extract()` de items uit de array als gewone variabelen beschikbaar stellen. `$data['title']` wordt dan `$title` (een string) en `$data['medewerkers']` wordt `$medewerkers` (een 2-dim array). U hoeft dan niets aan uw reeds eerder gemaakte templates aan te passen!

```

// *****
function include_template($name, $data)
{
    // bouw pad naar template op met constante uit config
    $template = TEMPLATE_PATH . $name . '.tpl.php';

    // als de template bestaat sluit hem dan in
    if (file_exists($template)) {

        // extract variabelen in huidige scope
        extract($data, EXTR_SKIP);

        // sluit content in
        include($template);

    } else {
        die('template ' . $template . ' is niet beschikbaar...');
    }
}
// *****

```

Het optionele tweede argument is een constante, waarmee u aangeeft wat er moet gebeuren als er al een variabele met dezelfde naam aanwezig is. Met `EXTR_SKIP` geeft u aan dat als een variabele reeds bestaat, dat deze niet wordt overschreven. De kans dat dit gebeurt in een template is natuurlijk bijzonder klein omdat het niet de bedoeling is om hierin met PHP variabelen te definiëren...

Met een array `$data` gebruikt u de `include_template()` functie als volgt:

```

$conn = db_connect();
$data['medewerkers'] = get_medewerkers($conn);

$data['title'] = 'Medewerkers overzicht';

include_template('medewerkers_tonen', $data);

```

U kunt er echter ook voor kiezen om de variabelen pas bij het aanroepen in een array te zetten:

```

$conn = db_connect();
$medewerkers = get_medewerkers($conn);

$title = 'Medewerkers overzicht';

include_template('medewerkers_tonen', array(
    'title' => $title,
    'medewerkers' => $medewerkers
));

```

Modulariseren

Deze werkwijze komt u in diverse PHP frameworks tegen. Wij kiezen tijdens de cursus in principe voor de eerste methode.

Oefening	<i>Voer de <code>include_template()</code> functie door in de applicatie. Denk eraan hem ook toe te passen in database functies zoals <code>db_connect()</code> en <code>get_rows()</code>!</i>

7.3.23.12 Includes organiseren

U zult in de loop der tijd waarschijnlijk steeds meer handige functies schrijven om in uw applicatie(s) te gebruiken. Het is dan niet handig om deze allemaal in een grote functiebibliotheek te zetten. U kunt ze beter op functionaliteit groeperen en alleen die includes insluiten die u per script nodig heeft. U zou bijvoorbeeld zo te werk kunnen gaan:

- `functions.db.inc.php`
- `functions.debug.inc.php`
- `functions.form.inc.php`
- etc..

Includes kunnen elkaar insluiten zoals u zag bij het voorbeeld met het configuratiebestand, maar het is natuurlijk wel zaak om dit zo overzichtelijk mogelijk te houden en niet te lange 'kettingen' te maken!

7.3.23.13 functie `print_array()`

Denk aan een functie `print_array()` die u in een include `functions.debug.inc.php` kunt zetten die u alleen insluit als u aan het ontwikkelen bent (door de constante `DEBUG` op `true` te zetten in uw `config.inc.php`):

```
//*****
// DEBUG Functions
//*****

//*****
function print_array($arr, $details=false, $abort=false)
{
    echo PHP_EOL;
    echo '<pre style="padding: 1em;
        border: 1px black solid;
        background-color: #EEE;
        margin-bottom: 1em">';
    echo PHP_EOL;

    if ($details) {
        var_dump($arr);
    } else {
        print_r($arr);
    }

    echo PHP_EOL;
    echo '</pre>';
    echo PHP_EOL;

    if ($abort) {
        exit();
    }
}
//*****
```

Als u in uw `config.inc.php` dan de debug functions in laat sluiten (alleen wanneer de `DEBUG` constante op `TRUE` staat) dan hoeft u wanneer u de array `$week` netjes geformatteerd weg wilt schrijven enkel dit te doen:

```
print_array($week);
```

En dat levert dan gelijk zoiets op:

```
Array
(
    [0] => Zondag
    [1] => Maandag
    [2] => Dinsdag
    [3] => Woensdag
    [4] => Donderdag
    [5] => Vrijdag
    [6] => Zaterdag
)
```

7.3.24 VERDIEPING: Callback functies

Een **callback functie**, of kortweg een **callback** of **callable**, is een functie die als argument kan worden doorgegeven aan een andere functie, waarbinnen de callback code dan zal worden uitgevoerd.

Deze techniek, die u mogelijk ook al bent tegengekomen in JavaScript, wordt gebruikt om flexibele, modulaire code te schrijven. Met callbacks kunt u eenvoudiger wisselende code laten uitvoeren, bijvoorbeeld bij het optreden van een bepaald event. Dat is handiger dan proberen alles op te nemen in een alles overkoepelende functie of het opnemen van dezelfde logica in twee of meer functies.

Een aantal in PHP ingebouwde functies maakt er gebruik van, bijvoorbeeld array functies als `array_map()` en `array_filter()`, maar ook een functie als `set_exception_handler()` die we later tegen zullen komen.

Wanneer u in een door u zelf aangemaakte functie (een *user defined function*) gebruik wilt maken van een callback, dan geeft u de naam van de functie als string door. Binnen uw functie definitie maakt u dan gebruik van een **variabele functienaam** of beter nog van de `call_user_func()` functie. Een andere optie is gebruik te maken van een **anonieme functie**. Die mogelijkheid bespreken we zo meteen.

Het idee is met de volgende code op een eenvoudige manier goed aanschouwelijk te maken:

```
function transform($callback, $var)
{
    return $callback($var);
}

echo transform('strtoupper', 'abc'); // ABC
echo transform('ucfirst', 'abc');   // Abc
```

Bovenstaande code maakt gebruik van bestaande PHP functies, maar u kunt de callbacks natuurlijk ook zelf definiëren:

```
function a() {echo 'uitvoer callback functie a...';}
function b() {echo 'uitvoer callback functie b...';}

function demo_with_callback($x, $callback)
{
    echo "doe iets met $x...<br>";
    echo "roep dan functie $callback aan...<br>";
}
```

Modulariseren

```
$callback();  
}  
  
demo_with_callback('x', 'a');
```

Het tweede argument 'a' wordt een aanroep voor functie a() en dit geeft het volgende resultaat:

```
doe iets met x...  
roep dan functie a aan...  
uitvoer callback functie a...
```

Als u de functie met een andere callback aanroept, dan wordt deze uitgevoerd:

```
demo_with_callback('y', 'b');
```

Dit geeft:

```
doe iets met y...  
roep dan functie y aan...  
uitvoer callback functie b...
```

U zult minimaal willen controleren of het tweede argument de naam van een bestaande functie is, bijvoorbeeld:

```
...  
if (is_string($callback) && function_exists($callback)) {  
    echo "roep dan functie $callback aan<br>";  
    $callback();  
}  
...
```

Nu geeft het aanroepen van een niet bestaande functie tenminste geen fatal error:

```
demo_with_callback('z', 'c');
```

```
doe iets met z...
```

Beter is het echter om voor de controle de speciaal daarvoor bedoelde functie `is_callable()` te gebruiken, deze werkt ook als de callback functie de method van een object is of een statische method van een class. Om diezelfde reden kunt u ook beter de functie `call_user_func()` gebruiken om de callback functie uit te laten voeren:

```
...  
if (is_callable($callback)) {  
    echo "roep dan functie $callback aan<br>";  
    call_user_func($callback);  
}  
...
```

Als de callback functie parameters heeft dan geeft u deze komma gescheiden mee na de naam:

```
call_user_func($callback, $arg1, $arg2, ...);
```

Of u zet de parameters in een array en maakt gebruik van `call_user_func_array()`:

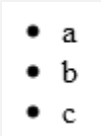
```
call_user_func_array($callback, $args);
```

Zie ook:

- <http://www.phpriot.com/articles/php-callbacks>
- <http://blog.cribznetwork.com/2013/01/php-5-function-callbacks/>
- <http://blog.peterhaza.no/callback-functions-explained/>

Verdieping *Maak in de map oefeningen/ een script `callback.php` aan. Definieer hierin een functie `show_range($min, $max, $callback)`. In de functie roept u eerst een user defined functie `get_data($min, $max)` aan. De functie retourneert een array `$data` met een range die loopt van `$min` tot `$max` (gebruik hiervoor de PHP `range()` functie). De callback functies die u gebruikt om `$data` te tonen noemt u `html_list()` en `html_table()`.*

`show_range("a", "c", "html_list")` geeft:



en `show_range("a", "c", "html_table")` geeft:



7.3.25 VERDIEPING: Anonieme functies (Closures)

Wanneer u gebruik maakt van callback functies op de hierboven beschreven manier dan zult u soms functies moeten definiëren die u misschien maar een keer nodig heeft:

```
function is_even($val)
{
    return $val % 2 == 0;
}
$arr_even = array_filter($arr, 'is_even');

function square($val)
{
    return $val * $val;
}
$arr_square = array_map('square', $arr);
```

Vanaf PHP 5.3 kunt u dit oplossen met **anonymous functions**, ook wel **closures** of **lambda functions** genoemd. Dergelijke functies zijn anoniem omdat ze geen naam hebben. De code wordt dan:

```
$arr_even = array_filter($arr, function ($val) {
    return $val % 2 == 0;
});

$arr_square = array_map(function ($val) {
    return $val * $val;
}, $arr);
```

Zo wordt ook in JavaScript (jQuery) genoteerd!

Modulariseren

U kunt de anonieme functies ook toekennen aan een variabele:

```
$square = function ($val) {  
    return $val * $val;  
};  
$arr_square = array_map($square, $arr);
```

Maar dat leest misschien iets minder prettig...

```
var_dump($square);
```

geeft:

```
object(Closure)#1 (1) { ["parameter"]=> array(1) { ["$val"]=> string(10) "" } }
```

Wanneer u wilt controleren of een callback functie anoniem is dan kunt dat doen met:

```
if (is_object($callback) && ($callback instanceof Closure)) {...}
```

Want op de achtergrond wordt een object van de native PHP Closure class aangemaakt. Meer hierover in de cursus PHP OOP.

7.3.25.1 create_function()

Overigens kon dit ook al vanaf PHP 4.0.1 met de functie `create_function()`:

```
$square = create_function('$val', 'return $val * $val;');
```

U geeft dan eerst de argumenten als string en daarna de functiebody als string. Denk eraan dat deze met een ; wordt afgesloten!

```
var_dump($square);
```

geeft:

```
string(9) "lambda_1"
```

Intern wordt nu echter gebruik gemaakt van `eval()` met alle veiligheidsrisico's van dien. Bovendien is het traag en inefficiënt. U kunt het dan ook beter **niet gebruiken!**

Update: Deze functie is **deprecated** in PHP 7.2.0. **Gebruik hem dus niet!**

7.3.26 Aanbeveling en tips

Aarzel niet om zelf functies te schrijven! Ze maken uw code vele malen beter te lezen, te onderhouden en te hergebruiken!

Hou de volgende zaken goed in de gaten:

- Het werken met functies bestaat altijd uit twee stappen: het definiëren van een functie en het aanroepen van de functie.
- U mag in PHP een functie aanroepen voordat hij gedefinieerd is, als de definitie maar binnen het script beschikbaar is.
- Functienamen zijn case-insensitive.

- Functies kunnen direct output leveren of waarden retourneren. Ze doen dit in beide gevallen op de plaats van de aanroep. Wanneer een functie een waarde retourneert moet er sprake zijn van een side-effect: de waarde moet worden weggeschreven of opgeslagen.
- Functies kunnen argumenten gebruiken om input te krijgen. Het standaardgedrag is argument passing by value, de waarde van de bron variabele wordt niet aangepast.
- De scope van variabelen binnen de functie is lokaal. Buiten de functie gedefinieerde variabelen hebben een global scope.
- Binnen functies kunnen zo alleen variabelen voorkomen die binnen die functie zijn gedefinieerd of afkomstig zijn van de formele parameters.
- Als u binnen een functie naar de waarde van een globale variabele wilt verwijzen dan kunt u dit doen door middel van het keyword `global` (of door de variabele by reference aan de functie te geven).
- U kunt argumenten met een default waarde gebruiken. De volgorde en het aantal zijn dan (net als bij meerdere gewone argumenten) van groot belang.

7.4 Alternatieven

De hierboven beschreven werkwijze is, zoals al eerder aangegeven, natuurlijk maar één manier om een applicatie te bouwen. Voor deze cursus is het in ieder geval een aardige oplossing. In de praktijk kunt u vele andere varianten tegenkomen om de pagina's in een site op te bouwen.

Veel oplossingen maken gebruik van **templates** om een betere scheiding te verkrijgen tussen de logica die de data ophaalt en de presentatie die de data moet tonen. Met de kennis die we tot nu toe hebben is dat echter nog niet te realiseren. Zo hebben we bijvoorbeeld op dit moment nog een sterke koppeling tussen de PHP code om de lus te maken waarin de data uit de MySQL database wordt getoond. Nadat we in Hoofdstuk 2 hebben geleerd hoe we de records in een 2-dimensionale array kunnen gieten, kunnen we iets vergelijkbaars maken door deze array aan een HTML template te geven waarin enkel wat presentatielogica staat om deze data in een lus te kunnen tonen. Wij zullen hiervoor overigens wel PHP blijven gebruiken. Sommige template engines zoals het bekende **Smarty** (<http://www.smarty.net>), hebben hun eigen presentatielogica om te voorkomen dat een templateontwikkelaar ongewenste of foutieve PHP code in de template zou kunnen opnemen. Dat geeft echter wel veel meer overhead. Wat dan weer gecompenseerd wordt door een uitgebreid caching mechanisme.

De mooiste oplossingen maken gebruik van **object georiënteerde PHP** code die volgens het beproefde **Model-View-Controller design pattern (MVC)** werkt. Dat is op dit moment echter nogal wat stappen te ver, omdat daar eerst een heleboel kennis over OOP voor nodig is.

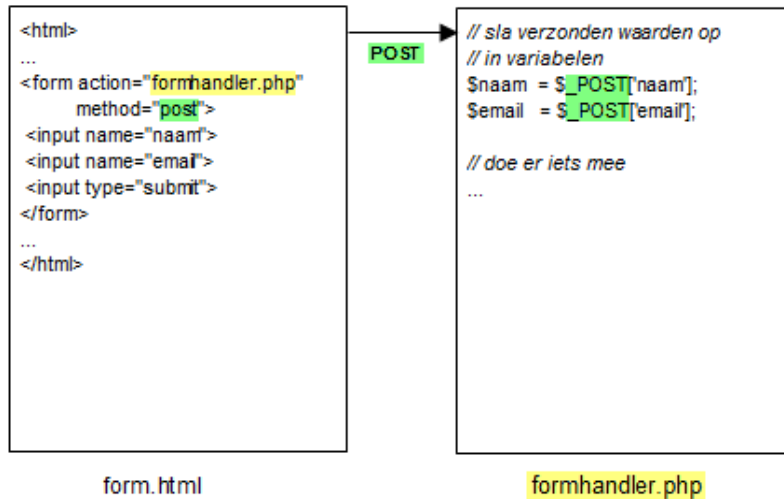
Er is niet op dit moment voor u dan ook niet één beste manier om een PHP applicatie te schrijven. Elke ontwikkelaar komt na verloop van tijd met een eigen manier.

Welke manier van werken voor u handig is hangt gedeeltelijk af van uw voorkeur, maar is natuurlijk ook sterk afhankelijk van uw collega's. En het organiseren van code, op welke wijze dan ook, is natuurlijk altijd beter dan het niet organiseren van code!

7.5 Self-submitting forms

Tot nu toe hebben we tijdens de oefeningen in de Basis cursus met losse pagina's gewerkt: een apart HTML formulier stuurt data naar een PHP pagina die de name-value pairs uit de `$_GET` of `$_POST` superglobal arrays haalt om er vervolgens mee verder te werken. Op deze manier werden onder andere dynamische SQL statements opgebouwd en uitgevoerd:

Modulariseren

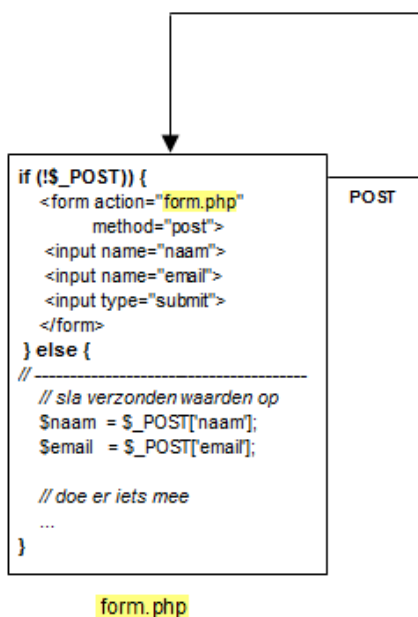


Met deze manier van werken is op zich niets mis maar omdat u vaak meerdere formulieren nodig heeft in een applicatie heeft u uiteindelijk vrij **veel pagina's nodig**. Om alle basale databasehandelingen online uit te kunnen voeren had u bijvoorbeeld al gauw een zevental pagina's nodig. Dat is **niet** echt **onderhoudsvriendelijk**... Bovendien moet u **voorzorgen** nemen zodat een gebruiker niet (per ongeluk?) de formhandler scripts **los** kan **opvragen**.

Soms is het dan ook efficiënter om functionaliteit in één pagina onder te brengen. Een pagina die informatie naar zichzelf stuurt via een form of via een hyperlink met een querystring wordt **self-submitting**, **self-referencing** of **self-referring** genoemd.

Zo kunt u de te ondernemen acties over de querystring versturen en met een switch afvangen om de bij elke actie horende functies aan te roepen. Een andere handige bijkomstigheid is dat **Server-side validatie** op deze manier heel **elegant** is **toe te voegen** (zie **hoofdstuk 4**). Enig nadeel is mogelijk de ietwat complexere opbouw per pagina.

Het idee is grofweg als volgt (de letterlijke code komt zo meteen aan bod):



7.5.1 Opbouw formulier

We gaan dus leren hoe u een formulier naar zichzelf kunt laten posten. Dat houdt in dat we de invoer en de verwerking door dezelfde pagina laten plaats vinden: de pagina bevat zowel het formulier als het verwerkende script. Onze pagina's zijn vanwege de modulaire opbouw die ze op dit moment al hebben vrij makkelijk in dit formaat te gieten.

7.5.1.1 Action

Om een formulier naar zichzelf te laten verzenden kunt u gewoon als waarde voor het action attribuut hard de **naam van de pagina** opgeven:

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

Het nadeel van deze methode is dat wanneer u het bestand hernoemt, u niet moet vergeten de waarde van het action attribuut aan te passen!

Om dat probleem op te heffen wordt er vaak voor gekozen om de **servervariabele** `$_SERVER['PHP_SELF']` te gebruiken:

```
<form action="<?php echo $_SERVER['PHP_SELF'] ?>" method="post">
```

Deze servervariabele geeft het pad naar het huidige bestand, opgebouwd vanuit de webroot. Als het bestand `selfsubmit.php` bijvoorbeeld in een directory `cursus` staat dan wordt het pad simpelweg:

```
/cursus/selfsubmit.php
```

Dit zorgt ervoor dat het formulier altijd naar zichzelf verzendt. Ook als u de pagina verplaatst of zou hernoemen!

Deze manier is echter **niet** helemaal **veilig voor XSS Attacks** (zie hoofdstuk 4), daarom gebruiken we de **allermakkelijkste** en tegelijkertijd ook de **veiligste** manier en dat is om de **waarde** voor het **action attribuut** gewoon **leeg** te laten:

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

Dit is volledig valide (X)HTML. We zullen deze manier dan ook tijdens de cursus gebruiken.

Update: In HTML5 mag het **action** attribuut echter **niet leeg** zijn! Gebruik dan de *hardcoded filename* of de veilige manier zoals in hoofdstuk 4 wordt besproken:

```
<form action="<?php echo htmlspecialchars($_SERVER['PHP_SELF']) ?>" ...>
```

7.5.1.2 Method

De meeste self-submitting formulieren zullen waarschijnlijk de method POST gebruiken, maar GET kan uiteraard ook...

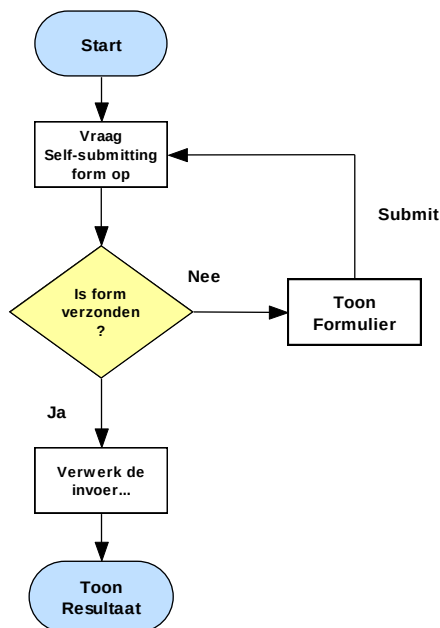
7.5.2 Logica

We zullen nu wel bij de uitvoer van een dergelijke pagina op de een of andere manier onderscheid moeten maken in hoe die pagina werd aangeroepen. Wordt de pagina voor de eerste keer opgevraagd, bijvoorbeeld via een link of het rechtstreeks intypen van de URL, dan is er in principe nog geen invoer en hoeft er niets verwerkt te worden.

Als de pagina daarentegen wordt opgevraagd nadat het formulier is ingevuld en op de submit knop is gedrukt (en er dus wel invoer is) dan moet de verwerking juist wel plaats vinden. We moeten met PHP dus de status van het formulier uitvinden en navenant handelen.

Modulariseren

Stel, we willen dat wanneer iemand de pagina voor het eerst bezoekt een formulier getoond wordt waarop om een voornaam gevraagd wordt. Het formulier wordt dan naar zichzelf verzonden en daarna mag alleen nog maar het resultaat te zien zijn. De beslissing wat te tonen zal ons script moeten maken. In een Flow Chart ziet het er ongeveer zo uit:



Let op: In eerste instantie gaan we er nu voor het gemak van uit dat de pagina maar **twee stadia** kent: verstuurd of niet verstuurd. In de praktijk zou u natuurlijk best meer na kunnen gaan (is er op reset of misschien op een helpknop gedrukt?).

Maar hoe gaan we dit coderen? Er zijn verschillende manieren om dit op te lossen. Maar ze maken in ieder geval allemaal gebruik van een controlestructuur (`if else` of `switch`). We nemen hieronder een aantal van de meest voorkomende manieren (en hun mogelijke valkuilen) door.

Voor het tonen van het formulier en het verwerken van de data gebruiken we in de voorbeelden 2 functies:

1. `show_form()`
genereert een self-submitting form. We gaan er in de voorbeelden vanuit dat de data middels POST verzonden wordt.
2. `process_form()`
verwerkt de door het formulier verzonden data. In de praktijk kan dat van alles inhouden (mailen, database) maar in de voorbeelden gebruiken we een eenvoudig echo statement.

We werken de functies hier verder niet uit, dat mag u straks zelf in de oefeningen doen! Wat we hierna wel doen, is een generieke en robuuste manier aanbieden om dit soort formulieren te maken.

7.5.3 Controle op Submit button

U geeft de submit button een naam en u gebruikt die waarde om te controleren of het formulier verstuurd is:

```
<input type="submit" name="submit" value="OK" />
```

Vervolgens kan de code er als volgt uit zien:

```

if ($_POST['submit'] == 'OK') {
    // formulier is verzonden: verwerk de data
    process_form();
} else {
    // formulier niet verzonden: toon het formulier
    show_form();
}

```

Dit zou zelfs nog wat compacter kunnen omdat een niet lege string in een conditie naar `true` evalueert:

```

if ($_POST['submit']) {
    // formulier is verzonden: verwerk de data
    process_form();
} else {
    // formulier niet verzonden: toon het formulier
    show_form();
}

```

Deze manier van werken kent echter een drietal problemen:

1. Bij een **hoog error_reporting level** geeft bovenstaande code **waarschuwingen** omdat de index `submit` bij de eerste keer opvragen van het script nog niet gedefinieerd is.
2. **Onder bepaalde condities** zal het **niet mogelijk** zijn om het formulier in **Internet Explorer** te verwerken.
3. De volgorde leest wat **onhandig**. Het zou prettiger zijn als de volgorde in het if-else statement omgedraaid was.

Het eerste probleem is eenvoudig op te lossen met de functie `isset()` en voor het tweede probleem kunnen we een **hidden field** gebruiken. Het derde probleem lossen we op met een functie en kunt u als **verdieping** beschouwen.

7.5.3.1 `isset()`

Als u `error_reporting(E_ALL)` aan hebt staan (en dat is natuurlijk een goed idee tijdens het ontwikkelen!) dan leveren bovenstaande vergelijkingen een waarschuwing op bij de eerste keer opvragen:

Notice: Undefined index: submit

Dat is ook best logisch want als het formulier voor de eerste keer wordt opgevraagd dan gebeurt dit door middel van een GET request. Op dat moment zit er dus nog geen element met de index `submit` in de `$_POST` array. U kunt de foutmelding eenvoudig voorkomen met de functie `isset()` waarmee u controleert of de variabele wel bestaat:

```

if (isset($_POST['submit'])) {
    // formulier is verzonden: verwerk de data
    process_form();
} else {
    // formulier niet verzonden: toon het formulier
    show_form();
}

```

Nog netter is om vervolgens te controleren of `submit` ook de gewenste waarde heeft:

```

if (isset($_POST['submit']) && $_POST['submit'] == 'OK') {
    // formulier is verzonden: verwerk de data
    process_form();
} else {

```

```
// formulier niet verzonden: toon het formulier
show_form();
}
```

U hoeft in het tweede deel van de vergelijking zoals u ziet geen gebruik van `isset()` te maken. PHP hanteert voor dit soort situaties een principe dat **short circuiting** wordt genoemd: als het eerste deel van een samengestelde conditie al niet waar is dan wordt het tweede deel door de parser genegeerd...

7.5.3.2 Quick and dirty

Overigens kunt u het probleem ook snel oplossen door simpelweg te controleren op de aanwezigheid van elementen in de `$_POST` array. In dat geval zou u er van uit kunnen gaan dat het formulier getoond, ingevuld en verzonden is. En aangezien in PHP een array met meer dan 0 elementen in een vergelijking naar true evalueert, zou u het ook heel compact kunnen oplossen met:

```
if ($_POST) {
    // formulier is verzonden: verwerk de data
    process_form();
} else {
    // formulier niet verzonden: toon het formulier
    show_form();
}
```

U ziet dat u zelfs `isset()` niet hoeft te gebruiken. De array `$_POST` bestaat immers sowieso. Hij is alleen bij het voor de eerste keer opvragen van de pagina nog niet gevuld...

Het **grote nadeel** van deze werkwijze is dat **elk POST request** ervoor **zorgt** dat de **code voor het verwerken** van het formulier **wordt uitgevoerd**. Dat kan een probleem opleveren wanneer u meerdere formulieren op een pagina heeft staan (registratie, login, zoeken, aanmelden nieuwsbrief, het komt best vaak voor...).

Dus hoewel verleidelijk is dit niet echt een robuuste oplossing voor een productie omgeving! Tijdens de cursus is het echter wel een snelle manier als u toch maar met een formulier werkt...

7.5.4 Controle op hidden field

Wanneer u op eerder geschetste manier een **formulier met één tekstveld** aanbiedt in **Internet Explorer** (8 en eerder) én de gebruiker drukt op de **[Enter]-toets** om het formulier te verzenden, in plaats van door op de submit button te klikken, dan zal steeds het formulier getoond worden. In een dergelijk geval wordt het **name-value pair** voor de **submit button** namelijk **niet verzonden!** Vanaf versie 9 gebeurt dat overigens wel.

De oplossing is wederom simpel. U neemt speciaal voor de verzendcontrole een hidden field in het formulier op:

```
<input type="hidden" name="action" value="postback">
```

en vervolgens checkt u of deze waarde met het formulier is meegestuurd:

```
if (isset($_POST['action']) && $_POST['action'] == 'postback') {
    // formulier is verzonden: verwerk de data
    process_form();
} else {
    // formulier niet verzonden: toon het formulier
    show_form();
}
```

Let op: U kunt hidden fields het beste gelijk na de `<form>` openingstag plaatsen. Anders loopt u kans dat de velden niet worden meegezonden wanneer iemand op [enter] drukt om het formulier te verzenden (zie mogelijkheid 1). Bovendien ziet u zo in één oog opslag of een formulier gebruik maakt van hidden fields.

In XHTML STRICT DocTypes mag een veld (ook een hidden field) echter niet zomaar los in het formulier worden geplaatst en moet het worden opgenomen in een container als een `<div>`.

7.5.5 Verdieping: Volgorde omdraaien en functie

Met het hidden field hebben we een redelijk robuuste oplossing maar we hebben nog wel een "probleem" met de volgorde, want nu staat de functie voor de verwerking als eerste bij het `if` statement waarna bij het `else` statement de functie voor het tonen van het formulier staat. Daar is op zich niets mis mee, maar het leest voor de meeste mensen prettiger als de code voor het tonen van het formulier als eerste komt. We moeten de statements daarvoor omdraaien en dan wordt het iets als:

```
if (! isset($_POST['action']) || $_POST['action'] != 'postback') {
    // formulier niet verzonden: toon het formulier
    show_form();
} else {
    // formulier is verzonden: verwerk de data
    process_form();
}
```

Helaas wordt hierdoor de conditie juist minder leesbaar...

We kunnen dan ook beter een functie `is_postback()` maken die `true` of `false` retourneert voor de check op een naam-waarde paar:

```

//*****
function is_postback($name='', $value='')
{
    $postback = false;
    if (isset($_POST[$name]) && $_POST[$name] == $value) {
        $postback = true;
    }
    return $postback;
}
//*****

```

Of simpelweg:

```

//*****
function is_postback($name='', $value='')
{
    return isset($_POST[$name]) && $_POST[$name] == $value;
}
//*****

```

Dit kunnen we op de volgende manier gebruiken:

```
if (! is_postback('action', 'postback')) {
    show_form();
} else {
    process_form();
}
```

Modulariseren

U kunt de argumenten weglaten als u 'action' en 'postback' als defaultwaarden gebruikt, terwijl u tegelijkertijd nog de flexibiliteit houdt om andere naam-waarde paren voor de controle op te geven:

```
//*****
function is_postback($name='action', $value='postback')
{
    return isset($_POST[$name]) && $_POST[$name] == $value;
}
//*****
```

Dan kunt u het simpelweg zo houden:

```
if (! is_postback()) {
    show_form();
} else {
    process_form();
}
```

Let op: Wanneer u later in uw formulier nog velden hebt met name="action" maar met een andere value, dan overschrijven deze de postback waarde!

U zou zelfs ondersteuning voor GET kunnen inbouwen maar dan is de naam `postback` niet meer zo toepasselijk. Dit is meer iets voor een switch op een action parameter (Zie ook **1.8**).

Wel biedt de functie mogelijkheden om later met een referer check en een sessie stamp de herkomst van het formulier te controleren en zo spoofen te voorkomen... We zullen hier in respectievelijk **hoofdstuk 4** en **hoofdstuk 6** naar kijken.

Ander idee: maak een functie `postback_field()` die het benodigde hidden field wegschrijft in het formulier (deze kan later eventueel ook de sessie check tegen form spoofing bevatten. Zie verder **hoofdstuk 6**).

```
//*****
function postback_field($name='action', $value='postback')
{
    return '<input type="hidden"
           name="' . $name . '"
           value="' . $value . '">';
}
//*****
```

7.5.6 Multiple Submits en Form Spoofing

Het is belangrijk dat u zich realiseert dat geen van de hier beschreven manieren bescherming biedt tegen multiple submits en dat form spoofing (het verzenden van data naar de formhandler middels een ander, eigen formulier) er ook niet mee voorkomen wordt. In **hoofdstuk 9 (Sessies)** leert u een techniek om dit laatste te voorkomen.

Oefening

Mak in de map /oefeningen een pagina groet.php die bij de eerste keer bezoeken het volgende formulier laat zien:

Groet

Dit is de eerste keer dat je deze pagina bezoekt. Vul je naam in:

voornaam:

© 2008 Arno

Het formulier heeft als action zichzelf en geeft na invullen en verzenden alleen nog maar het resultaat, bijvoorbeeld:

Groet

Hoi **Arno**, bedankt voor het invullen!

© 2008 Arno

Verdieping

Mak in de map /oefeningen een pagina groet_sticky.php. Dit is een variant op groet.php maar nu laat u het formulier en de invoer staan:

Groet

voornaam:

Hoi **Arno**, bedankt voor het invullen!

© 2008 Arno

Als bovenstaande oefening(en) gelukt is/zijn kunt u dezelfde werkwijze gebruiken voor het toevoegen van een nieuwe medewerker:

Oefening	<i>Combineer <code>medewerker_toevoegen.html</code> en <code>medewerker_toevoegen.php</code> tot één pagina. Bouw deze pagina net zo op als u dat in de voorgaande oefening gedaan heeft met <code>medewerkers_tonen.php</code>. Maak nu echter gebruik van twee, nog te schrijven functies, <code>toon_formulier</code> en <code>medewerker_toevoegen</code> en zorg voor een juiste verwerking.</i>

Merk op dat u niet alleen via een formulier informatie naar dezelfde pagina kunt sturen. U kunt dit ook doen met behulp van een aan een hyperlink 'geplakte' querystring:

```
<a href="?action=insert">Nieuw</a>
```

Merk op dat u hier ook geen bestandsnaam hoeft te gebruiken... In hoofdstuk 2 zult u soortgelijke code toepassen bij het pagineren en sorteren van recordsets.

7.6 Redirecting

Met behulp van deze techniek kunt u het algemene verloop (*general flow of execution*) van uw applicatie beïnvloeden door de browser een andere pagina te laten opvragen waarna de code van dit nieuwe script kan worden uitgevoerd.

In technische termen: bij een redirect vertelt u de client dat hij een request voor de door u gespecificeerde URL moet zenden naar de webserver. Er is dus altijd een **extra roundtrip** naar de server voor nodig (zie figuur).

Een voorbeeld:

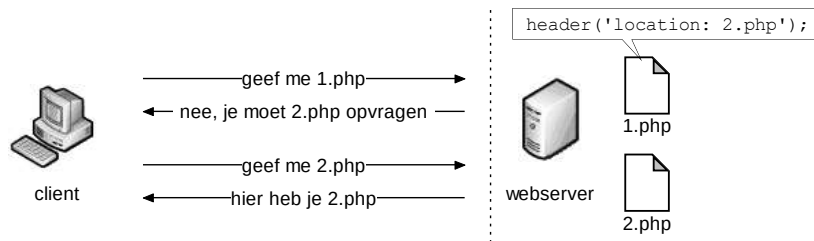
```
// dit is pagina 1.php
// met een redirect naar pagina 2.php
header("location: 2.php");
exit();
```

Omdat we hierbij van de browser afhankelijk zijn of de gewenste pagina wel wordt opgevraagd, is het zaak om na een dergelijke redirect ervoor te zorgen dat de rest van de PHP code in ieder geval niet meer kan worden uitgevoerd. Dit doet u door een `exit()` (of `die()`) commando te geven. Hiermee wordt het uitvoeren van het script beëindigd.

Bovenstaande code werkt alleen als er nog geen informatie naar de client verzonden is. Wat er namelijk gebeurt is dat de benodigde informatie in de HTTP Response Header verstuurd wordt:

```
HTTP/1.1 302 Object moved
...
Location: 2.php
```

Zoals u weet kunt u met PHP de HTTP headers manipuleren. Er wordt in dit geval een response met status code 302 naar de browser gezonden. Voor de browser is dit het teken dat de pagina verplaatst is naar een nieuwe locatie die in de location header staat aangeduid. De browser zal vervolgens een GET request voor deze nieuwe URL doen:



In FireBug kunt u mooi zien dat een request voor 1.php als resultaat het opvragen van 2.php heeft:

URL	Status	Domein	Omvang	IP-adres op afstand	Tijdslijn
GET 1.php	302 Found	localhost	331 B	::1:80	3ms
GET 2.php	200 OK	localhost	331 B	::1:80	2ms
2 verzoeken			662 B		14ms (onload: 64ms)

U kunt dan ook mooi zien dat in de response header van 1.php de location opdracht voor 2.php staat:

URL	Status	Domein	Omvang	IP-adres op afstand	Tijdslijn
GET 1.php	302 Found	localhost	331 B	::1:80	3ms

Headers	Antwoord	Cache	HTML	Cookies
Antwoordheaders				
Connection	Keep-Alive			
Content-Length	331			
Content-Type	text/html			
Date	Wed, 09 May 2012 15:04:12 GMT			
Keep-Alive	timeout=5, max=100			
Location	2.php			
Server	Apache/2.4.2 (Win32)			
Verzoekheaders				

Het is dus heel belangrijk dat u zich realiseert dat u de `header()` functie alleen maar kunt gebruiken **voordat** de HTTP headers verzonden zijn. Dat kan allereerst door de functie aan te roepen voordat er iets naar de client verzonden is. Er mag dus nog geen `echo` of HTML-code geschreven zijn.

Is dat wel het geval dan krijgt u een foutmelding als deze:

Warning: Cannot add header information - headers already sent.

Let op: er mag zelfs geen spatie verzonden zijn! Onderstaande code zou ook een foutmelding geven:

```

1  <?php
2  header('location: 2.php');
3  ?>

```

Het allereerste karakter op regel 1 is een spatie, deze wordt meteen verzonden bij het opvragen van de pagina en wanneer op regel 2 de headers aangepast moeten worden kan dat niet meer omdat ze al, voor de spatie uit, verzonden zijn...

Hetzelfde verhaal gaat ook op voor het zetten van **cookies**. Deze worden namelijk ook in de HTTP Response header verzonden (zie Hoofdstuk 5).

7.6.1 Output buffering

Er is wel een uitzondering op bovenstaande regel: namelijk als u ervoor gezorgd hebt dat de response *gebufferd* wordt. Output kan namelijk op twee manieren verzonden worden: ongebufferd en gebufferd. Ongebufferde output wordt onmiddellijk na verwerking naar de browser verzonden

Modulariseren

terwijl gebufferde output wordt opgeslagen in een buffer op de server. De buffer wordt in één keer automatisch verzonden als het eind van het script bereikt is.

In PHP zet u met de `ob_start()` functie de **output buffering** aan. Dit dient u uiteraard wel aan te geven voordat u informatie verzonden hebt, dus zet het bovenaan in uw script!

Nu zal regel 4 geen problemen opleveren:

```
1 <?php
2 // zet output buffering aan
3 ob_start();
4 echo 'dit wordt nu gebufferd en niet meteen verzonden...';
5 header('location: 2.php');
6 ?>
```

Met het commando `ob_end_flush()` kunt u eventueel de gebufferde data naar de client zenden voordat het einde van het script bereikt is.

7.6.2 Debugging

Let op: Wanneer u in een script `insert.php` na een database insert actie gelijk een redirect doet naar het script `overview.php` dan zult u geen database foutmelding zien wanneer er iets mis gaat in `insert.php`. U wordt, ook al gaat er iets mis, gelijk naar het overzicht gevoerd waardoor de foutmeldingen dus nooit in beeld komen!

Als u wilt gaan debuggen omdat u in het overzicht merkt dat er iets niet goed is gegaan, dan moet u dus altijd eerst even het `header()` statement in `insert.php` (tijdelijk) uitcommentariëren...

7.6.3 Absolute URL

In de voorbeelden hierboven gebruikten we relatieve URL's. De meeste clients accepteren weliswaar relatieve URL's maar HTTP/1.1 verlangt officieel een absolute URL als argument voor Location, dus inclusief het schema, de hostname en een absoluut pad. In plaats van

```
header('Location: login.php');
```

zou u dus eigenlijk iets als

```
header('Location: http://www.server.nl/app/login.php');
```

moeten gebruiken.

7.6.3.1 functie `redirect()`

Met de volgende functie voldoet u aan de specificatie maar houdt u uw code toch portabel:

```
function redirect($relative_url)
{
    header("Location: http://" . $_SERVER['HTTP_HOST']
          . dirname($_SERVER['PHP_SELF'])
          . "/" . $relative_url);

    exit();
}
```

Merk ook op dat u zo ook nooit `exit();` kunt vergeten!

7.6.4 Alternatieven

Realiseert u zich overigens dat u dezelfde functionaliteit ook zonder PHP voor elkaar kunt krijgen. Een pure HTML oplossing:

```
<meta http-equiv="refresh" content="10;url=next.php">
```

Deze tag zorgt ervoor dat de browser na 1 seconde een request voor de pagina `next.php` doet. U kunt die vertraging eventueel voorkomen door als waarde 0 te kiezen.

Ook met Client-side JavaScript kunt u een redirect afdwingen:

```
<script>location.href = "next.php";</script>
```

7.6.5 Querystring

Het is belangrijk dat u zich realiseert dat u met een redirect opnieuw naam-waarde paren over de querystring kunt meezenden om zo scripts aan te kunnen sturen:

```
header("location: errors.php?error=no_rights");
exit();
```

Hier gelden uiteraard weer de beperkingen die we al in de basiscursus hebben besproken: veiligheidshalve kunt u beter niet meer dan 255 karakters in de querystring plaatsen. Als u **meer data** wilt meegeven dan kunt u dat het beste doen via **sessie variabelen**. Deze worden behandeld in **Hoofdstuk 6**.

7.7 Praktijkvoorbeeld: self-referencing page

De medewerkers applicatie die u in de basiscursus heeft gemaakt bestond minimaal uit een vijftal pagina's. Met de kennis die u nu heeft zou u de applicatie ook zo kunnen herschrijven dat de verschillende acties door één pagina kunnen worden afgehandeld:

```
<?php
// sluit standaardfuncties in
include('includes/common.inc.php');

//*****
$conn = db_connect();

// roep functie aan afhankelijk van gekozen actie
switch ($_REQUEST['action']) {
    case 'insert':
    case 'update':
        toon_formulier($_GET['action']);
        break;
    case 'insert_save':
        insert_medewerker();
        break;
    case 'update_save':
        update_medewerker($_POST['id']);
        break;
    case 'delete':
        delete_medewerker($_GET['id']);
        break;
    default:
        toon_medewerkers();
        break;
}
//*****
```

Zo kunt u de te ondernemen acties over de querystring versturen en met een switch afvangen om de bij elke actie horende functies aan te roepen.

Modulariseren

Alle handelingen kunnen nu door één pagina worden uitgevoerd! Bovendien is in één oogopslag te zien wat het script allemaal kan doen. De functies die worden aangeroepen zijn gedefinieerd in includes of binnen de pagina, afhankelijk van het feit of ze door de hele site herbruikbaar zijn of niet.

Ook prettig aan deze werkwijze is dat u zo de code voor het formulier kunt gebruiken voor zowel de insert als de update acties. Aanpassingen aan het formulier hoeft u dan maar op een plaats te doen.

Na het hoofdprogramma in de switch kunt u alle afzonderlijke functies onderin het bestand opnemen, maar mooier is het om ze te bundelen in de /data map zoals eerder besproken.

Verdieping	<i>Maak een pagina <code>medewerkers.php</code> waarin u op bovenstaande wijze alle databasehandelingen uit de basiscursus verwerkt.</i>

Werken met cookies

8.1 Inleiding

Een webapplicatie lijkt voor een groot deel sterk op de desktop applicaties die op uw computer draaien. De interface van een webapplicatie bestaat echter uit een aantal webpagina's en de code resideert voornamelijk op de webserver waarvan u die pagina's opvraagt. Dat is handig: u kunt zo makkelijk en platform-onafhankelijk functionaliteit aanbieden aan gebruikers, het enige dat ze nodig hebben is een browser, die ze toch al hadden. Vandaar dat webapplicaties ook wel als **zero-client applicaties** worden aangeduid.

HTTP is echter een **stateless protocol** en dat levert problemen op die we niet kennen bij desktop applicaties of traditionele client-server applicaties waarbij meerdere geauthenticeerde gebruikers elk een 'eigen' permanente verbinding met de server hebben. Hierbij is het mogelijk om elk request tot een bepaalde client te herleiden en waarden in het geheugen te houden die specifiek zijn voor die bepaalde client.

HTTP werkt niet op deze manier. De hele transactie is eigenlijk uiterst onpersoonlijk. Een kopie van een opgevraagde pagina wordt, indien beschikbaar, toegezonden waarna de verbinding verbroken wordt. De webserver heeft daarbij geen idee of we al eerder een pagina hebben opgevraagd of dat we dat voor de eerste keer doen. De server houdt niet bij wat we gedaan hebben en maakt ook geen aannamen over wat we verder willen. Er wordt geen gebruikersinformatie vastgehouden. Met andere woorden: onze **status** is onbekend.

Dit heeft ondermeer tot gevolg dat de variabelen in onze PHP pagina's niet meer beschikbaar zijn nadat het script is uitgevoerd. U kunt ze dus niet in een tweede pagina opnieuw uitlezen, tenzij u ze als naam-waarde paren via de querystring of een formulier naar deze pagina heeft gestuurd.

Dit is altijd een groot probleem geweest bij het maken van websites waar dat juist wel nodig is, zoals bij webapplicaties. Een webapplicatie is een verzameling pagina's die een bepaalde taak uitvoeren of tot een bepaald resultaat leiden. Er is meestal een bepaalde route door de site. Denk bijvoorbeeld aan het online bestellen van een product. Dan moet u iets uit een catalogus kunnen kiezen en in een virtueel winkelwagentje kunnen stoppen om het aan het einde af te rekenen. En dan natuurlijk alleen uw producten, niet die van een andere klant!

De statusloosheid en anonimiteit die gebruikelijk waren bij vrij over het net surfen gaan dus niet meer op bij een webapplicatie. Kortom, we willen informatie over meerdere pagina's vast kunnen houden. De Engelse term hiervoor luidt: **maintaining a persistent state**.

Maar hoe gaan we dat nu doen?

Het **IP-adres** van de client geeft geen afdoende zekerheid of we met dezelfde gebruiker te maken hebben. Zo hebben gebruikers die achter een firewall of proxy zitten hetzelfde IP-adres. De methode om een gebruiker te kunnen identificeren moet dus op HTTP gebaseerd zijn.

De manieren die u tot nu toe kent om informatie door te geven naar een volgende pagina zijn:

1. het posten van informatie in (verborgen) formulervelden en
2. het doorgeven van informatie over de querystring via een formulier of hyperlink.

Het mag duidelijk zijn dat u niet op deze manier informatie mee kunt blijven sturen voor een gebruiker die door uw site navigeert. Los van de programmeer- en onderhoudsproblematiek staat het feit dat bij een uitstapje buiten de site (een prijsvergelijking op een andere site?) in één keer alle opgebouwde informatie verloren gaat.

In PHP heeft u echter twee andere manieren beschikbaar die wel op een makkelijke manier status kunnen vasthouden: **cookies** en **sessies**. In dit hoofdstuk bespreken we het werken met cookies. Sessies komen in het volgende hoofdstuk aan de orde.

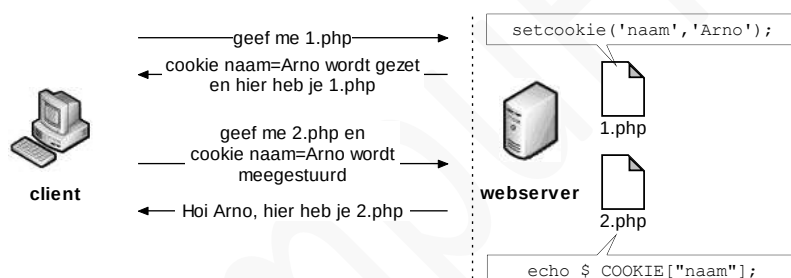
8.2 Cookies

Eén van de meest gebruikelijke manieren om status tussen opeenvolgende requests vast te houden is door middel van *cookies*. Cookies werden al ten tijde van Navigator 1.0 (1994) door Netscape ontworpen. Later is het gebruik van cookies vastgelegd in een (voorstel tot een) standaard: *RFC 2965: HTTP State Management Mechanism*.

8.2.1 Werking

Een cookie is in feite een naam-waarde paar dat door ons, de programmeurs, Server-side in de HTTP Response Header wordt gezet. Het gaat voor de pagina uit naar de client, in de meeste gevallen een webbrowser. Een browser die Cookies ondersteunt zal vervolgens het naam-waarde paar vasthouden en met elk volgend request naar hetzelfde domein opnieuw meesturen, nu in de HTTP Request Header. Hierdoor is aan de server kant het naam-waarde paar voor deze client weer beschikbaar waardoor wij er met onze PHP code op kunnen reageren en er mee verder werken, eigenlijk op dezelfde manier als u invoer uit een formulier zou aannemen.

In schema ziet dat er zo uit:



Cookies worden traditioneel met CGI programma's of met Server-side scripting gezet en uitgelezen maar het kan sinds jaar en dag ook met Client-side scripting zoals JavaScript. In deze cursus houden we ons echter bij de Server-side manier van werken...

8.2.2 Soorten

Er bestaan twee soorten cookies:

1. **non-persistent** cookies ook wel in-memory cookies
2. **persistent** cookies ook wel disk-based cookies

Een **non-persistent** cookie kent **geen vervaldatum** en verdwijnt uit het geheugen van de client wanneer de browsersessie wordt beëindigd. Ze worden daarom ook wel **in-memory** cookies genoemd.

Persistent cookies hebben **wel een vervaldatum** en worden door de browser naar de harddisk van de client geschreven, waar ze bewaard blijven totdat de vervaldatum bereikt is. Ze worden daarom ook wel **disk-based** cookies genoemd.

8.2.3 Beperkingen

Er zijn wel een aantal beperkingen in de standaard voorgesteld. Zo mag een **cookie niet groter zijn dan 4 Kb**. De meeste zijn echter een stuk kleiner. En er mogen **niet meer dan 20 cookies per domein** worden weggeschreven.

Update: In FireFox en Chrome gelden op dit moment (2020) geen limieten meer voor wat betreft aantal en grootte... <https://www.lifewire.com/cookie-limit-per-domain-3466809>

Op zichzelf is het gebruik van cookies relatief ongevaarlijk maar gebruikers willen ze toch nog wel eens uitzetten. Dat komt allereerst omdat cookies in het verleden een slechte naam hebben gekregen omdat ze door ontwikkelaars verkeerd werden ingezet. Zo werden bijvoorbeeld allerlei persoonlijke gegevens rechtstreeks, onversleuteld in een cookie weggeschreven en u kunt zich voorstellen dat een gebruiker er niet blij van wordt wanneer hij zijn creditcard nummer in een cookie op zijn PC terugvindt... Het vervelende is namelijk dat een andere gebruiker op dezelfde PC deze cookie ook zou kunnen openen en lezen.

Let op: Om met cookies te kunnen werken moet de browser van de gebruiker wel cookies ondersteunen. Alle moderne browsers doen dit standaard, maar een gebruiker kan besluiten de ondersteuning voor cookies uit te schakelen.

Wat de ontwikkelaar had moeten doen is de cookie gebruiken om de gebruiker uniek te identificeren met een **unieke code** en aan de hand daarvan de op de server opgeslagen data aan deze gebruiker ter beschikking stellen (zodat hij zijn gegevens niet steeds opnieuw hoeft in te vullen). Dit is ook de werkwijze die in de later te bespreken **sessies** wordt gevolgd.

Ook heden ten dage hebben cookies voor sommige gebruikers een bijmaak omdat ze ook gebruikt worden om gegevens over paginabezoeken op te slaan (*tracking*). Als dat binnen een site gebeurt om te achterhalen hoe gebruikers er precies doorheen navigeren om dit eventueel te kunnen verbeteren dan is dat niet erg, maar als het bezoek over meerdere sites wordt bijgehouden door aanbieders van banners om gericht advertenties te kunnen aanbieden dan is dat voor veel mensen wat teveel van het goede.

Let op: Omdat informatie aan de Client-side wordt opgeslagen is het mogelijk dat deze door de gebruiker wordt verwijderd (of aangepast!). Je zou bijna kunnen zeggen dat cookies gevaarlijker zijn voor ontwikkelaars dan voor gebruikers....

8.2.4 Toepassingen

Omdat de cookies door de client kunnen worden verwijderd of gewijzigd, worden ze vaak ingezet om niet kritieke voorkeursinstellingen vast te houden. Wanneer een cookie onverhoopt niet meer beschikbaar is dan worden gewoon de default instellingen gebruikt.

Cookies worden onder andere gebruikt om **voorkeursinstellingen** vast te houden (bijvoorbeeld voor **taal** of **indeling**), om een ingelogde status vast te houden zodat u niet steeds opnieuw hoeft in te loggen op een applicatie of om een **splashscreen** alleen bij de eerste **pagevisit** te tonen. Kortom, in allerlei situaties waarbij voor een bepaalde tijd een niet kritieke instelling voor een bepaalde gebruiker moet worden vastgehouden.

8.3 Zetten en uitlezen non-persistent cookie

In PHP is het werken met cookies enorm vereenvoudigd ten opzichte van de traditionele methode waarin u een CGI script de headers laat schrijven of met JavaScript.

8.3.1 Name en value

Het principe is heel eenvoudig. Met de functie `setcookie()` zet u een cookie met een naam en een waarde die u met de `$_COOKIE` superglobal array weer uit kunt lezen.

Let op: Omdat de informatie in cookies verstuurd wordt via de HTTP Headers is het wel belangrijk dat er nog **geen HTML** naar de client verstuurd mag zijn. Een andere mogelijkheid is om de **output** te **bufferen** via `php.ini` of door bovenaan uw pagina `ob_start()` op te nemen. Dit alles werkt op dezelfde manier als bij de in 1.6 besproken `header()` functie.

Laten we eens kijken naar een eenvoudig voorbeeldscript `cookie_set.php`:

```
<?php
// hier mag wel PHP code staan maar er mag (nog)
// niets naar de client worden geschreven!
// tenzij ob_start() aanstaat...

setcookie('naam', 'Arno');
?>

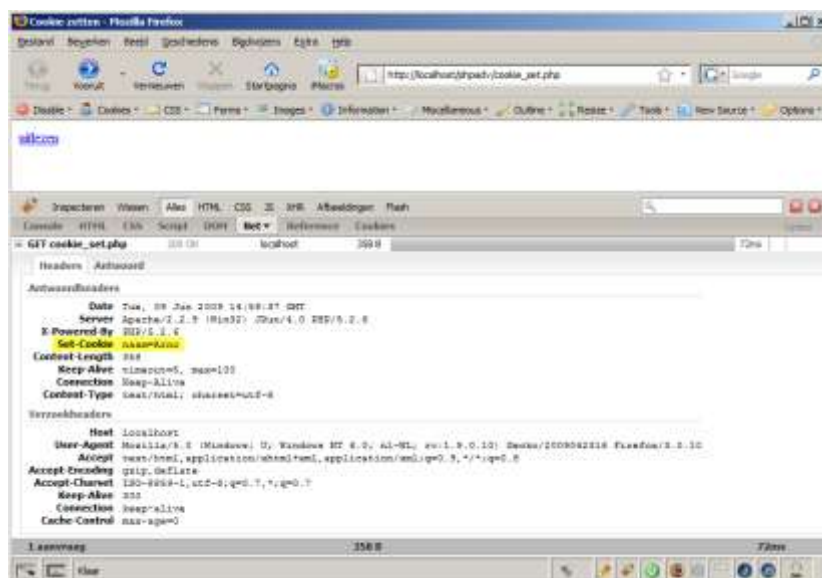
...
<!-- alvast een hyperlink naar de pagina waar we de
      cookie straks uit zullen lezen... -->
<a href="cookie_read.php">uitlezen</a>
```

De functie `setcookie()` heeft één verplicht argument en dat is de naam van de cookie. U geeft de naam als string. U zult de functie echter vrijwel altijd gebruiken met minimaal 2 argumenten, waarbij het tweede argument de waarde is die bij de naam hoort. De functie verwacht voor deze waarde eveneens een string. U kunt echter ook een datatype gebruiken dat impliciet naar een string geconverteerd kan worden. In de praktijk betekent dat ook dat u integers, floats en booleans weg kunt schrijven. Later in het hoofdstuk zult u zien dat u na een speciale behandeling zelfs arrays en objecten als waarde mee kunt geven.

Let op: wanneer u als waarde de Boolean `false` gebruikt, dan wordt de cookie **niet** gezet! U kunt dan ook beter voor booleans de waarde `0` of `1` gebruiken.

Uiteraard mag de waarde die u toekent ook afkomstig zijn uit een **variabele**, dus als tweede argument kunt u ook bijvoorbeeld `$naam` gebruiken.

Als u bovenstaande code opneemt in een PHP pagina `cookie_set.php` en deze vervolgens in de browser opvraagt, dan wordt in de response het naam-waarde paar `naam=Arno` in de HTTP Response header naar de client gestuurd:

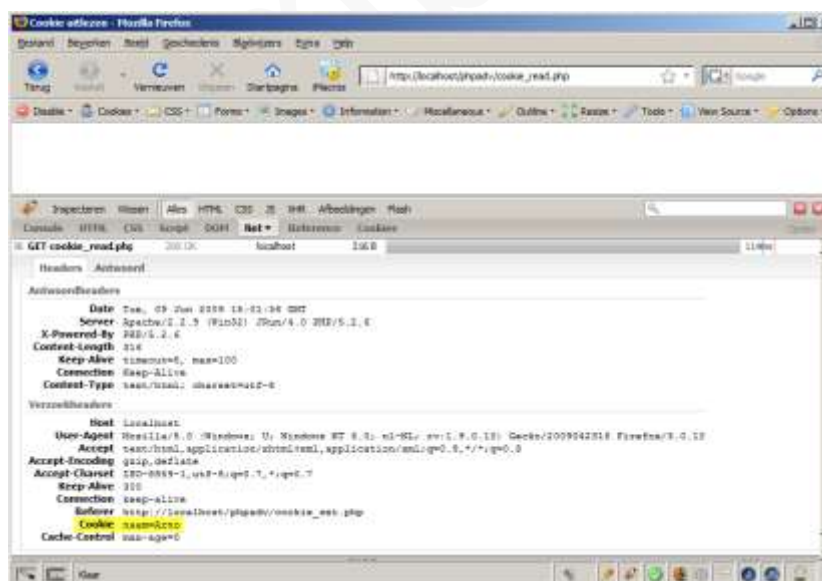


Het enige wat deze pagina doet is een cookie zetten met de naam **'naam'** en de waarde **'Arno'** en vervolgens een link aanbieden naar de pagina `cookie_read.php` waar we straks de cookie uit zullen lezen.

Bovenstaand inzicht in de opbouw van de HTTP Headers verkrijgt u met de superhandige **Firebug** extensie.

Let op: U kunt de waarde van deze cookie niet gelijk op dezelfde pagina uitlezen als waarin de cookie werd weggeschreven! De inhoud van de cookie wordt, wanneer de pagina die de cookie zet wordt opgevraagd, eerst als een naam-waarde paar in de HTTP request header naar de client gestuurd. Pas als deze een nieuw request doet (voor dezelfde pagina of een andere pagina) dan kunnen we met PHP het door de client teruggezonden naam-waarde paar uitlezen.

Wanneer we nu de link volgen naar de pagina `cookie_read.php` dan zal de browser het door ons gezette naam-waarde paar in het request meesturen:



8.3.2 Uitlezen

De letterlijke inhoud van de cookie is terug te vinden in de `$_SERVER` array. Wanneer u in de pagina `cookie_read.php` de volgende code opneemt:

```
echo $_SERVER['HTTP_COOKIE'];
```

Dan ziet u alle cookies die met het page request werden meegezonden. In ons voorbeeld levert dit dan het volgende naam-waarde paar uit de HTTP Request header op:

```
naam=Arno
```

Een dergelijke string is natuurlijk niet heel erg handig om in uw scripts te gebruiken. U kunt de waarde van een weggezette cookie makkelijker rechtstreeks benaderen door de betreffende naam als index voor de superglobal array `$_COOKIE` op te geven, waarna u het kunt gebruiken in uw code, bijvoorbeeld voor een groet:

```
echo 'Hoi ' . $_COOKIE['naam'] . '!';
```

Hiermee schrijft u de waarde van de cookie `naam` weg in de PHP pagina. U kunt de waarde uiteraard ook weer in een variabele opslaan en er verder mee werken, op een vergelijkbare manier als u eerder ook al deed met `$_GET`, `$_POST` en `$_SERVER` variabelen:

```
$naam = $_COOKIE['naam'];  
echo 'Hoi ' . $naam . '!';
```

Bovenstaande is echter wel heel kort door de bocht. Bij het uitlezen van een cookie moet u namelijk altijd aan twee zaken denken:

1. Omdat een cookie zonder vervaldatum verdwijnt als de browser wordt gesloten en een cookie met vervaldatum door de gebruiker kan worden verwijderd is het niet zeker of u de waarde van uw gezette cookies wel kunt uitlezen. U kunt er dus nooit zeker van zijn dat een door u gezette cookie ook daadwerkelijk door de client wordt teruggestuurd. U zult dus **altijd** op de **aanwezigheid** van de cookie moeten **controleren**.
2. Hoewel de cookie in principe zonder tussenkomst van de gebruiker wordt teruggezonden door de browser is het natuurlijk heel goed mogelijk dat een (kwaadwillende) gebruiker de inhoud van de cookie heeft aangepast. U zou de cookie dan ook moeten **behandelen als onveilige gebruikersinvoer** zoals besproken in het vorige hoofdstuk.

We bespreken beide gevallen hier onder:

8.3.3 Aanwezigheid cookie controleren met `isset()`

Als een gebruiker de browser afsluit dan wordt onze non-persistent cookie uit het geheugen gewist. Wanneer de gebruiker zijn browser weer opstart en rechtstreeks naar onze pagina `cookie_read.php` zou gaan dan is het afhankelijk van het ingestelde niveau van `error_reporting` of hij wel of geen waarschuwing in beeld krijgt. Alleen bij het maximale waarschuwningsniveau krijgt de gebruiker deze melding:

```
Notice: Undefined index: naam in /cookie_read.php on line 10
```

In de overige gevallen neemt PHP als waarde een lege string waardoor u zoiets zou krijgen:

```
Hoi!
```

Ook niet ideaal dus! U kunt dus maar beter, net als bij formulieren, er voor zorgen dat u altijd met `isset()` controleert of de cookie die u wilt uitlezen wel aanwezig is. Is dat het geval dan kunt u hem gebruiken, is dat niet het geval dan doet u of niks of u maakt gebruik van een default actie of default waarde:

```
<?php
if (isset($_COOKIE['naam'])) {
    echo 'Hoi ' . $_COOKIE['naam'] . '!';
} else {
    echo 'Hoi onbekende!';
}
?>
```

U kunt natuurlijk ook heel goed gebruik maken van de **ternary control operator** syntax om een variabele te vullen:

```
$naam = isset($_COOKIE['naam']) ? $_COOKIE['naam'] : 'onbekende';
```

8.3.4 Inhoud cookie controleren / veilig weergeven

Sinds het vorige hoofdstuk weet u dat data die via de HTTP Request headers verzonden wordt tamelijk eenvoudig gemanipuleerd kan worden. Dit geldt dus ook voor cookies. En omdat persistent cookies ook nog eens op de machine van de client worden weggeschreven kunnen ze hier ook makkelijk geopend en aangepast worden.

Omdat de inhoud van de cookie gemanipuleerd kan zijn door de gebruiker dient u te controleren of deze nog overeenkomt met wat u er in gezet heeft. Dat kan met sessie variabelen die we in het volgende hoofdstuk zullen bespreken. Een gewone variabele voldoet natuurlijk niet omdat u die niet over meerdere requests en meerdere pagina's kunt vasthouden. Dat was ook precies de reden waarom we in de eerste plaats iets als cookies nodig hebben!

Een basale beveiliging bestaat er uit om de inhoud van de cookie niet zo maar klakkeloos weg te schrijven maar potentieel gevaarlijke inhoud met `htmlspecialchars()` onschadelijk te maken zodat XSS Attacks geen kans krijgen:

```
<?php
if (isset($_COOKIE['naam'])) {
    echo 'Hoi ' . htmlspecialchars($_COOKIE['naam']) . '!';
} else {
    echo 'Hoi onbekende!';
}
?>
```

Oefening 7.1

Maak een pagina `cookie_set.php` met een formulier waarin om een voornaam gevraagd wordt. Het formulier is self-submitting en na verzenden wordt een cookie voornaam gezet. Als de cookie gezet is, wordt een link getoond naar `cookie_read.php` waarop u de cookie uitleest en de gebruiker met zijn voornaam begroet.

Sluit ook de browser eens en vraag `cookie_read.php` rechtstreeks op.

8.4 Zetten en uitlezen persistent cookie

Wanneer u op bovenstaande wijze een cookie zet, dan is deze niet in uw cookies folder terug te vinden. Dat komt omdat een cookie waarvoor geen **vervaldatum** is opgegeven niet op de harde schijf wordt weggeschreven. Hij wordt in het geheugen bewaard en is uit te lezen zolang u uw browser niet sluit. Zou u dat wel doen, dan wordt de cookie 'vernietigd'. Een dergelijke cookie wordt daarom **non-persistent** genoemd, zoals we in het voorbeeld hiervoor al zagen.

Om een **persistent** cookie te kunnen zetten, zult u een **vervaldatum** moeten opgeven. Dat doet u met een derde argument dat een datum in de vorm van een **timestamp** verwacht. Een timestamp is een lang getal dat het aantal seconden sinds het verstrijken van de UNIX epoch (1 januari 1970) representeert. Op basis hiervan wordt dan door PHP automatisch de datum berekend.

De vervaldatum voor de cookie kan absoluut zijn of relatief:

8.4.1 Absolute vervaldatum

Een absolute vervaldatum wil zeggen dat onafhankelijk van wanneer de cookie wordt gezet, de vervaldatum altijd hetzelfde specifieke moment zal zijn.

In PHP kunt u de timestamp voor een specifieke datum maken met de functie `mktime()`. Als u deze zonder argumenten zou gebruiken dan geeft deze net als de functie `time()` het tijdstip van aanroepen terug:

```
// stel het is nu 9 juni 2009 15:43:26
echo time();
// geeft dan 1244555006
echo mktime();
// geeft dan ook 1244555006
```

Wanneer u de functie echter met argumenten gebruikt dan kunt u de timestamp voor een specifiek moment (in de toekomst of het verleden) aanmaken. Let op de misschien wat onlogische volgorde van de argumenten:

```
// maak timestamp voor 10 juni 2009 om 12 uur 's middags
// volgorde argumenten: (uur, min, sec, maand, dag, jaar)
echo mktime(12,0,0,6,10,2009);
// geeft dan ook 1244628000
```

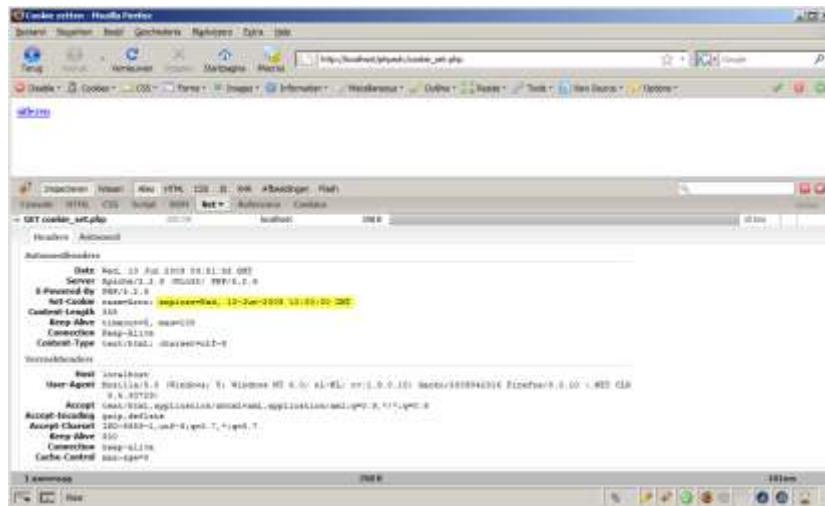
Met behulp van de in de basis cursus besproken `date()` functie kunt u de door `time()` en `mktime()` geretourneerde timestamps tonen als netjes geformatteerde datums maar dat is in combinatie met `setcookie()` niet nodig. Sterker nog, het kan zelfs niet want zoals gezegd verwacht de functie juist een timestamp voor de vervaldatum:

```
// Cookie vervalt 10 juni 2009 om 12 uur 's middags
setcookie('naam', 'Arno', mktime(12,0,0,6,10,2009));
```

Het rechtstreeks noteren van de timestamp mag natuurlijk ook maar komt de leesbaarheid van uw code natuurlijk niet ten goede:

```
// Uh, wanneer zegt u?
setcookie('naam', 'Arno', 1244628000);
```

Hoe dan ook, u kunt met behulp van FireBug zien dat de vervaldatum meegestuurd wordt in de HTTP Response header:



De tijd die u hier ziet staan, scheelt echter precies 2 uur met wat we opgaven! Hoe kan dat? Dat komt omdat de vervaldatum in de header als Greenwich Mean Time wordt weggeschreven. Omdat de cookie in het voorbeeld werd gezet op een webserver met een Nederlandse locale (GMT+1) scheelt dat al een uur en omdat 10 juni in de zomertijd valt scheelt het nog eens een uur...

De zomertijd eindigt in 2009 op 25 oktober om 03:00 en de klok gaat dan een uur achteruit. Het verschil zou daarna dus nog maar 1 uur zijn:

```
// Cookie vervalt 10 november 2009 om 12 uur 's middags
setcookie('naam', 'Arno', mktime(12,0,0,11,10,2009));
```

Inderdaad:

Antwoordheaders

```
Date Wed, 10 Jun 2009 09:58:30 GMT
Server Apache/2.2.9 (Win32) PHP/5.2.6
X-Powered-By PHP/5.2.6
Set-Cookie naam=Arno; expires=Tue, 10-Nov-2009 11:00:00 GMT
Content-Length 358
Keep-Alive timeout=5, max=100
Connection Keep-Alive
Content-Type text/html; charset=utf-8
```

Als dit soort lokale afwijkingen ongewenst zijn dan kunt u ook de functie `gmmktime()` gebruiken, die hetzelfde werkt als `mktime()` maar nu leveren de parameters niet de waarden voor een lokale timestamp maar direct voor een GMT timestamp:

```
// cookie vervalt 10 juni 2009 om 12 uur 's middags GMT
setcookie('naam', 'Arno', gmmktime(12,0,0,6,10,2009));
```

Dit geeft dan het volgende resultaat:

Antwoordheaders

```
Date Wed, 10 Jun 2009 10:00:52 GMT
Server Apache/2.2.9 (Win32) PHP/5.2.6
X-Powered-By PHP/5.2.6
Set-Cookie naam=Arno; expires=Wed, 10-Jun-2009 12:00:00 GMT
Content-Length 358
Keep-Alive timeout=5, max=100
Connection Keep-Alive
Content-Type text/html; charset=utf-8
```

U kunt de **vervaldatum alleen instellen**. Het is **niet** mogelijk deze **uit te lezen** zoals bij een waarde wel kan. De vervaldatum wordt namelijk **niet** door de client **meegezonden** met de cookie:

Verzoekheaders

```
Host localhost
User-Agent Mozilla/5.0 (Windows; U; Windows NT 6.0; nl-NL; rv:1.9.0.10)
3.5.30729)
Accept text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=
Accept-Encoding gzip,deflate
Accept-Charset ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive 300
Connection keep-alive
Cookie naam=Arno
Cache-Control max-age=0
```

8.4.2 Relatieve vervaldatum

Vaak is het echter niet zozeer de bedoeling dat een cookie op een specifiek moment moet verlopen, als wel dat hij vanaf het moment van zetten voor een bepaalde tijdsduur moet blijven bestaan, zeg een maand of een jaar. U kunt zelfs cookies tegenkomen met een vervaldatum over 50 jaar (!). Alsof u dan nog op dezelfde computer met dezelfde browser zou werken... Maar het idee erachter is natuurlijk dat op deze manier instellingen erg lang vastgehouden kunnen worden (als de gebruiker ook niet tussentijds zijn cookies opschoont).

Het is meestal handiger om met *relatieve* data te werken. Een relatieve datum baseert u op de huidige datum. Vervolgens kunt u daar seconden bij gaan optellen, bijvoorbeeld:

```
// Cookie vervalt over 1 minuut
setcookie('naam', 'Arno', time() + 60);

// Cookie vervalt over 1 uur
setcookie('naam', 'Arno', time() + 60*60);

// Cookie vervalt over 1 dag
setcookie('naam', 'Arno', time() + 60*60*24);

// Cookie vervalt over 1 jaar
setcookie('naam', 'Arno', time() + 60*60*24*365);
```

Uiteraard mag u in plaats van de vermenigvuldigingen ook gelijk het gewenste aantal seconden opgeven, maar bovenstaande notaties zijn wel erg duidelijk.

Oefening 7.2 *Pas de pagina `cookie_set.php` uit de vorige oefening zo aan dat de cookie `voornaam` gezet wordt met een relatieve vervaldatum van een week.*

Sluit ook nu weer de browser en vraag `cookie_read.php` rechtstreeks op.

8.4.3 Storage Location

Het wegschrijven van een persistent cookie wordt door de browser zelf geregeld, wij hebben daar verder geen invloed op. De plek waar dat gebeurt, verschilt per browser en besturingssysteem, maar het kan altijd alleen via de browser en op een vooraf aangegeven plek.

We laten voor het idee twee populaire browsers op het Windows platform zien, dezelfde die we ook tijdens de cursus gebruiken, namelijk Internet Explorer en FireFox.

8.4.3.1 Internet Explorer

Internet Explorer op Windows 98 slaat cookies op in de map:

```
C:\Windows\Cookies
```

Voor Windows 2000 en XP is de map:

```
C:\Documents and Settings\[Username]\Cookies
```

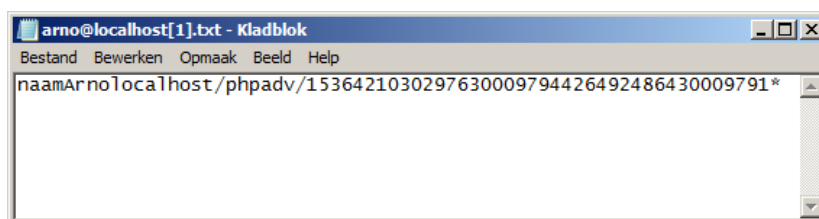
En onder Vista is het pad nog wat langer, namelijk:

C:\Users\[Username]\AppData\Roaming\Microsoft\Windows\Cookies

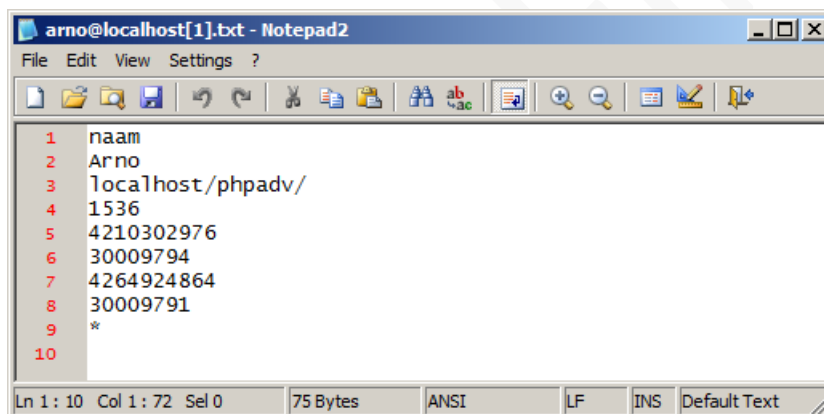
Het in het voorbeeld gezette cookie zou daar dan bijvoorbeeld zo kunnen staan:

Naam	Grootte	Gewijzigd op	Type
Low		10-6-2009 13:58	Bestandsmap
arno@google[2].txt	1 kB	10-6-2009 14:03	TXT-bestand
arno@localhost[1].txt	1 kB	10-6-2009 13:38	TXT-bestand

Het getal 1 tussen de vierkante haakjes geeft aan dat er voor het betreffende domein 1 cookie is gezet. U kunt er gewoon op klikken en de inhoud bekijken. Er zal zo iets in staan als dit:



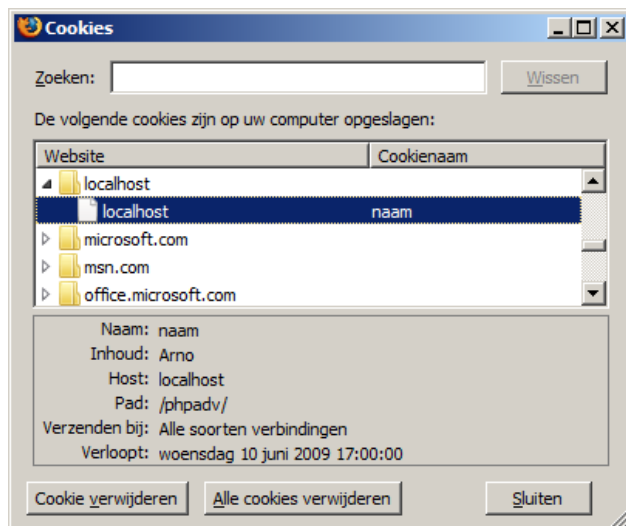
Wanneer u de inhoud bekijkt in een editor als Notepad2 of een andere editor die beter met NewLine Characters kan omgaan dan ziet het er iets logischer uit:



U herkent de waarden van de cookie en het domein en verder een hele reeks getallen. Die zullen ondermeer de vervaldatum aangeven en worden door MSIE voor het cookie management gebruikt. De door PHP gezette timestamp (1244628000) vinden we er echter niet in terug...

8.4.3.2 Firefox

In Firefox kunt u uw cookies inzien via **Extra → Opties... → Privacy → Cookie tonen...**



Met deze Cookie Manager heeft u snel toegang tot individuele cookies. Bent u echt nieuwsgierig naar de plek waar de cookies worden weggeschreven? Voor versie 3.0 stonden de cookies allemaal bij elkaar in een tekstbestand, genaamd `cookies.txt`. Maar vanaf versie 3.0 worden de cookies in een SQLite database opgeslagen die toepasselijk `cookies.sqlite` heet. U vindt het databasebestand in een folder met een naam als:

```
C:\Users\[Username]\AppData\Roaming\Mozilla
\Firefox\Profiles\xxxxxxx.default\
```

U kunt eventueel SQLite gebruiken om de data in deze database te selecteren en te manipuleren:

```
Microsoft Windows [versie 6.0.6001]
Copyright (c) 2006 Microsoft Corporation. Alle rechten voorbehouden.

C:\Users\Arno>cd C:\Program Files\SQLite3\

C:\Program Files\SQLite3>sqlite3
C:\Users\Arno\AppData\Roaming\Mozilla\Firefox\Profiles\r43cir9a.default\cookies.sqlite
SQLite version 3.6.14.2
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
moz_cookies
sqlite> .mode column
sqlite> .headers on
sqlite> SELECT id, name, value, host, path, expiry FROM moz_cookies;
-----
id            name      value    host      path      expiry
-----
1244655168105000 naam      Arno     localhost /phpadv/ 1244667600
sqlite>
```

U kunt ook vanaf de command line naar andere formaten exporteren, maar goed dat is verder niet echt een onderwerp voor deze cursus...

8.5 Cookie wijzigen

Wanneer u de waarde van een bestaande cookie wilt wijzigen, dan overschrijft u simpelweg de oude waarde van deze cookie. U doet dat eveneens met `setcookie()`. Als eerste argument gebruikt u de naam van de cookie die u wilt aanpassen en als tweede argument geeft u de nieuwe waarde mee:

```
// wijzig waarde bestaande non-persistent cookie
setcookie('naam', 'Eric');
```

Bij het eerstvolgende request na het uitvoeren van deze instructie zal de browser de nieuwe waarde meesturen.

Let op: Als u een **persistent** cookie wilt aanpassen, dan moet u de **vervaldatum** weer **meegeven** anders maakt u er een non-persistent cookie van:

```
// wijzig waarde bestaande persistent cookie
setcookie('naam', 'Eric', mktime(12,0,0,6,10,2009));
```

Bovenstaande wijziging handhaaft de vervaldatum ongeacht het tijdstip waarop de wijziging wordt uitgevoerd. Bij een relatieve vervaldatum wordt automatisch de vervaldatum opnieuw verlengd:

```
// wijzig waarde bestaande persistent cookie
// cookie vervalt over 1 jaar na deze wijziging
setcookie('naam', 'Eric', time() + 60*60*24*365);
```

8.6 Cookie verwijderen

Er bestaat geen PHP functie om een cookie te verwijderen. Een dergelijke functie zou ook niet logisch zijn, want de cookie staat immers op de computer van de gebruiker!

Standaard is het zo dat een cookie zonder vervaldatum automatisch van het client systeem verdwijnt wanneer de browser wordt afgesloten.

Een cookie met een vervaldatum verdwijnt van de client wanneer die vervaldatum bereikt is. Dat mechanisme kunt u zich ongeveer zo voorstellen: bij elk request kijkt de browser of er een cookie gezet is voor het betreffende domein. Als dat het geval is, dan wordt er gekeken of de vervaldatum niet verlopen is. Als dat nog niet het geval is, dan wordt de cookie meegezonden. Is de vervaldatum wel bereikt, dan wordt de cookie niet meegezonden en meteen verwijderd.

Wanneer u een cookie op een bepaald moment vanuit uw applicatie wilt verwijderen, gebruikt u wederom de `setcookie()` functie. U kunt er een cookie op de volgende manieren mee verwijderen:

- Geef de cookie de waarde `false`
- Zet de **vervaldatum** in het **verleden**
- Combineer beide methoden

Door voor de cookie die u wilt verwijderen de waarde `false` in te stellen en dit te combineren met een vervaldatum in het verleden, bijvoorbeeld door 24 uur af te trekken van de huidige tijd, zorgt u ervoor dat de browser de cookie niet langer meestuurt en deze verwijdert:

```
// verwijder cookie
setcookie('naam', false, time()-86400);
```

De volgende oefening combineert alle basishandelingen met een array cookie. Er wordt gebruik gemaakt van een viertal losse pagina's. U mag deze natuurlijk combineren.

8.7 Developer Tools

Gebruikers kunnen cookies inzien. Maar u kunt dat als ontwikkelaar die lokaal test natuurlijk ook heel goed doen om te zien wat er precies gebeurt.

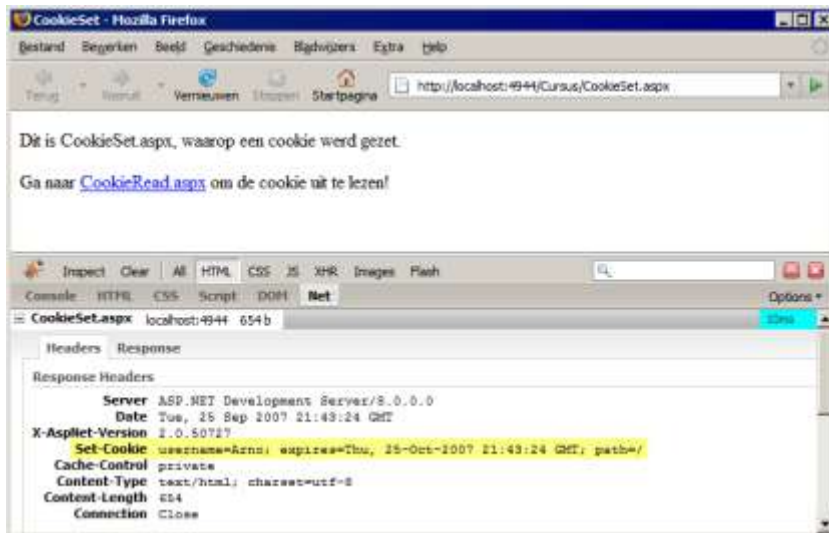
Let op: Elke browser slaat cookies op zijn eigen manier op. **Verschillende browsers kunnen elkaars cookies niet gebruiken!** Dus een cookie die door Internet Explorer is gezet, wordt niet door Firefox meegestuurd en vice versa.

Er bestaat ook een leuk tooltje **IECookiesView** waarmee u beter inzicht krijgt in alle door IE gezette cookies. U kunt het gratis downloaden op <http://www.nirsoft.net/utils/iecookies.html>.

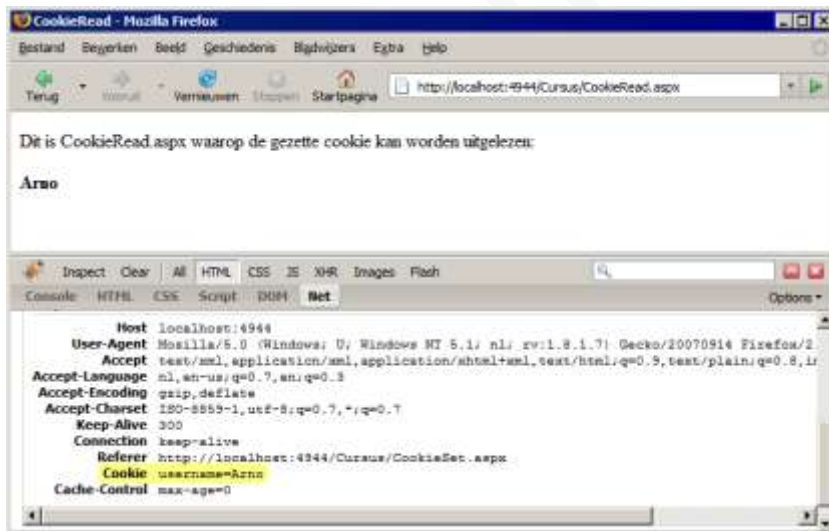
8.7.1 Met de FireBug Extensie in FireFox

Een van de meest inzichtelijke manieren om te zien wat er gebeurt bij het zetten en uitlezen van een cookie is met behulp van de **FireBug Extensie** in FireFox. U kunt hem gratis downloaden via <https://addons.mozilla.org/firefox/1843/> of via de eigen website <http://www.getfirebug.com>.

FireBug maakt het mogelijk om een schat aan informatie over elke webpagina in FireFox te ontginnen. Voor ons is het nu vooral interessant omdat u snel kunt zien hoe een cookie wordt gezet door de PHP pagina:



En hoe de client de cookie weer mee terugstuurt in het HTTP Request:



8.7.2 FireCookie

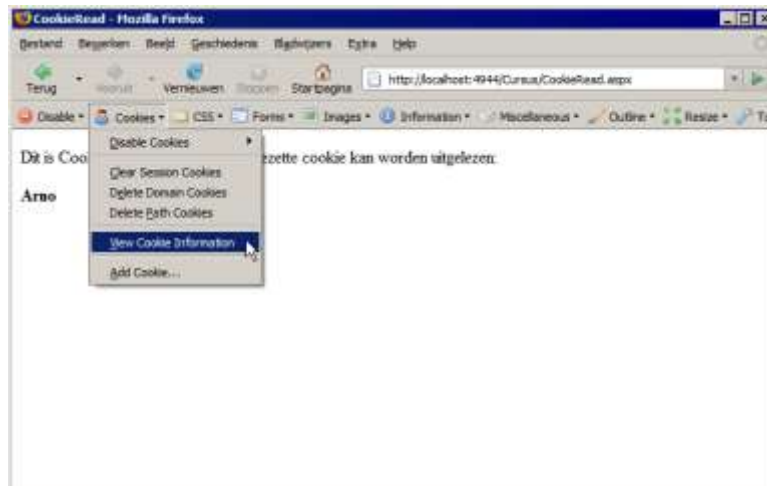
Er bestaat ook een Add-on voor de FireBug Add-on, jawel! Hartstikke handig. Maar denk eraan dat alleen de cookies getoond worden die bij de huidige tab (pagina/site) horen en dat die ook meteen bij binnenkomst de gezette cookies laat zien. Zijn te bewerken en te verwijderen, etc. Maar denk eraan dat het verversen van een pagina waarop cookies worden gezet, bijvoorbeeld om te kijken of ze er nog zijn, de cookies weer opnieuw zet en zelfs verlengt in het geval van relatieve vervaldatum...

8.7.3 Met de Web Developers Toolbar in FireFox

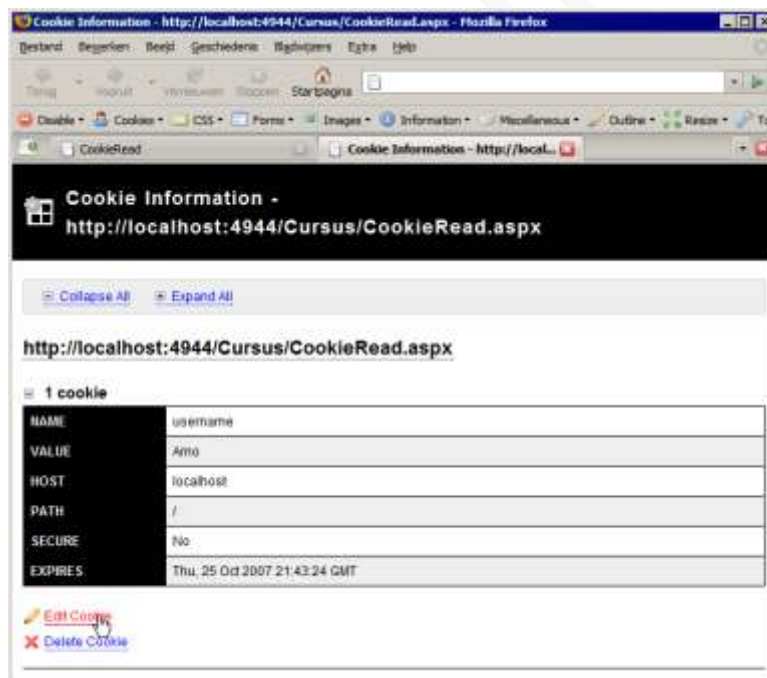
Een andere manier in FireFox om snel inzicht te krijgen in de inhoud van een Cookie is met behulp van de onvolprezen gratis Web Developers Toolbar.

De Web Developers extensie is gratis te downloaden via <https://addons.mozilla.org/firefox/60/> of de website van de auteur: <http://chrispederick.com/work/webdeveloper/>

Via Cookies → View Cookie Information...

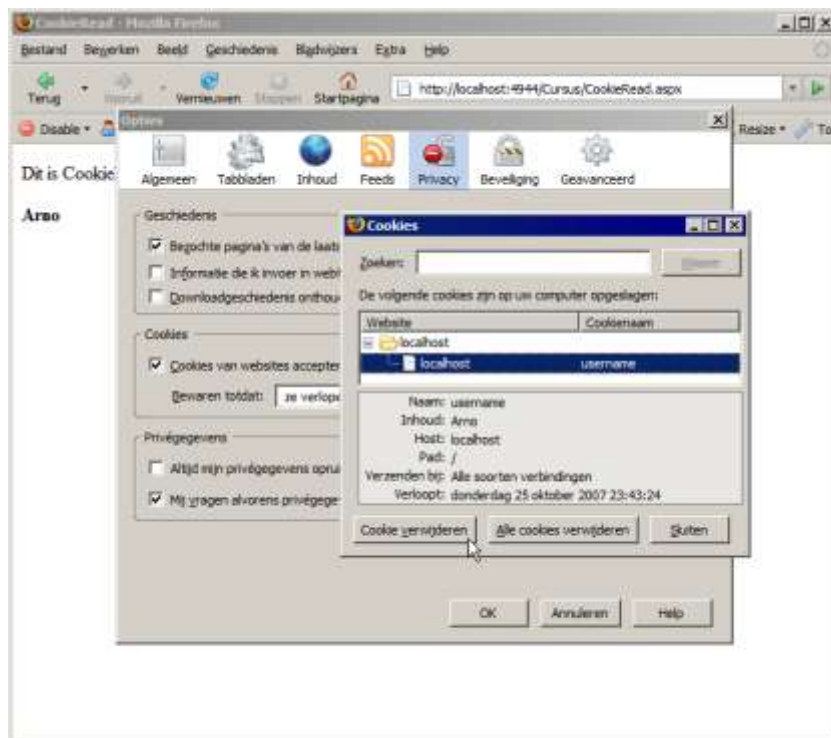


Krijgt u de inhoud van de cookie(s) voor het betreffende domein snel te zien, waarna u het ook snel kunt aanpassen of verwijderen:

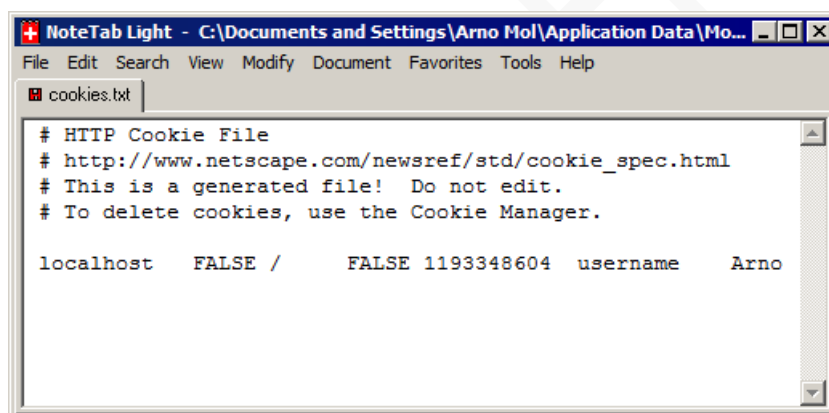


8.7.4 In FireFox

Ook standaard biedt FireFox de mogelijkheid om Cookies in te zien en te verwijderen via **Extra → Opties... → Privacy → Cookie tonen...**



Met deze Cookie Manager heeft u snel toegang tot individuele cookies. Bent u echt nieuwsgierig naar de plek waar de cookies worden weggeschreven? Ze staan allemaal in het bestand `cookies.txt`:

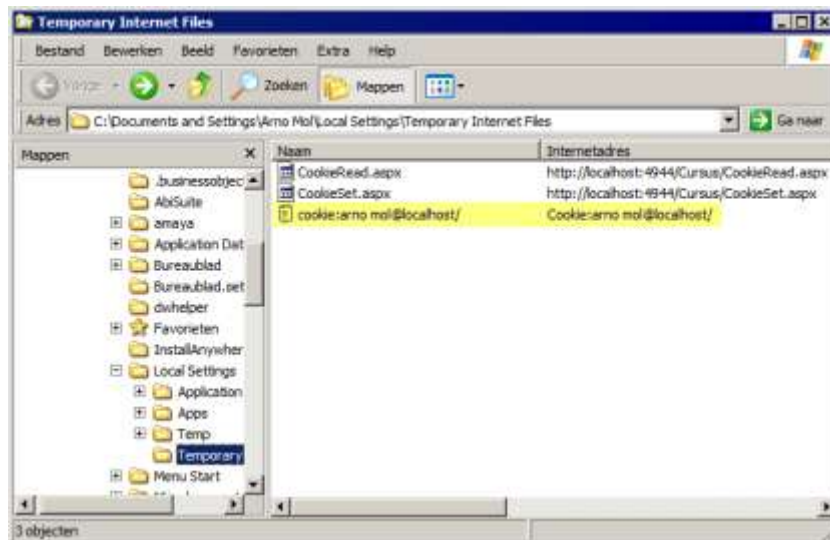


U vindt het in een folder met een naam als

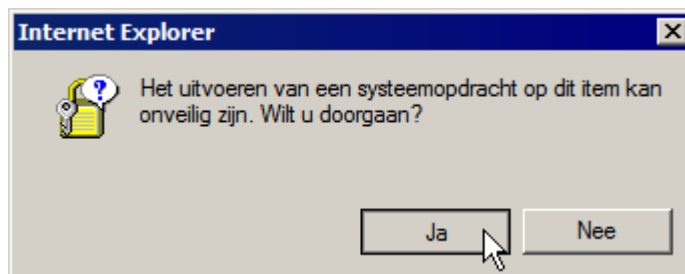
`C:\Documents and Settings\<user>\Application Data\ Mozilla\Firefox\Profiles\xxx.default`

8.7.5 In Microsoft Internet Explorer

In Internet Explorer heeft u wat minder mogelijkheden. Een persistent cookie zou u als tekstbestand in uw Cookies folder moeten zien, bijvoorbeeld:



U kunt er gewoon op klikken. Er verschijnt een waarschuwing:



U kunt gewoon op Ja klikken om de inhoud te bekijken. Standaard opent het in Kladblok. Er zal zoiets in staan als dit:

```
usernameArnolocalhost/1536125314560298903902585959229884355*
```

Wanneer u de inhoud bekijkt in een editor als NoteTab of een andere editor die beter met NewLine Characters kan omgaan dan ziet het er iets logischer uit:

```
Username
Arno
localhost/
1536
125314560
29890390
258595920
29884355
*
```

U ziet de waarden van de cookie en het domein en verder een hele reeks getallen. Die geven ondermeer de vervaldatum aan (in seconden gerekend vanaf 1970).

U kunt Cookies hier handmatig per stuk verwijderen of allemaal tegelijk via de browser: **Extra → Browsergeschiedenis verwijderen:**



8.8 Scope

Normaal gesproken worden alle cookies voor een site meegezonden bij elk request. Oftewel elke pagina krijgt altijd alle cookies voor de site binnen.

U kunt de scope van de cookies op drie manieren aanpassen:

1. Door de scope te beperken tot een folder op de website. Hierdoor kunnen cookies specifiek voor een applicatie gebruikt worden. U doet dit met het **path** argument.
2. Door de scope te beperken tot een bepaald subdomein van de site. U doet dit met het **domain** argument.
3. Door cookies alleen te verzenden over een beveiligde verbinding. U doet dit met het **secure** argument.

8.8.1 Path

De functie `setcookie()` heeft een optionele vierde parameter **path** waarmee u de directory aan kunt geven waarvoor de cookie actief is. PHP geeft als defaultwaarde de directory waarin het scripts staat dat de cookie zet. Dus als u een cookie `test` zet in een pagina `set_cookie.php` in de map `/cursus/php/` dan is die cookie alleen in die map (en onderliggende subdirectories) uit te lezen. Zet u echter het path op `"/`, dan betekent dit dat de cookie actief is voor de hele site. Met een waarde als `"/forum/"` is het cookie alleen geldig in die directory (en onderliggende subdirectories):

```
// Cookie is alleen geldig in de /forum/ directory
setcookie('naam', 'Arno', time() + 60*60*24, '/forum/');
```

Merk op dat de cookie niet in de map `/forum/` gezet hoeft te zijn!

8.8.2 Domain

Een vijfde parameter **domain** geeft aan in welk subdomein de cookie actief is. Een waarde als `"mail.compufirm.nl"` geeft aan dat de cookie alleen hier actief is en niet in `www.compufirm.nl`. Met `".compufirm.nl"` is het cookie in alle subdomeinen actief.

8.8.3 Secure

Met de laatste (zesde) optionele parameter **secure** geeft u aan wanneer de cookie verzonden mag worden. De standaardwaarde is 0 en dan wordt de cookie zowel over een onbeveiligde (HTTP) als beveiligde verbinding (HTTPS) verzonden. Als u de waarde 1 opgeeft dan wordt de cookie alleen over een beveiligde verbinding verzonden.

8.9 Meerdere cookies

U hoeft het niet bij een cookie te laten. U kunt meerdere naam-waarde paren meesturen:

```
setcookie('voornaam', 'Arno', time() + 60*60*24);
setcookie('achternaam', 'Mol', time() + 60*60*24);
```

Dat ziet er in de HTTP headers zo uit:

```
Antwoordheaders
Date: Fri, 12 Jun 2009 07:30:25 GMT
Server: Apache/2.2.9 (Win32) PHP/5.2.6
X-Powered-By: PHP/5.2.6
Set-Cookie: voornaam=Arno; expires=Sat, 13-Jun-2009 07:30:25 GMT
           achternaam=Mol; expires=Sat, 13-Jun-2009 07:30:25 GMT
Content-Length: 624
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

Uitlezen gaat dan natuurlijk per cookie:

```
if (isset($_COOKIE['voornaam']) &&
    isset($_COOKIE['achternaam'])) {
    echo 'Hoi '. htmlentities($_COOKIE['voornaam']) . ' '.
               htmlentities($_COOKIE['achternaam']) . '!';
}
```

Houd er echter rekening mee dat er in de officiële specificatie niet meer dan 20 per domein gezet mogen worden. Dat is in de meeste browsers echter inmiddels opgehoogd tot 50. Zet u cookie nummer 51 dan is het principe *first in, first out*. Kijk maar eens naar deze lus die er 51 probeert te zetten:

```
for ($i=1; $i<=51; $i++) {
    setcookie('naam_' . $i, 'waarde_' . $i, time()+30);
}
```

Tip: U kunt de inhoud van de cookie array `$_COOKIE` tijdens het programmeren snel visualiseren met `var_dump()` of `print_r()`, net als u dat met `$_GET` of `$_POST` gewend bent:

```
echo '<pre>';
print_r($_COOKIE);
echo '</pre>';
```

Door het gebruik van de `<pre>` tags krijgt u in het browservenster de meer inzichtelijke opmaak zoals die ook in de broncode zichtbaar is.

In het request blijken dan inderdaad nummer 2 t/m 51 beschikbaar:

Cookie uitlezen

```
Array
(
    [naam_2] => waarde_2
    [naam_3] => waarde_3
    [naam_4] => waarde_4
    [naam_5] => waarde_5
    [naam_6] => waarde_6
    [naam_7] => waarde_7
    [naam_48] => waarde_48
    [naam_49] => waarde_49
    [naam_50] => waarde_50
    [naam_51] => waarde_51
)
```


Het uitlezen van meerdere, bij elkaar horende waarden gaat op deze manier niet echt makkelijk. Net als bij gewone variabelen kan het dan ook handig zijn om bij elkaar horende waarden als een array te kunnen benaderen.

Update: In FireFox en Chrome gelden op dit moment (2020) geen limieten meer voor wat betreft aantal en grootte... <https://www.lifewire.com/cookie-limit-per-domain-3466809>

8.10 Meerdere cookies als array uitlezen

U kunt meerdere cookies groeperen door gebruik te maken van de inmiddels bekende array notatie:

```
setcookie('gegevens[voornaam]', 'Arno', time()+3600);
setcookie('gegevens[achternaam]', 'Mol', time()+3600);
```

Let op: er worden nu nog steeds **2 aparte cookies** gezet. Deze manier van werken is dus niet bedoeld om meer naam-waarde paren te kunnen zetten dan de toegestane 20 (of 50) per domein. Het is alleen een handige manier om de naam-waarde paren bij elkaar te houden en de cookies makkelijker te kunnen uitlezen.

In de headers ziet u dat er 2 cookies verzonden zijn:

Antwoordheaders

```
Date: Fri, 12 Jun 2009 08:14:50 GMT
Server: Apache/2.2.9 (Win32) PHP/5.2.6
X-Powered-By: PHP/5.2.6
Set-Cookie: gegevens[voornaam]=Arno; expires=Fri, 12-Jun-2009 09:14:50 GMT
Set-Cookie: gegevens[achternaam]=Mol; expires=Fri, 12-Jun-2009 09:14:50 GMT
Content-Length: 462
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
```

Als u nu met `print_r()` kijkt:

```
echo '<pre>';
print_r($_COOKIE['gegevens']);
echo '</pre>';
```

dan ziet u dat u de twee cookies effectief in een array heeft ondergebracht:

Cookie uitlezen

```
Array
(
    [voornaam] => Arno
    [achternaam] => Mol
)
```

U kunt een gezette waarde dan uitlezen door 2 indici op te geven, de eerste index is de naam van de cookie, de tweede index is de naam van het naam-waarde paar waarvan u de waarde wilt uitlezen:

```
// één element uit de array uitlezen:
echo $_COOKIE['gegevens']['voornaam'];
```

U kunt nu ook alle waarden doorlopen met een foreach lus:

```
if (isset($_COOKIE['gegevens'])) {
    foreach ($_COOKIE['gegevens'] as $key => $value) {
```



```

    echo $key.' = ' . $value . "<br>\n";
}
}

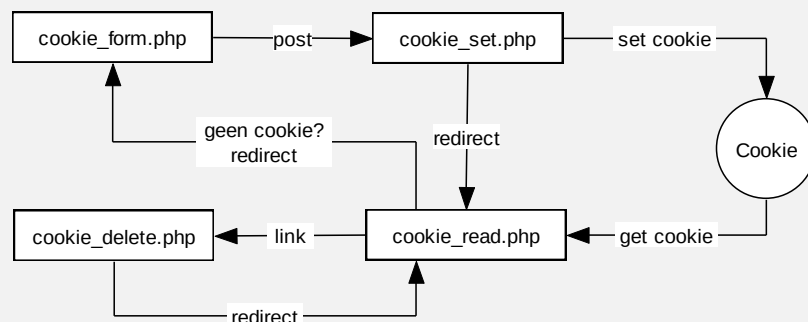
```

Oefening 5.3

Maak een pagina `cookie_form.php` aan die bestaat uit een formulier met twee velden: `voornaam` en `achternaam`. Dit formulier wordt verzonden naar de pagina `cookie_set.php`, waar een cookie `remember_me` aangemaakt wordt met de waarden afkomstig uit het formulier. Bovendien worden ook de huidige datum en tijdstip weggeschreven. De vervaldatum van het cookie is over een maand.

Stuur uw gebruiker met een redirect naar de pagina `cookie_read.php`, waar gecontroleerd wordt of het cookie bestaat. Is dat het geval, dan wordt het cookie uitgelezen en verschijnt een vriendelijke 'persoonlijke' boodschap voor de gebruiker op het scherm. Is er geen cookie gevonden, dan stuurt u de gebruiker weer automatisch door naar `cookie_form.php`. Maak tenslotte op `cookie_read.php` een link naar een pagina `cookie_delete.php` die het cookie verwijdt.

In Schema:



8.11 Meerdere waarden in één cookie

De cookies zoals we ze hiervoor hebben besproken worden ook wel **single value** cookies genoemd. Een dergelijke cookie bevat steeds slechts één waarde. In sommige Server-side technologieën zoals ASP en ASP.NET kunt u echter ook meerdere waarden per cookie meegeven wat aanzienlijk efficiënter is dan voor elke waarde een aparte cookie aan te maken (denk aan het maximum van 20 of 50!). Een dergelijke cookie wordt een **multiple value** cookie genoemd. U verzendt dan als het ware meerdere naam-waarde paren binnen één cookie. Deze noemt men dan weer **subkeys**. En omdat een cookie tot wel 4 Kb groot kan worden heeft u dan dus mogelijkheden genoeg...

Een multiple value cookie met subkeys wordt in de HTTP Response header dan als volgt verzonden:

```
Set-cookie: voornaam=Arno&achternaam=Mol;
```

Waarna ze automatisch als een Collection, zeg maar een associatieve array, beschikbaar worden gemaakt in de aannemende code. Helaas ondersteunt PHP dit principe standaard niet!

8.11.1 Waarde kan geen array zijn

Als u toch meerdere waarden in één cookie wil zetten dan is het een logische stap om te proberen in PHP een array aan te maken en deze dan als waarde aan een cookie toe te kennen.

Helaas gaat dit mis:

```
$gegevens = array('Arno', 'Mol');  
setcookie('gegevens', $gegevens, (time() + 60*60));
```

U krijgt een foutmelding als:

Warning: setcookie() expects parameter 2 to be string, array given in **cookie_array.php** on line 3

Omdat `setcookie()` als waarde een string verwacht, wat op zich natuurlijk ook logisch is, omdat de waarde ook als string in de Response header gezet moet worden.

8.11.2 Explode()

Als alternatief zou u zelf iets kunnen bedenken door een *character separated string* samen te stellen en deze in de cookie te zetten:

```
$gegevens = 'Arno,Mol';  
setcookie('gegevens', $gegevens, (time() + 60*60));
```

In de aannemende code kunt u met `explode()` de string als array benaderen:

```
$gegevens_str = $_COOKIE['gegevens'];  
$gegevens_arr = explode(',', $gegevens_str);  
  
echo '<pre>';  
print_r($gegevens_arr);  
echo '</pre>';
```

En voila, de waarden in de cookie gegevens zijn beschikbaar als array waar u naar hartenlust mee verder kunt werken:

Cookie uitlezen

```
Array  
(  
    [0] => Arno  
    [1] => Mol  
)
```

8.11.3 parse_str() en http_build_query()

Dergelijke arrays hebben echter een numerieke index, waardoor de juiste volgorde van de waarden wel erg belangrijk is. Als u een associatieve index zou willen gebruiken dan zou u de string op kunnen stellen op dezelfde manier als ASP en ASP.NET het doen:

```
$gegevens = 'voornaam=Arno&achternaam=Mol';  
setcookie('gegevens', $gegevens, (time() + 60*60));
```

Als u nu een `explode()` loslaat met het `&`-karakter dan krijgt u de twee naamwaarde paren als array elementen in een numerieke index. Hier zou u weer allerlei bewerkingen op los kunnen laten, maar het kan gelukkig wat eenvoudiger met de functie `parse_str()`.

Deze functie neemt als eerste verplichte argument een querystring met een of meer parameters en maakt deze vervolgens beschikbaar als globale variabelen. Daar zitten we natuurlijk niet op te wachten, maar het optionele, tweede argument maakt het mogelijk om deze variabelen in een array te zetten! En dat is precies wat we willen!

```
$gegevens_str = $_COOKIE['gegevens'];
parse_str($gegevens_str, $gegevens_arr);
```

Met als resultaat de gewenste associatieve array:

Cookie uitlezen

```
Array
(
    [voornaam] => Arno
    [achternaam] => Mol
)
```

Denk er wel aan dat dit alleen werkt als u de datastring net als een querystring opmaakt. En dat de data in de array nog steeds tainted is, oftewel nog steeds ongecontroleerde en daardoor gevaarlijke gebruikersinvoer bevat.

Als u al een associatieve array heeft dan moet u die eerst naar het querystring formaat toebrengen. Dit kan heel makkelijk met de functie `http_build_query()` die een associatieve array omzet naar een URL encoded string:

```
$gegevens_arr = array(
    'voornaam' => 'Arno',
    'achternaam' => 'Mol'
);

$gegevens_str = http_build_query($gegevens_arr);
echo $gegevens_str;
```

Geeft:

```
voornaam=Arno&achternaam=Mol
```

Deze string kunt u vervolgens als waarde aan een cookie meegeven.

8.11.4 serialize() en unserialize()

Een andere oplossing bestaat eruit een reeds bestaande array met de functie `serialize()` om te zetten naar een speciaal geformatteerde string:

```
$gegevens_str = serialize($gegevens_arr);
echo $gegevens_str;
```

De functie zet PHP complexe variabelen als arrays en objecten om naar een speciale string notatie die makkelijk opgeslagen of verzonden kan worden:

```
a:2:{s:8:"voornaam";s:4:"Arno";s:10:"achternaam";s:3:"Mol";}
```

U kunt deze string zo lezen: dit is een array (a) die 2 elementen bevat. Het eerste element heeft als index een string (s) 'voornaam' van 8 karakters lang en heeft als waarde de string 'Arno' van 4 karakters. Het tweede element heeft eveneens een string als index, namelijk 'achternaam' van 10 karakters lengte en dit element heeft de waarde 'Mol', een string van 3 karakters.

Deze string kan vervolgens zonder problemen als stringwaarde in een cookie worden weggeschreven.

Met de functie `unserialize()` kan PHP de string weer terug zetten naar de oorspronkelijke array:

```
$gegevens_arr = unserialize($_COOKIE['gegevens']);  
print_r($gegevens_arr);
```

Maar afhankelijk van uw PHP configuratie kan dit gepaard gaan met de volgende foutmelding:

Notice: unserialize(): Error at offset 9 of 36 bytes in `cookie_array.php`

U krijgt een dergelijke melding als de setting `magic_quotes_gpc` aan staat. In dat geval worden de quotes in de cookie waarden geëscaped met backslashes (de `c` in `gpc` staat voor cookies). En zoals u hierboven al kon zien staan er nogal wat (dubbele) quotes in de geserialiseerde string. Wanneer u de meegestuurde cookie zou uitlezen zou u dit te zien krijgen:

```
a:2:{s:8:"voornaam";s:4:"Arno";s:10:"achternaam";s:3:"Mol";}
```

U kunt de backslashes verwijderen met de in hoofdstuk 4 besproken `stripslashes()` en dan is het probleem opgelost:

```
$gegevens = unserialize(stripslashes($_COOKIE['gegevens']));  
print_r($gegevens);
```

Dit geeft weer netjes de array terug:

```
Array ( [voornaam] => Arno [achternaam] => Mol )
```

8.12 Cookies 'versleutelen'

Om te voorkomen dat een gebruiker gelijk ziet wat er in een cookie is opgeslagen en dan op het idee zou kunnen komen om dit aan te passen, zou u kunnen overwegen de waarde van het cookie te versleutelen.

Een eenvoudige manier om dit principe te illustreren is met behulp van base64:

```
setcookie('naam', base64_encode($_POST['naam']), time() + 60*60*24*365);
```

Wanneer een gebruiker nu "Arno" invult in het formulier dan zal dit als "QXJubw==" worden weggeschreven in de cookie, iets wat de gemiddelde gebruiker niet veel zal zeggen. Om de cookie uit te lezen gebruikt u dan:

```
echo 'Hoi ' . base64_decode($_COOKIE['naam']);
```

Let op: Dit is geen echte encryptie, maar een vorm van *character mapping*. Het is voor een ervaren gebruiker geen probleem om alsnog de cookiewaarde uit te lezen en aan te passen.

Echte encryptie zou u bijvoorbeeld kunnen doen met behulp van `mcrypt()`, maar dit is geen standaard PHP functie. Bovendien valt een bespreking van encryptie buiten het bestek van deze cursus. Meer informatie over encryptie en het versleutelen van cookies vindt u bijvoorbeeld op:

http://www.phpmag.net/itr/online_artikel/psecom.id,667,nodeid,114.html

Een handige class voor het versleutelen van cookies is bijvoorbeeld `clsCookie` (<http://www.phpclasses.org/browse/package/2007.html>).

8.13 Voorbeeld: instelling vasthouden

Cookies worden, zoals we in de inleiding al zagen, in de praktijk veel gebruikt voor het vasthouden van instellingen die een gebruiker maakt. U moet dan niet alleen controleren of de cookie al is gezet of niet, maar er ook rekening mee houden dat een gebruikerskeuze, via GET of POST, gelijk moet worden doorgevoerd en met een cookie moet worden 'vast' gezet.

Hieronder ziet u een klein voorbeeldje waarin de gekozen waarde van een selectbox met een cookie wordt vastgehouden. Bijvoorbeeld om het aantal records per pagina voor een gepagineerde recordset in te stellen. U zou gelijksoortige functionaliteit echter ook kunnen gebruiken voor bijvoorbeeld een taalkeuze op een multilingual site.

```
// geef toegestane keuzes op
$keuzes = array(5, 10, 15, 20, 25);
// stel standaardwaarde in
define('STANDAARD_AANTAL', 10);

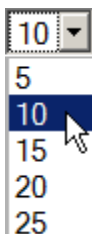
// heeft de gebruiker zojuist een keuze gemaakt?
if (isset($_GET['aantal'])) {
    // gebruik dan deze waarde
    $aantal = intval($_GET['aantal']);
    // en zet hem in een cookie voor de volgende keer
    setcookie('aantal', $aantal, time() + 60*60*24*365);
// geen keuze, maar is er wel een cookie?
} elseif (isset($_COOKIE['aantal'])) {
    // gebruik dan de cookie
    $aantal = intval($_COOKIE['aantal']);
// geen keuze en geen cookie
} else {
    // gebruik dan de default waarde
    $aantal = STANDAARD_AANTAL;
}

// als aantal onverhoopt een ongeldige waarde heeft
// val dan terug op defaultwaarde
if (!in_array($aantal, $keuzes)) {
    $aantal = STANDAARD_AANTAL;
}

...

<form method="get">
<select onchange="submit()" name="aantal">
<?php
    // geef vijftallen als keuze
    foreach ($keuzes as $keuze) {
        // selecteer de huidige waarde
        $selected = $aantal == $keuze ? ' selected="selected"' : '';
        echo ' <option' . $selected . '>' . $i . '</option>' . PHP_EOL;
    }
?>
</select>
</form>
```

In dit voorbeeld wordt een selectbox met een vijftal opties getoond:



De eerste keer dat de pagina rechtstreeks wordt opgevraagd is er geen GET parameter en ook geen cookie. De waarde voor de variabele aantal wordt dan de standaardwaarde 10. Deze optie wordt tevens voorgeselecteerd weergegeven in de selectbox.

Als een gebruiker een ander aantal kiest, dan wordt het formulier naar zichzelf verzonden en wordt de waarde in de GET parameter gebruikt voor aantal. Daarbij wordt ook gelijk de waarde in een cookie vast gezet, zodat, wanneer de pagina een volgende keer rechtstreeks wordt opgevraagd, de waarde van cookie gebruikt kan worden.

Als een gebruiker de cookie zou verwijderen dan wordt automatisch weer de standaardwaarde 10 gebruikt en is er verder niks aan de hand.

Als een gebruiker de waarde van de cookie zou aanpassen dan zorgt intval() ervoor dat tekst de waarde 0 krijgt, maar dat is natuurlijk een waarde die problemen oplevert. Bovendien kan een gebruiker nog steeds getallen als -1 en 23 opgeven en dat willen we ook niet, vandaar dat we echt controleren of de waarde in de array met toegestane waarden zit. Is dat niet het geval dan vallen we ook weer terug op de standaardwaarde.

Verdieping	<i>Maak twee extra stylesheets voor uw website door de bestaande te kopiëren en onder andere namen op te slaan. Maak een paar eenvoudige aanpassingen op de bestaande opmaak, bijvoorbeeld door andere achtergrondkleuren en andere lettertypes te kiezen. Maak vervolgens een mogelijkheid waarmee een gebruiker kan kiezen tussen de drie verschillende stijlen voor uw site. Op het moment dat een gebruiker voor een andere stijl kiest wordt een cookie gezet voor dit thema. Op basis van de cookiewaarde besluit u welke stylesheet u insluit in uw functie begin_page(). Alle pagina's in uw site hebben nu (tijdelijk) de nieuwe stijl!</i>

8.14 Samenvatting

Met cookies kunt u leuke dingen doen. Het grootste voordeel is dat u tamelijk eenvoudig voor een lange tijd instellingen kunt vasthouden. Maar er kleven wel een aantal nadelen aan het concept. Omdat cookies aan de Client-side worden opgeslagen, kunt u niet op hun aanwezigheid vertrouwen. U kunt er ook geen gevoelige data inzetten. Bovendien zijn ze makkelijk door kwaadwillenden aan te passen. Tot slot kunt u er maar minimale hoeveelheden data mee vasthouden.

Gelukkig kent PHP nog een ander mechanisme om status en dus data voor specifieke gebruikers vast te houden: sessies. In het volgende hoofdstuk zult u zien dat sessievariabelen de eerder genoemde nadelen niet kennen. Ze worden aan de Server-side aangemaakt en u zou ze als een soort Server-side Cookies kunnen beschouwen waar u behoorlijke hoeveelheden data op een veilige manier kunt opslaan.

Sessies maken echter ook gebruik van Client-side cookies zoals we in het volgende hoofdstuk zullen zien. Het enige wat hierin echter vastgehouden wordt is een uniek ID voor de bezoeker.

Werken met Sessies

9.1 Inleiding

De in het vorige hoofdstuk besproken **cookies** houden status vast aan de **Client-side**. De client stuurt door ons gezette naam-waarde paren in de HTTP Request headers mee bij elk request. Maar gebruikers kunnen cookies aanpassen of weggooien, vandaar dat we cookies dan ook als (onveilige) user data moeten behandelen.

Het is dan vaak ook handiger om de status aan de **Server-side** vast te houden. We gebruiken daartoe het principe van **sessions** (sessies). Een session is een manier om **sessie variabelen** te identificeren en te beheren voor een bepaalde gebruiker. Wanneer een sessie gestart wordt krijgt de gebruiker een zogenaamde session identifier uitgedeeld (meestal in de vorm van een cookie). In plaats van steeds alle data te verzenden, hoeft de client nu alleen deze unieke code mee te sturen waarmee de bij de sessie van die gebruiker behorende data op de server benaderd kan worden. Het principe lijkt wel een beetje op het bonnetje van een garderobe, waarmee een concertbezoeker zijn jas en tas kan ophalen.

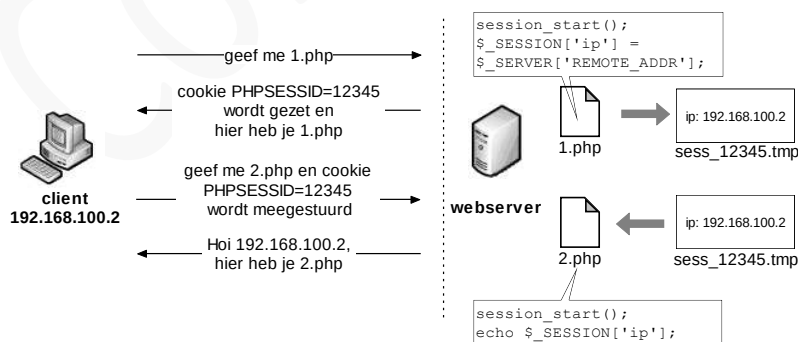
Vanaf PHP4 kunt u wanneer u met sessies wilt gaan werken, gebruik maken van het **ingebouwde session management** om data voor specifieke gebruikers op te slaan. In PHP3 moest u nog alle voorzieningen zelf treffen.

In dit hoofdstuk bespreken we de werking en de (on)mogelijkheden, waarna we sessies zullen gebruiken om gebruikers in te laten loggen op onze website. Alleen ingelogde gebruikers mogen onze medewerkers tabel zien en aanpassen, maar daarover later meer.

Laten we eerst maar naar de globale werking van het mechanisme kijken:

9.2 Werking

Bij het starten van een nieuwe sessie wordt een **non-persistent cookie** met daarin een speciaal ID naar de client gestuurd. De client stuurt vervolgens dit ID bij elke volgende aanvraag weer mee. Dit mechanisme maakt het mogelijk om op de server data vast te houden voor deze specifieke gebruiker en alleen deze gebruiker daar toegang toe te geven:



Een groot verschil met cookies is dat de naam-waarde paren nu **op de server** bewaard worden in tijdelijke bestanden. Dat is op zich veiliger maar de gegevens gaan wel verloren wanneer de browser wordt gesloten en de sessie eindigt, tenzij u speciale maatregelen treft.

9.3 Sessie starten

PHP start niet automatisch een sessie op voor elke gebruiker. Dat komt omdat de entry `session.autostart` in de `php.ini` standaard *disabled* is. Meestal is dit de meest logische oplossing omdat anders elk request voor een PHP pagina op de server een sessie zou opstarten en dat is meestal helemaal niet nodig en zou onnodig veel resources kosten...

Het houdt wel in dat u alle pagina's waarop u sessies wilt gaan gebruiken daarom moet beginnen met de functie `session_start()`:

```
<?php
// hier mag wel PHP code staan maar er mag (nog)
// niets naar de client worden geschreven!
// tenzij ob_start() aanstaat...
session_start();
?>
```

Deze plaatst u helemaal bovenaan de pagina, óf u zorgt er in ieder geval voor dat er nog geen uitvoer wordt gegenereerd, óf dat deze gebufferd wordt middels `ob_start()`. De reden is dezelfde als we al eerder bij `header()` en `set_cookie()` zagen: Omdat de HTTP Response headers door deze functies worden aangepast mogen deze niet al reeds verzonden zijn door eerdere output!

Tip: Als de pagina's in uw applicatie worden opgebouwd door het insluiten van een `common.inc.php` die op zijn beurt een `config.inc.php` insluit (zoals besproken in Hoofdstuk 1) dan is het handig om `ob_start()` en `session_start()` bovenaan uw `config.inc.php` te zetten zodat u er verder geen omkijken naar heeft:

```
<?php
//dit is config.php

// maximaal waarschuwningsniveau
error_reporting(E_ALL);
// zet outputbuffer aan
ob_start();
// start sessie
session_start();

...
?>
```

Vanaf nu is het dan mogelijk om op elke pagina met sessievariabelen te werken!

De function `session_start()` doet de volgende drie dingen:

1. De functie kijkt allereerst of er een sessie id werd meegezonden in het request.
2. Als dat niet het geval is dan wordt een nieuw sessie id gegenereerd en als non-persistent cookie naar de client gestuurd: een nieuwe is sessie gestart.
3. Is het sessie id al aanwezig dan maakt de functie de sessievariabelen voor deze specifieke sessie beschikbaar in de `$_SESSION` super global array.

Laten we eerst eens naar het sessie ID kijken. De eerste keer dat iemand een pagina bezoekt met `session_start()` zal er nog geen cookie met een session ID zijn aangemaakt en kan de client deze dus ook niet meezenden.

Er wordt nu namelijk een unieke SessionID aangemaakt en een nieuwe sessie gestart. De SessionID is een random hexadecimaal getal van 32 karakters en dit wordt opgeslagen in een **non-persistent cookie** dat standaard `PHPSESSID` wordt genoemd (de naam is eventueel in `php.ini` te wijzigen). De cookie wordt in de HTTP Response header naar de gebruiker verzonden:

Antwoordheaders

```

Date Tue, 23 Jun 2009 09:42:46 GMT
Server Apache/2.2.9 (Win32) PHP/5.2.6
X-Powered-By PHP/5.2.6
Set-Cookie PHPSESSID=qdbaes4es5l197ndp7tr1uar57; path=/
Expires Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma no-cache
Content-Length 468
Keep-Alive timeout=5, max=100
Connection Keep-Alive
Content-Type text/html; charset=utf-8

```

Het unieke ID wordt daarna door de client met elk HTTP Request mee teruggestuurd:

Verzoekheaders

```

Host localhost
User-Agent Mozilla/5.0 (Windows; U; Windows NT 6.0; nl-NL; rv:1.9.0.11) Gecko
3.5.30729)
Accept text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding gzip,deflate
Accept-Charset ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive 300
Connection keep-alive
Referer http://localhost/phpadv/session_set.php
Cookie PHPSESSID=qdbaes4es5l197ndp7tr1uar57

```

U kunt de aan de sessie toegekende ID eventueel uitlezen met de opdracht

```
echo $_COOKIE["PHPSESSID"];
```

of met de functie `session_id()`:

```
echo session_id();
```

Dit geeft in beide gevallen hetzelfde resultaat, de unieke waarde van de session identifier, bijvoorbeeld:

```
qdbaes4es5l197ndp7tr1uar57
```

Een volgende gebruiker zou een ander SessionID toegewezen krijgen, waarmee de bij zijn sessie behorende data kan worden benaderd.

9.4 Sessie variabele aanmaken

Nadat u op een PHP pagina heeft aangegeven dat u een sessie wilde starten, kunt u vervolgens sessievariabelen aanmaken en er een waarde aan toekennen. U doet dat door een nieuwe index aan te maken in de superglobal array `$_SESSION` en er een waarde aan toe te kennen, bijvoorbeeld:

```

<?php
// sessie starten
session_start();

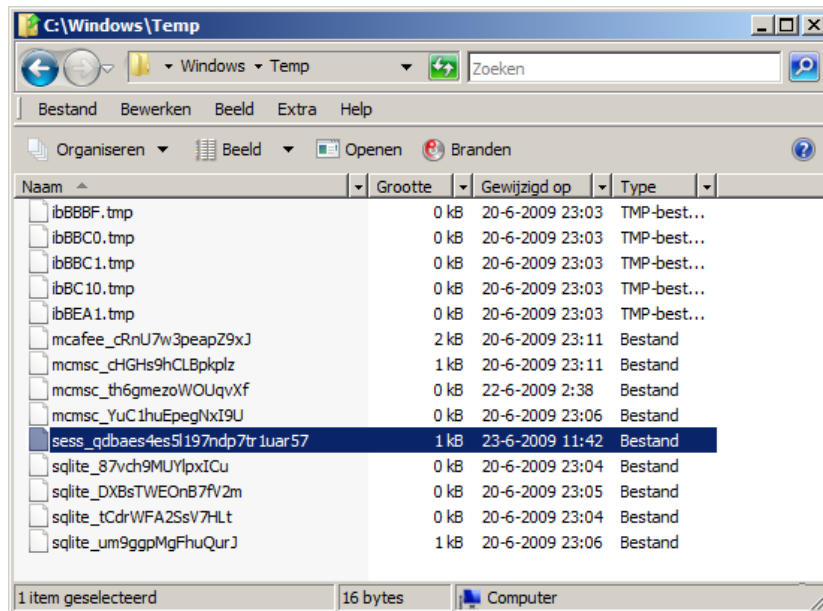
// sessie variabele zetten
$_SESSION['naam'] = 'Arno';
?>

```

Uiteraard had de waarde voor de sessievariabele ook uit een andere variabele kunnen komen, bijvoorbeeld uit een formulier...

Door het starten van de sessie wordt een tijdelijk bestand aangemaakt **op de server**. Het wordt aangemaakt in de directory die gespecificeerd is in de `session.save_path` entry in het `php.ini` configuratiebestand. Het bestand krijgt de naam `sess_` gevolgd door de SessionID. Ook als er geen sessievariabele wordt aangemaakt dan zal dit bestand worden aangemaakt, het is dan alleen leeg...

Als u PHP onder Windows draait zou u het bestand bijvoorbeeld in `C:\Windows\Temp` kunnen zien staan:



In het sessie bestand worden alle waarden van de sessievariabelen voor deze sessie opgeslagen. Dit gebeurt **automatisch in geserialiseerd formaat**. De inhoud van het sessiebestand is in bovenstaand voorbeeld:

```
naam|s:4:"Arno";
```

Oftewel de sessie variabele `naam` bevat een string (`s`) van 4 karakters lengte met de waarde `"Arno"`. Zoals u ziet lijkt de syntax sterk op het resultaat van `serialize()` uit het vorige hoofdstuk.

9.4.1 Save_PathValue

U kunt de locatie waar de sessiebestanden worden opgeslagen wijzigen in `php.ini` via instelling `session.save_pathvalue`. Het is handig om in een shared omgeving (zoals bij een provider gebruikelijk is) een map binnen uw eigen directory tree te kiezen zodat andere gebruikers er niet bij kunnen.

9.5 Sessie variabele uitlezen

In tegenstelling tot een cookie kunt u de waarde van de sessie variabele **meteen uitlezen op dezelfde pagina** als waarop u de variabele instelde. Het naam-waarde paar wordt namelijk niet, zoals bij een cookie, door de client meegestuurd maar blijft gewoon op de server staan.

Voor het **uitlezen** van de sessievariabele **op een andere pagina** op de website moet u er echter **wederom** `session_start()` **gebruiken** bovenaan de pagina. PHP bekijkt nu of er al een sessie gestart is voor de gebruiker met dit ID. Zo ja, dan worden de bijbehorende variabelen uit het bijbehorende sessiebestand gedeserialiseerd en ingelezen in de `$_SESSION` array. Zo nee, dan

wordt een nieuwe sessie gestart en is `$_SESSION` in eerste instantie leeg. Dit alles gaat geheel automatisch, u hoeft er verder niets voor te doen.

Als u vergeet de sessie opnieuw te starten dan krijgt u afhankelijk van het `error_reporting` niveau of niets, of een waarschuwing, te zien. De waarschuwing ziet er zo uit:

Notice: Undefined variable: `$_SESSION` in `/sessie.php` on line 3

Merk op dat er melding gemaakt wordt van een **undefined variable** en niet van een **undefined index**. Wanneer u bij `$_GET`, `$_POST` en `$_COOKIE` superglobals probeert een niet bestaande waarde uit te lezen, krijgt u alleen een melding over een niet bestaande index. Deze superglobal arrays zijn er namelijk **altijd**. De `$_POST` array bestaat gewoon, ook als u het script via GET oproept. Hij bevat dan alleen geen elementen. In het geval van de `$_SESSION` array is het dus zo dat deze in zijn geheel niet beschikbaar is als u de sessie niet eerst gestart heeft met `session_start()`!

De oplossing bestaat eenvoudig uit het toevoegen van `session_start()` boven aan de pagina. Daarna kunt u de waarde van de sessievariabele gewoon uitlezen:

```
<?php
// sessie is beschikbaar, lees variabelen in
session_start();

// sessie variabele weergeven
echo 'Hoi ' . $_SESSION['naam'] . '!';
?>
```

9.5.1 Inlezen in globale variabele

Er wordt vaak voor gekozen om de inhoud van een sessie variabele in een normale, globale variabele in te lezen en die verder in uw code te gebruiken:

```
$naam = $_SESSION['naam'];
```

Dit is niet verplicht maar maakt volgens sommigen de code wat leesbaarder. Bovendien hoeft de waarde maar 1x vanuit het sessiebestand ingelezen te worden in plaats van elke keer opnieuw als u er op de pagina aan refereert.

9.5.2 Bereik

U kunt nu in principe **overal op uw website** gebruik maken van de sessievariabele `$_SESSION['naam']`. Het path voor de PHPSESSID cookie wordt namelijk standaard op `"/"` gezet, zodat de sessie ID naar elke pagina binnen het domein wordt meegestuurd.

Het enige waaraan u moet denken is dat u op elke pagina waarop u iets met sessie variabelen wilt doen u de sessie ook daadwerkelijk moet hebben gestart...

9.5.3 Controle

Uiteraard is het daarbij verstandig om met `isset()` te controleren of de sessie variabele wel aanwezig is, voordat u er mee aan de slag gaat:

```
if (isset($_SESSION['naam'])) {
    echo 'Hoi ' . $_SESSION['naam'] . '!';
} else {
    echo 'Hoi onbekende!';
}
```

Dit werkt op precies dezelfde manier als we al eerder hebben gezien bij `$_GET`, `$_POST` en `$_COOKIE`.

9.5.4 Meerdere gebruikers

Het mooie van sessies is niet alleen dat u heel eenvoudig waarden voor een gebruiker kunt vasthouden, maar vooral ook dat u dat voor meerdere gebruikers tegelijkertijd kan doen! Elke gebruiker krijgt namelijk zijn eigen unieke sessieID en in het bijbehorende sessiebestand worden alleen zijn of haar waarden vastgehouden. Zo kunnen tientallen gebruikers tegelijkertijd een ander gevuld winkelwagentje hebben, terwijl u in uw code maar met één sessie variabele hoeft te werken!

9.5.5 Toepassingen

Sessie variabelen zijn heel makkelijk in het gebruik en tamelijk veilig (zie verderop) en worden daarom graag en vaak gebruikt door ontwikkelaars. Sessievariabelen kunnen natuurlijk voor tal van klussen ingezet worden waarbij waarden over meerdere pagina's beschikbaar moeten zijn.

Denk bijvoorbeeld aan:

- Winkelwagentjes
- Multi-page forms

Maar zoals we straks zullen zien worden sessie variabelen ook veel gebruikt om **delen van een website af te schermen** zodat alleen ingelogde gebruikers toegang hebben. U kunt zich voorstellen dat het absoluut onwerkbaar zou zijn als een gebruiker op elke pagina opnieuw zou moeten inloggen. Met een sessievariabele kunt u het feit dat een gebruiker succesvol is ingelogd gebruiken om die persoon toegang te geven tot alle beveiligde pagina's. Wanneer die sessie variabele niet bestaat of niet de juiste waarde heeft dan zijn die pagina's echter verboden terrein.

We zullen dit aan het einde van dit hoofdstuk uitwerken in een praktijkvoorbeeld.

9.5.6 Meerdere Sessievariabelen

Wanneer u meerdere sessievariabelen gebruikt dan worden deze allemaal in hetzelfde tekstbestand weggeschreven:

```
// meerdere sessie variabelen zetten
$_SESSION['voornaam'] = 'Arno';
$_SESSION['achternaam'] = 'Mol';
```

geeft in het sessiebestand dit:

```
voornaam|s:4:"Arno";achternaam|s:3:"Mol";
```

U ziet dat de geserialiseerde naam-waarde paren worden gescheiden door punt-komma's.

U kunt in principe zoveel variabelen aanmaken als u wilt. In principe is de schijfruimte/werkgeheugen op de server de beperkende factor.

9.5.7 Datatypes

U kunt vrijwel elk datatype in een sessievariabele zetten vanwege de automatische (de)serialisatie. Dus ook (multi-dimensionale) arrays en objecten.

Bij objecten geldt wel de beperking dat er geen circulaire referenties gemaakt mogen worden. Bovendien moet PHP de beschikking hebben over de class definities van elk sessie object, en dus

moeten deze op elke pagina worden ingesloten, ook al wordt een class misschien helemaal niet op de betreffende pagina gebruikt.

Het enige wat echt niet in een sessievariabele kan zijn resources, zoals bijvoorbeeld een database link identifier of een file handle.

9.6 Sessie variabele aanpassen

Wanneer u de waarde van een sessie variabele wilt aanpassen dan kunt u gewoon de bestaande waarde overschrijven, net zoals u dat bij gewone variabelen zou doen.

9.6.1 Sessie variabelen en register_globals

Voorzichtigheid is geboden wanneer in `php.ini` de waarde voor `register_globals` op `on` staat. In dit geval overschrijven globale variabelen met een zelfde naam als de index in de sessievariabelen de waarde van deze sessie variabelen!

Neem dit voorbeeld:

```
session_start();

$_SESSION['test'] = 'session';

// LET OP: globale variabele met dezelfde naam
// als de index van de sessie variabele
$test = 'global';

// als u nu de sessie variabele wegschrijft
// heeft deze de waarde van $test gekregen!

echo $_SESSION['test']; // print 'global'
```

Gelukkig is `register_globals` vanaf PHP 5.3 vervallen en sinds PHP 5.4 verwijderd.

9.7 Duur van de sessie

Bij sessies wordt dat op de server opgeslagen. Maar een belangrijke vraag is natuurlijk hoe lang moet die opslag duren? In veel applicaties wordt in- en uitgelogd en bij uitloggen kunnen we de data van de sessie natuurlijk gerust opruimen. Maar als er niet wordt uitgelogd? De server kan niet weten of een gebruiker er nog is. Niets doen is geen optie, want dan zou data achterblijven en dit neemt onnodig ruimte in beslag en levert mogelijk ook veiligheidsrisico's op. De data moet dus opgeruimd worden nadat een sessie beëindigd is. Maar wanneer eindigt dan een sessie?

In principe blijft een sessie lopen zolang het Sessie ID wordt meegestuurd. Zolang de gebruiker de browser niet sluit en steeds pagina's opvraagt uit het domein zal dat het geval zijn.

Wanneer een gebruiker niets doet dan zal na een bepaalde tijd de sessie als beëindigd worden beschouwd. Dit wordt de **Session TimeOut** genoemd en deze is standaard **1440 seconden** oftewel **24 minuten**. Misschien heeft u deze melding al eens op het inlogscherf van phpMyAdmin gezien...

Wanneer de gebruiker zijn browser sluit, dan verdwijnt de sessie cookie waardoor bij het naderhand opnieuw opvragen van een pagina een nieuw sessie id gegenereerd zal worden en de oude sessie dus niet meer benaderd kan worden (al zijn de gegevens op de server misschien nog niet verwijderd...).

Het daadwerkelijke opruimen van oude sessie data, de **garbage collection**, wordt door PHP op de achtergrond geregeld. Het is een samenspel van een aantal instellingen in `php.ini` die daarvoor zorgt.

Oefening 6.1

Maak een pagina `session_set.php` waarin u een sessievariabele `test` aanmaakt met een link naar een pagina `session_read.php` waar u de sessievariabele uitleest.

Gooi ook de browser eens dicht en vraag `session_read.php` direct op. Zorg er voor dat er een default waarde getoond wordt.

Oefening 6.2

Maak een pagina `session_counter.php` die controleert of er een sessievariabele `count` is. Zo niet dan wordt deze aangemaakt en op 1 gezet. Zo ja, dan wordt hij met 1 opgehoogd. Vervolgens toont u hoe vaak de pagina bezocht is (tijdens de sessie).

Tip: Plaats een naar de pagina zelf verwijzende hyperlink om het ophogen makkelijk te kunnen testen.

9.8 Sessie variabele verwijderen

Een sessie variabele verwijdert u net als andere variabelen, met `unset()`. Denk er wel aan dat u eerst de sessie met `session_start()` toegankelijk moet hebben gemaakt:

```
<?php
session_start();
// ...
unset($_SESSION['count']);
?>
```

Let op: Doe dat **niet** op de hele `$_SESSION` superglobal (als in `unset($_SESSION)`) omdat het dan niet langer mogelijk is om überhaupt sessie variabelen aan te maken!

Als u alle variabelen in een keer wilt verwijderen dan kunt u natuurlijk de sessie beëindigen (zie hierna) maar u kunt ook eenvoudig weer een lege array maken:

```
$_SESSION = array();
```

9.9 Session beëindigen

Een sessie eindigt in principe al wanneer de browser wordt gesloten. Soms wilt u een sessie echter expliciet beëindigen, bijvoorbeeld in wanneer een gebruiker uitlogt. Dit kunt u realiseren met `session_destroy()`:

```
<?php
session_start();
session_destroy();
?>
```

Let op: U dient dus een sessie eerst weer met `session_start()` toegankelijk te maken, voordat u hem kunt vernietigen!

Hiermee worden alle voor de sessie opgeslagen waarden vernietigd, oftewel het bij de sessie behorende temp bestand wordt verwijderd. De PHPSESSID cookie in de browser blijft echter bestaan! Wanneer deze waarde opnieuw wordt verzonden door de gebruiker dan ziet PHP dat er geen bijbehorend bestand is. Dit wordt dan opnieuw aangemaakt, maar er wordt dan weer met een schone lei begonnen... De oude waarden zijn niet meer beschikbaar!

Let op: Voor een uitlog pagina hoeft u wellicht niet de hele sessie te vernietigen omdat u dan ook alle andere sessie variabelen voor het publieke deel weggooit!

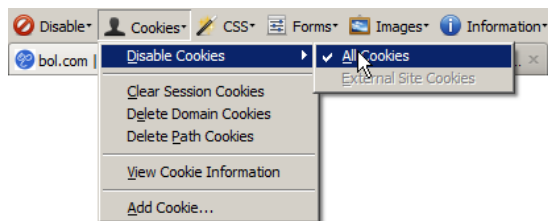
9.10 Client zonder cookie support, wat nu?

Wanneer een gebruiker het accepteren van cookies in zijn browser uitzet dan hebben we een probleem want de hele werking van sessies is, zoals u inmiddels weet, gebaseerd op cookies. Om die reden waarschuwen veel webapplicaties, zoals bijvoorbeeld webshops, dan ook dat zonder cookies alleen beperkte functionaliteit mogelijk is:

Informatie over cookies

De browser die je gebruikt laat geen cookies toe.
Om veiligheidsredenen is het niet mogelijk om te bestellen bij bol.com zonder cookies.
Producten zoeken en bekijken kan wel zonder cookies.

U kunt dit zelf snel ondervinden door in de Web Developer Toolbar van FireFox de cookies tijdelijk uit te schakelen:



Omdat PHP wel een sessie ID genereert zou u ervoor kunnen kiezen om deze op een alternatieve manier door te geven door zelf het id handmatig aan alle uitgaande links toe te voegen:

```
<a href="test.php?PHPSESSID=<?php echo session_id(); ?>">test</a>
```

Dit kan overigens iets korter door de ingebouwde SID constante te gebruiken die de hele benodigde querystring bevat:

```
<a href="test.php?<?php echo SID; ?>">test</a>
```

Beiden leveren het gewenste resultaat (te zien in de HTML broncode):

```
<a href="test.php?PHPSESSID=u055bee8ns6qgif64k2ebf5q17">test</a>
```

Op deze manier is het Session ID toch op elke pagina van de applicatie beschikbaar en kunt u de sessie variabelen gewoon blijven gebruiken. Denk er overigens wel aan dat als u al een querystring begonnen was u de sessie id er met een ampersand achter moet plakken:

```
<a href="test.php?x=y&?&?php echo SID; ?>">test</a>
```

Maar een dergelijke oplossing is natuurlijk nog al tijdrovend en foutgevoelig. U vergeet heel snel een hyperlink of formulier action. Dat zorgt ervoor dat er dan geen Sessie ID gevonden wordt waardoor PHP automatisch een nieuwe genereert en de informatie uit de vorige sessie niet meer toegankelijk is. Bovendien moet u natuurlijk ook nog ergens een mechanisme inbouwen dat ervoor zorgt dat de Sessie ID's alleen worden toegevoegd wanneer cookies niet worden ondersteund.

Gelukkig is er een manier om dit automatisch door PHP te laten doen, als PHP tenminste met de optie `--enable-trans-sid` geïnstalleerd is. Geef vervolgens in `php.ini` de optie `session.use_trans_sid` de waarde 1:

```
session.use_trans_sid = 1
```

Dan wordt de SessionID automatisch achter elke **relatieve** link geplakt! U hoeft dan verder op link/formulier niveau niets toe te voegen.

Als u geen wijzigingen in `php.ini` kunt aanbrengen omdat u bij een provider host dan kunt u het ook op pagina niveau regelen met `ini_set()`:

```
<?php
// kan op pagina niveau worden aangezet
ini_set('session.use_trans_sid', '1');
session_start();

// vul sessie variabele
$_SESSION['browser'] = $_SERVER['HTTP_USER_AGENT'];

// als cookies niet gesupport worden
// dan plakt PHP nu automatisch de SessionID
// achter alle uitgaande links...
?>

<a href="test.php">test</a>
```

Wanneer u nu de broncode van deze pagina bekijkt, dan zult u zien dat PHP zelf achter de uitgaande link het sessieID heeft geplakt:

```
<a href="test.php?PHPSESSID=u055bee8ns6qgif64k2ebf5q17">test</a>
```

En na het volgen van de hyperlink is de inhoud van de sessievariabele `$_SESSION['browser']` gewoon beschikbaar in `test.php`.

Let op: U moet met deze manier van werken echter wel oppassen! Omdat de SessionID nu als onderdeel van de URL verzonden wordt kunnen anderen er ook makkelijk aan komen. Een eindgebruiker kan bijvoorbeeld zonder verdere bedoelingen nu een link in een mailtje zetten waarin ook zijn SessionID staat. Of dat SessionID via de browser history achterlaten op een publiek toegankelijke computer. Hij kan ook een link op zijn eigen PC in de bookmarks opnemen en daardoor steeds hetzelfde SessionID blijven gebruiken. Kortom de mogelijkheden tot misbruik nemen enorm toe... In een volgend onderdeel gaan we uitgebreid in op dit soort problemen.

9.11 Applicatie variabelen

In PHP heeft u niet de beschikking over Applicatie variabelen, zoals u die bijvoorbeeld wel heeft in Classic ASP en ASP.Net. Applicatie variabelen zijn variabelen die bij het opstarten van de applicatie in het geheugen van de webserver worden ingelezen en voortdurend beschikbaar zijn in alle scripts. De waarden zijn aanpasbaar maar hebben bij uitlezen een gelijke waarde voor elke gebruiker. Ze worden in ASP bijvoorbeeld gebruikt om connectionstrings in te bewaren, om dynamisch gegenereerde data te cachen en om tellertjes bij te houden.

Zoals gezegd bestaan dergelijke variabelen niet in PHP. Mocht u er toch dringend behoefte aan hebben dan kunt u wel zelf een global array aanmaken die u bijvoorbeeld `$_APPLICATION` noemt, maar die is dus niet superglobal en u zult zelf een mechanisme moeten bedenken om de waarden in te lezen en weg te schrijven. Het makkelijkst is het waarschijnlijk om daar een tekstbestand voor te gebruiken.

Een dergelijke oplossing wordt duidelijk beschreven op:
<http://www.leosingleton.com/projects/code/phpapp/>

9.12 PRAKTIJK: Login

Onze medewerkers gegevens zijn op dit moment voor iedereen vrij in te zien en te wijzigen, zelfs verwijderbaar! Dat kan natuurlijk niet en daarom zullen we met de kennis die u over cookies en sessies hebt opgedaan login functionaliteit aan de applicatie toevoegen. Men heeft het in dit verband ook wel over **Form based Web Authentication and Authorization**. Authentication is het proces waarbij we de gebruiker identificeren en Authorization het proces waarbij we bepalen welke rechten de gebruiker heeft.

9.12.1 Voorbereidingen

Zorg ervoor dat uw applicatie, als dat nog niet het geval is, over een homepage beschikt. Deze zal voor iedereen toegankelijk zijn (en blijven). Verder dient op iedere pagina in de applicatie een navigatiemenu aanwezig te zijn, met hierin hyperlinks naar:

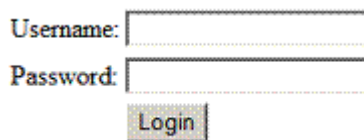
- home
- medewerkers
- contact
- about
- login

9.12.2 Login Form

Om in te kunnen loggen is een login form nodig. U kunt daar een self-submitting formulier dat u al eerder gemaakt heeft voor hergebruiken:

Open `groet.php` (of een ander self-submitting form) en sla op als `login.php`.

Maak van de `groet_form()` een `login_form()` met daarin een inlogformulier met een veld `username` en een veld `password`:



The image shows a simple login form. It has two text input fields. The first is labeled 'Username:' and the second is labeled 'Password:'. Below these fields is a button labeled 'Login'.

9.12.3 Login Check

Na verzenden komt u in de functie `login_check()` waar u controleert of `$_POST['username']` en `$_POST['password']` overeenkomen met de door u gekozen waarden.

Tip: Neem voor het testen eerst super simpele waarden als 'a' en 'b' voor de username en password... Het gaat om de principes, u wilt niet dat mogelijke typefouten roet in het eten kunnen gooien!

Als zowel username als password overeen komen dan zet u **twee Session variabelen**, een voor access en een voor de username en daarna sluist u de gebruiker via een `header()` functie door naar `medewerkers_tonen.php`:

```
if ($_POST['password'] == 'a' && $_POST['username'] == 'b') {
    // zet sessievariabelen
    $_SESSION['access'] = true;
    $_SESSION['username'] = $_POST['username'];
    // naar medewerkers_tonen.php
    header('location: medewerkers_tonen.php');
    exit;
}
```

LET OP: Denk er aan dat u `session_start()` aangeroepen moet hebben voordat u sessie variabelen kunt zetten en dat er voor deze aanroep **geen output** verzonden mag zijn! U kunt daarom het beste helemaal bovenin uw `common.inc.php` (of `config.inc.php`) de volgende coderegels opnemen:

```
ob_start();
session_start();
```

Als de login fout is dan roept u `loginform()` opnieuw aan... met eventueel een foutmelding eronder...

```
...
if ($_POST['password'] == 'a' && $_POST['username'] == 'b') {
    ...
} else {
    loginform();
    echo 'Login mislukt. Probeer het nog eens!';
}
...
```

Wanneer de controle niet slaagt, toont u opnieuw het inlogformulier met een passende foutmelding (login mislukt, probeer het nog een keer). Het is beter om geen informatie te geven wat niet correct was: de username of het password. Wanneer het om het password gaat weten kwaadwillenden dan al dat de username wel geldig is...

De bijbehorende template `login_form_tpl.php` zou er zo uit kunnen zien:

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

<form method="post" action="login.php">
    <!-- hidden field voor de controle op verzonden -->
    <input type="hidden" name="action" value="postback">

    <label for="username">Username</label>
    <input type="text" name="username" id="username">

    <label for="password">Password</label>
    <input type="password" name="password" id="password">

    <button>Login</button>
</form>

<?php if (isset($error)): ?>
    <p class="error"><?php echo $error; ?></p>
<?php endif; ?>

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

Al een gebruiker correct is ingelogd dan komt deze nu op de pagina `medewerkers_tonen.php`. Maar op dit moment is het nog steeds zo dat iedere gebruiker die rechtstreeks naar deze pagina navigeert de pagina nog kan gebruiken!

U wilt dan ook bovenaan `medewerkers_tonen.php` controleren of `$_SESSION['access']` bestaat en de waarde `true` heeft `true` staat. Als dat niet het geval is verwijst u met een redirect terug naar `login.php`:

```
if (!isset($_SESSION['access']) || $_SESSION['access'] !== true) {
    header('location: login.php');
    exit;
}
```

Het is makkelijker om zoals hierboven de ontkenning te zoeken dan de bevestiging. In het laatste geval krijgt u namelijk een onhandige `else` constructie waarin u naar de login pagina moet verwijzen...

9.12.3.1 `is_logged_in()`

Nog handiger is het om de controle in een functie `is_logged_in()` te gieten die u in `common.inc.php` plaatst:

```
// *****
function is_logged_in()
{
    return (isset($_SESSION['access']) && $_SESSION['access'] === true);
}
// *****
```

Als u dat combineert met de eerder aangemaakte functie `redirect()` dan kunt u de toegang eenvoudig zo controleren:

```
if (!is_logged_in()) {
    redirect('login.php');
}
```

Denk eraan dat u de controle op **elke pagina** die u wilt beveiligen moet zetten! Dus ook op de insert, update en delete pagina's! Dit gaat nog makkelijker wanneer u de controle in een functie opneemt:

9.12.3.2 `check_access()`

U kunt de controle opnemen in een functie `check_access()` die u in `common.inc.php` definieert:

```
// *****
function check_access()
{
    if (!is_logged_in()) {
        redirect('login.php');
    }
}
// *****
```

Zo hoeft u bovenaan de beveiligde pagina's enkel deze functie aan te roepen:

```
<?php
// *****
// MAIN
// *****
include "../includes/common.inc.php";

check_access();

$conn = db_connect();
...
```

Denk eraan dat u de functie `check_access()` pas kan aanroepen **nadat** `common.inc.php` is ingesloten.

Nu kunt u eventuele aanpassingen eenvoudig op een centrale plek, namelijk in de functiedefinitie, regelen. Bijvoorbeeld om de functie uit te breiden met userlevels zoals we straks zullen doen...

9.12.4 Logout

Nadat een gebruiker succesvol is ingelogd zal deze ook moeten kunnen uitloggen. Als eerste zullen we de pagina `logout.php` op een logische manier toegankelijk moeten maken. U kunt dit doen door in de `header.tpl.php` de login link conditioneel te veranderen in een link naar de logout pagina:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title><?php echo $title; ?></title>
  <link rel="stylesheet" href="styles/style.css">
</head>

<body>

<h1><?php echo COMPANY_NAME; ?></h1>

<nav>
  <ul>
    <li><a href="home.php">Home</a></li>
    ...
    <li><a href="contact.php">Contact</a></li>
    <?php if (is_logged_in()): ?>
      <li><a href="logout.php">Logout</a></li>
    <?php else: ?>
      <li><a href="login.php">Login</a></li>
    <?php endif; ?>
  </ul>
</nav>

<h2><?php echo $title; ?></h2>
```

Als een gebruiker niet is ingelogd krijgt hij/zij een login link te zien, maar als er succesvol is ingelogd dan zal deze veranderen in een logout link.

De pagina `logout.php` is verder heel eenvoudig. Het enige wat u wilt doen is `$_SESSION['access']` op `false` zetten, of beter helemaal verwijderen. U moet daartoe natuurlijk wel eerst de sessie starten. Dat kan door `common.inc.php` in te sluiten maar omdat u daar verder niets uit nodig heeft is dat niet eens nodig. Daarna kunt u gebruikers doorsturen naar de login pagina. Dit maakt het meteen duidelijk dat ze zijn uitgelogd:

```
<?php
start_session();
unset($_SESSION['access']);
header('location: login.php');
?>
```

Merk op dat u, omdat `common.inc.php` **niet** is ingesloten, niet de functie `redirect()` kunt gebruiken!

Als u in het publieke deel verder geen sessie variabelen gebruikt dan kunt u bij het uitloggen zelfs de hele sessie vernietigen:

```
<?php
start_session();
destroy_session();
header('location: login.php');
?>
```

9.12.5 Rest Navigatie aanpassen

U moet nu ook de rest van de navigatie nog aanpassen. De hyperlink naar de medewerkers mag alleen getoond worden als er is ingelogd. Bovendien moet dan de login link vervangen worden door een logout...

Oefening

In deze oefening zorgt u er voor dat de hyperlink in de navigatie voor het beheer van de medewerkers alleen toegankelijk is na inloggen. U maakt hierbij gebruik van sessievariabelen. Maak daarbij gebruik van de in dit hoofdstuk opgedane kennis.

9.12.6 Login tegen de database

Maar we willen natuurlijk niet dat er maar één user met een vast password kan inloggen. We willen de credentials uit de database kunnen halen!

We gaan daarbij uit van de volgende tabel `users` (maar u kunt natuurlijk ook de tabel `mw` uitbreiden). Hierin staat minimaal deze velden: `username`, `password`, `userlevel`:

```
CREATE TABLE users (
  id tinyint(4) NOT NULL auto_increment,
  username varchar(8) NOT NULL default '',
  password varchar(8) NOT NULL default '',
  userlevel tinyint(1) NOT NULL default '1',
  PRIMARY KEY(id)
) TYPE=MyISAM;

INSERT INTO users VALUES (1, 'guest', 'guest', 1);
INSERT INTO users VALUES (2, 'user', 'user', 2);
INSERT INTO users VALUES (3, 'admin', 'admin', 3);
```

Voor het gemak hebben we slechts 3 users met een overeenkomstig password aangemaakt, maar wel elk met een ander userlevel (role).

Let op: Sla **nooit** zoals in dit voorbeeld is gedaan **passwords onversleuteld** op in de database! We zullen dat straks beter regelen, maar we focussen ons eerst op de logica...

Na succesvol inloggen hangt het van het userlevel af welke acties een gebruiker kan uitvoeren op `medewerkers_tonen.php`:

userlevel >= 1	select
userlevel >= 2	idem + insert, update
userlevel >= 3	idem + delete + afdelingen beheer

We gaan dit regelen door de links voor deze acties alleen te tonen als een gebruiker de juiste rechten heeft, en we zullen er natuurlijk ook voor zorgen dat al te ondernemende gebruikers geen acties kunnen uitvoeren die hun userlevel te boven gaan!

De eerste stap bestaat er uit om in `login.php` de `check_login()` functie te vervangen door `check_login_db()`. In deze functie hebben we database verbinding nodig om te controleren of er voor de opgegeven username en password een record bestaat in de tabel `users`.

Wat moet er daarvoor in de fieldlist van uw SELECT statement komen te staan? Alleen het id en het userlevel zijn eigenlijk al voldoende. Want met id in de sessie kunt u later met vervolg queries alles wat u nodig heeft voor de gebruiker ophalen (al zal dat met de beperkte users tabel die we nu hebben niet nodig zijn...). En op basis van het userlevel zullen we straks in het beveiligde deel beslissen wat we gaan laten zien. In de WHERE clause checkt u op username en password, met LIMIT 1 bent u er van verzekerd dat er hooguit 1 record matcht:

```
$query = "SELECT id, userlevel
          FROM users
          WHERE username = :username
          AND password = :password
          LIMIT 1";
```

De parameters `:username` en `:password` worden gevuld met de waarden uit het login form. Uw `check_login_db()` functie zou er zo uit kunnen zien:

```
//*****
function check_login_db($username, $password)
{
    $user = false;

    $db = db_connect();

    $query = "SELECT id, userlevel
              FROM users
              WHERE username = :username
              AND password = :password LIMIT 1";

    $stmt = $db->prepare($query);
    $stmt->bindValue(':username', $username, PDO::PARAM_STR);
    $stmt->bindValue(':password', $password, PDO::PARAM_STR);
    $stmt->execute();
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    return $user;
}
//*****
```

Hierbij komen `$username` en `$password` gefilterd uit het login form.

NB: Het is overigens een **leuk experiment** om bij deze query even niet met parameters te werken en gewoon in de query te concateneren en niet te escapen om zo aan den lijve te ervaren hoe SQL Injection Attacks werken. Probeer nu maar eens deze waarde voor het password: `' OR 1=1 --` Juist, oeps... Gauw terug naar PDO!

U heeft een match als het aantal records in de resultset groter is dan 0 (er even van uitgaand dat er geen zelfde usr/pwd combinaties in de database kunnen zitten!). Als u gebruik maakt van de `mysqli_fetch_assoc()` functie en de geretourneerde waarde opslaat in een variabele `$user` dan bevat deze als er geen match wordt gevonden de waarde `false`. Wordt er wel een match gevonden dan heeft u een array met een valide gebruiker. Als dit het geval is dan zet u wederom de sessievariabelen voor access en username, maar nu ook voor id en userlevel, waarna u wederom redirect naar het medewerkeroverzicht:

```
...
$user = check_login_db($_POST['username'], $_POST['password']);
if ($user) {
    $_SESSION['access'] = true;
    $_SESSION['username'] = $_POST['username'];
    $_SESSION['userlevel'] = $user['userlevel'];
    header('location: medewerkers.php');
} else {
    $data['error'] = 'Helaas Pindakaas!';
    echo get_template('login', $data);
}
...
```

U kunt nu de gebruiker op de pagina `medewerkers_tonen` begroeten met zijn username en zijn userlevel laten zien:

```
Hoi <?php echo $_SESSION['username']; ?>
met userlevel <?php echo $_SESSION['userlevel']; ?>
```

Tip: Maak username en userlevel zichtbaar onder het menu:

```
<?php if (is_logged_in()) : ?>
<p>
    username: <strong><?php echo $_SESSION['username']; ?></strong><br>
    userlevel: <strong><?php echo $_SESSION['userlevel']; ?></strong>
</p>
<?php endif; ?>
```

Oefening *Haal de inloggegevens uit een (zelf aan te maken) databasetabel users met daarin de velden id, username en password.*

9.12.7 Userlevels

Zorg ervoor dat in de template de links voor insert en update alleen getoond worden bij userlevel 2 en hoger en de delete link alleen bij userlevel 3, bijvoorbeeld op deze manier:

```
<?php if ($_SESSION['userlevel'] > 1): ?>
    <p><a href="medewerker_toevoegen.php">Nieuwe medewerker toevoegen</a></p>
<?php endif; ?>
...
<?php if ($_SESSION['userlevel'] > 1): ?>
    <td><a href="medewerker_aanpassen.php">edit</a></td>
<?php endif; ?>
...
<?php if ($_SESSION['userlevel'] > 2): ?>
    <td><a href="medewerker_verwijderen.php">delete</a></td>
<?php endif; ?>
```

Houd er rekening mee dat als u besluit een hele cel (een <td>) niet meer te tonen dat dan ook de bijbehorende header (<th>) ook niet getoond mag worden.

Controleer ook op de pagina's `medewerker_insert.php`, `medewerker_update.php` en `medewerker_delete.php` of de gebruiker wel voldoende rechten heeft en zo niet breek dan het script af met een melding over onvoldoende rechten. Een hyperlink naar een script niet tonen is één ding, maar als een gebruiker met te weinig rechten alsnog bij het script kan komen heeft u daar niet zoveel aan. Wanneer een user met userlevel 2 een update link ziet met de opbouw `medewerkers.php?id=3&action=update` dan is de stap naar het in de adresbalk invoeren van `medewerkers.php?id=3&action=delete` snel gemaakt!

Dit moeten we voorkomen en de functie `check_access()` leent zich hier uitstekend voor. U kunt er een extra argument `$userlevel` aan meegeven. Door dit default op 1 te zetten kunt u de functie gewoon zonder argument blijven gebruiken en dan controleert hij in feite alleen op correct ingelogd zijn (het userlevel is minimaal 1), maar u kunt nu bij een script dat minimaal een userlevel 2 verwacht dat op deze manier aangeven: `check_access(2)`.

```
// *****
function check_access($userlevel=1)
{
    if (!is_logged_in()) {
        redirect('login.php');
    }
    if ($_SESSION['userlevel'] < $userlevel) {
        $data['title'] = 'Onvoldoende rechten';
        $data['errors']['no_rights'] = ERROR_NO_PERMISSION;
        echo get_template('no-rights', $data);
    }
}
```

```

        exit();
    }
}
//*****

```

--

Zorg ervoor dat de links voor insert en update alleen getoond worden bij userlevel 2 en hoger en de delete link alleen bij userlevel 3. Houd er rekening mee dat als u besluit een hele TD niet meer te tonen dat dan ook de bijbehorende TH niet getoond mag worden.

Controleer ook op de pagina's `medewerker_insert.php`, `medewerker_update.php` en `medewerker_delete.php` of de gebruiker wel voldoende rechten heeft en zo niet breek dan het script af met een melding over onvoldoende rechten. Een hyperlink naar een script niet tonen is een ding, maar als een gebruiker met te weinig rechten alsnog bij het script kan komen heb je daar niet zoveel aan.

De functie `check_access()` leent zich hier uitstekend voor. U kunt er een extra argument `$userlevel` aan meegeven. Door dit default op 1 te zetten kunt u de functie gewoon zonder argument blijven gebruiken en dan controleert hij in feite alleen op correct ingelogd zijn (het userlevel is minimaal 1), maar u kunt nu bij een script dat minimaal een userlevel 2 verwacht dat op deze manier aangeven: `check_access(2)`

```

function check_access($userlevel=1)
{
...
}

```

Oefening	<i>Breid de tabel <code>users</code> uit met een veld <code>userlevel</code>, zodat u per user onderscheid kunt maken in de mogelijkheden die hij in het beschermde gebied heeft. Sla de waarde op in een sessievariabele en toon afhankelijk van het niveau waarop ingelogd is, een aantal hyperlinks (of tekst o.i.d.) juist wel of niet.</i>

9.12.8 Password hashing

Het is een bijzonder slecht idee om de passwords onversleuteld in de database te hebben staan. Als een onbevoegde toegang zou krijgen tot de database dan heeft deze gelijk de beschikking over alle username/password combinaties van alle gebruikers. En daarmee mogelijk ook toegang tot hun accounts bij andere diensten, wanneer zij steeds hetzelfde password gebruiken!

U moet gebruik maken van **hash algoritmes**. Deze functies zetten een *message*, een string van een willekeurige lengte, om naar een *digest*: een samenvatting in de vorm van een *hash* van altijd dezelfde lengte, die u in de database zet. Het algoritme werkt maar **een kant op**, het hele idee is dat de hash niet is terug te leiden tot de oorspronkelijke waarde, er hoeft alleen maar gecontroleerd of de originele strings matchen. U zult het password van de gebruiker (die het onversleuteld invoert) bij het inloggen dus moeten omzetten naar de versleutelde waarde en deze vergelijken met de in de database opgeslagen, versleutelde versie.

De werkwijze die u volgt komt globaal hierop neer:

- De gebruiker maakt een account aan (of er wordt er een voor hem aangemaakt)
- Voor het gekozen password wordt een hash aangemaakt en deze wordt in de database opgeslagen. Het oorspronkelijke wachtwoord zelf wordt dus nergens opgeslagen!
- Bij het inloggen wordt de aan de Server-side gemaakte hash van het ingevoerde password gecontroleerd tegen de in de database opgeslagen hash.

- Alleen als deze hashes overeenkomen wordt de gebruiker toegelaten.

Merk op dat wanneer gebruikers bij registreren zelf een password mogen kiezen, dat u als database beheerder het originele password niet kunt teruggeven wanneer de gebruiker het vergeten is. In dergelijke gevallen wordt tijdelijk een nieuw password aangemaakt waarmee de gebruiker kan inloggen. Na succesvol inloggen kan hij dan zelf een ander password aanmaken.

U kunt het password op zich eenvoudig versleutelen met behulp van de `md5()` of `sha1()` functies in PHP en MySQL. Deze manier werd in het verleden veel gebruikt. Om veiligheidsredenen zou u ze echter niet meer moet gebruiken!

We laten ze omdat ze zoveel gebruikt werden toch eerst zien en geven daarna de veiliger alternatieven.

9.12.8.1 MD5 (Let op: Niet gebruiken!)

MD5 staat voor Message Digest Algorithm 5. Het is een in 1991 ontworpen en sindsdien veelgebruikte cryptografische hashfunctie. Het werd veel gebruikt in veiligheidstoepassingen en nog steeds veel om de integriteit van bestanden te controleren. Als u er een willekeurige string mee versleutelt dan levert dit altijd een string van 32 karakters.

```
$hash = md5($password); // 32 karakters
```

Meer over MD5 leest u op <http://en.wikipedia.org/wiki/MD5>.

9.12.8.2 SHA1 (Let op: Niet gebruiken!)

SHA1 is de afkorting van US Secure Hash Algorithm 1. Het werd in 1995 ontwikkeld door de NSA en zet strings om naar een 40 karakters lange string.

```
$hash = sha1($password); // 40 karakters
```

Meer over SHA1 leest u op <http://en.wikipedia.org/wiki/SHA-1>.

9.12.8.3 SHA2 (Let op: Niet gebruiken!)

De opvolgers worden met SHA2 aangeduid en bestaan uit algoritmes als SHA-224, SHA-256, SHA-384 en SHA-512. PHP heeft daar geen native functies voor, u moet de `hash()` functie gebruiken om ze toe te kunnen passen. Als eerste argument geeft u het gewenste algoritme als string en als tweede argument wat moet worden versleuteld:

```
$hash = hash('sha224', $password); // 56 karakters
$hash = hash('sha256', $password); // 64 karakters
$hash = hash('sha384', $password); // 96 karakters
$hash = hash('sha512', $password); // 128 karakters
```

Meer over SHA2 leest u op <http://en.wikipedia.org/wiki/SHA-2>.

Meer over `hash()` vindt u op <http://php.net/manual/en/function.hash-equals.php>

Tip: U kunt snel zien hoe strings versleuteld worden op <http://www.hashgenerator.de/>

9.12.8.4 Werkwijze (Let op: niet doen!)

De werkwijze zou bij MD5 en SHA1 dan als volgt gaan. Sla de passwords met behulp van de gekozen functie op in de database. Dit kan vanuit een PHP script of rechtstreeks via SQL:

Column	Type	Function	Null	Value
id	tinyint(4)			3
username	varchar(8)			admin
password	varchar(40)	SHA1		admin
userlevel	tinyint(1)			3

Go

Die wordt dan eenmalig over geselecteerde veld in de query uitgevoerd... Het password 'admin' wordt dan zo in het UPDATE statement gebruikt:

```
UPDATE users SET password = SHA1('admin') WHERE id = 3;
```

en staat als deze string van 40 karakters in de database:
d033e22ae348aeb5660fc2140aec35850c4da997

Denk eraan dat u bij het SELECT statement in de logincheck dan ook de `sha1()` functie loslaat op het password dat de gebruiker natuurlijk gewoon als 'admin' invoert bij het inloggen:

U kunt dat met SQL doen maar net zo makkelijk in PHP:

```
...
$stmt->bindValue(':password', sha1($password), PDO::PARAM_STR);
```

Met de SHA2 functies kunt u niet rechtevreeks via SQL werken. U moet de wachtwoorden vanuit een PHP script opslaan:

```
...
$query = "INSERT INTO users (username, password)
        VALUES (:username, :password)";
$stmt = $db->prepare($query);
$stmt->bindValue(':username', $username);
$stmt->bindValue(':password', hash('sha256', $password));
$stmt->execute();
...
```

En u zou ze dan als volgt uit kunnen lezen:

```
$query = "SELECT id, userlevel, password
        FROM users_hash
        WHERE username = :username
        LIMIT 1;";

$stmt = $db->prepare($query);
$stmt->bindValue(':username', $username, PDO::PARAM_STR);
$stmt->execute();
$user = $stmt->fetch(PDO::FETCH_ASSOC);

// check opgegeven password tegen hash uit database
if (hash('sha256', $password) == $user['password']) {
    // haal password uit array
    unset($user['password']);
    // hoera!
}
```

9.12.8.5 Waarom niet gebruiken?

De MD5 en SHA* functies mag u echter **niet** voor password hashing gebruiken! De toegepaste algoritmes zijn ontworpen om zo snel en efficiënt mogelijk te werken. Dat is om de integriteit van data te controleren fantastisch maar voor het opslaan van passwords juist funest. Het is met moderne hardware dan te gemakkelijk om met **brute force** de originele invoer te achterhalen.

Om een idee te geven: een gewone computer kan per seconde de MD5 hash voor zo'n 330 MB aan data berekenen. Als u gebruikers passwords laat opslaan in een lowercase, alfanumeriek formaat van 6 karakters lang, dan kunnen in 40 seconden alle passwords van die grootte worden geprobeerd! En met een beetje investeren in hardware kunt u zo maar naar 700.000.000 passwords per seconde gaan!

Hierbij wordt overigens handig gebruik gemaakt van het feit dat dezelfde invoer steeds dezelfde uitvoer geeft door zogenaamde **rainbow tables** in te zetten. Dit zijn lijsten met bekende (wacht)woorden en hun hash. Kijk maar eens op sites als:

- **Online MD5 decrypter**
<http://md5online.org>
- **HashKiller**
<http://www.hashkiller.co.uk/sha1-decrypter.aspx>

Deze sites beschikken over databases met miljarden hashes en hun originele invoer. Simpelweg een hash invullen geeft u vrijwel instant de oplossing...

Het mag duidelijk zijn dat om deze reden security professionals adviseren om deze algoritmes te vermijden!

Zie ook: <http://stackoverflow.com/questions/2235079/is-it-possible-to-reverse-a-sha1>

1.1.1 Salt

U kunt de MD5 en SHA* functies veiliger maken door middel van een *salt*. Een salt is een extra string die u aan de originele invoer toevoegt om het decrypten moeilijker te maken. Dit kan op verschillende manieren, statisch of dynamisch en publiek of geheim.

Het mooist is om ze **zo random mogelijk** te maken, dus compleet verschillend voor elk wachtwoord dat u opslaat. Bovendien moeten ze **lang** zijn om veel verschillende salts te kunnen garanderen.

Voor het random maken wordt soms gekozen voor `mt_rand()` en/of de systeemtijd maar dat is eigenlijk al te voorspelbaar. U zou eigenlijk met een Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) moeten werken, bijvoorbeeld met `mcrypt_create_iv()`.

Om de controle te kunnen doen moet het steeds veranderende salt dus wel voor elk record worden opgeslagen. Dat kan in een tweede veld in de database maar ook als onderdeel van de gehashede string!

Bij de `crypt()` en `password_hash()` functies die we zo zullen bespreken wordt het salt echt random aangemaakt en wordt het in de hash zelf opgeslagen, wat wel zo makkelijk is.

De boodschap is eigenlijk vooral om niet zelf oplossingen te verzinnen om salts te gaan genereren!

U leest meer over salts op [http://en.wikipedia.org/wiki/Salt_\(cryptography\)](http://en.wikipedia.org/wiki/Salt_(cryptography))

9.12.9 Slow Hash algoritmes

Wat u behalve de salt echter echt nodig heeft, is een zogenaamd **slow hash algorithm** met een **cost factor**, zo dat u controle heeft over de tijd die nodig is om een hash aan te maken, zoals:

- **PBKDF2**
<https://en.wikipedia.org/wiki/PBKDF2>
- **BCrypt**
<https://en.wikipedia.org/wiki/Bcrypt>
- **SCrypt**
<https://en.wikipedia.org/wiki/Scrypt>

Deze algoritmes zijn speciaal bedoeld om brute force cracking van hashes tijdrovend te maken.

9.12.10 Crypt()

Met de `crypt()` functie kunt u de eerder genoemde Slow Hash algoritmes gebruiken. Deze Unix functie bestaat al sinds 1976 maar het is nog best lastig met de keuze uit algoritmes, salts en preventie tegen **timing attacks**. U moet het dan ook in combinatie met de `hash_equals()` functie gebruiken <http://php.net/manual/en/function.hash-equals.php> (vanaf PHP 5.6)

Vanaf PHP 5.5 is er echter een handige password API, in feite een wrapper om `crypt()`. En die zullen wij dan ook gaan gebruiken!

9.12.11 Password API

Al het gedoe met `crypt()` kunt u voorkomen met de handige, in PHP 5.5 geïntroduceerde **Password API**. Dit is de aanbevolen manier! De Password API bestaat uit verschillende functies. De belangrijkste zijn:

- `password_hash()`
- `password_verify()`

Naast deze functies bestaan ook `password_get_info()` waarmee u informatie over de hash kunt krijgen (het gebruikte algoritme en de cost) en `passwords_needs_rehash()` waarmee u kunt controleren of de opgegeven hash voldoet aan de opgegeven instellingen. In de loop van de tijd kan het gebruikte algoritme of de cost namelijk wijzigen en dan kan het password opnieuw gehashed worden. Wij gaan daar tijdens deze cursus verder niet op in...

9.12.12 password_hash

`password_hash()` is een wrapper om de `crypt()` functie en het is compatible met daarmee reeds aangemaakte hashes. U kunt er eenvoudig een hash mee aanmaken op de volgende manier:

```
$hash = password_hash($password, PASSWORD_BCRYPT);
```

Met de constante geeft u aan welke encryptiemethode u wilt gebruiken. `PASSWORD_BCRYPT` gebruikt het BLOWFISH algoritme en dat is op dit moment het sterkste algoritme dat u kunt gebruiken. Het produceert een standaard met `crypt()` compatibele hash van 60 karakters. De string `'admin'` wordt nu bijvoorbeeld:

```
'$2y$10$VMNA0koet3efaNgIe3YLVevJEtROtg2apLxyw2QDyZVzNqdqZtd8i'.
```

Als u de functie nog een keer aan zou roepen krijgt u, in tegenstelling tot MD5 en SHA* een andere uitvoer! Deze begint overigens in het geval van `PASSWORD_BCRYPT` wel steeds met de identifier `'$2y$'`.

Let op: Als het om wat voor reden dan ook niet zou lukken om de hash aan te maken dan retourneert de functie `false`. Het is dus belangrijk om goed te controleren of het maken van de hash gelukt is voordat u het in de database opslaat.

De functie `password_hash()` is *future proof* want een ander algoritme dat wordt ondersteund is `PASSWORD_DEFAULT`. Dit gebruikt de sterkste encryptie waarover PHP op het moment beschikt. Op dit moment is dat hetzelfde als `PASSWORD_BCRYPT`. Maar in toekomstige PHP versies zou dit kunnen veranderen... Als u deze versie wilt gebruiken wordt het dan ook sterk aangeraden om voor de opslag een `VARCHAR(255)` veld te gebruiken zodat nieuwere, sterkere, hashes niet worden afgekapt.

9.12.12.1 Cost

U kunt als derde argument options meegeven in de vorm van een array. U kunt zelf een salt opgeven en de cost instellen. Het zelf aanmaken van een salt wordt echter afgeraden... De default variant is helemaal OK! Met de cost geeft u aan wat de CPU cost voor het gekozen algoritme is. Deze kan lopen van 4 tot 31. Hoe hoger de waarde hoe meer tijd het kost om de hash aan te maken en te verifiëren. Het streven is om zo hoog mogelijk te gaan maar de tijd voor een interactieve login onder de 0.1 seconde te houden. De standaardwaarde is 10 en in de meeste gevallen voldoet dat prima. De tijd blijft daarbij op een modern systeem zelfs onder de 0.05 seconde.

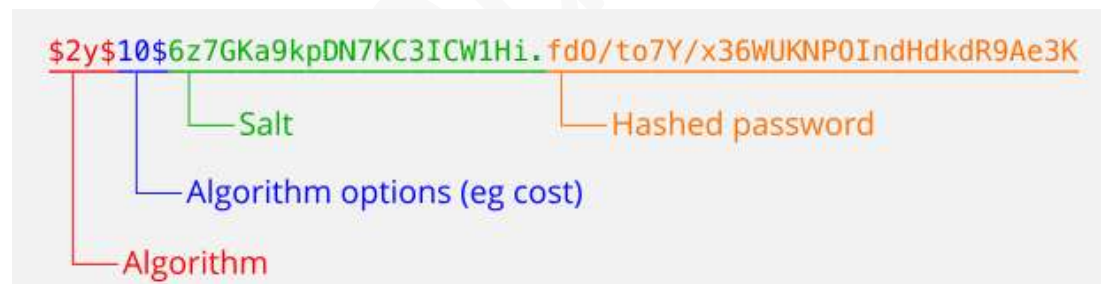
Tip: In de PHP manual vindt u een handig code fragment om de beste cost voor uw hardware te bepalen.

U kunt een andere cost zo opgeven:

```
$hash = password_hash($password, PASSWORD_BCRYPT, array("cost" => 11));
```

9.12.13 password_verify()

De met `password_hash()` aangemaakte hash bevat het gebruikte algoritme, de cost en het salt:



De hash bevat dus alle informatie die nodig is om hem te kunnen verifiëren. U hoeft het salt of het algoritme bij het aanmaken niet apart op te slaan in de database of configuratie file.

Het verifiëren doet u met de functie `password_verify()`:

```
if (password_verify($password, $hash)) {
    // Valid
} else {
    // Invalid
}
```

Voor de login check moet u nu niet op een matchend password in de WHERE clause zoeken maar het password ophalen in de fieldlist en het daarna verifiëren. Bijvoorbeeld zo:

```

//*****
function check_login_db_hash($username, $password)
{
    $return = false;

    $db = db_connect();

    $query = "SELECT id, userlevel, password
              FROM users
              WHERE username = :username
              LIMIT 1;";

    $stmt = $db->prepare($query);
    $stmt->bindValue(':username', $username, PDO::PARAM_STR);
    $stmt->execute();
    $user = $stmt->fetch(PDO::FETCH_ASSOC);

    // check opgegeven password tegen hash uit database
    if ($user && password_verify($password, $user['password'])) {
        // haal password uit array
        unset($user['password']);
        $return = $user;
    }
    // return user array of false
    return $return;
}
//*****

```

Merk op dat we de het opgegeven password alleen valideren als er überhaupt een user gevonden wordt met de betreffende username!

9.12.14 Alternatieven

Heeft uw host nog geen PHP 5.5? Dan kunt u de bibliotheek **password_compat** gebruiken. Deze is geschreven door de ontwikkelaar van de password_* API en bootst deze na voor PHP versies vanaf 5.3.7. U kunt de bibliotheek gratis downloaden via https://github.com/ircmaxell/password_compat.

Voor nog oudere PHP versies (zelfs vanaf PHP 3!) is er Openwall's **PHPass** (Portable PHP password hashing framework) dat u kunt downloaden op <http://www.openwall.com/phpass/>. Deze bibliotheek is echter wel al bijna 10 jaar oud en kent inmiddels wel een paar veiligheids issues, maar het is waarschijnlijk hoe dan ook beter dan zelf wat prutsen...

Oefening	<i>maak gebruik van <code>password_hash()</code> en <code>password_verify()</code> in uw loginprocedure.</i>

9.13 Links

Meer informatie over het werken met sessies in PHP vindt u op:
<http://php.net/session>.

Meer informatie over password hashing in het algemeen:

- <http://codahale.com/how-to-safely-store-a-password/>
- <https://crackstation.net/hashing-security.htm>
- <http://www.securityfocus.com/blogs/262>
- http://en.wikipedia.org/wiki/Dictionary_attack
- http://en.wikipedia.org/wiki/Rainbow_table
- <http://stackoverflow.com/questions/3897434/password-security-sha1-sha256-or-sha512?rq=1>

En specifiek voor PHP:

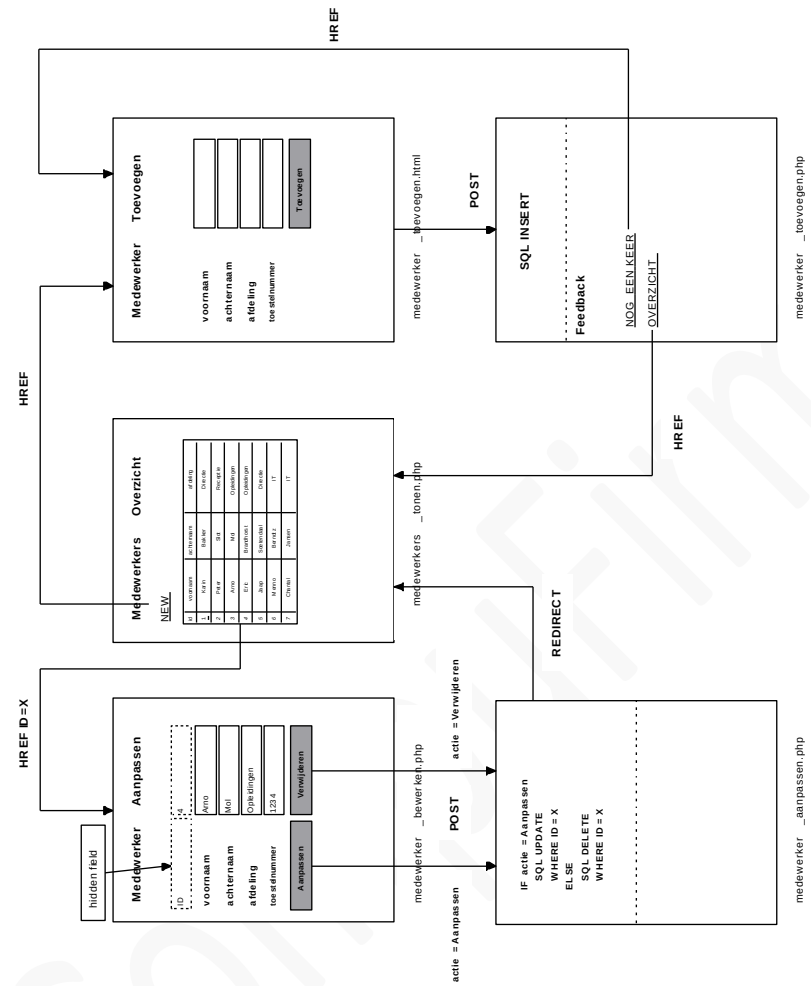
- <http://php.net/manual/en/faq.passwords.php>
- <http://www.sitepoint.com/password-hashing-in-php/>
- <http://www.sitepoint.com/hashing-passwords-php-5-5-password-hashing-api/>
- <http://www.openwall.com/articles/PHP-Users-Passwords>
- <http://stackoverflow.com/questions/14992367/using-php-5-5s-password-hash-and-verify-function-am-i-doing-it-right>
- <http://stackoverflow.com/questions/25372636/password-hash-equivalent-for-php-5-4>

ComputFirm.nl

CompuFirm.nl

Bijlagen

10.1 Schema mini-applicatie



10.2 Extra Oefeningen

Mogelijk hebt u na deze cursus behoefte om extra te oefenen met de stof die we tot nu toe behandeld hebben. Deze stof is namelijk het uitgangspunt voor de **PHP Vervolg cursus**, we gaan daarin verder waar we hier opgehouden zijn. Als u zich nog niet geheel vertrouwd voelt met de materie en nog wat verder wil oefenen dan zijn de onderstaande oefeningen wellicht interessant voor u.

Allereerst zou ik u er op willen wijzen dat er in dit cursusboek een aantal **verdiepingsoefeningen** zijn opgenomen die u wellicht nog niet gedaan heeft. Het is natuurlijk zeer aan te raden deze wel te doen!

Als dat allemaal gelukt is, dan zou u de volgende extra oefeningen kunnen maken. De oefeningen zijn grotendeels 'getrapt', waarbij u ze steeds iets mooier maakt (maar waarvoor u ook wel wat meer moet puzzelen...). Kijk maar hoever u komt!

En mocht u tot het einde gekomen zijn dan heeft u hopelijk de smaak te pakken gekregen en heeft u zelf nog meer uitdagingen/verbeteringen bedacht. Die kunnen we mooi aan het begin van de vervolgcursus doornemen!

10.2.1 Formulier Datum Tonen

Maak een formulier `datum_kiezen.php` waarin u 3 selectboxes zet: een voor dag, een voor maand en een voor jaar. Die voor dag loopt van 1-31, die voor maand van 1-12 en die voor jaar van 2015-2025:

Datum kiezen
Datum:

Uiteraard gaat u de options er **niet met de hand** inzetten maar doet u dat met behulp van een **drietal lussen!**

Verstuur de gegevens via GET naar de formhandler `datum_tonen.php` waarin u de gekozen datum laat zien, bijvoorbeeld:

Datum tonen
Datum: 31-12-2016

10.2.1.1 Verdieping

Zorg ervoor dat de **formhandler** `datum_tonen.php` **niet los opgevraagd** kan worden. Redirect in dat geval naar het formulier `datum_kiezen.php`.

10.2.1.2 Verdieping

Als dit gelukt is zorg er dan voor dat de selectboxes op het formulier worden **voorgeselecteerd op de huidige datum**.

Als het vandaag 24 juni 2020 is, dan zou u dus dit moeten zien:

Datum kiezen

Datum:

Ter herinnering, u maakt in HTML een option op de volgende manier geselecteerd:

```
<option selected>...</option>
```

Hint: gebruik variabelen als `$day = date('j')` om het dagnummer voor vandaag te bepalen. U gebruikt hier het format character `j` omdat het dagnummer in de option geen *leading zero* heeft (zie <http://php.net/date>). Controleer in de lus met een `if-else` statement wanneer de teller gelijk is aan de huidige dag en maak dan de option voorgeselecteerd.

10.2.1.3 Verdieping

Maak nu het begin en eindpunt van de lus voor het jaar **dynamisch** door altijd vijf jaar voor en vijf jaar na het huidige jaartal te nemen. In 2020 is de range dus 2015-2025 en het jaar daarna wordt dat dan automatisch van 2016-2021!

10.2.1.4 Verdieping

Gebruik vervolgens in `datum_tonen.php` de functie `checkdate` (zie <http://php.net/checkdate>) om te controleren of er een **valide datum** is samengesteld. Geef een foutmelding als dat niet het geval is (bijvoorbeeld bij een datum als 31 februari) en toon een hyperlink terug naar het formulier om een nieuwe datum te kunnen kiezen:

Datum tonen

Datum: 31-2-2016

Dit is geen geldige datum!

[Kies een nieuwe datum](#)

10.2.1.5 Verdieping

Maak een kopie van `datum_tonen.php` en noem deze `datum_tonen_function.php`. Maak hierin een functie `select()` aan die de benodigde HTML voor de selectboxes op een nette manier retourneert. De functie heeft daarvoor 4 parameters. Drie daarvan `name`, `start` en `end` zijn verplicht. De vierde, `select` is optioneel en bedoeld om een waarde voor te kunnen selecteren. Vervolgens roept u in uw pagina de functie 3x aan om achtereenvolgens de selectboxes voor dag, maand en jaar aan te maken.

Hint: om de selectbox voor dag aan te maken die voorselecteert op de huidige dag zou u uiteindelijk dus alleen maar dit hoeven noteren in het HTML deel van uw pagina:

```
<?= select('dag', 1, 31, date('j')) ?>
```

10.2.2 Formulier Tabel Maken

Maak een formulier `tabel_maken.html` waarin u twee invoervelden zet, waarin maximaal 1 cijfer gezet kan worden, bijvoorbeeld zo iets:

Tabel maken

rij(en) x kolom(men)

Verstuur deze via GET naar `tabel_tonen.php` waarin u met behulp van **twee geneste lussen** een HTML tabel aanmaakt met het ingevoerde aantal rijen en kolommen, waarin u de letter X zet:

Tabel tonen (3 x 5)

X	X	X	X	X
X	X	X	X	X
X	X	X	X	X

10.2.2.1 Verdieping

Als dit gelukt is, laat dan in de cel zien in welke rij en in welke kolom deze zit:

Tabel tonen (3 x 5)

rij 1 - kolom 1	rij 1 - kolom 2	rij 1 - kolom 3	rij 1 - kolom 4	rij 1 - kolom 5
rij 2 - kolom 1	rij 2 - kolom 2	rij 2 - kolom 3	rij 2 - kolom 4	rij 2 - kolom 5
rij 3 - kolom 1	rij 3 - kolom 2	rij 3 - kolom 3	rij 3 - kolom 4	rij 3 - kolom 5

10.2.2.2 Verdieping

Probeer ook eens in de buitenste lus (die de rijen genereert) een `class="alt"` toe te kennen. U doet dat met behulp van de modulo operator `%`. Hiermee kunt u snel controleren of een getal even of oneven is:

```
if ($i % 2 == 0) {
    // $i is een even getal
} else {
    // $i is een oneven getal
}
```

Met behulp van een embedded stylesheet kunt u nu een andere achtergrondkleur aan de oneven tabelrijen geven:

```
.alt {
    background-color: #CCC;
}
```

Dit zou er dan zo uit kunnen zien:

Tabel tonen (3 x 5)

rij 1 - kolom 1	rij 1 - kolom 2	rij 1 - kolom 3	rij 1 - kolom 4	rij 1 - kolom 5
rij 2 - kolom 1	rij 2 - kolom 2	rij 2 - kolom 3	rij 2 - kolom 4	rij 2 - kolom 5
rij 3 - kolom 1	rij 3 - kolom 2	rij 3 - kolom 3	rij 3 - kolom 4	rij 3 - kolom 5

Let op: Dit kan natuurlijk veel makkelijker met de CSS `:nth-child()` pseudo class. Maar het blijft een goede oefening! Als u niet weet hoe het met CSS only werkt: zoek het op! Of beter: volg de **CSS cursus...**

10.2.2.3 Verdieping

Pas het formulier aan zodat er tot wel 100 rijen en 10 kolommen kunnen worden toegevoegd. Toon de data in de cellen vervolgens met **leading zero's** met behulp van `sprintf()`:

Tabel tonen (3 x 5)

rij 001 - kolom 01	rij 001 - kolom 02	rij 001 - kolom 03	rij 001 - kolom 04	rij 001 - kolom 05
rij 002 - kolom 01	rij 002 - kolom 02	rij 002 - kolom 03	rij 002 - kolom 04	rij 002 - kolom 05
rij 003 - kolom 01	rij 003 - kolom 02	rij 003 - kolom 03	rij 003 - kolom 04	rij 003 - kolom 05

10.2.2.4 Verdieping

Voeg beveiliging toe aan `tabel_tonen.php` zodat deze **niet los** kan worden **opgevraagd**. Redirect in dat geval naar `tabel_maken.html`.

Voeg daarna beveiliging toe zodat er niet minder dan 1 of meer dan 100 rijen en/of minder dan 1 of meer dan 10 kolommen getoond kunnen worden. Toon foutmeldingen als volgt (en bouw dan de tabel uiteraard niet op:

Tabel tonen (0 x 11)

- Aantal rijen kan niet lager zijn dan 1
- Aantal kolommen kan niet groter zijn dan 10

10.2.2.5 Verdieping

Zet afhankelijk van het gekozen aantal kolommen automatisch de juiste tableheaders boven de kolommen.

Toon in de laatste rij van de tabel (in een `<tfoot>`) de som van het aantal rijen en vermenigvuldig dat met het kolomnummer. Bij 3 rijen zou dat $1+2+3 = 6$ moeten zijn voor de eerste kolom (*1) maar 12 voor de 2^e kolom (*2), etc:

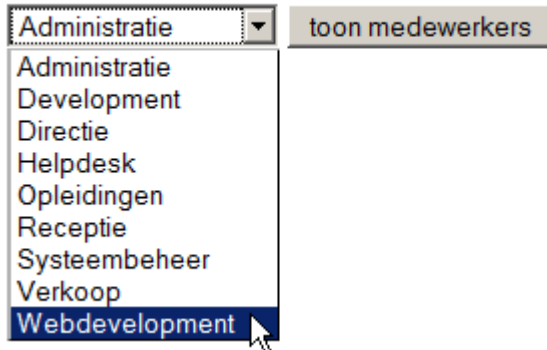
Tabel tonen (3 x 5)

kolom 01	kolom 02	kolom 03	kolom 04	kolom 05
kolom 01 - rij 001	kolom 02 - rij 001	kolom 03 - rij 001	kolom 04 - rij 001	kolom 05 - rij 001
kolom 01 - rij 002	kolom 02 - rij 002	kolom 03 - rij 002	kolom 04 - rij 002	kolom 05 - rij 002
kolom 01 - rij 003	kolom 02 - rij 003	kolom 03 - rij 003	kolom 04 - rij 003	kolom 05 - rij 003
Totaal:	6	Totaal: 12	Totaal: 18	Totaal: 24
				30

10.2.3 Database Extra 1

Maak een pagina `afdelingen_tonen.php`. Hierin toont u de afzonderlijke afdelingen zoals die in de `medewerkers` tabel staan in een selectbox:

Afdelingen



Om elke afdeling slechts 1x te tonen gebruikt u het keyword `DISTINCT` in uw SQL `SELECT` statement:

```
SELECT DISTINCT afdeling FROM medewerkers
```

U kunt nu het `value` attribuut achterwege laten: de afdeling die getoond wordt in de optie is ook de waarde die wordt verzonden. U verzendt de waarde via `GET` naar een pagina `medewerkers_per_afdeling.php` waar u laat zien welke medewerkers er op de gekozen afdeling werken:

Webdevelopment

id	voornaam	achternaam	afdeling	toestelnummer
9	Davidof	Kromme	Webdevelopment	2213
10	Kathy	Bukowicz	Webdevelopment	2213

[Maak een nieuwe keuze...](#)

10.2.4 Database Extra 2

Maak de alternatieve presentatie voor de `medewerkers_tonen.php` zoals beschreven in 6.19 (vanaf blz. 228) van uw cursusboekje:

Overzicht Medewerkers

 [Nieuwe medewerker](#)

Voornaam	Achternaam	Afdeling	Toestelnummer	Acties
Sheila	Lavanter	Verkoop	1201	 
Peter	Spanje	Verkoop	1201	 
Jaap	Soetendaal	Directie	1299	 
Arno	Mol	Opleidingen	1234	 
Kees	Prins	Opleidingen	1234	 
Denise	Smit	Verkoop	1201	 

U zult er dan een pagina bij moeten maken: `medewerker_verwijderen.php` die ervoor zorgt dat een medewerker wordt verwijderd als iemand op de delete afbeelding (het kruisje) klikt. U kunt nu de deleteknop weghalen uit `medewerker_bewerken.php` en haal dan ook gelijk de delete query weg uit `medewerker_aanpassen.php`.

Vergelijkbare icoontjes vindt u via Google. Zoek afbeeldingen in klein formaat voor insert, update en delete.

10.2.5 Database Extra 3

Breid de medewerkers tabel uit met de velden `email`, `gsm` en `foto`. Zet er zinnige waarden in. Voeg een extra actie show (of details) toe aan `medewerkers_tonen.php`. Het overzicht laat nu dus maar een deel van de gegevens zien:

Overzicht Medewerkers

 [Nieuwe medewerker](#)

Voornaam	Achternaam	Afdeling	Toestelnummer	Acties
Sheila	Lavanter	Verkoop	1000	  
Peter	Spanje	Verkoop	1000	  
Jaap	Soetendaal	Directie	999	  
Arno	Mol	Opleidingen	1245	  
Kees	Prins	Opleidingen	123	  
Denise	Smit	Verkoop	219	  

De nieuwe actie verwijst naar een pagina `medewerker.php` en verstuurt het id over de querystring en haalt alle gegevens voor de geselecteerde medewerker op. Toon de foto en maak het email adres klikbaar (zie 6.13 vanaf blz. 142):

Sheila Lavanter

	Verkoop
	Toestel: 1000
	Email: sheila@compufirm.nl
	GSM: 06-12345678

10.2.6 Database Extra 4

Voeg nog twee extra velden toe aan de medewerkers tabel. Een veld tussenvoegsel voor de naam en een geboortedatum en toon deze in detailview. Zorg ervoor dat wanneer er geen tussenvoegsel is voor een medewerker dat deze ook niet getoond wordt in de opgebouwde naam.

10.2.7 Database Extra 5

Voeg aan de beheerpagina's de eenvoudige Server-side validatie toe die in 4.8 beschreven wordt. Markeer in de formulieren duidelijk welke velden verplicht zijn en laat een foutmelding zien als deze velden leeg worden verstuurd. Controleer ook of het toestelnummer wel een getal is. U kunt dat voorlopig eenvoudig doen met de functie `is_numeric()`. Nog leuker wordt het wanneer u alleen waarden tussen de 1000 en 2000 goedkeurt...

In **HTML5** kunt u dit natuurlijk ook **snel client-side** regelen met wat eenvoudige markup maar het gaat er juist om hoe dit Server-side te implementeren!

10.2.8 Database Extra 6

Voeg aan de medewerkers tabel een extra kolom tonen toe en zet deze standaard op 1. In de beheerpagina's kunt u nu de mogelijkheid inbouwen om met een checkbox de medewerker wel of niet te tonen. U implementeert zo de in het cursusboekje al kort genoemde **soft delete** functionaliteit.

U moet dan voor INSERT of UPDATE kijken of de POST waarde voor tonen is gezet. Zo ja, gebruik hem dan (1) of zet hem anders op 0 (want dan is hij dus uitgevinkt).

Maak ook een pagina aan die alleen de actieve medewerkers toont (met een WHERE clause). Deze is bedoeld om aan de eindgebruikers te tonen. In de CRUD moet u dat natuurlijk niet zo doen anders heeft u geen mogelijkheid meer om de inactieve users weer te activeren!

10.2.9 Database Extra 7

Doe alle basishandelingen (SELECT, INSERT, UPDATE en DELETE) zoals je die voor de medewerkers hebt uitgevoerd nog eens over voor een zelf aan te maken tabel, bijvoorbeeld CDs, waarin je de velden id, artiest, titel, label, jaar en cover opneemt. Of verzin zelf andere gegevens (Boeken, Films o.i.d.). Hou het eenvoudig zodat u het in één tabel kunt zetten. In de **PHP Vervolg cursus** gaan we met meerdere tabellen aan de slag!

Succes!

10.3 PHP Vervolg cursus

De **PHP Vervolg cursus** is het logische vervolg op de basiscursus PHP. In deze vervolgcursus breidt u de reeds opgedane basiskennis verder uit.

We beginnen met het **modulariseren** van onze code. U leert hoe u met includes, zelfgeschreven functies en classes van anderen, code makkelijk kunt hergebruiken.

U leert hoe u de **database** toegang kunt **optimaliseren** en hoe u de uit de database afkomstige informatie prettiger aan de gebruiker kan presenteren. U leert eveneens hoe u data uit (tekst)bestanden kunt uitlezen en hoe u naar deze bestanden kunt schrijven.

U leert over het belang van data **validatie** en hoe deze te realiseren.

U leert de voor- en nadelen kennen van de verschillende manieren die PHP kent om **status** vast te houden. Naast de al bekende methoden als het gebruik van verborgen velden en het verzenden van

informatie over de querystring wordt uitvoerig aandacht besteed aan twee andere manieren: **cookies** en **sessies**.

Aan het einde van de training leert u ook nog hoe u vanuit PHP **email** kunt **versturen** en hoe u **bestanden** naar de webserver kunt **uploaden**.

U past alle technieken toe in de medewerkers applicatie waarmee u al in de basiscursus bent begonnen en die dus steeds verder wordt uitgebreid.

ComputFirm.nl