Victor Peters

Basiscursus PHP 6

en MySQL

ACADEMIC SERVICE

Inhoud

1 Aan de slag met PHP

PHP is voor een interactieve website wat de motor is voor een auto. Hoe mooi een auto ook is vormgegeven, hoe schitterend de lak en hoe comfortabel de stoelen ook zijn, je hebt er niets aan als er onder de motorkap geen goed lopende motor ligt.
Zou u in deze vergelijking HTML en CSS vergelijken met het uiterlijk van de auto – met HTML en CSS bouwt u immers de zichtbare kant van een website – zo kunt u PHP vergelijken met de motor die onmerkbaar in de achtergrond voor allerlei zaken zorgt, waar de argeloze gebruiker geen weet van heeft.

1.1 In dit hoofdstuk

- Kennismaken met begrippen als server en client
- Andere talen voor het web
- Het paradigma Model, View, Controller
- Objectgeoriënteerd programmeren
- Databases en MySQL
- WAMP – de webserver voor uw pc
- Editors voor PHP
- Denken als een computer

1.2 Webserver en browser

Websites worden altijd geserveerd door een webserver. De webserver stuurt alle benodigde gegevens voor een website over het internet naar uw computer, waarin de browser de ontvangen webpagina weergeeft – dat is de client.

De webpagina die u ziet is opgebouwd uit HTML en andere talen voor de browser en bevat geen enkele regel PHP-code. Uw browser zou ook niet weten wat hij met PHP-code aanmoet. De webserver is het domein van PHP. Met PHP-code bepaalt de programmeur welke handelingen de server moet verrichten als een bezoeker in de browser op een knop of hyperlink klikt. De PHP-code zorgt voor afhandeling van webformulieren, bewaart gegevens van de bezoekers in een database en stuurt weer nieuwe gegevens als HTML-code naar de browser.

PHP is dus niet een op zichzelf staande programmeertaal. De programmeur zorgt dat zijn (of haar) PHP-code HTML-code genereert, of CSS-code (voor stijlbladen) of zelfs JavaScript-code.

1.3 HTML, Ajax, XML, CSS en JavaScript

PHP is een taal met veel mogelijkheden, maar kan weinig uitrichten zonder de drie basistalen voor browsers: HTML, CSS en JavaScript. PHP doet eigenlijk niets anders dan verzoeken die van de gebruiker komen, verwerken en opslaan, en vervolgens daarop reageren door het terugzenden van stukken HTML en JavaScript. Het stijlblad (gemaakt met CSS) zorgt voor het juiste uiterlijk van de HTML-pagina.

Ook XML speelt hierin een rol. XML is een taal waarmee gegevens tussen server en browser gestructureerd uitgewisseld kunnen worden. De XML-structuur zorgt ervoor dat de ontvangen gegevens correct begrepen en verwerkt kunnen worden. Ajax zorgt in de browser meestal voor verzending en ontvangst van met XML gecodeerde gegevens.

In dit boek ontkomt u dus niet aan de nodige HTML-code. CSS, XML en JavaScript zult u minder tegenkomen, omdat die met de browserafhandeling te maken hebben, wat dus niet het domein van PHP is. Wel komt aan de orde hoe u met PHP JavaScript-code en CSS-stijlelementen aan HTML kunt toevoegen. Kennis van HTML is onontbeerlijk om zelfstandig met PHP te kunnen werken. In dit boek wordt uitgegaan van praktische kennis van HTML.

1.4 WordPress, Joomla en PHPBB

PHP mag gerust martkleider worden genoemd als ontwikkeltaal voor het web. Er zijn natuurlijk vele programmeertalen die zich lenen voor het ontwikkelen van interactieve websites, zoals Perl, Java, ASP.net, Ruby on Rails, of Django. PHP heeft zich in de loop der jaren echter op een aantal fronten bewezen als 'winnaar'. De taal is veruit het meest toegankelijk, is zeer flexibel en vergevingsgezind voor de programmeur, draait op vrijwel elke webserver – ook bij heel goedkope hostingpakketten – en wordt toegepast in veel grote platforms als WordPress, Joomla en PHPBB. Wie goed kan programmeren in PHP, is in deze tijden verzekerd van een goede bron van inkomsten. Aan het eind van deze Basiscursus bent u al een heel eind op weg om dat te bereiken.

1.5 PHP is een rommelpot

Voor dat laatste is het echter wel nodig dat de programmeur meer kan dan wat regels code achter elkaar plakken. Waarmee ook meteen de grote zwakte van PHP wordt aangestipt. Doordat PHP zo flexibel is als elastiek, nodigt het ook uit om 'snel' even wat code te bakken, ad hoc-aanpassingen aan een webproject te doen of zonder vooropgezet plan iets in elkaar te knutselen.

Veel doe-het-zelfprogrammeurs gaan op deze manier te werk en menigeen durft zichzelf PHP-programmeur te noemen zonder planmatig te kunnen programmeren. Waakt u ervoor in deze zelfde valkuil te springen. Zomaar iets programmeren, betekent meestal beduidend meer nawerk en soms zelfs helemaal opnieuw beginnen, voordat het project ook goed draait.

1.6 Model, View, Controller

Dit boek gaat om deze reden niet alleen over een grondige basis van PHP, maar ook over de basis van goed programmeerwerk.

Een goed gebruik bij allerlei ontwikkelplatformen en frameworks voor het web, is ontwikkelen volgens het paradigma MVC: Model, View, Controller. Daarvoor is een goede reden: het web werkt namelijk volgens dit paradigma. In het kort komt het erop neer, dat de View bepaalt wat de gebruiker van een website in zijn browser ziet; het Model bepaalt hoe de gegevens die in de website gebruikt worden, op de server zijn opgeslagen; de Controller koppelt beide zaken aan elkaar en regelt wat er moet gebeuren met de input van de gebruiker.

Ook als u een heel klein programmaatje voor een website maakt, bijvoorbeeld een peiling (poll) of een gebruikerslogin, kunt u werken met dit paradigma. Het grote voordeel is dat uw systeem weinig fouten of verrassingen kan bevatten en bijvoorbeeld dat het makkelijk uitbreidbaar is.

1.7 OOP

Goed en gestructureerd programmeren betekent ook dat u gebruikmaakt van klassen en objecten. Objectgeoriënteerd programmeren is niet voorbehouden aan complexe talen als Java of C en is ook niet alleen bedoeld voor grote projecten. Het kleinste project, zoals de eerdergenoemde peiling of gebruikerslogin heeft voordeel bij het werken met klassen. Klassen maken een project overzichtelijk en eenvoudig te veranderen of uit te breiden. Klassen maken het bovendien makkelijk om eerder ontwikkelde onderdelen in nieuwe projecten toe te passen. Minder dubbel werk dus. Ook Object Oriented Programming (OOP) wordt in dit boek besproken en waar mogelijk toegepast.

1.8 SQL en MySQL

Een PHP-project is nauwelijks denkbaar zonder database. De database bewaart en ordent de gegevens van de gebruikers. PHP is geen database en heeft ook geen database. Daarvoor moet u dus gebruikmaken van een van de bekende databases voor het web: SQLite, MySQL, Postgre SQL of Oracle. MySQL wordt veruit het meest gebruikt op gewone webservers en is bovendien gratis. Een voor de hand liggende keuze dus. Maar daarmee bent u er nog niet.

PHP zelf kan geen database aanspreken. Daarvoor is speciale databasetaal nodig: SQL. Het leuke van SQL is dat alle genoemde databases deze verstaan.

In dit boek leert u tevens de basis van SQL, omdat SQL en PHP vrijwel onlosmakelijk met elkaar verbonden zijn.

1.9 WAMP, LAMP, MAMP op uw eigen computer

Om PHP-projecten te kunnen draaien, hebt u eigenlijk een webserver nodig. PHP draait immers op een webserver en niet in de browser. Op het web zijn allerlei manieren te vinden om van uw eigen computer een webserver te maken. Zo'n webserver is echter volstrekt niet geschikt voor het permanent hosten van een website op het web – daar komt veel meer bij kijken – maar is prima voor het ontwikkelen van webprojecten.

De meest hapklare oplossing is zonder twijfel WAMP voor Windows of MAMP voor de Mac. WAMP is het prachtige acroniem voor Windows Apache (de webserver), MySQL (de database) en PHP. XAMP is de OS X-versie daarvan.

Werkt u met een Linux-werkstation, dan bent u ongetwijfeld ook bekend met de ingebouwde Apache-server en MySQL-component. In dit boek wordt uitgegaan van Windows als werkcomputer. In een volgend hoofdstuk wordt WAMP geïnstalleerd als webserver. Daarin komt ook aan de orde hoe u WAMP met PHP 6 kunt installeren – een hapklare versie van WAMP met PHP 6 is bij het ter perse gaan van dit boek nog niet beschikbaar.

1.10 Editors

Een ander punt van belang bij het programmeren in PHP is de tekstverwerker waarmee u de code schrijft. PHP wordt geschreven en opgeslagen als gewone tekst. Met Windows Kladblok kunt u dus PHP-code schrijven. Met WordPad of Word kan dat niet, omdat deze editors allerlei opmaakcode onzichtbaar rond uw teksten plaatsen.

Het kan natuurlijk wel veel handiger dan met Kladblok. Er zijn editors die u helpen met het indelen van de PHP-code. Er zijn er die helpen PHP-code voor u te typen en er zijn editors die complete projecten met meerdere bestanden voor u managen en zelfs de code voor u kunnen uitvoeren om te debuggen (fouten opsporen en oplossen). In een volgend hoofdstuk komen deze editors aan de orde. In dit boek wordt een eenvoudige editor gebruikt, zodat u zich kunt concentreren op het leren van PHP en niet eerst ook nog eens verward raakt door de duizend mogelijkheden van de editor.

1.11 'You will be assimilated'

Leren programmeren is leren denken als een computer. Liever zouden we de computer leren denken als een mens, maar aangezien mensen het onderling al niet eens kunnen worden over 'hoe te denken', is de eerste optie veruit het handigst. U moet uw denken dus aanpassen aan de beperkingen van de computer. You will be assimilated…

U moet als programmeur dus denken als een computer. Dat is binair denken, tweewaardig denken. Iets kan, of iets kan niet. Iets staat aan of uit. Een beetje, misschien, of ongeveer bestaan niet in het binaire denken. Wilt u programmeren, dan moet u denken in keuzes en cycli. Laten we dit concept eens loslaten op de manier waarop kinderen geleerd wordt over te steken:

```
kijk naar links
kijk naar rechts
kijk weer naar links
oversteken
klaar
```

Een mooie methode, maar zou een computer deze methode toepassen, dan gaat het dus meteen al mis als er een auto aankomt. We zijn namelijk in de code vergeten in te calculeren, dat er niet alleen gekeken maar ook geëvalueerd moet worden. Kijken is niet voldoende, er moet bepaald worden wat de computer ziet en er moeten beslissingen genomen worden. Dat wordt dan dus als volgt:

```
kijk naar links
ALS( er niets aankomt ) DAN:
     kijk naar rechts
     ALS( er niets aankomt ) DAN:
          kijk naar links
          ALS( er niets aankomt ) DAN:
               oversteken
KLAAR
```

Dat lijkt al beter. Er wordt immers nu alleen maar overgestoken als er echt van links en rechts niets aankomt. Maar wat moet onze computer doen, als er wel iets aankomt? Ook dat moet worden ingebakken:

```
kijk naar links
ALS( er niets aankomt ) DAN:
     kijk naar rechts
     ALS( er niets aankomt ) DAN:
          kijk naar links
          ALS( er niets aankomt ) DAN:
               oversteken
```

```
        ANDERS:
            Wacht
        KLAAR
    ANDERS:
        wacht
    KLAAR
ANDERS:
    wacht
KLAAR
```

1.11.1 Syntaxis

Door regels code in te laten springen en door het woordje KLAAR toe te voegen, moet in het voorbeeld hierboven duidelijk zijn wat er in deze code bij de verschillende voorwaarden hoort. U ziet dat het bij dit kleine stukje code al minder overzichtelijk wordt, nu er meerdere voorwaarden in elkaar genest zijn. Tijd dus om meteen maar over te stappen op de syntaxis van PHP. Woorden zoals ALS en ANDERS worden dus ook meteen in het Engels vermeld, want PHP is zoals vrijwel alle programmeertalen opgebouwd in het Engels.

Een syntaxis definieert de manier waarop een programmeertaal (of eigenlijk elke taal) is opgebouwd. De ene programmeertaal gebruikt inspringen om structuur in de code aan te brengen (zoals Python), PHP gebruikt accolades, haakjes en puntkomma's hiervoor. De ene programmeertaal gebruikt woorden als IF THEN ELSE voor de beslissingen, terwijl andere (waaronder PHP) het ook zonder THEN kunnen. Oversteken in PHP zou er als volgt uit kunnen zien – met natuurlijk hier en daar nog veel Nederlands ertussen, waar geen PHP-interpreter iets van snapt:

```
kijk_naar_links;
if( er_niets_aankomt ){
    kijk_naar_rechts;
    if( er_niets_aankomt ){
        kijk_naar_links;
        if( er_niets_aankomt ){
            oversteken;
        }else{
            wacht;
        }
    }else{
        wacht;
    }
}else{
    wacht;
}
```

Dezelfde code, maar nu meer in PHP-stijl. Deze beslissingsstructuur is opgebouwd rond de IF-THEN-ELSE-structuur. In PHP ziet die er als volgt uit:

```
if( voorwaarde ){
    opdrachten_1;
}else{
    opdrachten_2;
}
```

Als aan de voorwaarde wordt voldaan, wordt de reeks `opdrachten_1` uitgevoerd. Dat kan één regeltje code zijn, maar ook een compleet programma van duizenden regels. Als niet aan de voorwaarde wordt voldaan, wordt de reeks `opdrachten_2` uitgevoerd – dat zijn dus de opdrachten tussen de accolades bij `else`.

De cursieve tekst in de code is natuurlijk geen PHP. PHP houdt ook niet van spaties in namen, dus hier zijn underscores (_) geplaatst. Het inspringen is niet nodig voor de werking van een PHP-programma, maar wel voor de leesbaarheid van de code voor u als programmeur, of voor iemand anders die na u met de code verder wil. De puntkomma's (;) achter de statements zijn wel noodzakelijk voor PHP. Ze maken het ook mogelijk om code anders op te maken:

```
kijk_naar_links; if( er_niets_aankomt ){ kijk_naar_rechts; if( er_niets_aankomt ){
kijk_naar_links; if( er_niets_aankomt ){ oversteken; }else{ wacht; }}else{ wacht;
}}else{ wacht;}
```

Wat PHP betreft gebeurt er nu hetzelfde als in het vorige oversteekvoorbeeld, maar voor een programmeur is deze code onleesbaar. Dit soort broddelwerk vraagt gewoon om fouten.

1.11.2 Herhaling

Helaas staat onze computer met deze code, ondanks alle duidelijke voorwaarden en beslissingen nog steeds op de stoep te wachten en is nog niet overgestoken. Er kwam namelijk een fietser van links. De computer moest dus de opdracht `wacht` uitvoeren en toen was de code afgelopen.

We zijn vergeten te vertellen dat hij deze beslissingsstructuur net zolang moet herhalen, tot hij met succes is overgestoken. Er moet dus naast alle conditionals (voorwaarden) ook nog een loop (lus) worden gemaakt. En niet zomaar een loop maar een conditional loop – een lus met een voorwaarde. Namelijk de voorwaarde `nog_niet_overgestoken`. Want zolang de computer niet is overgestoken, moet gekeken worden naar links en naar rechts, net zolang tot wel overgestoken kan worden. De benodigde code kan ook meteen in correct PHP worden gezet:

```
while( nog_niet_overgestoken ){
    kijk_naar_links;
    if( er_niets_aankomt ){
        kijk_naar_rechts;
        if( er_niets_aankomt ){
            kijk_naar_links;
            if( er_niets_aankomt ){
                oversteken;
            }else{
                wacht;
            }
        }else{
            wacht;
        }
    }else{
        wacht;
    }
}
```

1.11.3 Vele wegen naar Rome

Om heelhuids over te steken zijn er in programmacode heel wat mogelijkheden. Zo is in het voorbeeld hierboven ervoor gekozen om de drie beslissingen te nesten. Dat is ook het meest logisch, omdat het weinig zin heeft om verkeer van rechts te controleren als er van links al een vrachtwagen aan komt denderen. In programmacode kan het echter toch handig zijn om zo'n beslissingstructuur anders op te bouwen, zonder dat het resultaat anders wordt:

```
while( nog_niet_overgestoken ){
    kijk_naar_links;
    if( er_iets_aankomt ){
        continue;
    }
    kijk_naar_rechts;
    if( er_iets_aankomt ){
        continue;
    }
    kijk_naar_links;
    if( er_iets_aankomt ){
        continue;
    }
    oversteken;
}
```

Deze code is al een stuk beter leesbaar. De opdracht `continue` is een PHP-opdracht die ervoor zorgt dat alle volgende opdrachten binnen de `while`-lus niet meer worden uitgevoerd en dat de lus vanaf het begin gecontinueerd wordt, al-

thans, als aan de conditie van de `while`-lus voldaan wordt. Komt er dus iets van links, dan wordt er niet meer opnieuw naar rechts of links gekeken, maar begint de lus opnieuw met naar links kijken. De 'menselijke' opdracht `wacht` uit de code hiervoor deed natuurlijk niets. Want tijdens dat wachten wordt gewoon opnieuw geobserveerd.

1.11.4 Functies

In feite wordt dus driemaal dezelfde handeling uitgevoerd: kijken en oordelen. Nu gebruiken we de computer om ons werk makkelijker te maken en vooral om minder herhalend werk te doen. Daar kunnen we dus nu meteen mee beginnen.

Vrijwel alle moderne programmeertalen hebben een manier om stukken code te bundelen. Naast allerlei programmeertechnische voordelen (dit onderdeel komt later aan de orde), maakt dit de code ook beter leesbaar. In ons oversteekprogramma zou het onderdeeltje 'kijken en oordelen' als volgt kunnen worden:

```
function erKomtIetsVan($richting){
    kijk_naar($richting);
    return(er_iets_aankomt);
}
```

Alleen de cursieve woorden zijn nu nog 'mensentaal'. De functie (`function`) `erKomtIetsVan($richting)` is gedefinieerd volgens de PHP-syntaxis:

- `function` – hiermee wordt een nieuwe functie gedefinieerd;
- `erKomtIetsVan` – de naam van de functie;
- `()` – staat altijd achter de naam van een functie, tussen de haken staan eventuele parameters;
- `$richting` – de naam van een variabele (die in dit voorbeeld staat voor de richting waarin gekeken moet worden). De naam van een variabele begint in PHP altijd met het dollarteken (`$`).
- tussen de `{` en `}` staan de opdrachten die achtereenvolgens in de functie worden uitgevoerd;
- `return` – geeft een waarde terug aan het eind van de functie (in dit voorbeeld dus of er iets aankomt of niet – Ja of Nee bijvoorbeeld).

In gewone mensentaal komt het erop neer dat deze functie naar `richting` kijkt net als in de voorbeelden hierboven en het antwoord teruggeeft met de opdracht `return`. Dat antwoord kan zijn: Ja, er komt iets aan, of Nee, er komt niets aan.

Ook het testen of we overgestoken zijn, moet natuurlijk met een functie gebeuren:

```
function nietOvergestoken(){
    return(nog_niet_overgestoken);
}
```

Tot slot moet ook nog het oversteken zelf in een functie worden ondergebracht:

```
function veiligOversteken(){
    heel_veel_moeilijke_code_om_over_te_steken;
}
```

De code van ons oversteekprogramma ziet er met deze drie nieuwe functies als volgt uit:

```
while( nietOvergestoken() ){
    if( erKomtIetsVan('links') ){
        continue;
    }
    if( erKomtIetsVan('rechts) ){
        continue;
    }
    if( erKomtIetsVan('links') ){
        continue;
    }
    veiligOversteken();
}
```

Dit stuk code bevat geen cursieve tekst, omdat het een correct werkend stuk PHP-code zou kunnen zijn.

In de functies wordt dus het kijken, het oversteken en het resultaat geregeld en in de while-lus de hoofdstructuur. De waarden `'links'` en `'rechts'` staan tussen aanhalingstekens. Deze worden binnen de while-lus als parameters meegegeven aan de functie `erKomtIetsVan()`, waarin ze worden bewaard in de variabele `$richting`.

Het hoofdprogramma is duidelijk leesbaar dankzij de veelzeggende namen van onze functies, zoals `erKomtIetsVan()` en `veiligOversteken()`. Maak er een gewoonte van om de namen van functies, variabelen en klassen op deze manier duidelijk en beschrijvend te maken.

1.11.5 Complexe factoren

Zolang er links of rechts iets aankomt, zal onze computer blijven wachten op de stoep. En telkens nadat er iets aankwam, begint het hele verhaal weer opnieuw. En zo hoort het ook als een computer moet leren oversteken. Want risico's incalculeren, is voor 'gevorderde overstekers'. De bijbehorende code van een ge-

vorderde oversteker is ook een stuk complexer. De risico's worden bepaald aan de hand van de conditie `( er_iets_aankomt )`. Bij het calculeren van het risico betrekken we ook hoe snel iets nadert. Een voortrazende auto geeft een ander risico dan oma op de fiets.

Heeft de gevorderde oversteker haast, dan kan ook nog aan de hand van de snelheid van het naderende object worden bepaald hoe snel het oversteken moet gebeuren. Ons eenvoudige stukje code wordt dankzij onze mooie structuur met de drie functies echter nauwelijks anders als we al deze factoren gaan toevoegen.

1.11.6 Denken als een computer

Inmiddels hebt u, denkend als een computer, de computer geleerd veilig over te steken. U hebt ook al kennisgemaakt met conditions, loops en functions en hebt de variabele `$richting` gebruikt. Kortom: programmeren in een notendop. In de loop van dit boek komen deze en nog veel meer zaken uitgebreid aan de orde.

Onze code om over te steken zit nog vol met 'menselijke' aspecten. PHP begrijpt natuurlijk nog geen snars van het statement 'er iets snels aankomt' of van het statement 'oversteken'. Ook 'kijk naar links' betekent weinig.

Stel dat onze computer al een oog zou hebben om te kijken en stel dat de computer al onderscheid kan maken tussen een stoep, een lantaarnpaal en een voortrazende auto, dan nog moet er heel wat code geschreven worden om de vraag te beantwoorden: 'er iets aankomt' en hoe snel dat dan is en wanneer het object ons pad kruist.

Maar uitgaande van het bestaan van al deze technologie, zou je de computer inderdaad kunnen vragen: komt er een object aan (1), hoe ver is dat object nu verwijderd (2) en op welk moment kruist dat object ons pad (3)?

Want aan de hand van deze drie gegevens kan worden berekend met welke snelheid de lijn van een eventueel aankomende object moet worden gekruist, om het object niet te raken (1), en of die snelheid voor de wandelaar haalbaar en wenselijk is (2). Het oversteken zelf is ook nog weer een 'menselijke' handeling.

Maar hoe complex al deze zaken ook zijn: aan de structuur van ons simpele hoofdprogramma in de `while`-lus hoeft weinig te veranderen om ook deze complexe factoren toe te voegen. Daarvoor zijn de drie functies, die eindeloos zijn uit te breiden. In feite hoeft alleen maar een oversteekadvies te worden bepaald dat zegt: niet oversteken, of oversteken met een bepaalde snelheid, die dan weer binnen de gewenste en mogelijke snelheid van de wandelaar ligt.

1.12 Top-down refinement, stapsgewijze verfijning

Deze methode van programmeren heet top-down refinement. Eerst de grote lijnen vastleggen, waarbij goed nadenken en uittekenen van de structuren erg belangrijk is, en dan pas de details aanbrengen. Net als het ontwerpen van een gebouw:

- Eerst wordt bepaald wat de mogelijkheden van een gebouw moeten zijn en of er nog uitbreidingen mogelijk moeten zijn.
- De architect bedenkt een passende structuur, maar bemoeit zich niet met waar welke baksteen moet liggen.
- De bouwkundige berekent en tekent de technische structuur en de substructuren, maar bemoeit zich evenmin met waar welke baksteen moet liggen.
- De uitvoerder bepaalt wie wat, wanneer en waar doet binnen de gedefinieerde structuren, wie muren metselt, wie kozijnen timmert en in welke volgorde dat moet gebeuren, maar bemoeit zich nog steeds niet met waar welke baksteen moet liggen.
- de metselaar legt tot slot stenen en cement op de juiste plek en bepaalt welke baksteen waar ligt.

Top-down refinement is een handige manier om te programmeren met structuur en zonder later tegen verrassingen aan te lopen. Simpelweg beginnen met code bakken, is vrijwel altijd een lange en heilloze weg. Komen er later factoren bij, dan moet opnieuw naar de hoofdstructuur van het programma gekeken worden om de nieuwe zaken in te voegen.

1.13 Tot slot

Hopelijk bent u inmiddels warmgelopen om met PHP aan de slag te gaan. In het volgende hoofdstuk wordt WAMP gedownload en geïnstalleerd, zodat u op uw eigen computer met PHP, MySQL en de browser uw site kunt ontwikkelen en testen.