Kapitel 2

Grundlagen von PHP

Angefangen hat alles 1994. Damals entwickelte Rasmus Lerdorf einige Makros, um die Funktionalität seiner Homepage zu erweitern. Die Sammlung nannte er »Personal Home Page Tools«, also etwa Werkzeuge für die (Gestaltung) der persönlichen Homepage. Die Tools konnten von jedermann genutzt werden – und das wurden sie unter dem Kürzel PHP dann auch.

Im Laufe der Zeit wurde aus den bescheidenen Makros und Tools eine mächtige Scriptsprache. Die Auflösung des Kürzels schien nicht mehr angemessen, schließlich kann man mit PHP nicht nur seine private Homepage ein wenig aufpeppen, sondern vollständige und komplexe Webauftritte organisieren. Heute steht PHP für *PHP Hypertext Preprocessor* und ist damit eine jener rekursiven Abkürzungen, die in der Formulierung, die sie abkürzen sollen, bereits enthalten sind (ein anderes Beispiel für diese etwas seltsame Art der Abkürzung ist GNU, das für *GNU is Not Unix* steht). PHP kann als Preprocessor gesehen werden, weil zuerst die PHP-Anweisungen ausgeführt werden, bevor der HTML-Code vom Webbrowser interpretiert und angezeigt werden kann.

PHP ist ein typisches Open-Source-Produkt, das heißt, dass der Quellcode von PHP frei zugänglich ist und jeder, der Verbesserungen oder Ergänzungen anbringen will, dies problemlos tun kann. PHP entwickelte sich auf diese Weise kontinuierlich weiter, bis im Juni 1998 schließlich die Version PHP 3 vorlag. PHP 3 sorgte für den endgültigen Durchbruch der Scriptsprache – heute gehört PHP zu den beliebtesten und leistungsfähigsten Webserver-Ergänzungen im World Wide Web. Aus der Erweiterung für private Webseiten ist im Laufe der Jahre eine wichtige Plattform für dynamische Internetauftritte geworden, die von großen Konzernen genauso genutzt wird wie von privaten Homepagebastlern.

Aktuell ist die Version PHP 4. Welche Version auf einem Webserver installiert ist, lässt sich mit einem einfachen PHP-Kommando ermitteln. Wie das genau geht, erfahren Sie im nächsten Kapitel.

Mit PHP programmieren Sie verschiedene Scripts zur Erledigung der unterschiedlichsten Aufgaben. Was genau bedeutet das bzw. was macht man eigentlich, wenn man »programmiert«?

Wer programmiert, schreibt ein Programm. Ein Programm besteht aus einer Reihe von Anweisungen, Befehlen oder Kommandos (welchen Begriff Sie hier benutzen, bleibt Ihnen überlassen), deren sequentielle Abarbeitung eine bestimmte Aufgabe löst.

Eine (sehr kurze) Geschichte von PHP

Was heißt »programmieren«?

Ein einfaches Beispiel mag den Sachverhalt verdeutlichen. Angenommen, man möchte ein Programm schreiben, das den Benutzer zur Eingabe zweier Zahlen auffordert, die beiden Zahlen addiert und das Ergebnis auf dem Bildschirm anzeigt. Die hierzu nötigen Anweisungen könnten so aussehen:

```
1 Fordere den Anwender zur Eingabe zweier Zahlen auf.
2 Addiere die beiden Zahlen.
3 Merk dir das Ergebnis.
4 Schreibe "Die Summe der beiden Zahlen lautet:".
5 Schreibe das Ergebnis der Addition.
```

Das sieht schon ganz ordentlich aus, aber bei näherem Hinsehen stellt man fest, dass zwischen dem ersten und dem zweiten Schritt vielleicht noch eine weitere Anweisung gehört:

```
1a Merk Dir die beiden eingegebenen Zahlen.
```

Schließlich sind Computer bekanntlich außerordentlich sture Dinger und tun ausschließlich das, was man ihnen sagt. Wenn Sie in Ihrem Programm nicht ausdrücklich darauf hinweisen, dass sich der Computer die beiden Zahlen merken soll, die der Anwender eingegeben hat, dann tut er das auch nicht und hat im nächsten Schritt nichts, was er addieren könnte.

Da Sie einem Computer wirklich jeden noch so kleinen Schritt vorgeben müssen, entpuppt sich auch schon der erste Schritt in dem kleinen Programm als zu allgemein – wir legen nirgendwo fest, dass der PC nach der Aufforderung zur Eingabe diese überhaupt entgegennehmen soll. Also formulieren wir die einzelnen Schritte noch einmal neu – wie man sieht, wächst die Liste mit Anweisungen schon bei vermeintlich einfachen Aufgaben sehr rasch an.

```
1 Schreibe "Geben Sie bitte eine Zahl ein: ".
2 Lies die Zahl, die der Anwender eingibt.
3 Merk dir diese Zahl.
4 Schreibe "Geben Sie bitte eine weitere Zahl ein: ".
5 Lies die Zahl, die der Anwender eingibt.
6 Merk dir diese Zahl.
7 Addiere die erste und zweite gemerkte Zahl.
8 Merk dir das Ergebnis der Addition.
9 Schreibe "Die Summe der beiden Zahlen lautet: ".
10 Schreibe das Ergebnis.
```

Da ein Computer mit umgangssprachlichen Sätzen nichts anfangen kann, müssen diese Anweisungen nun noch in die Sprache übersetzt werden, die der Computer versteht, und man erhält ein Programm, das den Anwender zur Eingabe zweier Zahlen auffordert und die Summe der beiden Zahlen addiert. Die Sprache, in der das Programm formuliert ist, heißt Programmier- oder im Falle von PHP, Javascript und ähnlichen Sprachen Scriptsprache. Dabei handelt es sich um so genannte »Hochsprachen«, da hier noch mit einigermaßen verständlichen – englischen – Begriffen gearbeitet werden kann. So sähe der erste Schritt in PHP zum Beispiel so aus: `echo "Geben Sie bitte eine Zahl ein:";`. Allerdings ist die einzige Sprache, die ein Computer versteht, eine Abfolge von Nullen und Einsen: 100101011010101 und so weiter und so fort. Die Programmiersprache muss also erst noch in diesen Maschinencode übersetzt werden – doch darum müssen Sie sich nicht kümmern, das ist Aufgabe des so genannten Compilers oder Interpreters.

Dieser gesamte, hier rudimentär und vereinfacht beschriebene Prozess ist gemeint, wenn von »Programmieren« die Rede ist. Wer programmiert, erledigt mehrere Dinge auf einmal. Zuerst wird die Aufgabe analysiert, die der Computer lösen soll, und ein Lösungsweg entwickelt. Dieser Lösungsweg wird dann in möglichst einfachen und präzise definierten Schritten beschrieben. Anschließend wird diese Beschreibung in der Programmier- bzw. Scriptsprache formuliert, mit der man den Computer programmiert.

Der Computer arbeitet die erteilten Anweisungen sequentiell ab, das heißt, er beginnt mit der ersten Anweisung, führt diese aus, geht zur zweiten Anweisung, führt diese aus und macht so weiter, bis er die letzte Anweisung gelesen und ausgeführt hat.

In dem kleinen Beispielprogramm taucht mehrfach die Anweisung auf, der Computer möge sich bestimmte Zahlen merken. Das klingt eigentlich ganz einfach, schließlich ist der Computer von Haus genau dazu da, sich Dinge zu merken bzw. zu speichern, doch – vielleicht ahnen Sie es schon – ganz so einfach ist es nun auch wieder nicht. Wenn der Programmierer nicht eindeutig angibt, wie und wo die eingegebene Zahl gespeichert, also »gemerkt« werden sollen, wird der Computer dies auch nicht tun.

Dafür werden so genannte Variablen eingesetzt. Damit sind symbolische Bezeichner für Speicherbereiche gemeint, in denen der Computer beliebige Werte ablegen kann. Um die konkrete Verwaltung des Arbeitsspeichers muss sich der Programmierer in aller Regel nicht kümmern. Er muss nur einen symbolischen Bezeichner als Variable deklarieren und der Variablen einen Wert zuweisen, den Rest übernimmt der Computer.

Variablen und Konstanten

Das kleine Additionsprogramm könnte mit dem Einsatz von Variablen etwa so aussehen:

```
1 Schreibe „Geben Sie bitte eine Zahl ein: „.
2 Lies die Zahl, die der Anwender eingibt.
3 Speicher die Zahl in der Variablen ZAHL_1.
4 Schreibe „Geben Sie bitte eine weitere Zahl ein: „.
5 Lies die Zahl, die der Anwender eingibt.
6 Speicher die Zahl in der Variablen ZAHL_2.
7 Addiere ZAHL_1 und ZAHL_2.
8 Speicher das Ergebnis in der Variablen ERGEBNIS.
9 Schreibe „Die Summe der beiden Zahlen lautet: „.
10 Schreibe ERGEBNIS.
```

`ZAHL_1`, `ZAHL_2` und `ERGEBNIS` sind innerhalb dieses schematischen Programms die Bezeichner für drei unterschiedliche Variablen, denen im Laufe des Programms verschiedene Werte zugewiesen werden. Bei jedem Durchlauf wird der Wert von `ZAHL_1` und `ZAHL_2` durch die Benutzereingabe neu festgelegt, wodurch sich, logisch, auch der Wert von `ERGEBNIS` bei jedem Durchlauf ändert. Kurz, die Werte dieser Zahlen sind nicht ein für alle Mal definiert, sondern variabel. Welche konkreten Werte die beiden Variablen `ZAHL_1` und `ZAHL_2` im Programm annehmen, ist für die korrekte Ausführung nicht weiter wichtig, sie werden auf jeden Fall addiert und die Summe wird in `ERGEBNIS` abgelegt.

Mitunter benötigt man aber auch etwas Dauerhaftes in seinem Programm, die so genannten Konstanten. Auch hierbei handelt es sich um Speicherbereiche, denen ein Wert zugewiesen wird, doch dieser Wert ist während der Programmausführung nicht mehr veränderbar, sondern bleibt konstant. Als Beispiel diene ein rudimentäres Programm, das die Benutzereingaben verdoppelt:

```
1 Setze die Konstante MULTIPLIKATOR gleich 2.
2 Lies die Zahl, die der Anwender eingibt.
3 Speicher die Zahl in der Variablen ZAHL.
4 Multipliziere ZAHL mit MULTIPLIKATOR.
5 Speicher das Ergebnis in der Variablen ERGEBNIS.
6 Schreibe „Das Doppelte von „.
7 Schreibe ZAHL.
7 Schreibe „ ist gleich: „.
8 Schreibe ERGEBNIS.
```

ZAHL und ERGEBNIS sind in diesem Beispiel erneut Variablen, während MULTIPLIKATOR anfangs definiert wird und danach konstant bleibt. Je nachdem, welchen Wert der Konstanten MULITPLIKATOR zu Beginn zugewiesen wird, kann man das Programm dazu benutzen, Benutzereingaben zu verdoppeln, zu verdreifachen, zu vervierfachen und so weiter.

Alle bisherigen Beispiele gleichen schnurgeraden programmtechnischen Straßen, ohne Abzweigung, Kurven, Ausweichmöglichkeiten oder alternative Streckenführung. Das Programm beginnt am Anfang, hangelt sich der Reihe nach durch die einzelnen Anweisungen und hört auf, sobald es die letzte Anweisung erreicht hat. Damit lassen sich zwar schon eine ganze Reihe von Aufgaben lösen, aber ohne Entscheidungsmöglichkeiten und Verzweigungen im Programm wird man kaum auskommen.

Nehmen wir als Beispiel die Aufforderung an den Benutzer, eine Zahl einzugeben. Nichts hindert ihn daran, an dieser Stelle ein Wort oder beliebige Zeichen einzutippen. Doch wenn das Programm statt der erwarteten Zahl nur eine Zeichenkette wie »Feinfinger« oder »!§“&@*%« als Eingabe bekommt, wird der Programmablauf nicht mehr dem entsprechen, was sich der Programmierer gedacht hat. Anders gesagt: Das Programm wird wahrscheinlich abstürzen.

Um das zu verhindern, könnte man zum Beispiel folgende Abfrage vornehmen:

```
1 Schreibe „Geben Sie bitte eine Zahl ein: „.
2 Lies die Zeichen, die der Anwender eingibt.
3 Sind die Zeichen eine Zahl?
  3a Ja. Gehe zur Anweisung 4.
  3b Nein. Gehe zurück zur Anweisung 2.
4 Speicher die Zahl in der Variablen ZAHL_1.
```

Mit der Überprüfung und der Verzweigung in Abhängigkeit von der Eingabe in Anweisung 3 ist sichergestellt, dass das Programm auch tatsächlich eine Zahl bekommt, wenn es eine Zahl erwartet.

Bislang wird ein Programm bzw. ein Block aus Anweisungen genau einmal durchlaufen. Doch ein Computer ist endlos geduldig und entfaltet seine wahre Stärke erst dann, wenn er einen bestimmten Anweisungsblock so lange wiederholt, wie Sie das wollen. Diese so genannten Schleifen sind das Kernstück eines

Verzweigungen

Schleifen

jeden Programms. Auch hierzu ein Beispiel. Nehmen wir an, es sollen die Zahlen von 1 bis 10 ausgegeben werden. Nach unseren bisherigen Kenntnissen müsste man diese Aufgabe folgendermaßen lösen:

```
1 Schreibe „1".
2 Schreibe „2".
3 Schreibe „3".
4 Schreibe „4".
5 Schreibe „5".
6 Schreibe „6".
7 Schreibe „7".
8 Schreibe „8".
9 Schreibe „9".
10 Schreibe „10".
```

Das würde zwar funktionieren, ist allerdings sehr plump, fehleranfällig (jede Zeile, die Sie tippen müssen, ist eine potentielle Fehlerquelle), wenig effektiv und letztlich völlig unzureichend: Was machen Sie, wenn Sie nicht die Zahlen bis 10, sondern bis 100, 1000 oder gar 100.000 ausgeben möchten?

Eleganter ist hier der Einsatz einer Schleife und einer Variablen:

```
1 Weise der Variabeln ZAHL den Wert 1 zu.
2 Schreibe ZAHL.
3 Erhöhe ZAHL um 1.
4 So lange ZAHL kleiner oder gleich 10 ist, gehe zurück zu
Anweisung 2.
```

Zuerst wird also der Variablen `ZAHL` ein Startwert zugewiesen und dieser Wert ausgegeben. Auf dem Bildschirm steht nun eine 1. Danach wird ZAHL um 1 erhöht, hat nun also den Wert 2. Im nächsten Schritt wird der aktuelle Wert von `ZAHL` mit 10 verglichen. So lange `ZAHL` kleiner oder gleich 10 ist, springt das Programm zu Anweisung Nummer 2 zurück, gibt den Wert also aus. Auf dem Bildschirm stehen nun die beiden Zahlen 1 und 2. `ZAHL` wird erneut um 1 erhöht, enthält also 3. Wieder ist die Bedingung in Anweisung Nummer 4 nicht erfüllt, das Programm springt also zurück zu Anweisung Nummer 2, nun stehen auf dem Bildschirm die Zahlen 1, 2 und 3.

Das wird so lange fortgesetzt, bis `ZAHL` den Wert 10 angenommen hat. Dann wird ein letztes Mal zur Anweisung Nummer 2 zurückgesprungen, `ZAHL` ausgegeben und um 1 erhöht. Nun hat `ZAHL` den Wert 11, ist also weder kleiner noch gleich 10, sondern größer, und das Programm wird beendet.

Die Schleife ist nicht nur kompakter und übersichtlicher als die vorherige Langversion, sie ist auch deutlich flexibler. Wollen Sie nicht nur von 1 bis 10, sondern von 15 bis 45 in Zweierschritten zählen (15, 17, 19 ...), müssen Sie hier lediglich ein paar Werte anpassen:

```
1 Weise der Variabeln ZAHL den Wert 15 zu.
2 Schreibe ZAHL.
3 Erhöhe ZAHL um 2.
4 So lange ZAHL kleiner oder gleich 45 ist, gehe zurück zu
Anweisung 2.
```

Schleifen machen ein Programm flexibler und leistungsfähiger und sind neben den Variablen und den Ablauf- und Kontrollstrukturen der Verzweigungen die dritte große Säule, auf der alle Programme – ganz gleich, in welcher Programmiersprache sie geschrieben sind – ruhen.

Wir können das Schleifen-Beispiel noch weiter ausbauen, indem wir die Berechnung möglichst unabhängig von konkreten Zahlen machen und stattdessen ausschließlich Variablen einsetzen:

```
1 Schreibe ZAHL.
2 Erhöhe ZAHL um WERT.
3 So lange ZAHL kleiner oder gleich ENDE ist, gehe zurück zur
Anweisung 1.
```

Das sieht nun allerdings nicht nach einem ausführbaren Stückchen Code aus, schließlich bleibt hier völlig unbestimmt, welchen Wert die Variablen `ZAHL`, `WERT` und `ENDE` eigentlich haben. Und mit unbestimmten bzw. nicht definierten Werten lässt sich nun mal nicht rechnen. Doch das lässt sich leicht ändern. Wir müssen das kleine Programm nur ein wenig modifizieren:

```
1 Lies die Variablen ZAHL, WERT und ENDE.
2 Schreibe ZAHL.
3 Erhöhe ZAHL um WERT.
4 So lange ZAHL kleiner oder gleich ENDE ist, gehe zurück zur
Anweisung 2.
```

Nun sieht es zwar so aus, als hätten wir überhaupt nichts gewonnen und stattdessen das kleine Programm nur ein wenig undurchsichtiger gemacht, aber der Schein täuscht erneut. Da alle wichtigen Werte nun als Variablen vorliegen, kann dieser Programmschnipsel in nahezu beliebigen Kontexten eingesetzt werden

Funktionen

und die unterschiedlichsten Zahlenreihen ausgeben. Es muss lediglich sichergestellt werden, dass für die Variablen `ZAHL`, `WERT` und `ENDE` passende Werte bereitgestellt werden: Bei diesem Beispielcode handelt es sich im Kern nun um eine so genannte Funktion. Funktionen können einen Namen bekommen – etwa: Zahlenreihe – und dann von beliebiger Stelle im Programm mit ihrem Namen aufgerufen werden, was die Arbeit an einem Programm doch erheblich erleichtert und dem Programmierer völlig neue Möglichkeiten eröffnet.

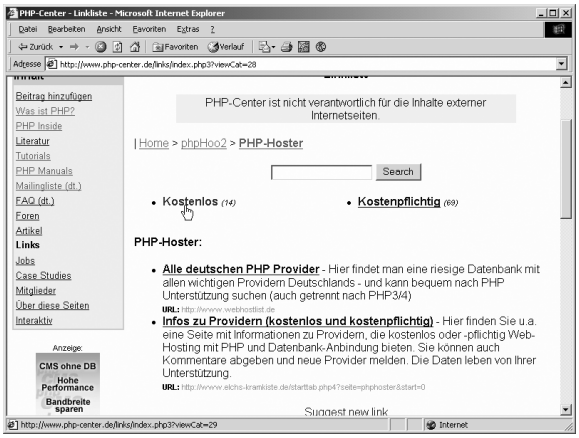
Doch damit ist erst einmal Schluss mit der Theorie, ab dem nächsten Kapitel wird es praktisch: Die ersten echten PHP-Scripts sind nicht mehr weit entfernt.

Jetzt wird's ernst – nach der Theorie geht es jetzt zügig zur Praxis. In diesem Kapitel erfahren Sie, wie Sie einen (kostenlosen) PHP-Anbieter finden und wie PHP in HTML integriert wird. Danach wissen Sie genug, um mit Ihren ersten PHP-Scripts endgültig den Schritt in die Praxis zu tun.

Kapitel 3

Die ersten PHP-Scripts

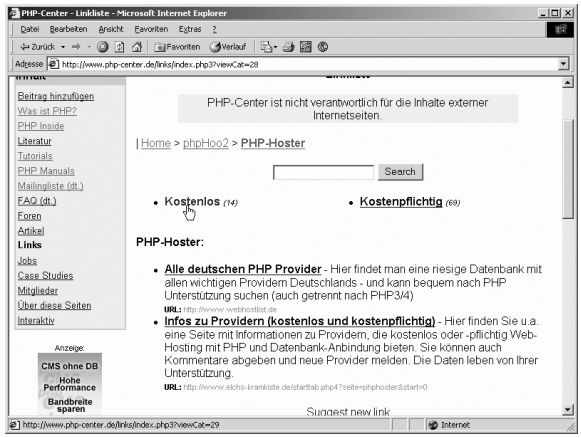
Da PHP eine serverbasierte Scriptsprache ist, benötigen Sie für den Einsatz von PHP einen Server, auf dem PHP installiert ist. Die sind zwar im Netz weit verbreitet, gehören aber (noch) nicht zur Standardausrüstung der Provider. So bieten etwa die beiden größten deutschen Internetzugangsanbieter T-Online oder AOL ihren Kunden zwar die Möglichkeit, eine eigene Homepage abzulegen, aber die Ausführung von PHP-Scripts ist dabei nicht vorgesehen (wenn es Sie tröstet: anderen serverbasierten Scriptsprachen wie zum Beispiel Perl geht es da nicht besser). Ähnlich sieht es bei den zahlreichen Anbietern von kostenlosem Webspace aus, fast immer wird serverseitige Scriptausführung nicht angeboten. Das ist zwar einerseits verständlich, da eine serverbasierte Scriptsprache einen nicht unerheblichen Aufwand bei der Installation und Wartung erfordert, aber dennoch ärgerlich.



Doch nicht verzagen. Auch wenn die großen und populären Anlaufstellen in Sachen PHP noch ein wenig enttäuschen – es gibt sie, die kostengünstigen oder auch komplett kostenlosen PHP-Hoster, auch im deutschsprachigen Internet. Eine gute Übersicht bietet die Linksammlung des PHP-Centers. Steuern Sie auf der Homepage `www.php-center.de/` im Inhaltsverzeichnis den Eintrag *Links* an. Dort finden Sie einen Verweis auf PHP-Hoster, der wiederum in kostenlose und kostenpflichtige Anbieter unterteilt wird.

Die kostenlosen Angebote sind natürlich fast immer in irgendeiner Form eingeschränkt. Meist werden Werbebanner eingeblendet und ist das Datenvolumen stark begrenzt. Doch für Tests und erste Probeläufe reichen diese Angebote allemal aus. Wenn Sie danach zu der Überzeugung kommen, dass Sie auf PHP in Zukunft nicht mehr verzichten wollen, können Sie immer noch zu einem der kostenpflichtigen PHP-Hoster wechseln.

So finden Sie einen (kostenlosen) PHP-Anbieter



Sobald Sie einen PHP-Hoster haben, steht der PHP-Programmierung nichts mehr im Weg.

Damit der Webserver (bzw. das PHP-Modul oder der PHP-Interpreter, aber um diese Details können wir uns im Rahmen dieses Buches nicht kümmern) erkennt, dass die abgelegte Datei eine PHP-Datei ist, die er bearbeiten muss, bevor er sie an den Besucher der Website weiterreicht, müssen Sie ihm das mitteilen. Das geschieht ganz einfach durch eine entsprechende Dateiendung. Hier hat es sich eingebürgert, eine Datei, die PHP-Code enthält, den der Server ausführen soll, mit .php statt .html zu speichern.

Der PHP-Code selbst besteht aus beliebig vielen Anweisungen, die durch ein Semikolon voneinander getrennt werden. Damit der Interpreter den Code eindeutig von der HTML-Umgebung, in die er eingebettet ist, trennen kann, wird er standardmäßig durch ein `<?PHP` eingeleitet und mit `?>` abgeschlossen.

Ein Stückchen PHP-Code sieht also ungefähr so aus:

```
<?PHP
  Eine PHP-Anweisung;
  noch eine PHP-Anweisung;
  und eine dritte;
?>
```

Dieser Code kann an beliebiger Stelle in der HTML-Datei stehen (ja, er muss nicht einmal in einer HTML-Umgebung auftauchen, aber das lassen wir für den Anfang auch außen vor).

Die `<?PHP` / `?>`-Klammer ist die XML-kompatible Version. Daneben gibt es noch drei andere Methoden, um PHP-Code zu markieren.

```
<?
  Eine PHP-Anweisung;
  noch eine PHP-Anweisung;
  und eine dritte;
?>
```

In der Kurzform wird der PHP-Code durch `<?` und `?>` eingeschlossen.

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

PHP-Code in HTML einbetten

1

```
  Eine PHP-Anweisung;
  noch eine PHP-Anweisung;
  und eine dritte;
</script>
```

In der Script-Variante kennzeichnet ein Script-Tag den Code. Mit `<script language="php">` wird der PHP-Block eingeleitet, ein `</script>` beendet ihn.

```
<%
  Eine PHP-Anweisung;
  noch eine PHP-Anweisung;
  und eine dritte;
%>
```

Schließlich können Sie den PHP-Code noch ASP-konform, also zwischen `<%` und `%>` notieren.

Normalerweise sollten Sie mit keiner Variante ein Problem bekommen, doch welche der vier Versionen Sie einsetzen können, hängt nicht zuletzt von Ihrem PHP-Hoster ab. Im Zweifelsfalle probieren Sie es einfach aus. In diesem Buch wird ausschließlich die XML-konforme Version `<?PHP` / `?>` benutzt.

Fassen wir die einzelnen Ausführungen noch einmal in Stichpunkten zusammen:

Eine Webseite mit PHP-Code endet auf .php.

Der PHP-Code wird zwischen `<?PHP` und `?>` eingeschlossen.

Jede PHP-Anweisung endet mit einem Semikolon.

Die Kombination von PHP und HTML hat noch sehr viel mehr zu bieten, doch für den Anfang soll das genügen, es wird Zeit für die ersten Fingerübungen in PHP.

Wie Sie im vorigen Kapitel erfahren haben, gibt es von PHP unterschiedliche Versionen, aktuell ist PHP 4, aber auch PHP 3 findet man noch recht häufig. Es ist also keine schlechte Idee, einmal nachzuschauen, welche PHP-Version der Hoster eigentlich installiert hat, und eine gute Gelegenheit, das erste PHP-Script zu schreiben. Das besteht zwar nur aus einer einzigen Anweisung, aber immerhin.