**PHP lernen**

**Anfangen, anwenden, verstehen**

ISBN 3-8273-2000-3

# 3 PHP kennen lernen


## 3.1 Die Funktionsweise von PHP

PHP ist eine Skriptsprache, die auf dem Server ausgeführt wird. Wenn Sie eine PHP-Datei im Browser aufrufen, wird diese Anfrage an den Webserver weitergeleitet. Auf dem Webserver ist ein Parser (Sprachanalysator) installiert, der den PHP-Code interpretiert und eine HTML-Seite ausgibt, die an den Client, also Ihren Browser, zurück geschickt wird.

An der Dateiendung erkennt der Webserver, dass er es mit einer PHP-Datei zu tun hat. Übliche Endungen sind dabei *.php*, *.php3* oder *.php4*, je nach PHP-Version und Konfiguration des Webservers.

Wenn Sie einen Blick in den Quellcode der Ausgabedatei werfen, werden Sie darin keinen PHP-Code mehr finden, da es sich um eine interpretierte Datei handelt.

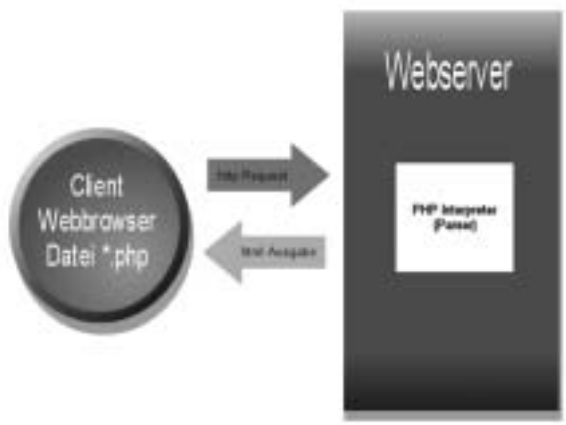


*Abbildung 3.1: Funktionsweise von PHP*

## 3.2 Sprachelemente und Variablen

### 3.2.1 PHP-Einbettung

PHP-Code wird direkt in HTML-Seiten eingebunden und kann an jeder beliebigen Stelle einer HTML-Datei auftreten.

Anfang und Ende einer PHP-Anweisung werden durch PHP-Tags kenntlich gemacht. Dafür gibt es mehrere Möglichkeiten. Zwei gebräuchliche werden hier vorgestellt.

Beide Möglichkeiten benutzen die Syntax einer SGML- bzw. XML "prozessing instruction", bei der der Code in <? ... ?> eingebettet wird. SGML steht für Standard Generalized Markup Language, XML für Extensible Markup Language.

Möglichkeit 1:

```
<?
   echo „Willkommen bei PHP";
?>
```

Möglichkeit 2:

```
<?php
   echo „Willkommen bei PHP";
?>
```

Empfohlen wird die Verwendung der Langform, bei der zu Beginn eines Prozessing-Instruction Blocks explizit angegeben wird, dass es sich bei dessen Inhalt um PHP-Code handelt. Dies geschieht durch das Anhängen der Buchstaben `PHP` direkt nach dem ersten ? (Groß- und Kleinschreibung ist dabei irrelevant).

Die Langform wird auf jeden Fall unterstützt und wird daher auch in diesem Buch bevorzugt eingesetzt.

Wird innerhalb eines Dokumentes nur PHP verwendet, so genügt die Kurzform, wie im ersten Codebeispiel gezeigt.

In Zusammenhang mit XML oder XHTML-Dokumenten benutzen aber auch andere Komponenten Prozessing-Insctruction-Blöcke, so z. B. die Kopfzeilen aller XML und XHTML-Dateien:

```
<?xml version="1.0">
```

In diesem Fall muss verhindert werden, dass PHP auch diese Blöcke zu interpretieren versucht. Dies geschieht durch das Ausschalten der Kurzform über `short_tags=Off` in der *php.ini* und die Verwendung der Langform `<?php ... ?>`.

Der auszugebende Text wird in Anführungszeichen gesetzt. Jede Codezeile wird mit einem Semikolon abgeschlossen. Vielleicht kennen Sie das schon von Javascript-Anweisungen. Die Syntax von PHP ähnelt der Syntax der Programmiersprachen C und Java bzw. der Skriptsprache JavaScript.

Im Browser erscheint daraufhin der in Anführungszeichen angegebene Text:


*Abbildung 3.2: Textausgabe im Browser*

**Übung 3.1:**

Erstellen Sie eine Datei mit dem Namen *hallo.php*, die im XML-Stil geschrieben ist und auf dem Bildschirm den Text „Hallo Welt“ ausgibt.

## 3.2.2 Kommentare

Kommentare sind unverzichtbarer Bestandteil eines guten Programms. Mit ihrer Hilfe bleibt der Code für Sie selbst und auch für andere überschaubar und nachvollziehbar.

In PHP werden Kommentare mit /* eingeleitet und mit */ abgeschlossen. Sie können eine oder mehrere Zeilen umfassen. Alternativ ist es auch möglich, die von C bekannte Notation mit // zu verwenden oder Kommentare mit einem jeder Zeile vorangestellten # einzuleiten, wie man es von Perl-Scripts kennt. Alle drei Variationen sind sowohl für ein- bis mehrzeilige Kommentare als auch für kurze Kommentare am Ende der jeweiligen Anweisung geeignet. Die erste Variante empfiehlt sich besonders für längere Kommentare. Generell sollten Sie sich aber für eine Methode entscheiden und diese durchgängig einsetzen.

```
<?php
   /* Die folgende Anweisung gibt den Text "Willkommen bei PHP"
   auf dem Bildschirm aus */
   echo "Willkommen bei PHP"; // gibt Text auf Bildschirm aus
   echo "Willkommen bei PHP"; /* gibt Text auf Bildschirm aus */
   # Die vorangehende Anweisung gab den Text "Willkommen bei PHP"
   # auf dem Bildschirm aus
?>
```

Achtung: Enthält ein einzeiliger Kommentar ein PHP-Endetag ?>, so endet der Kommentar bereits hier und nicht erst am Zeilenende. Beispiel:

```
<p>HTML-Text mit PHP <?php echo "test"; #test ?> Ausgabe</p>
```

Dieses Listing hat jetzt noch einen kleinen Schönheitsfehler, da die Zeile „Willkommen bei PHP nun zweimal direkt hintereinander ausgegeben wird, ohne Leerzeichen und Zeilenumbruch.


*Abbildung 3.3: Textausgabe ohne Zeilenumbruch*

Um einen Zeilenumbruch einzufügen, wird der HTML-Befehl <br> in den Code eingefügt, auf die gleiche Weise wie die Textausgabe:

```
<?php
   # Die folgende Anweisung gibt den Text "Willkommen bei PHP"
   # auf dem Bildschirm aus
   echo "Willkommen bei PHP";
   echo "<br>"; # Zeilenumbruch
   echo "Willkommen bei PHP";
?>
```

Im Browser stehen die beiden Zeilen dann sauber untereinander:


*Abbildung 3.4: Textausgabe mit Zeilenumbruch*

Wenn Sie jetzt einen Blick in den HTML-Quelltext werfen, werden Sie aber feststellen, dass die Codeausgabe in einer einzigen Zeile erfolgt:

```
Willkommen bei PHP<br>Willkommen bei PHP
```

### 3.2.3 Escape-Sequenzen

Um den ausgegebenen HMTL-Code besser zu strukturieren, empfiehlt es sich daher, so genannte Escape-Sequenzen einzufügen. Zudem brauchen Sie Escape-Zeichen, wenn Sie innerhalb eines Strings Zeichen verwenden wollen, die sonst als Programmcode interpretiert und eine Fehlermeldung provozieren würden, wie etwa Anführungszeichen oder das Dollarsymbol.

| Escape-Sequenz | Bedeutung |
|---|---|
| \n | neue Zeile |
| \r | Wagenrücklauf |
| \t | Tabulator |
| \\ | Backslash |
| \$ | Dollarsymbol |
| \" | Anführungszeichen |
| \' | Apostroph |

*Tabelle 3.1: Escape-Sequenzen*

In diesem Fall wird die Escape-Sequenz `\n` benötigt, der Code sieht dann folgendermaßen aus:

```
<?php
   # Die folgende Anweisung gibt den Text "Willkommen bei PHP"
   # auf dem Bildschirm aus
   echo "Willkommen bei PHP";
   echo "\n<br>"; // Zeilenumbruch
   echo "Willkommen bei PHP";
?>
```

In der Bildschirmausgabe erscheint der HTML-Quelltext dann strukturiert:

```
Willkommen bei PHP
<br>Willkommen bei PHP
```

**Übung 3.2:**

Verändern Sie die Datei *hallo.php* so, dass das Wort „Welt“ in Anführungszeichen ausgegeben wird.

### 3.2.4 Anweisungen

Ein PHP-Skript ist nichts anderes als eine Reihe von Anweisungen. Der PHP-Interpreter arbeitet diese Anweisungen nacheinander ab. Jede Anweisung muss mit einem Semikolon enden. Die erste Anweisung haben Sie schon kennen gelernt: Sie haben den Parser angewiesen, mit Hilfe des Funktionsaufrufs `echo` den Text „Willkommen bei PHP“ auf den Bildschirm des Clients auszugeben.

### 3.2.5 Ausdrücke, Variablen und Konstanten

PHP wird als ausdrucksorientierte Sprache bezeichnet. Ein Ausdruck unterliegt genau wie ein grammatikalisch korrekter Satz einer gewissen Syntax und muss einen Wert haben. Dabei kann es sich auch um den Wert 0 oder eine leere Zeichenfolge "" handeln.

Beispiel für einen elementaren Ausdruck:

```
77
```

Hierbei handelt es sich um eine Integer-Konstante mit dem Wert 77.

Noch ein Beispiel:

```
$alter
```

Für sich allein gestellt ist diese Variable allerdings sinnlos. Die Kombination aus beidem ergibt einen zusammengesetzten Ausdruck:

```
$alter = 77;
```

Die Variable `$alter` besitzt nun den gleichen Wert wie die Integer-Konstante und kann im weiteren Verlauf des Scripts verwendet werden:

```
<?php
   $alter = 77;
   echo "Oma ist heute $alter Jahre alt geworden.";
?>
```

Im Browser sieht das dann so aus:

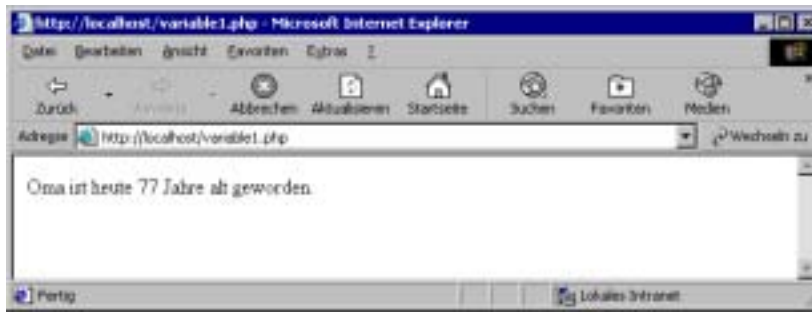


*Abbildung 3.5: Ausgabe unter Verwendung einer Variablen*

Das vorangestellte Dollarzeichen kennzeichnet eine Variable. Eine Variable ist ein Platzhalter, dem Zeichenketten, Zahlen oder ganze Texte zugewiesen werden können. Damit können Sie beispielsweise Text mehrmals verwenden, ohne ihn jedes Mal neu schreiben zu müssen. Oder Sie verarbeiten Benutzereingaben aus einem Formular, beispielsweise für eine persönliche Begrüßung oder für ein Gästebuch. Wie das geht, wird im Kapitel 4, „PHP und Formulare“, erklärt.

### 3.2.6 Erweitern von Variablen

Eine Variable lässt sich auch erweitern oder verlängern:

```
<?php
     $alter = 77;
     $alter .= " Jahre";
     echo "Oma ist heute $alter alt geworden.";
?>
```

Der Punkt vor dem Gleichheitszeichen markiert eine Erweiterung. Würde er fehlen, würde die Variable mit dem neuen Wert überschrieben, und die Textausgabe wäre dann „Oma ist heute Jahre alt geworden.“ So aber bringt die Ausgabe im Browser das gleiche Ergebnis wie vorher.

PHP kennt zwei Typen von Zahlen: Integer (Ganzzahlen) und Fließkommazahlen. Diese können positiv oder negativ sein, aber keine Brüche oder Zahlen mit mehr als einem Trennzeichen (Beispiel: Datumsformat).

Gültige Zahlenwerte:

```
1
1977
-10
1.0
19.77
-1.0
```

Ungültige Zahlenwerte:

```
1 ½
10.09.66
```

Zur Schreibweise von Variablen: Nach dem obligatorischen Dollarzeichen folgt die Variable, deren Name entweder mit einem Buchstaben oder einem Unterstrich beginnen muss, worauf eine beliebige Anzahl und Kombination von Buchstaben, Unterstrichen oder Zahlen folgen kann, nur Leerzeichen dürfen nicht enthalten sein. Achten Sie darauf, dass zwischen Groß- und Kleinschreibung unterschieden wird! Empfehlenswert ist außerdem, sprechende Namen zu wählen, die nicht miteinander verwechselt werden können.

Erlaubte Namen für Variablen:

```
$alter
$_alter
$Alter
$alter1
```

Nicht erlaubte Namen für Variablen:

```
$1alter
$1 alter
$alter 1
$ alter
```

Handelt es sich bei dem Wert um eine Zeichenkette, muss diese in Anführungszeichen gesetzt werden:

```
$gratulation = "Alles Gute zum Geburtstag!"
```

Im Programmierdeutsch spricht man nicht von Zeichenkette, sondern von String. Ein solcher String kann eine beliebige Kombination von Buchstaben, Zahlen, Leerzeichen und Symbolen enthalten, die durch Anführungszeichen (") oder Hochkommata (') eingeschlossen werden. Auch Variablennamen können Bestandteile eines Strings sein. Beachten Sie dabei, dass Variablenwerte nur bei Verwendung von Anführungszeichen ausgegeben werden.

Beispiel:

```
<?php
   $alter = 77;
   echo 'Oma ist heute $alter Jahre alt geworden.';
?>
```

In diesem Fall wird auf dem Bildschirm der Text `Oma ist heute $alter Jahre alt geworden` ausgegeben, und nicht `Oma ist heute 77 Jahre alt geworden`. Hier sehen Sie also nur den Namen der Variablen und nicht ihren Wert. Das Gleiche würde passieren, wenn Sie der Variablen ein Escape-Zeichen voranstellen:

```
<?php
   $alter = 77;
   echo "Oma ist heute \$alter Jahre alt geworden.";
?>
```

### Übung 3.3:

Erweitern Sie das Listing so, dass im Browser `Oma ist heute 77 Jahre alt geworden. Alles Gute zum Geburtstag!` zu lesen ist. Verwenden Sie dabei zwei Variablen, der Text soll außerdem in zwei Zeilen ausgegeben und die Gratulation fett formatiert werden.

Wenn Sie im weiteren Verlauf des Codes die gleiche Variable mit einem neuen Wert belegen, wird der ursprüngliche Wert überschrieben:

```
$alter = 77;
$alter = 78;
```

In diesem Fall würde dann „78“ als Wert für die Variable `$alter` ausgegeben. Umso wichtiger ist es, dass Sie darauf achten, nicht aus Versehen zwei Variablen mit demselben Namen zu erzeugen.

### 3.2.7 Vordefinierte Variablen

Ein weiterer Typ von Variablen sind die vordefinierten Variablen, die der Webserver und das PHP-Modul benutzen. Solche vordefinierten Variablen dürfen Sie nicht mit eigenen Werten belegen.

Um zu sehen, welche Variablen auf Ihrem Server benutzt werden, erstellen Sie folgendes PHP-Skript:

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

Speichern Sie das Skript unter dem Namen *phpinfo.php* ab und rufen Sie es im Browser auf. Sie erhalten daraufhin umfassende Informationen über das auf Ihrem Server installierte PHP-Modul, darunter auch über die vordefinierten Variablen.


*Abbildung 3.6: Informationen über die installierte PHP-Version*

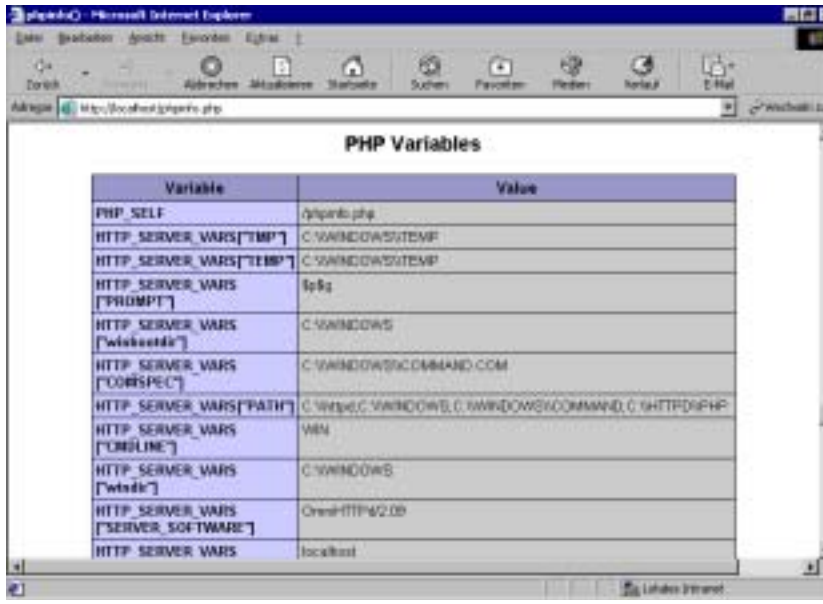


*Abbildung 3.7: Informationen über die verwendeten PHP-Variablen (mit phpinfo)*

## 3.2.8 Operatoren

Bisher haben Sie die Variablen nur ausgegeben. Um sie aber interagieren zu lassen, gibt es Operatoren. Dabei wird unterschieden zwischen arithmetischen Operatoren und String-Operatoren.

***Arithmetische Operatoren***

Arithmetische Operatoren entsprechen den Grundrechenarten und ermöglichen es, Variablenwerte zu addieren, zu subtrahieren, zu multiplizieren und zu teilen.

| Operator | Bedeutung |
|---|---|
| + | Addieren |
| - | Subtrahieren |
| * | Multiplizieren |
| / | Dividieren |
| % | Modulus (Divisionsrest) |

*Tabelle 3.2: Arithmetische Operatoren*

PHP erkennt selbstständig anhand der Anführungszeichen, die einen String einschließen, anhand eines ganzzahligen Zahlenwertes *(Integer)* oder eines Fließkommas, mit welchem Variablentyp es zu tun hat.

Beispiel:

Hier werden zwei Variablen für das aktuelle Jahr und für das Geburtsjahr definiert und den Variablen ganzzahlige Werte zugewiesen. Dann lässt sich daraus die Variable `$alter` berechnen:

```
$akt_jahr = 2002;
$geb_jahr = 1925;
$alter = $akt_jahr - $geb_jahr;
```

In den PHP-Code des Beispiels übertragen bedeutet das:

```
<?php
   $akt_jahr = 2002;
   $geb_jahr = 1925;
   $alter = $akt_jahr - $geb_jahr;
   echo "Oma ist heute $alter Jahre alt geworden.";
?>
```

Das Ergebnis auf dem Bildschirm ist das gleiche wie vorher, nur ist das kleine Programm jetzt schon wesentlich intelligenter und flexibler.

***String-Operatoren***

String-Operatoren funktionieren etwas anders.

Beispiel:

```
<?php
   $a = "Alles Gute zum Geburtstag";
   echo $a + ", liebe Oma!";   # wird nicht funktionieren
?>
```

Im Browser würde in diesem Fall `0` ausgegeben, weil PHP die String-Variablen vor der Ausgabe in einen Zahlenwert umwandelt.

Strings werden in PHP mit einem Punkt verbunden:

```
<?php
   $a = "Alles Gute zum Geburtstag";
   echo $a . ", liebe Oma!";   # wird funktionieren
?>
```

In diesem Fall werden die Strings korrekt zusammengesetzt:

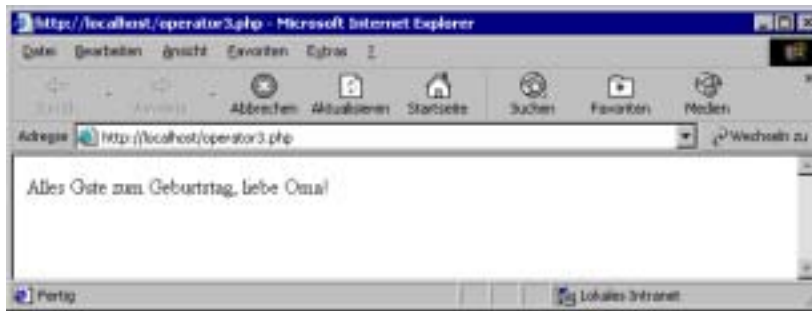


*Abbildung 3.8: Die Bildschirmausgabe bei korrekter String-Addition*

***Zuweisungs-Operatoren***

Sie haben ihn schon die ganze Zeit über eingesetzt: den zentralen Zuweisungsoperator von PHP, das Gleichheitszeichen.

Beispiel:

```
$akt_jahr = 2002
```

Auf der linken Seite des Gleichheitszeichens steht die Variable, auf der rechten Seite der Wert, der dieser Variablen zugewiesen werden soll. Dieses Gleichheitszeichen ist aber nicht im mathematischen Sinn zu verstehen! Für das mathematische „ist gleich“ wird in PHP ein doppeltes Gleichzeitszeichen == verwendet, das Sie unter der Überschrift „Vergleichsoperatoren“ noch näher kennen lernen werden.

***Inkrementieren***

In PHP gibt es noch weitere Zuweisungsoperatoren, Abkürzungen der offiziellen Syntax, die dem Programmierer das Leben erleichtern.

Um den Wert einer Variablen, zum Beispiel eines Counters, um 1 zu erhöhen, könnten Sie folgenden Code schreiben:

```
<?php
   $count1 = 0;
   $count1 = $count1 + 1;
   echo $count1;
?>
```

Genauso und wesentlich platzsparender funktioniert es aber auf diese Weise:

```
<?php
   $count1 = 0;
   $count1++;
   echo $count1;
?>
```

Das doppelte Pluszeichen wird dabei als Abkürzung für `$count1 + 1` verwendet. ++ zählt also immer um eins hoch.

***Dekrementieren***

Bei der Dekrementierung verhält es sich genau umgekehrt – statt des Plus- wird ein Minuszeichen verwendet und immer um eins tiefergezählt.

Darüberhinaus gibt es noch weitere platzsparende Möglichkeiten der Notierung, sogenannte Auslassungen:

| Langform | Kurzform |
|---|---|
| $x = $x + 2; # Erhöhung von $x um 2 | $x += 2; # Erhöhung von $x um 2 |
| $y = 5<br>$y = $y * 4; # $y hat jetzt den Wert 20 | $y *= 20; |
| $y += 1; | $y++; |

*Tabelle 3.3: Auslassungen in PHP*

### Datentypen

In PHP haben wir es mit sechs verschiedenen Datentypen zu tun, hier eine kurze Übersicht. Einige der erwähnten Datentypen lernen Sie in diesem Buch näher kennen.

| Datentyp | Beschreibung |
|---|---|
| boolean | TRUE (=1) or FALSE (=0) |
| integer | Ganzzahl |
| double | Kommazahl |
| string | Zeichenkette |
| array | Feldtyp (Liste) |
| object | definiertes Objekt |

*Tabelle 3.4: Datentypen in PHP*

### Übung 3.4:

Weisen Sie der Variablen `$count2` den Wert 10 zu und verringern Sie diese Variable um 1, so dass in der Bildschirmausgabe 9 zu lesen ist. Verwenden Sie beide Möglichkeiten.

### 3.2.9 Funktionen

In PHP gibt es einige vordefinierte Funktionen, die bestimmte Aktionen ausführen. Einige Funktionen haben Sie schon kennen gelernt:

*Funktion phpinfo()*

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

Diese Funktion liefert Informationen über die installierte PHP-Version und die vordefinierten Variablen.

*Funktionen echo und print()*

Mit `echo` und `print()` haben Sie außerdem schon zwei Ausgabefunktionen von PHP kennen gelernt. Bei beiden sind die Klammern optional und werden in der Praxis in der Regel weggelassen, insbesondere bei `echo`.

*Funktion time()*

Eine weitere wichtige Funktion ist die Datumsfunktion `time()`, die den Unix-Timestamp (Sekunden seit dem Jahresanfang 1970 Greenwich-Zeit) ermittelt und als Zahlenwert zurückgibt:

```
<?php
   $timestamp = time();
?>
```

Um das Datum auf den Bildschirm auszugeben, verwenden Sie `echo`:

```
<?php
   $timestamp = time();
   echo $timestamp;
?>
```

Wenn Sie sich das Ergebnis auf dem Bildschirm ansehen, werden Sie allerdings feststellen, dass das noch nicht allzu aussagekräftig ist:

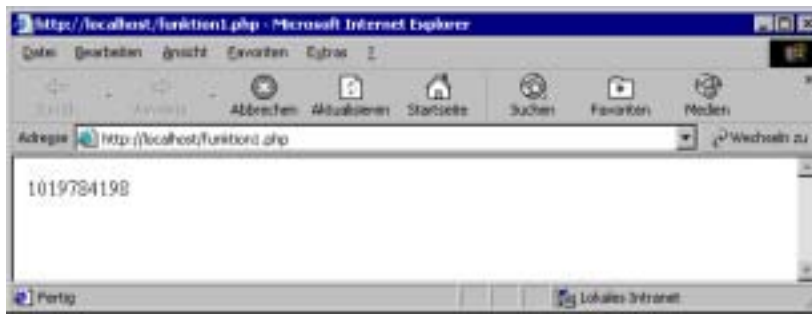


*Abbildung 3.9: Unformatierte Ausgabe des Unix-Timestamps*

Was ist passiert? Der Unix-Timestamp *(Zeitstempel)* ermittelt, wie viele Sekunden seit dem 1.1.1970 um 00:00 Uhr vergangen sind, und stellt diese Zahl dar. Um diese Zahl aber in vertrautem Datumsformat anzuzeigen, muss sie erst noch aufbereitet werden.

***Funktion date()***

Deshalb gibt es in PHP die Funktion `date()`. Die Notierung erfolgt dabei im Format `date(Format, Variablenname Timestamp)`. Der Variablenname für den Zeitstempel ist optional. Wird er weggelassen, verwendet PHP den aktuellen Zeitstempel, ansonsten wird auf die vorher bereits ermittelte Variable zurückgegriffen. Für die Formatierung gibt es umfangreiche Möglichkeiten. Einige wichtige werden hier aufgelistet:

| Format | Beschreibung | Beispiel |
|---|---|---|
| d | Tag des Monats, mit führender Null | 03, 28 |
| j | Tag des Monats, ohne führende Null | 3, 28 |
| m | Nummer des Monats, mit führender Null | 01, 17 |
| n | Nummer des Monats, ohne führende Null | 1, 17 |
| F | Monat ausgeschrieben | April |
| y | Jahr zweistellig | 66, 02 |
| Y | Jahr vierstellig | 1966, 2002 |
| H | Stunde im 24-Stunden-Format, mit führender Null | 07, 18 |
| G | Stunde im 24-Stunden-Format, ohne führende Null | 7, 18 |
| h | Stunde im 12-Stunden-Format, mit führender Null | 07, 06 |
| g | Stunde im 12-Stunden-Format, ohne führende Null | 7, 6 |
| i | Minuten, zweistellig | 08, 45 |
| s | Sekunden, zweistellig | 06, 56 |
| w | Wochentag in Zahlenwert | 2, 6 |
| D | Wochentagname abgekürzt | Fri |
| l | Wochentagname ausgeschrieben | Friday |
| r | nach RFC 822 formatiertes Datum (seit PHP 4.0.4) | Fri, 26 Apr 2002 01:43:12 +0200 |

*Tabelle 3.5: Formatierungsmöglichkeiten für date()*

Wenn Sie also ein Datum im Format `00.00.0000 - 00:00 h` ausgeben wollen, kommt folgende Schreibweise zum Einsatz:

```
<?php
   $datum = date("d.m.Y - H:i \h");
   echo $datum;
?>
```

Beachten Sie, dass Sie vor dem `h` ein Escape-Zeichen setzen müssen, ansonsten wird `h` als Formatierungsanweisung für `date` interpretiert und stattdessen die aktuelle Stunde im 12-Stunden-Format ausgegeben.

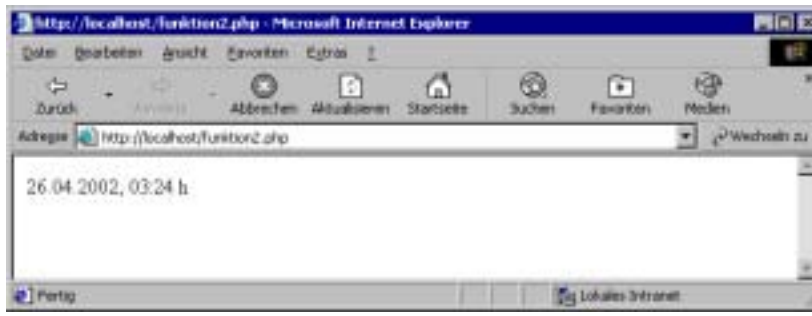


*Abbildung 3.10: Formatierte Ausgabe von Datum und Uhrzeit*

Interessante Ergebnisse erhalten Sie auch bei dieser Notierung:

```
<?php
   $datum = date("d.m.Y - H:i Uhr");
   echo $datum;
?>
```

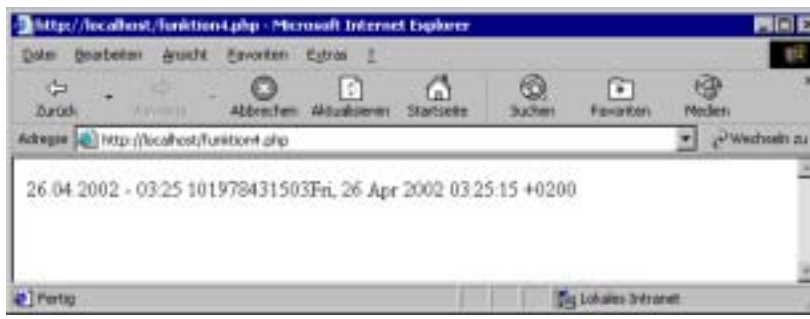


*Abbildung 3.11: Fehlerhafte Ausgabe bei nicht korrekter Notierung*

Was ist passiert? `U` wurde als Unix-Timestamp interpretiert, `h` gibt direkt danach die Stunde im 12-Stunden-Format aus, gefolgt vom Datum im RFC-Format, das mit `r` aufgerufen wird.

In diesem Fall empfiehlt es sich, das zu machen, was grundsätzlich die elegantere Methode ist, weil vielseitiger verwendbar:

```
<?php
   $timestamp = time();
   $datum = date("d.m.Y",$timestamp);
   $uhrzeit = date("H:i",$timestamp);
   echo $datum," - ",$uhrzeit," Uhr";
?>
```

Zunächst wird der aktuelle Unix-Zeitstempel ermittelt und in der Variablen `$timestamp` gespeichert. Danach werden zwei Variablen erzeugt, `$datum` und `$uhrzeit`, beide unter Verwendung der Anweisung `date()` bezogen auf den in `$timestamp` zwischengespeicherten Wert, aber mit unterschiedlichen Formatierungen. Schließlich werden diese beiden Variablen zusammen mit den „Füllwörtern“ in einem String zusammengesetzt.

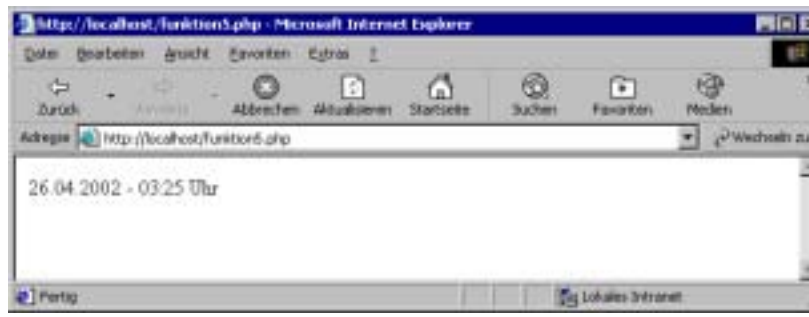


*Abbildung 3.12: Korrekte Ausgabe unter Verwendung von zwei Variablen für Datum und Zeit*

Die separat ermittelten Variablen können auf diese Weise auch getrennt voneinander weiterverwendet werden. Grundsätzlich empfiehlt es sich, so modular wie möglich zu arbeiten, das spart viel Schreibarbeit und minimiert Fehlerquellen, wenn ein Programm eines Tages vielleicht noch komplexer wird.

### Übung 3.5

Schreiben Sie ein Skript, das das aktuelle Datum in folgendem Format ausgibt: 1.1.2002, 00:00:00 h.

***Funktion mktime()***

Was aber, wenn Sie ein Datum haben und dieses Datum in eine Unix-Timestamp umwandeln wollen?

Dafür gibt es die Funktion `mktime()`. Die Werte in der Klammer werden in folgender Reihenfolge von links nach rechts notiert: Stunde, Minute, Sekunde, Monat, Tag, Jahr. Um also die Timestamp für den 24.04.2002, 20:15 h zu ermitteln, brauchen Sie folgenden Code:

```
<?php
   $timestamp = mktime(20,15,0,4,24,2002);
?>
```

Sie können das natürlich auch noch überprüfen, indem Sie die Gegenprobe machen und die so gewonnene Variable wieder in der richtigen Formatierung ausgeben:

```
<?php
   $timestamp = mktime(20,15,0,4,24,2002);
   $datum = date("d.m.Y, H:i \h",$timestamp);
   echo $datum;
?>
```

In der Bildschirmausgabe sehen Sie dann, dass der Code stimmt:

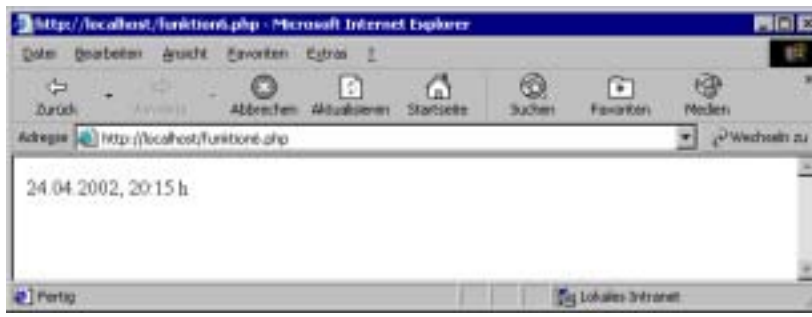


*Abbildung 3.13: Erfolgreiche Gegenprobe für mktime()*

**Funktion mail()**

Sehr nützlich ist auch die Funktion `mail()`, die aus einem PHP-Skript heraus Mails versendet. Das funktioniert allerdings nur, wenn das Skript auf einem „richtigen“ Webserver mit SMTP-Server, meist *sendmail*, läuft, *OmniHTTPd* wird hier einen Fehler auswerfen. Die Sendmail-Einstellungen können Sie mit dem Skript *phpinfo.php* überprüfen:


*Abbildung 3.14: Sendmail-Einstellungen auf dem lokalen Server(OmniHTTPd)*

Hier erkennen Sie auch, dass unter *OmniHTTPd* die notwendigen Parameter für *sendmail* nicht gesetzt sind.

Anders sieht es auf dem Linux-Webserver aus:

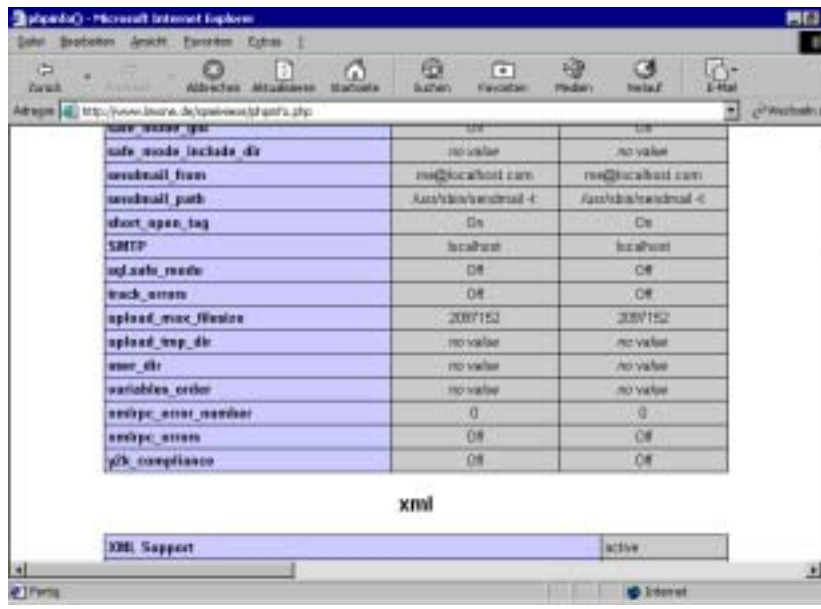


*Abbildung 3.15: Sendmail-Einstellungen auf dem Linux-Webserver beim Provider*