PHP 8 und MySQL

Das umfassende Handbuch

Kapitel 4
Grundlagen der Sprache

Dieses Kapitel beginnt mit der Syntax von PHP und zeigt Ihnen, wie Sie mit PHP programmieren. Für fortgeschrittene Leser bildet es ein gutes Nachschlagewerk, wenn einzelne Konstrukte unklar sind.

PHP ist nicht schwer zu erlernen. Dieses Versprechen steht am Anfang einer umfangreichen Spracheinführung, die alle wesentlichen Aspekte der Sprache beleuchten wird. Sie finden viele kleine, einfache Codestücke. Dadurch können Sie auch später schnell einzelne Fakten nachschlagen und so immer tiefer in PHP einsteigen.

Tipp

Sollten Sie es eilig haben und seltenere Details nicht benötigen, sondern schnell und kompakt die Sprache lernen wollen, lassen Sie beim ersten Lesen einfach Überschriften der vierten Ebene weg. Dort finden Sie meist Hintergrundinformationen zu einzelnen Themen, die aber erst im Einzelfall wirklich wichtig werden.

4.1 PHP in HTML

Eine Reise in die Tiefen und Untiefen von PHP beginnt bei HTML. PHP wurde als serverseitige Programmiersprache konzipiert, die eng in HTML integriert ist. Dies steht im Gegensatz zu dem Ziel anderer Programmiersprachen, Code und Inhalt zu trennen. Natürlich ist eine solche Trennung auch in PHP möglich, indem Sie Code in eine externe PHP-Datei einschließen.[1] Häufiger aber wird der PHP-Code direkt in die HTML-Datei eingefügt. Die Datei erhält dabei die Endung *.php*.[2]

PHP-Anweisungen können in diese Dateien auf verschiedene Arten eingebunden werden:

```
<?php
  //Code
?>
```

1 Siehe Abschnitt 4.1.3, »Externe Datei«.

2 Ab und an finden sich auch noch Versionsnummern in den Dateiendungen, z. B. *.php5* für Dateien, die mit PHP 5 geschrieben wurden.

- Dies ist die Standardvariante, PHP-Code einzubinden. Auch Großschreibung ist erlaubt: `<?PHP`. Außerdem wird in vielen modernen Projekten (unter anderem auch bei Frameworks wie *Zend Framework* und *Symfony*) das schließende Element `?>` in Dateien weggelassen, die nur PHP-Code enthalten (also kein HTML), da dahinter aus Versehen noch Whitespace folgen kann, der dann zu einer Fehlermeldung in Form eines `Cannot modify header`-Fehlers führt.

  ```
  <?  //Code  ?>
  ```

  Ein wenig kürzer geht es, wenn Sie `php` einfach weglassen und nur spitze Klammern und Fragezeichen verwenden. Allerdings ist diese Variante nicht XML-konform und kann in der *php.ini* über die Option `short_open_tag = Off` ausgeschaltet werden. Standardmäßig steht hier zwar `On`, aber dennoch sollten Sie sich nicht darauf verlassen. Deswegen raten wir vom Einsatz dieser Variante ab.

  ```
  <%
    //Code
  %>
  ```

- Diese Form entsprach ASP (*Active Server Pages*), der – inzwischen veralteten[3] – serverseitigen Programmiertechnologie von Microsoft. Diese Variante gibt es seit PHP 7 nicht mehr, ab dieser Version wirft die Variante einen Syntaxfehler. In älteren Versionen steht die Funktionalität noch zur Verfügung. Dafür müssen Sie den Eintrag `asp_tags` in der Konfigurationsdatei *php.ini*[4] auf `On` setzen. Allerdings raten wir vom Einsatz natürlich ab.

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

- Die letzte Form war schon immer in der Praxis ungebräuchlich, da sie sehr viel Tipparbeit bedeutet. Deswegen wurde sie ebenfalls in PHP 7 abgeschafft.

Allen Arten gemeinsam ist, dass es sich um PHP-Anweisungsblöcke handelt. Sie können beliebig viele PHP-Blöcke in eine HTML-Seite einbauen.

Hinweis

Wenn in einer PHP-Seite keine PHP-Anweisungen gefunden werden, gibt der PHP-Interpreter einfach den HTML-Code aus.

3 Abgelöst durch ASP.NET. Dort wird Code anders eingebunden.

4 Mehr zur Konfiguration von PHP finden Sie in Kapitel 34, »Konfigurationsmöglichkeiten in der »php.ini««.

4.1.1 Kommentare

Ein Kommentar ist Text im Quellcode, der vom PHP-Interpreter nicht ausgeführt wird. Kommentare dienen in der Praxis dazu, Teile des Codes vernünftig zu erklären oder sonstige Informationen mitzuliefern. PHP verwendet eine Syntax für Kommentare, die Sie vielleicht schon aus JavaScript oder anderen Sprachen kennen:

```
// Kommentar
```

steht für einen einzeiligen Kommentar. Alle Zeichen nach // sind auskommentiert.

```
# Kommentar
```

steht ebenfalls für einen einzeiligen Kommentar.

```
/* Mehrzeiliger
Kommentar */
```

kommentiert einen Block zwischen /* und */ aus, der sich auch über mehrere Zeilen erstrecken darf.

Hinweis

In der Praxis häufig im Einsatz ist eine Variante mit zwei Sternchen beim öffnenden Kommentarzeichen: /** … */. Hierbei handelt es sich im Grunde um normale PHP-Kommentare, die aber besonders für *phpDoc* markiert sind (*https://phpdoc.org*). Mit phpDoc erstellen Sie Kommentare, die automatisch in eine API-Dokumentation überführt werden.

Tipp

Kommentieren Sie Ihren Code sinnvoll und verständlich. Denken Sie einfach an den armen Kollegen, der daran weiterarbeiten muss, oder an sich selbst: Nach einigen Jahren werden Sie vergessen haben, worum es sich bei dem Skript handeln sollte. In beiden Fällen werden Sie einen Menschen mit guten Kommentaren glücklich machen!

4.1.2 Anweisungen

Alle Zeichen innerhalb eines PHP-Anweisungsblocks, die nicht auskommentiert sind, bilden zusammen den PHP-Code, den der PHP-Interpreter ausführt. Jede Zeile in PHP, die eine Anweisung enthält, wird mit einem Strichpunkt beendet:

```
<?php
  echo "Text";
?>
```

gibt beispielsweise einen Text aus.

Hinweis

Zur Anweisung gehört auch der Begriff *Ausdruck* (engl. *Expression*). In PHP ist alles ein Ausdruck, was einen Wert besitzt. Die meisten Anweisungen sind insofern ebenfalls Ausdrücke. Diese Definition ist allerdings eher akademisch und für Ihre praktische Arbeit wohl nur selten relevant.

4.1.3 Externe Datei

Die Trennung von Code und Inhalt gehört zwar nicht zu den ursprünglichen Intentionen von PHP, ist aber über externe Dateien zu realisieren.[5] Auch sonst sind externe Dateien praktisch. Sie erlauben auch, häufig verwendete Codestücke auszulagern.

Zum Einbinden von externen Dateien verwenden Sie die Anweisungen `include()` und `require()`. Funktional unterscheiden sich beide beim Fehlerhandling. `include()` produziert nur eine Warnung (`E_WARNING`), wenn beispielsweise die externe Datei nicht gefunden wird, `require()` liefert einen Fehler (`E_ERROR`). Dies ist vor allem beim Fehlerhandling und bei den Konfigurationseinstellungen für Fehlertoleranz in der *php.ini* wichtig.[6]

Ein einfaches Beispiel illustriert die Funktionsweise der beiden Anweisungen. Die externe Datei enthält eine Ausgabe mit der `echo`-Anweisung:

```
<?php
   echo "Externe PHP-Datei!";
?>
```

Listing 4.1 Die externe Datei gibt einen Text aus (»extern.php«).

5 Beim Vergleich serverseitiger Technologien ist die Trennung von Code und Inhalt – eine Form des modularen Programmierens – eine wichtige Forderung, die beispielsweise ASP.NET sehr gut erfüllt. Allerdings muss man bedenken, dass PHP ursprünglich gerade in der engen Integration von PHP-Code und HTML-Code einen Vorteil gegenüber dem damaligen Marktführer Perl hatte. Dank externer Dateien können Sie mit PHP mittlerweile aber sowohl »getrennt« als auch »integriert« programmieren, sodass bei sauberer Programmierung kein Unterschied mehr besteht.

6 Mehr hierzu lesen Sie in Kapitel 34, »Konfigurationsmöglichkeiten in der »php.ini««. Im Testbetrieb sollten Sie `error_reporting` in der *php.ini* immer auf `E_ALL` belassen, damit alle Fehlermeldungen angezeigt werden und Sie Probleme schnell erkennen können. Außerdem sollten Sie in der Entwicklungs- und Testumgebung die Fehlermeldungen mit `display_errors=On` einblenden.

Tipp

PHP-Code muss ganz normal in einen PHP-Anweisungsblock eingeschlossen werden. Zusätzlich kann die externe Datei HTML-Quellcode enthalten. Wenn der PHP-Interpreter eine externe Datei aufruft, liest er das HTML ein und interpretiert die PHP-Blöcke (siehe Abbildung 4.1).

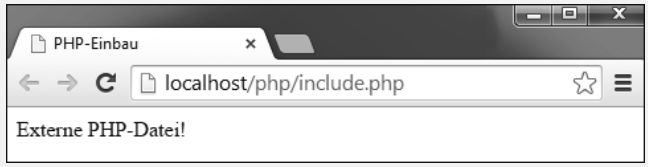

Abbildung 4.1 Der Inhalt der externen Datei wird ausgegeben.

Diese Datei wird dann mit `include()` in eine Datei eingebaut:

```
<html>
  <head>
    <title>PHP-Einbau</title>
  </head>
  <body>
    <?php
      include "extern.php";
    ?>
  </body>
</html>
```

Listing 4.2 »include()« bindet die externe Datei ein (»include.php«).

Befindet sich die Datei nicht im selben Verzeichnis bzw. nicht in einem Verzeichnis, das per `include_path`-Direktive in der *php.ini* angegeben ist, müssen Sie den vollständigen Pfad zur Datei angeben.

Hinweis

Windows unterscheidet bei Dateinamen nicht zwischen Groß- und Kleinschreibung. Insofern unterscheiden auch die Befehle zum Einbinden externer Dateien unter Windows z. B. nicht zwischen *extern.php* und *Extern.php*.

Die Syntax mit `require()` sieht genauso aus:

```
require "extern.php";
```

Hinweis

Anweisungen[7] sind von PHP angebotene Sprachkonstrukte, um ein bestimmtes Ziel zu erreichen. Die Parameter für Anweisungen werden in Anführungszeichen nach der Anweisung geschrieben. Alternativ ist hier auch eine Syntax mit runden Klammern möglich:

```
require("extern.php");
```

»include_once« und »require_once«

Neben `include()` und `require()` gibt es noch `include_once()` und `require_once()`. Diese beiden Sprachkonstrukte prüfen zuerst, ob die Datei bereits eingefügt wurde. Sollte sie schon eingebunden worden sein, geschieht dies nicht noch einmal.

Dieses Verhalten ist dann wünschenswert, wenn Ihr Skript wirklich Gefahr läuft, eine Datei mehrmals einzulesen. Dann kann es nämlich sein, dass bestehende Variablenwerte oder Funktionen erneut überschrieben werden bzw. bei Funktionen ein Fehler erscheint, da sie im selben Kontext immer nur einmal deklariert werden können.

Der Einsatz von `include_once()` und `require_once()` erfolgt genau wie der von `include()` und `require()`:

```
include_once "extern.php";
```

bzw.:

```
require_once "extern.php";
```

Rückgabewert

Liefert das Skript in der externen Datei einen Rückgabewert mit `return`[8], kann dieser auch in einer Variablen[9] gespeichert werden:

```
$wert = require("extern.php");
```

Besonderheiten in »if«-Anweisungen und Schleifen

Wird eine `include()`- oder `require()`-Anweisung in anderen Anweisungen wie `if`-Bedingungen oder Schleifen[10] eingebettet, *muss* diese Anweisung geschweifte Klammern besitzen, also ein abgeschlossener Block sein. Die Kurzform:

7 Hier ist die Nomenklatur nicht eindeutig. Eine Zeile in PHP, die mit einem Strichpunkt endet, heißt ebenfalls *Anweisung*. Sie enthält sogar meist ein PHP-Sprachkonstrukt, also eine Anweisung im engeren Sinne (alternativ: Befehl). Die Unterscheidung der Begriffe ist allerdings eher akademischer Natur und hat auf die Praxis keine Auswirkungen.

8 Siehe Kapitel 6, »Funktionen und Sprachkonstrukte«.

9 Siehe Abschnitt 1.3, »Das Konzept von PHP«.

10 Mehr Details dazu in Kapitel 5, »Programmieren«.

```
if (Bedingung)
  include "extern.php";
else
  include "extern2.php";
```

ist also nicht erlaubt, funktioniert allerdings in manchen PHP-Versionen dennoch. Korrekt ist:

```
if (Bedingung) {
  include "extern.php";
}
else {
  include "extern2.php";
}
```

Dateien über das Netzwerk

Wenn Sie Dateien über das Netzwerk mit absoluter URL öffnen möchten, muss in der *php.ini*-Konfigurationsdatei die Einstellung `allow_url_fopen` aktiviert sein.[11]

```
allow_url_fopen = On
```

»include_path«

In der *php.ini* findet sich noch eine zweite interessante Einstellung: Unter `include_path` legen Sie beliebige Pfade fest, in denen `include()`- und `require()`-Anweisungen automatisch nachsehen. Mehrere Pfade werden unter Linux mit Doppelpunkt, unter Windows mit Strichpunkt getrennt. Hier sehen Sie die Linux-Variante:

```
include_path = ".:/php/includes"
```

Und hier die Windows-Variante:

```
include_path = ".;c:\php\includes"
```

Die Konstante `PATH_SEPARATOR` enthält das Trennzeichen je nach Betriebssystem. Damit müssen Sie sich also nicht um dieses Detail kümmern, sondern schreiben einfach:

```
include_path = "." . PATH_SEPARATOR . "c:\php\includes"
```

Sie können die Einstellung `include_path` auch für das aktuelle Skript ändern. Dazu gibt es zwei verschiedene Wege:

- die Funktion `set_include_path()`:

  ```
  set_include_path("/includes");
  ```

- die Funktion `ini_set()`, um jede beliebige Einstellung der *php.ini* zu ändern:

  ```
  ini_set("include_path", "/includes");
  ```

11 Siehe hierzu auch Kapitel 32, »Sicherheit«.

4.2 Ausgabe mit PHP

Um richtig in PHP einzusteigen, müssen Sie testen können, wie die Syntax und die Programmierkonstrukte funktionieren. Dazu sollten Sie Daten ausgeben können. PHP besitzt zwei Sprachkonstrukte[12] für die Ausgabe:

- die `echo`-Anweisung:

```
<?php
  echo "Ausgabe";
?>
```

- die `print`-Anweisung:

```
<?php
  print "Ausgabe";
?>
```

Die beiden Anweisungen unterscheiden sich dadurch, dass `echo` einfach nur das Übergebene ausgibt, `print` dagegen einen Rückgabewert liefert.[13]

Dieser Rückgabewert kann in eine Variable (siehe Abschnitt 4.3, »Variablen«) gespeichert werden. Er beträgt `1`, wenn die Ausgabe funktioniert hat.

```
<?php
  $t = print "Ausgabe";
  echo $t;
?>
```

Listing 4.3 Rückgabewert von »print« (»print.php«)

Dieses Listing gibt

```
Ausgabe1
```

aus. In der Praxis kommt der Rückgabewert recht selten zum Einsatz.

Kurzfassung

Noch kürzer geht es, wenn Sie nur ein Gleichheitszeichen direkt nach dem Beginn des PHP-Blocks angeben:

```
<?="Kurze Ausgabe"?>
```

12 Ein *Sprachkonstrukt* (engl. *Statement*) ist eine Anweisung von PHP. In diesem Buch unterscheiden wir zwischen Sprachkonstrukten (synonym: Anweisungen, Sprachanweisungen) und Funktionen. Mehr hierzu folgt in Kapitel 6, »Funktionen und Sprachkonstrukte«.

13 Dieser Unterschied rührt daher, dass `print` eigentlich ein Operator ist. Lesen Sie hierzu auch den Abschnitt »print« in Abschnitt 5.1.5, »Operatoren, die aus der Reihe tanzen«.

Tipp

Bis zur PHP-Version 5.3 musste `short_open_tags` auf `on` gesetzt sein, um die Kurzform zu verwenden. Seit Version 5.4 ist `<?=` auch verfügbar, wenn `short_open_tags` deaktiviert ist.

4.2.1 Anführungszeichen

Da die Ausgabe in Anführungszeichen erfolgt,[14] ist die Frage, wie Anführungszeichen in der Zeichenkette behandelt werden. PHP erlaubt einfache und doppelte Anführungszeichen, um Ausgaben (respektive Zeichenketten) zu begrenzen.

Sie können also

```
echo "Ausgabe";
```

oder

```
echo 'Ausgabe';
```

schreiben.

Um doppelte oder einfache Anführungszeichen zu verwenden, müssen Sie die jeweils andere Anführungszeichen-Art einsetzen, um die Ausgabe zu begrenzen:

```
echo 'Er sagte: "Ich denke, also bin ich!"';
```

Die zugehörige Ausgabe sehen Sie in Abbildung 4.2.

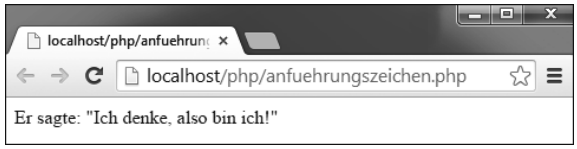

Abbildung 4.2 Anführungszeichen in der Ausgabe

Wenn Sie einfache und doppelte Anführungszeichen in einem String verwenden möchten, müssen Sie die jeweiligen Anführungszeichen per Backslash entwerten:

```
echo 'McDonald\'s-Esser: "Ich liebe nichts!"';
```

Mehr zum Entwerten lesen Sie in Abschnitt 4.3.4, »Variablen ausgeben«.

14 Sie ist eine Zeichenkette (auch String). Mehr dazu im nächsten Abschnitt.

4.3 Variablen

Eine Variable speichert einen Wert. Dieser Wert kann im Laufe eines Skripts geändert werden, er ist also variabel. Dieses Verhalten gibt der Variablen ihren Namen.

In PHP beginnen alle Variablen mit dem Dollarzeichen (`$`).[15] Im Gegensatz zu anderen Programmiersprachen fordert PHP nicht, dass eine Variable beim ersten Auftreten deklariert wird. Allerdings müssen Sie einer Variablen natürlich einen Wert zuweisen. Dies geht mit dem Gleichheitszeichen (`=`), dem sogenannten *Zuweisungsoperator*:

```
$text = "Wert";
```

weist also der Variablen `text` eine Zeichenkette mit dem Inhalt `"Wert"` zu.

4.3.1 Datentypen

Zeichenketten werden immer in Anführungszeichen geschrieben und heißen auch Strings. Zeichenketten sind allerdings nicht die einzigen Datentypen, die eine Variable annehmen kann. PHP unterscheidet außerdem noch folgende Datentypen:

- Integer (`integer` und `int`) sind ganze Zahlen.

  ```
  $zahl = 8;
  ```

- Double ist der Datentyp für Fließkommazahlen. In Double sind auch die ganzen Zahlen enthalten.

  ```
  $kommazahl = 8.4;
  ```

Hinweis

Beachten Sie, dass in PHP Kommazahlen immer mit Dezimalpunkt statt mit dem deutschen Komma geschrieben werden. Da das Komma in der Sprachsyntax eine völlig andere Bedeutung hat, kommt es in der Praxis meist zu einer Fehlermeldung.

- Real ist eine andere Bezeichnung für Double.
- Boolean (`boolean` oder `bool`) steht für einen Wahrheitswert. Ein Boolean hat nur die Werte `true` (wahr) oder `false` (falsch). Wahrheitswerte sind beispielsweise die Ergebnisse von Bedingungen und Überprüfungen.

  ```
  $wahr = true;
  ```

15 Diese Syntax lehnt sich an Perl (*Practical Extraction and Report Language*) an, eine sehr mächtige, aber teilweise auch recht komplizierte Skriptsprache. Insgesamt macht die Syntax von PHP viele Anleihen bei Perl und übernimmt beispielsweise auch die regulären Ausdrücke.

- `Object` steht für ein Objekt in PHP. Nähere Informationen hierzu erfahren Sie in Kapitel 11, »Objektorientiert programmieren«.
- Arrays können mehrere Werte speichern und sind für die Programmierung sehr wichtig. Mehr zu Arrays lesen Sie in Kapitel 8, »Arrays«.
- `Resource` ist ein intern von PHP verwendeter Datentyp, in dem beispielsweise Zugriffe auf Datenquellen gespeichert werden.
- `NULL` steht für keinen Wert, ist aber selbst auch ein Datentyp.

In den meisten Fällen müssen Sie sich nicht um den Datentyp kümmern, da PHP den Datentyp des Werts einer Variablen automatisch feststellt und ihn umwandelt, wenn er sich ändert. Die automatische Typkonvertierung funktioniert allerdings nicht immer wie erwartet und/oder erwünscht. Deswegen zeigen die nächsten beiden Unterabschnitte zuerst, wie Sie den Datentyp einer Variablen feststellen, und dann, wie Sie den Typ ändern.

Tipp

Sollten Sie schnell in PHP einsteigen wollen, überblättern Sie diese Abschnitte einfach, und lesen Sie sie später nach, wenn Sie sie für Ihre Anwendung benötigen.

Datentyp feststellen

Mit der Funktion `gettype(Variable)` können Sie den Datentyp einer Variablen herausfinden. Sie erhalten als Rückgabe den Datentyp in langer Form, also z. B. `boolean` statt `bool`.

```
<?php
  $a = "Text";
  echo gettype($a);
?>
```

Listing 4.4 Den Datentyp feststellen (»datentyp.php«)

Hinweis

Neben der allgemeinen Funktion `gettype()` gibt es noch viele einzelne Funktionen, die auf jeweils einen bestimmten Datentyp testen. `is_bool()` prüft auf Boolean, `is_string()` auf String, `is_numeric()` darauf, ob es sich um eine Zahl handelt, usw. Der Rückgabewert ist jeweils ein Wahrheitswert: `true`, wenn der Datentyp vorliegt, `false`, wenn nicht.

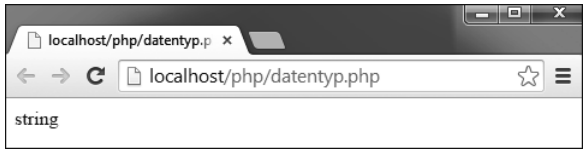

Abbildung 4.3 Die Variable hat den Datentyp »String«.

Typkonvertierung

Normalerweise müssen Sie sich in PHP um die Typkonvertierung nicht kümmern. Das folgende Skript würde in vielen Programmiersprachen die Zahl an den String anhängen. Da PHP allerdings für das Verbinden von Strings einen eigenen Operator, den Punkt (`.`), verwendet, funktioniert hier die Typkonvertierung richtig:

```
<?php
  $a = "3";
  $b = 5;
  $erg = $a + $b;
  echo $erg;
?>
```

Listing 4.5 Automatische Typkonvertierung (»typkonvertierung_auto.php«)

Das Ergebnis der Berechnung ist also:

```
8
```

Wenn Sie eine Typkonvertierung doch einmal benötigen, finden Sie in PHP die von C bekannte Typkonvertierung (engl. *Type Casting*). Sie schreiben den Datentyp (in Kurz- oder Langform) vor die Variable, die umgewandelt werden soll.

```
<?php
  $a = "true";
  $b = (bool) $a;
  echo $b;
?>
```

Listing 4.6 Typkonvertierung mit PHP (»typkonvertierung.php«)

Die Ausgabe des obigen Skripts ist der Wahrheitswert `1`, der für `true` steht:

```
1
```

Alternativ zur Konvertierung mit dem Datentyp vor der Variablen können Sie auch die Funktion `settype(Variable, Datentyp)` einsetzen. Der Datentyp wird dabei als String übergeben:

```
<?php
  $a = "true";
  $b = settype($a, "boolean");
  echo $b;
?>
```

Listing 4.7 »settype()« (»settype.php«)

Als Ausgabe erhalten Sie wie bei der Konvertierung oben die `1`.

4.3.2 Benennung

Der Name einer Variablen darf in PHP nur aus Buchstaben, Ziffern und Unterstrichen (_) bestehen. Beginnen darf er nur mit Buchstaben oder einem Unterstrich, *nicht* aber mit einer Ziffer.

Trotz dieser Einschränkungen gehört PHP bei Variablennamen zu den liberalsten Programmiersprachen: Die Namen von Sprachkonstrukten und Anweisungen wie `echo` oder `if` können als Variablennamen verwendet werden.[16]

```
$echo = "Wert";
echo $echo;
```

Obiger Code gibt `Wert` aus.

Dass etwas möglich ist, heißt natürlich nicht, dass man es auch verwenden sollte. Und so lässt der gute Programmierer von solchen »Experimenten« lieber die Finger. Auch sollten Sie Variablen immer aussagekräftig benennen. Eine Variable muss nicht nur aus drei Zeichen bestehen, und Durchnummerieren ist meist sehr unübersichtlich, vor allem wenn Sie ein Skript nachträglich erweitern.

Sie sollten die Namenskonventionen für Variablen in einem Projekt immer vorher festlegen. Hier einige Vorschläge:

- Bei zusammengesetzten Namen können Sie die einzelnen Wörter mit einem Unterstrich (_) trennen:

  ```
  $wert_links = 5;
  ```

- oder das neue Wort mit einem großen Anfangsbuchstaben beginnen:[17]

  ```
  $wertLinks = 5;
  ```

16 Diese Namen heißen auch *Schlüsselwörter*. In den meisten Programmiersprachen lassen sich Schlüsselwörter nicht als Variablennamen verwenden.

17 Dieses Verfahren heißt auch *Camel Case*, benannt nach den Höckern eines Kamels.

- Alternativ lassen Sie jedes Wort mit einem großen Anfangsbuchstaben beginnen:[18]

```
$WertLinks = 5;
```

4.3.3 Variable Variablen

Das Konzept der variablen Variablennamen funktioniert so: Sie weisen einer Variablen einen String zu. Diese Variable können Sie nun als Namen für eine weitere Variable festlegen. Dadurch wird eine Variable erzeugt, die als Namen den String der ersten Variablen und als Wert den Wert der zweiten Variablen besitzt (siehe Abbildung 4.4):

```
<?php
  $a = "text";
  $$a = "Text für die Ausgabe";
  echo $text;
?>
```

Listing 4.8 Der Variablenname als Variable (»variable_variablen.php«)

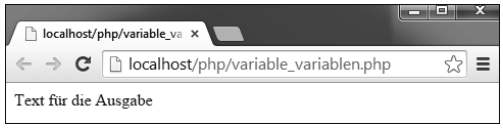

Abbildung 4.4 Der Text wird korrekt ausgegeben.

Hinweis

Das Zusammensetzen von Variablennamen ergibt vor allem dann Sinn, wenn Sie den Variablennamen dynamisch erzeugen möchten.

4.3.4 Variablen ausgeben

In den bisher gezeigten Beispielen wird oft der Wert einer Variablen mit `echo` (oder alternativ `print`) ausgegeben. Dies funktioniert problemlos:

18 Diese Variante wird nach der Programmiersprache Pascal auch *Pascal Case* genannt, oder, in PHP gebräuchlicher, *studlyCaps*. Pascal wiederum ist nach dem Mathematiker Blaise Pascal benannt.

```
$text = "Hallo PHP 8";
echo $text;
```

Die obigen Zeilen geben also Folgendes aus:

```
Hallo PHP 8
```

Sie können eine Variable allerdings auch in einer Zeichenkette ausgeben:

```
$text = "Hallo";
echo "$text PHP 8";
```

Diese zwei Zeilen produzieren als Ausgabe ebenfalls:

```
Hallo PHP 8
```

Das funktioniert allerdings nur, wenn Sie doppelte Anführungszeichen verwenden. Bei einfachen Anführungszeichen wird dagegen die Variable nicht eingebunden (siehe Abbildung 4.5):

```
$a = "Hallo";
echo '$a PHP 8';
```

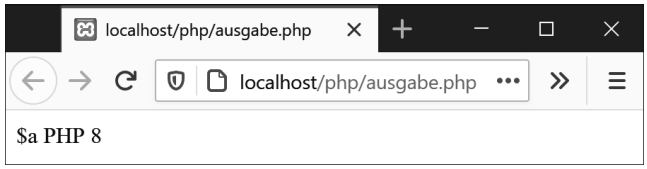

Abbildung 4.5 Der Variablenname wird ausgegeben, da einfache Anführungszeichen zum Einsatz kommen.

Diese Unterscheidung zwischen doppelten und einfachen Anführungszeichen ist nicht nur beim Einsatz von Variablen relevant, sondern auch bei Escape-Sequenzen. Bei einer Escape-Sequenz wird ein Zeichen mittels des Backslashs (\) entwertet, bzw. die Escape-Sequenz erzeugt eine bestimmte Wirkung. Die Unterscheidung zwischen doppelten und einfachen Anführungszeichen ist bei Escape-Sequenzen sehr einfach:

- Bei einfachen Anführungszeichen können Sie nur einfache Anführungszeichen und bei Bedarf den Backslash entwerten, wie Sie bereits gesehen haben:

  ```
  echo 'McDonalds\'-Esser: "Ich liebe nichts!"';
  ```

  Wenn Sie eine andere Escape-Sequenz einsetzen, wird diese nicht ausgeführt, sondern inklusive Backslash ausgegeben.

- Bei doppelten Anführungszeichen können Sie einfache Anführungszeichen sowieso verwenden, doppelte entwerten und zusätzlich einige Escape-Sequenzen einsetzen:

```
$version = "PHP 8";
echo "Die Variable \$version hat den Wert:\n $version";
```

Wenn Sie die Ausgabe des Beispiels betrachten, sehen Sie, dass `\n` nicht ausgegeben wird. Dies liegt daran, dass `\n` nur einen Umbruch im Quellcode und nicht in HTML erzeugt. Wenn Sie im Webbrowser den Quellcode ansehen, erkennen Sie den Zeilenumbruch:

```
Die Variable $version hat den Wert:
PHP 8
```

Escape-Sequenz	Beschreibung
\\	Gibt einen Backslash aus. Selbiges erreichen Sie, wenn Sie nur einen Backslash ohne Escape-Stringfolge danach ausgeben.
\"	Doppelte Anführungszeichen.
$	Dollarzeichen.
\n	Zeilenumbruch (ASCII 10), allerdings nicht in HTML. Hierfür benötigen Sie das HTML-Tag ` `.
\r	Wagenrücklauf (ASCII 13).
\t	Tabulator (ASCII 9).
\u	Beliebiges Unicode-Zeichen in hexadezimaler Form** (UTF-8).
\000	Ein bis drei Ziffern stellen eine Zahl in oktaler Notation dar.* Das entsprechende Zeichen wird dann ausgegeben.
\x00	Ein x und ein oder zwei Ziffern bilden eine Zahl in hexadezimaler Notation.**

*) Oktale Notation: Basis des oktalen Systems ist die 8. Alle Ziffern gehen von 0 bis 7. Die Umrechnung erfolgt so: Aus 245 wird 2 × 64 + 4 × 8 + 5, und das ergibt 165.

**) Hexadezimale Notation: Das hexadezimale System schreibt Zahlen auf der Basis von 16. Deswegen gibt es 16 Ziffern, nämlich die von 0 bis 9 sowie die Buchstaben A bis F. Eine hexadezimale Zahl aus zwei Ziffern rechnen Sie so um: Die erste Ziffer multiplizieren Sie mit 16 und addieren zum Ergebnis die zweite. Hexadezimale Zahlen kommen beispielsweise zur Farbnotation in HTML zum Einsatz.

Tabelle 4.1 Escape-Sequenzen für doppelte Anführungszeichen

4.3.5 Nützliches und Hilfreiches

In diesem Abschnitt sind Informationen versammelt, die Sie zum Arbeiten mit PHP nicht unbedingt brauchen, die aber für fortgeschrittene Aufgaben durchaus nützlich sind.

»isset()«

Die Hilfsfunktion `isset(Variable)` prüft, ob eine Variable existiert. Sie liefert als Ergebnis einen Wahrheitswert. Da es wenig spannend wäre, diesen Wahrheitswert einfach nur auszugeben, greifen wir ein wenig vor und zeigen bereits eine Fallunterscheidung, die erst im nächsten Kapitel genauer besprochen wird.

Das folgende Skript überprüft, ob eine Variable existiert. Wenn ja, wird sie ausgegeben. Ansonsten erscheint eine Alternativmeldung.

```
<?php
  $test = "Textvariable";
  if (isset($test)) {
    echo $test;
  } else {
    echo "Variable nicht gesetzt";
  }
?>
```

Listing 4.9 »isset()« (»isset.php«)

Im obigen Beispiel ist die Variable gesetzt und wird deswegen ausgegeben. Was aber, wenn Sie der Variablen gar keinen Wert zuweisen?

```
$test;
if (isset($test)) {
  echo $test;
} else {
  echo "Variable nicht gesetzt";
}
```

In diesem Fall wird der Alternativtext `Variable nicht gesetzt` ausgegeben.

> **Hinweis**
>
> `isset()` liefert auch `false`, wenn eine Variable den Wert `NULL` (kein Wert) hat.

»empty()«

Einen ähnlichen Test wie `isset()` führt `empty()` durch. `empty(Variable)` prüft, ob eine Variable leer ist (siehe Abbildung 4.6). Eine leere Variable ist allerdings auch ein leerer String oder `0`. Hierin liegt der Unterschied zu `isset()`.

```
<?php
  $test = "";
  if (empty($test)) {
    echo "Variable ist leer";
  } else {
    echo $test;
  }
?>
```

Listing 4.10 »empty()« (»empty.php«)

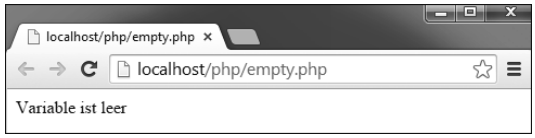

Abbildung 4.6 Hier liefert »empty()« »true«, da der String leer ist.

Hinweis

In der PHP-Dokumentation finden Sie eine recht interessante Vergleichstabelle der verschiedenen Testfunktionen (*www.php.net/manual/de/types.comparisons.php*, siehe Abbildung 4.7).

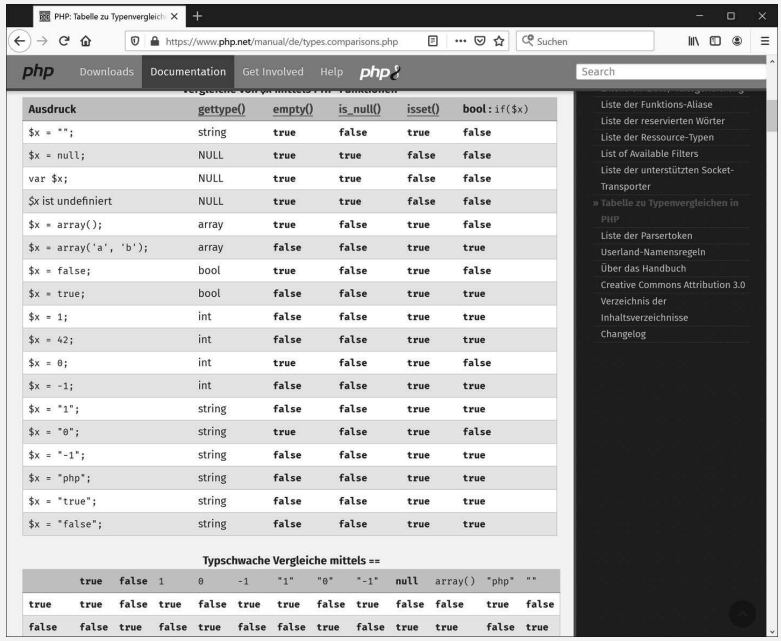

Abbildung 4.7 Der Vergleich der verschiedenen Funktionen ist sehr aufschlussreich, wenn Sie ein spezifisches Vergleichsproblem haben.

»is_null()«

Die Funktion `is_null(Variable)` gehört ebenfalls in die Riege der Hilfs- und Testfunktionen. Sie testet, ob eine Variable den Wert `NULL` (kein Wert) besitzt.

```
<?php
  $test = null;
  if (is_null($test)) {
    echo "Variable ist NULL";
  } else {
    echo "Variable ist nicht NULL, sondern" . $test;
  }
?>
```

Listing 4.11 »is_null()« (»is_null.php«)

Im obigen Fall ist die getestete Variable `null`, und deswegen wird Folgendes ausgegeben:

```
Variable ist NULL
```

> **Hinweis**
>
> Die Schreibweise der Funktionen in PHP ist leider teilweise etwas uneinheitlich. `isset()` wird zusammengeschrieben, `is_null()` hingegen mit Unterstrich. Dies hat historische Gründe: Die Funktionen wurden einfach irgendwann so genannt und konnten dann – um die Abwärtskompatibilität der Skripte zu erhalten – nicht mehr umbenannt werden.

»unset()«

Das Sprachkonstrukt `unset(Variable)` löscht eine Variable. Sie benötigen diese Funktion beispielsweise, wenn Sie bewusst im Hauptspeicher Platz schaffen möchten.

```
<?php
  $test = "Eine Variable.";
  echo $test;
  unset($test);
  echo $test;
?>
```

Listing 4.12 »unset()« (»unset.php«)

Dieses Beispiel gibt nur einmal den Text `Eine Variable.` aus. Bei der zweiten Ausgabe existiert die Variable schon nicht mehr. Hier zeigt PHP als Fehlermeldung die Notice `Undefined variable`.

Hinweis

Wenn Sie einen Parameter per Referenz an eine Funktion übergeben (siehe Kapitel 6, »Funktionen und Sprachkonstrukte«), wird mit `unset()` nur die lokale Variable gelöscht, nicht aber das Original, auf das die Referenz verweist.

Referenzen

Normalerweise hat eine Variable genau einen Wert. Der Wert der Variablen wird vom PHP-Interpreter im Hauptspeicher gespeichert. Sie können allerdings auch mehrere Variablen auf einen Wert verweisen lassen. Das funktioniert mit dem Et-Zeichen, das oft auch *kaufmännisches Und* oder, auf Englisch, *Ampersand* genannt wird (&). Und so geht es: Sie erstellen eine Variable und weisen dann mithilfe des Et-Zeichens einer anderen Variablen die Referenz auf diese Variable zu:

```
<?php
  $a = "Eine Variable";
  $b = &$a;
  $a = "Geänderte Variable";
  echo $b;
?>
```

Listing 4.13 Referenz auf eine Variable (»variablen_referenz.php«)

Wenn Sie dann die ursprüngliche Variable, hier `$a`, ändern, erhält auch die Variable mit der Referenz, hier also `$b`, den neuen Wert (siehe Abbildung 4.8). Übrigens, zwischen dem Gleichheitszeichen und dem Ampersand kann hier auch ein Leerzeichen folgen, oder das Ampersand-Zeichen kann direkt vor der Variablen `$a` stehen.

Abbildung 4.8 Der geänderte Wert wird ausgegeben, da »$b« die Referenz darauf enthält.

4.3.6 Vordefinierte Variablen

Eine Sprache wie PHP besteht natürlich nicht nur aus dem Sprachkern. Um PHP herum gibt es eine große Umwelt: HTML-Formulare, Cookies, also kleine Textdateien im Browser, und vieles mehr. Für diese Umwelt, die Sie im Laufe dieses Buchs noch kennenlernen werden, bietet PHP vordefinierte Variablen. Hier sehen Sie eine Auswahl:

- `$_GET` enthält die per GET aus einem Formular an die URL angehängten Werte.
- `$_POST` enthält die per POST von einem Formular versandten Werte.
- `$_COOKIE` enthält Informationen zu Cookies. Mehr dazu folgt in Kapitel 12, »Entwurfsmuster: MVC & Co.«.
- `$_REQUEST` enthält die Informationen aus den oben genannten drei Variablen. Mehr dazu lesen Sie in Kapitel 12, »Entwurfsmuster: MVC & Co.«, und in Kapitel 13, »Formulare«.
- `$_SESSION` liefert Daten aus Session-Variablen. Mehr dazu finden Sie in Kapitel 15, »Sessions«.
- `$_SERVER` enthält Informationen über die PHP-Installation und den Webserver.
- `$_ENV` bietet Informationen über die Umgebung, in der PHP läuft.
- `$_FILES` besteht aus Daten über hochgeladene Dateien. Dazu finden Sie Informationen in Kapitel 25, »Dateien«.
- `$GLOBALS` enthält alle globalen Variablen. Mehr dazu erfahren Sie in Abschnitt 6.1.2, »Gültigkeit von Variablen«.

Hinweis

Diese vordefinierten Variablen heißen auch *superglobale Arrays*, da sie überall in PHP zur Verfügung stehen. Sie gibt es seit PHP-Version 4.1.0. Davor existierten diese Arrays zwar auch schon, sie hießen aber anders und begannen immer mit `$HTTP_`, also beispielsweise `$HTTP_GET_VARS`. Mehr zu den superglobalen Arrays erfahren Sie in den einzelnen Kapiteln. Die wichtigsten lernen Sie in Kapitel 13, »Formulare«, kennen.

4.4 Konstanten

Konstanten haben, im Gegensatz zu Variablen, immer den gleichen Wert, der anfangs einmal festgelegt wird. In PHP definieren Sie ihn mit dem Schlüsselwort `const` oder – weniger gebräuchlich – mit der Funktion `define()`:

```
const Konstante = "Wert";
```

oder:

```
define("Konstante", "Wert");
```

Der Zugriff auf die Konstante erfolgt jederzeit mit ihrem Namen:

```
echo Konstante;
```

gibt ihren Wert, hier also den String `Wert` aus.

Alternativ greifen Sie auf Konstanten mit der Funktion `constant(Name)` zu:

```
echo constant("Konstante");
```

Diese Funktion kommt zum Einsatz, wenn der Konstantenname nur als Referenz, beispielsweise in einer Variablen oder als Parameter einer Funktion gespeichert, übergeben wird:

```
$Name = "Konstante";
constant($Name);
```

Hinweis

Beachten Sie, dass Konstanten im Gegensatz zu Variablen kein `$`-Zeichen besitzen. Außerdem gelten global definierte Konstanten automatisch im ganzen Skript.