PEARSON Education

Markt+Technik

PHP & MySQL

Dynamik für Ihre Webseiten!

GIESBERT DAMASCHKE

Markt+Technik

→leicht →klar →sofort

Kapitel 2

Entscheidungen, Schleifen und Funktionen

In diesem Kapitel lernen Sie, wie Sie in PHP Entscheidungen fällen, Befehlsfolgen mehrfach abarbeiten und eigene Anweisungen erzeugen können. Damit sind wir bereits bestens gerüstet, um Formulareingaben zu verwalten.

Das können Sie schon:

Sie haben Ihren lokalen Computer mit Xampp in einen Webserver mit PHP verwandelt und Ihre ersten Erfahrungen mit der Sprache gemacht. 16

Das lernen Sie neu:

Logische Vergleiche und Verknüpfungen

Bevor wir uns mit den verschiedenen Möglichkeiten beschäftigen, in PHP Entscheidungen zu fällen, müssen wir einen kleinen Abstecher in die Logik machen. Denn die Basis aller Entscheidungen in einem Programm oder Script ist der Vergleich zweier Werte. Dieser Vergleich wird durch so genannte Vergleichsoperatoren ausgedrückt.

Wenn Sie zwei Zahlen oder Variablen miteinander vergleichen, dann kann dieser Vergleich zu einem von drei möglichen Ergebnissen führen. Entweder ist der eine Wert kleiner als der andere, oder er ist größer oder beide Werte sind gleich. Um diese Verhältnisse zwischen zwei Werten zu beschreiben, gibt es drei verschiedene Operatoren, mit denen die Werte zu einem Ausdruck verbunden werden.

- `$a < $b`: Der Wert der Variablen `$a` ist **kleiner als** der Wert der Variablen `$b`.
- `$a > $b`: Der Wert der Variablen `$a` ist **größer als** der Wert der Variablen `$b`.
- `$a == $b`: Der Wert der Variablen `$a` **ist gleich** dem Wert der Variablen `$b` (da das Gleichheitszeichen in PHP bereits als Zuweisungsoperator benutzt wird, wird die Gleichheit zweier Werte durch ein doppeltes Gleichheitszeichen dargestellt).

Man kann diese drei Grundformen auch kombinieren und erhält so folgende Ausdrücke:

- `$x <= $y`: Der Wert der Variablen `$x` ist **kleiner oder gleich** dem Wert der Variablen `$y`.
- `$x >= $y`: Der Wert der Variablen `$x` ist **größer oder gleich** dem Wert der Variablen `$y`.

Schließlich besteht noch die Möglichkeit, die Ungleichheit zweier Werte zu formulieren:

- `$x != $y`: Der Wert der Variablen `$x` ist **ungleich** dem Wert der Variablen `$y`.
- In Abhängigkeit vom Wert der auf diese Art verknüpften Variablen ist ein so formulierter Ausdruck entweder wahr oder falsch.
- In PHP lassen sich diese Ausdrücke auswerten und je nach Ergebnis (`true` oder `false`) der weitere Verlauf über Wenn-dann-Strukturen des Scripts steuern: Wenn ein bestimmter Ausdruck `true` ist, dann mache dieses, andernfalls tue jenes.

Für $a = 1 und $b = 2 gilt:

Ausdruck	Wert	Erläuterung
`$a < $b`	`true`	Der Ausdruck ist wahr, weil 1 kleiner als 2 ist.
`$a > $b`	`false`	Der Ausdruck ist falsch, weil 1 nicht größer als 2 ist.
`$a == $b`	`false`	Der Ausdruck ist falsch, weil 1 nicht gleich 2 ist.
`$a != $b`	`true`	Der Ausdruck ist wahr, weil 1 ungleich 2 ist.
`$a <= $b`	`true`	Der Ausdruck ist wahr, weil 1 kleiner oder gleich 2 ist.
`$a >= $b`	`false`	Der Ausdruck ist falsch, weil 1 weder kleiner noch größer als 2 ist.

Die einfachen Vergleiche erlauben auch nur einfache Abfragen. Mitunter möchte man aber etwas komplexere Bedingungen formulieren und die Ausführung eines Programmcodes von mehr als nur einem Ausdruck abhängig machen, etwa: »Wenn die Variable `$a` kleiner als `$b` ist und die Variable `$c` kleiner als `$d`, dann ...«.

In diesem Fall müssen Sie die unterschiedlichen Ausdrücke mit einem logischen `and` (»und«) bzw. einem logischen `or` (»oder«) verknüpfen. Der so gebildete neue Ausdruck erzeugt in Abhängigkeit von den Wahrheitswerten der verknüpften Ausdrücke einen neuen Wahrheitswert.

1 Eine `and`-Verknüpfung ist dann (und nur dann) wahr, wenn alle verknüpften Ausdrücke wahr sind. Eine `and`-Verknüpfung lässt sich mit dem doppelten Ampersand `&&` abkürzen:

```
<?
($a < $b) and ($c > $d)
($a < $b) && ($c > $d)
?>
```

2 Eine `or`-Verknüpfung ist dann wahr, wenn mindestens einer der verknüpften Ausdrücke wahr ist. Eine `or`-Verknüpfung lässt sich mit dem doppelten Pipesymbol `||` abkürzen:

```
<?
($a < $b) or ($c > $d)
($a < $b) || ($c > $d)
?>
```

3 Eine `xor`-Verknüpfung ist eine Verschärfung der `or`-Verknüpfung. Sie ist dann (und nur dann) wahr, wenn entweder der eine oder der andere verknüpfte Ausdruck wahr ist. Wenn beide wahr sind, ist eine `xor`-Verknüpfung falsch, wenn beide verknüpften Aussagen falsch sind, natürlich auch. Für `xor` gibt es keine Abkürzung:

```
<?
($a < $b) xor ($c > $d)
?>
```

4 Schließlich können Sie jeden Wahrheitswert durch ein vorangestelltes `!` negieren. Ein negierter Ausdruck ist dann wahr, wenn der Ausdruck falsch ist und umgekehrt:

```
<?
!($a < $b)
?>
```

Wenn ... dann ... andernfalls!

Mit der Möglichkeit, Verhältnisse zwischen zwei (oder mehr) Variablen bzw. Aussagen zu beschreiben, sind wir in PHP in der Lage, Entscheidungen zu treffen.

Die klassische »Wenn-dann«-Verknüpfung wird in PHP mit einer `if`-Anweisung realisiert. Der umgangssprachliche Satz »Wenn eine bestimmte Bedingung zutrifft, dann tue dieses«, ließe sich in PHP abstrakt so formulieren:

```
if (Bedingung) Anweisung;
```

Wenn Sie mehr als nur eine Anweisung ausführen wollen, dann müssen Sie die folgenden Anweisungen wie gewohnt mit einem Semikolon voneinander trennen und mit einer Mengenklammer zusammenfassen:

```
if (Bedingung) {
  Anweisung 1;
  Anweisung 2;
  Anweisung 3;
}
```

Die in der Mengenklammer zusammengefassten Anweisungen werden dann – und nur dann! – ausgeführt, wenn die Bedingung den Wert `true` besitzt, ansonsten ignoriert PHP den gesamten Block.

Die Mengenklammer

Mit der Mengenklammer werden bei bestimmten Konstruktionen mehrere Anweisungen zusammengefasst. Die verschiedenen Anweisungen innerhalb der Mengenklammer werden wie gewohnt durch ein Semikolon getrennt. Nach der Mengenklammer steht jedoch kein Semikolon, auch wenn noch andere Anweisungen folgen sollten.

Jetzt sind wir in der Lage, ein kleines Script zu schreiben, das entscheiden kann, ob zwei übergebene Werte gleich sind, oder ob der erste kleiner bzw. größer als der zweite Wert ist. Dazu werden zuerst zwei mit `get` übergebene Werte in Variablen abgelegt. Anschließend überprüfen drei aufeinanderfolgende `if`-Abfragen die möglichen Beziehungen der beiden Werte und geben, falls sie zutreffen, einen kurzen Text aus.

```
<?
$a = $_GET['a'];
$b = $_GET['b'];
if ($a < $b) echo "A ist kleiner als B";
if ($a > $b) echo "A ist gr&ouml;&szlig;er als B";
if ($a == $b) echo "A ist gleich B";
?>
```

Speichern Sie das kleine Script im `<body>`-Teil von z. B. »test.php« und rufen Sie es mit zwei Werten auf, etwa als

http://localhost/test.php?a=5&b=1

Das funktioniert schon ganz gut, hat aber noch einen Schönheitsfehler. Bei jedem Aufruf werden alle `if`-Abfragen abgearbeitet, auch dann, wenn dies gar nicht nötig ist. Wenn etwa die erste Abfrage zutrifft und die `echo`-Anweisung ausgeführt wird, können die übrigen beiden `if`-Abfragen übersprungen werden, weil sie garantiert nicht mehr zutreffen können.

Für diesen Fall gibt es die `else`- Anweisung. Das englische Wort bedeutet so viel wie »sonst, anders, anderenfalls«. In Kombination mit `if` sind damit Konstruktionen der Art »Wenn die Bedingung zutrifft, tue jenes, andernfalls unternehme dieses« formulierbar. Trifft die Bedingung in der `if`-Abfrage zu, wird die damit verbundene Anweisung ausgeführt und der `else`-Teil ignoriert. Im umgekehrten Fall – die Bedingung ist falsch – wird der `if`-Teil ignoriert und die mit `else` verbundene Anweisung ausgeführt.

Die Konstruktion hat folgende Grundstruktur:

```
if (Bedingung) Anweisung;
else Anweisung;
```

Auch bei `else` können Sie mehrere, mit einem Semikolon getrennte Anweisungen mit einer Mengenklammer zusammenfassen (nach der, wie bei `if`, dann kein Semikolon mehr folgt):

```
if (Bedingung) {
  Anweisung 1;
  Anweisung 2;
} else {
  Anweisung 1;
  Anweisung 2;
}
```

Mit der `elseif`-Anweisung lässt sich diese Abfrage noch weiter ausbauen. Hiermit lassen sich mehrere Bedingungen nacheinander überprüfen. Am Schluss steht schließlich die `else`-Anweisung, die ausgeführt wird, wenn keine der Bedingungen erfüllt wurde:

```
if (Bedingung 1) Anweisung;
elseif (Bedingung 2) Anweisung;
else Anweisung;
```

Dabei können Sie auch mehrere `elsif`-Anweisungen aufeinander folgen und so verschiedene Bedingungen überprüfen lassen:

```
if (Bedingung 1) Anweisung;
elseif (Bedingung 2) Anweisung;
elseif (Bedingung 3) Anweisung;
elseif (Bedingung 4) Anweisung;
elseif (Bedingung 5) Anweisung;
else Anweisung;
```

Natürlich gibt es auch bei der `if-elseif-else`-Struktur die Möglichkeit, mit Mengenklammern mehrere Anweisungen zu bündeln. Hier gelten die gleichen Regeln wie bei `if` oder `else`:

```
if (Bedingung 1) {
  Anweisung 1;
  Anweisung 2;
} elseif (Bedingung 2) {
  Anweisung 1;
  Anweisung 2;
} else {
  Anweisung 1;
  Anweisung 2;
}
```

Damit können wir unser kleines Beispiel schon etwas eleganter formulieren und die überflüssigen `if`-Abfragen durch `elseif` und `else` ersetzen. Trifft die `if`-Abfrage zu, wird der entsprechende Text ausgegeben und das Script beendet. Andernfalls wird die `elseif`-Abfrage bearbeitet und erst, wenn diese ebenfalls nicht zutrifft, gelangt das Script zur `else`-Anweisung:

```
<?
$a = $_GET['a'];
$b = $_GET['b'];
if ($a < $b) echo "A ist kleiner als B";
elseif ($a > $b) echo "A ist gr&ouml;&szlig;er als B";
else echo "A ist gleich B";
?>
```

Auf Inhalt überprüfen

Nun führt das Script zwar keine überflüssigen Abfragen mehr durch, aber einen Haken hat die Sache doch noch. Es erkennt nicht, wenn keine Werte für `$a` und `$b` übergeben wurden und führt anschließend eher unsinnige Vergleiche aus. Das mag in diesem Beispiel noch ein akzeptabler Fehler ohne große Konsequenzen sein, aber wenn wir bei einer Formularauswertung nicht überprüfen können, ob das Formular korrekt ausgefüllt wurde, kann das fatale Folgen haben.

Hier hilft die Funktion `isset()`, der man in Klammern die Variable übergibt, die überprüft werden soll. Existiert die Variable, dann liefert `isset()` den Wert `true` zurück, andernfalls `false`. So lässt sich diese Funktion sehr schön in eine `if`-Abfrage integrieren:

```
<?
if (isset($a)) echo "A hat den Wert $a";
else echo "A ist nicht definiert";
?>
```

Häufig wird `isset()` in Verbindung mit der Negation `!` benutzt, um eine Konstruktion der Art »Wenn die Variable $a nicht existiert, dann ...« zu bilden, was in PHP so aussieht:

```
<?
if (!isset($a)) echo "Geben Sie einen Wert f&uuml;r A ein!";
else echo "A hat den Wert $a";
?>
```

Unser Beispielscript kann nun so umgebaut werden, dass es nur ausgeführt wird, wenn für `$a` und `$b` Werte übergeben wurden, andernfalls wird ein kurzer Hinweis ausgegeben.

Die `if`-Abfragen des bisherigen Scripts rutschen dabei komplett in den Anweisungsteil einer weiteren `if`-Abfrage und werden dann (und nur dann) ausgeführt, wenn sowohl `$a` also auch `$b` einen Wert besitzen, die Funktion `isset($a)` und `isset($b)` den Wert `true` liefern:

```
<?
$a = $_GET['a']; $b = $_GET['b'];
if (isset($a) && isset($b)) {
  if ($a < $b) echo "A ist kleiner als B";
  elseif ($a > $b) echo "A ist gr&ouml;&szlig;er als B";
  else echo "A ist gleich B";
} else echo "Geben Sie Werte f&uuml;r A und B ein !";
?>
```

Durch den Einsatz von isset() lässt sich überprüfen, ob eine Variable einen Wert besitzt und eine Weiterverarbeitung überhaupt sinnvoll ist.

Neben `isset()`, mit der man die Existenz einer Variablen überprüft, gibt es in PHP noch die Funktion `empty()`, die ähnlich arbeitet, aber mitunter seltsame Nebeneffekte produziert. Mit dieser Funktion wird überprüft, ob eine Variable einen Inhalt besitzt oder nicht. Fatalerweise wird allerdings der Wert 0 als »leer« gewertet, und zwar sowohl als numerischer Wert `($a=0)`, als auch als Zeichenkette `($a="0")`. Der Einsatz dieser Funktion will also gut überlegt sein.

Alternative Abfragen

Solange Sie nur zwei Werte vergleichen, werden Sie mit `if`, `elseif` und `else` keine Probleme bekommen. Sobald es aber darum geht, mehr als nur zwei oder drei Möglichkeiten zu beachten, tendieren `if`-Abfragen dazu, eher unübersichtlich zu werden. Hier greift man zur `switch`-Anweisung, mit der sich gewissermaßen beliebig viele `if`-Abfragen zusammenfassen lassen.

Die `switch`-Anweisung wertet einen Ausdruck aus und vergleicht anschließend beliebig viele Varianten mit dem Ergebnis. Sollte eine Variante (`case`)

zutreffen, wird der dazugehörige Programmcode ausgeführt und die Anweisung über das Kommando `break` verlassen.

Ein frei definierbarer `default`-Block enthält überdies Anweisungen, die ausgeführt werden, wenn keine der Vorgaben zutrifft. Die Struktur der `switch`-Anweisung hat folgenden Aufbau:

```
switch (Ausdruck) {
  case Ergebnis1:
    Anweisung1; break;
  case Ergebnis2:
    Anweisung2; break;
// beliebig viele weitere case-Einträge
  default:
    Anweisung_default;
}
```

Das Ergebnis des Ausdrucks wird mit den `case`-Vorgaben verglichen und im Falle einer Übereinstimmung der dazugehörige Code ausgeführt. Über das `break`-Kommando wird die `switch`-Anweisung verlassen und das Programm mit dem Code nach der abschließenden geschweiften Klammer fortgesetzt.

Falls keine der gemachten Vorgaben mit dem Ergebnis des Ausdrucks übereinstimmt, wird der `default`-Code ausgeführt (falls ein solcher Code vorhanden ist, diese Anweisung ist nicht zwingend erforderlich). Da dieser Code immer zuletzt steht, entfällt das `break`-Kommando. Achten Sie darauf, dass die `case`- und `default`-Zeilen mit einem Doppelpunkt, nicht mit einem Semikolon beendet werden!

Wenn Sie das `break`-Kommando vergessen, dann erhalten Sie zwar keine Fehlermeldung, aber unter Umständen ein unerwünschtes Ergebnis. Sobald eine `case`-Anweisung zutrifft, werden sämtliche folgenden Anweisungen ausgeführt, auch die anderen `case`-Anweisungen. Obendrein wird die `default`-Anweisung am Ende auf jeden Fall ausgeführt, was wohl kaum im Sinne des Erfinders ist.

Datumsangaben in PHP

Ein einfaches, praxisnahes Einsatzgebiet für `switch` liefert die Datumsabfrage. Auch dafür bietet PHP – Sie ahnen es – eine eigene Funktion. Streng genommen gibt es sogar mehrere Funktionen, die sich mit Daten und Zeiten beschäftigen, wir beschränken uns hier aber nur auf die griffigste: `date()`.

Der Einsatz der `date()`-Funktion ist recht einfach: Sie übergeben der Funktion einen oder mehrere bestimmte Parameter und die Funktion gibt Ihnen dafür das gewünschte Datum oder die gewünschte Uhrzeit zurück. Kompliziert wird die Sache nur dadurch, dass es für `date()` erstaunlich viele Parameter gibt, was für ein gewisses Durcheinander sorgt. Die folgende Tabelle stellt die wichtigsten `date()`-Parameter zusammen.

Parameter für Datumsangaben mit `date()`:

`d`	Tag im Monat als Zahl mit führender Null
`j`	Tag im Monat als Zahl ohne führende Null
`w`	Wochentag als Zahl von 0 – 6
`m`	aktueller Monat als Zahl mit führender Null (01-12)
`n`	aktueller Monat als Zahl ohne führende Null (1-12)
`D`	Wochentag, englischer Begriff, abgekürzt mit 3 Buchstaben
`l`	Wochentag, englischer Begriff, ausgeschrieben (Achtung: »l« ist ein kleines »L«, kein großes »i«)
`F`	Monat, englischer Begriff, ausgeschrieben
`M`	Monat, englischer Begriff, abgekürzt mit 3 Buchstaben
`y`	Das Jahr als zweistellige Zahl
`Y`	Das Jahr als vierstellige Zahl

Parameter für Zeitangaben mit `date()`:

`s`	Sekunden, Zahl mit führender Null
`i`	Minuten, Zahl mit führender Null
`H`	Stunden im 24-Stunden-Format mit führender Null
`G`	Stunden im 24-Stunden-Format ohne führende Null

Die Parameter können bei `date()` auch als Bestandteile einer auszugebenden Zeichenkette benutzt werden, in der die übergebenen Parameter durch die entsprechenden Angaben ersetzt und alle anderen Zeichen unverändert ausgegeben werden. So lassen sich Datumsabfragen als kompakter Code formulieren.

1 Benötigt man etwa das aktuelle Datum in Form von »17. 6. 2002«, dann geht das ein wenig umständlich so:

```
<?
$tag = date(j);
$monat = date(n);
$jahr = date(Y);
echo "Heute ist der $tag. $monat. $jahr";
?>
```

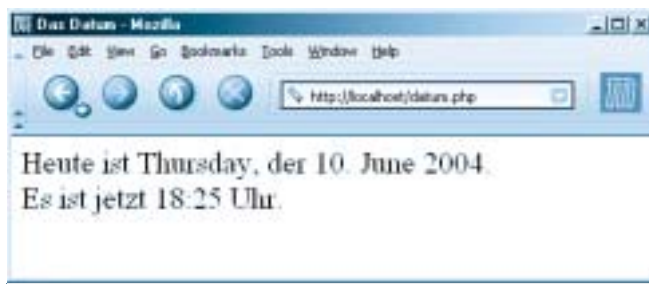
2 Diese vier Anweisungen lassen sich jedoch zu einer einzigen einschmelzen, indem man `date()` die benötigten Parameter zusammen mit den Punkten und Leerzeichen nach Tag und Monat als Zeichenkette übergibt (`date("j. n. Y")`) und diesen Ausdruck einfach mit einem Komma an den auszugebenden Text hängt:

```
<?= "Heute ist der ", date("j. n. Y"); ?>
```

3 Auf die gleiche Weise können wir das aktuelle Datum und die Uhrzeit mit `date()` ermitteln. Die Ausgabe soll nach dem Muster *Heute ist WOCHENTAG, der DATUM. Es ist jetzt UHRZEIT Uhr* erfolgen.

```
<?
  $wochentag = date(l);
  $datum = date("d. F Y");
  $uhrzeit = date("H:i");
  echo "Heute ist $wochentag, der $datum.<br>";
  echo "Es ist jetzt $uhrzeit Uhr.";
?>
```

4 Allerdings ist das Ergebnis nicht so ganz das, was wir uns vielleicht vorgestellt haben – schließlich wird die Ausgabe in einem Mischmasch aus Deutsch und Englisch erfolgen.



Die date()-Funktion liefert das aktuelle Datum und Uhrzeit – allerdings auf Englisch.

5 Hier können wir nun durch eine `switch`-Abfrage die englischen Tages- und Monatsbezeichnungen durch die deutschen Begriffe austauschen.

```
<?
$wochentag = date(l);
switch($wochentag) {
  case "Monday": $wochentag = "Montag"; break;
  case "Tuesday": $wochentag = "Dienstag"; break;
  // und so weiter bis Sunday
}
?>
```

Wie Sie sehen, können Sie die verschiedenen `case`-Einträge auch in einer Zeile notieren.

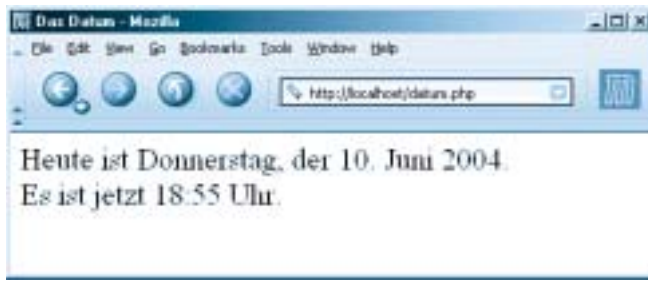
6 Bei der Ermittlung des Monatsnamens geht man ähnlich vor. Die acht Anweisungen – nein, nicht zwölf, die Monatsnamen »April«, »August«, »September« und »November« werden im Englischen wie im Deutschen geschrieben, hier muss das Script also nichts ersetzen – werden nach dem gleichen Muster notiert:

```
<?
$monat = date(F);
switch($monat) {
  case "January": $monat = "Januar"; break;
  case "February": $monat = "Februar"; break;
// und so weiter bis December
}
?>
```

7 Wenn in `$wochentag` und `$monat` nun die deutschen Bezeichnungen vorliegen, ist der Rest des Scripts rasch angepasst. Die Datumsangabe mit `date("d. F Y")` müssen wir nur noch durch

```
<?
$datum = date(d).".$monat ".date(Y);
?>
```

ersetzen, die `echo`-Anweisungen bleiben unverändert.



Mit einer einfachen switch-Anweisung lassen sich die englischen Begriffe der date()-Funktion durch die entsprechenden deutschen Bezeichnungen austauschen.

Das Datum auf der Homepage

Mit dem vorgestellten Script können Sie Besucher Ihrer Homepage mit dem aktuellen Datum und der Uhrzeit begrüßen. Setzen Sie dafür das Script (ohne die `echo`*-Anweisungen versteht sich) an den Anfang des Codes Ihrer Webseite. Innerhalb des HTML-Teils können Sie dann jederzeit mit* `<?=$datum?>` *und* `<?=$uhrzeit?>` *Datum und Uhrzeit einfügen und die Ausgabe mit den üblichen HTML-Möglichkeiten formatieren.*

Programmschleifen

Bislang werden die Anweisungen in den Beispielscripten 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 jeden Programms und machen es flexibler und leistungsfähiger.

Die einfachste Form einer Schleife in PHP ist die `while`-Konstruktion. »While« ist das englische Wort für »während, solange« und genauso funktioniert die Schleife auch. Solange eine bestimmte Abbruchbedingung nicht eintritt, solange werden die Anweisungen innerhalb der Schleife durchlaufen. Die Abbruchbedingung wird dabei als logischer Ausdruck notiert. In der allgemeinen Form sieht das so aus:

```
while (Ausdruck) {
  // Beliebige Anweisungen
}
```

Solange der Ausdruck den Wert `true` besitzt, werden die Anweisungen in den geschweiften Klammern ausgeführt. Mindestens eine dieser Anweisungen muss also den Wert des Ausdrucks so verändern, dass er irgendwann `false` ergibt, ansonsten wird die Abbruchbedingung nie erfüllt und die Schleife nie verlassen.

> **Hinweis**
>
> *Fehlende oder fehlerhaft formulierte Abbruchbedingungen sind eine der häufigsten Ursachen für einen Programmabsturz.*

Das klingt jetzt vielleicht etwas abstrakt und trocken, also probieren wir das rasch an einem einfachen Beispiel aus.

1 Einem Script werden mit `get` zwei Zahlen übergeben, die als Anfangs- und Endwert gespeichert werden. Das Script zählt dann in Einserschritten von der einen bis zur anderen Zahl und zeigt das Ergebnis am Bildschirm an:

```
<?
$anfang = $_GET['a']; $ende = $_GET['e'];
while ($anfang <= $ende) {
  echo $anfang," ";
  $anfang++;
}
?>
```

2 Da die Variable `$anfang` innerhalb der `while`-Schleife verändert wird, verändert sich auch der Wahrheitswert der Abbruchbedingung. Sobald der Wert von `$anfang` größer als `$ende` ist, wird die Schleife verlassen.

Streng genommen müssten wir noch einige Sicherheitsabfragen für den Fall einfügen, dass keine Werte übergeben werden oder dass der Endwert kleiner als der Anfangswert ist. Aber wirklich nötig sind sie in diesem Fall nicht, denn die Schleife wird dann (und nur dann) durchlaufen, solange `$anfang` kleiner oder gleich `$ende` ist. In den anderen Fällen passiert gar nichts.

`While` ist ein Beispiel für eine so genannte »abweisende Schleife«, bei der noch vor dem ersten Durchlauf die Abbruchbedingung überprüft wird. Das Gegenstück ist eine »annehmende Schleife«, bei der erst nach den Anweisungen überprüft wird, ob die Abbruchbedingung eingetreten ist. Eine solche Schleife wird also mindestens einmal, eine abweisende Schleife unter Umständen gar nicht durchlaufen.

Eine annehmende Schleife in PHP wird mit `do/while` konstruiert:

```
do {
  // Beliebige Anweisungen
} while (Ausdruck);
```

Neben der `while`- kennt PHP noch die `for`-Schleife, die in vielen Fällen das gleiche Ergebnis liefert, aber einen etwas kompakteren und übersichtlicheren Code ermöglicht.

Bei einer `while`-Konstruktion müssen Sie dafür sorgen, dass die Abbruchbedingung der Schleife erreicht wird, indem etwa innerhalb der Schleife ein Zähler mitgeführt oder durch kontinuierliche Akkumulation ein Grenzwert erreicht wird. Bei einer `for`-Konstruktion wird die Ausführungsbedingung und die kontinuierliche Erhöhung eines Zählers dagegen außerhalb und vor dem Start der eigentlichen Schleife definiert.

Die allgemeine Syntax einer `for`-Schleife sieht so aus:

```
for (Initialisierung; Bedingung; Veränderung) {
  // Beliebige Anweisungen
}
```

Einzeilig

Besteht die Schleife nur aus einer Anweisung, kann man die `for`-Konstruktion in einer Zeile schreiben und auf die Mengenklammern verzichten.

Kernstück der `for`-Schleife sind die drei durch ein Semikolon getrennten Anweisungen in Klammern. Zuerst wird üblicherweise eine Variable als Zähler initialisiert, danach wird die Bedingung angegeben, die erfüllt sein muss, damit die Schleife durchlaufen wird, und schließlich wird der Zähler erhöht (oder verringert). Um die Zahlen von 1 bis 10 auszugeben, benötigt man zum Beispiel diese Schleife:

```
<?
for ($x = 1; $x <= 10; $x++) echo $x;
?>
```

Das kleine `while`-Beispiel ließe sich also in einer `for`-Schleife folgendermaßen notieren:

```
<?
$anfang = $_GET['a']; $ende = $_GET['e'];
for ($wert = $anfang; $wert <= $ende; $wert++) echo $wert." ";
?>
```

Ein typischer Einsatz für die `for`-Schleife ist das Abzählen von Werten. Will man etwa die ASCII-Codes von 32 bis 255 ausgeben, so benötigt man dafür gerade mal eine Zeile PHP-Code.

HTML- und numerische Entities

In HTML können Zeichen durch so genannte Entities dargestellt werden. Eine Entity wird mit einem Ampersand & eingeleitet und einem Semikolon ; beendet. Das auszugebende Zeichen wird entweder mit seinem ISO-Namen angegeben (z. B. »auml« für »a-Umlaut«, also »ä«) oder mit seinem numerischen Zeichencode (»ä« hat den Code 228, notiert als #228). Ein »ä« lässt sich so in HTML entweder als »ä« oder als »ä« darstellen.

Man lässt in einer Schleife eine Variable von 32 bis 255 hochzählen und setzt diese Variable in einer `echo`-Anweisung als Wert für die numerische Entity ein:

```
<html>
<head><title>ASCII-Codes 32 bis 255</title></head>
<body>
<p>Dies sind die ASCII-Codes von 32 bis 255:</p>
<p><? for ($x = 32; $x <= 255; $x++) echo "&#$x; "; ?></p>
</body>
</html>
```

Eine einzige Zeile in PHP genügt, um sich die darstellbaren ASCII-Codes ausgeben zu lassen.

Das kleine Einmaleins

Ihre wahre Stärke erreichen Schleifen dann, wenn man sie verschachtelt, d. h. innerhalb einer Schleife eine weitere Schleife ausführt. Verschachtelte Schleifen sind nicht immer auf Anhieb verständlich und tendieren dazu, sehr komplex zu werden und zu eher unerwarteten Ergebnissen zu führen. Gehen wir also das Verfahren an einem Beispiel etwas ausführlicher durch.

Aufgabe ist es, das kleine Einmaleins zeilenweise über eine PHP-Anweisung auszugeben.

1 Wir benötigen zwei Schleifen, eine für die Reihen, die von eins bis zehn zählt und eine für die Spalten, die ebenfalls von eins bis zehn zählt und die eigentliche Multiplikation enthält. Das Grundgerüst könnte so aussehen:

```
<?
for ($r = 1; $r <= 10; $r++) {
  for ($s = 1; $s <= 10; $s++) echo ($r * $s);
  echo "<br>";
}
?>
```

2 Die erste Schleife benutzt die Variable `$r` (für »Reihe«) als Zähler von 1 bis 10. Die erste Anweisung ist eine weitere Schleife, die die Variable `$s` (für »Spalte«) von 1 bis 10 zählt und das Ergebnis der Multiplikation von Reihen- und Spaltenzähler ausgibt. Abschließend sorgt ein `<br>`-Tag dafür, dass eine neue Zeile begonnen wird.

3 Was passiert nun genau in dieser Schleife? Gehen wir sie im Kopf einmal kurz durch. Zu Beginn wird der Variablen `$r` der Wert 1 zugewiesen. Danach beginnt die zweite Schleife, in der die Variable `$s` die Werte 1 bis 10 durchläuft, während `$r` immer noch den Wert 1 besitzt. Innerhalb der zweiten Schleife werden `$r` und `$s` miteinander multipliziert und das Ergebnis ausgegeben. Insgesamt bekommt man also dieses Ergebnis:

```
1 2 3 4 5 6 7 8 9 10
```

4 Damit ist die innere Schleife beendet, die äußere wird fortgeführt. Es wird ein Zeilenumbruch erzwungen, danach wird der Wert für `$r` um eins erhöht. Da die Abbruchbedingung der äußeren Schleife noch nicht erreicht ist, beginnt nun alles von vorn, diesmal mit dem Wert 2 für die Variable `$r`, die Anzeige sieht nun so aus:

```
1 2 3 4 5 6 7 8 9 10

2 4 6 8 10 12 14 16 18 20
```

5 Wieder wird die innere Schleife beendet und die äußere fortgeführt. Dies wird so lange wiederholt, bis die Variable `$r` den Wert 11 erreicht hat und die äußere Schleife verlassen wird.

6 Damit die Ausgabe etwas ansprechender und übersichtlicher gerät, setzen wir den gesamten Code in ein `<table>`-Tag. Die äußere Schleife gibt die Reihen aus, muss also die innere Schleife mit einem `<tr>`-Element umgeben. Dort werden nun die einzelnen Werte pro Reihe ausgegeben, die jeweils von einem `<td>` umrahmt werden müssen. Zusammen ergibt sich dieser HTML-Code mit PHP-Anteil:

```
<table cellpadding="5" border="1">
<?
for ($r = 1; $r <= 10; $r++) {
  echo "<tr>";
  for ($s = 1; $s <= 10; $s++) {
    echo "<td>";
    echo ($r * $s);
    echo "</td>";
  }
  echo "</tr>";
}
?>
</table>
```

7 Die innere Schleife könnte mit einem Komma zu einer einzigen Anweisung zusammengefasst und so in einer Zeile notiert werden. Allerdings wird's dann ein wenig unübersichtlich:

```
<?
for ($s = 1; $s <= 10; $s++) echo "<td>",($r * $s),"</td>";
?>
```

Für den Anfang empfiehlt es sich, auf solche Verkürzungen zu verzichten und jeden Schritt in einer einzelnen Zeile zu schreiben.

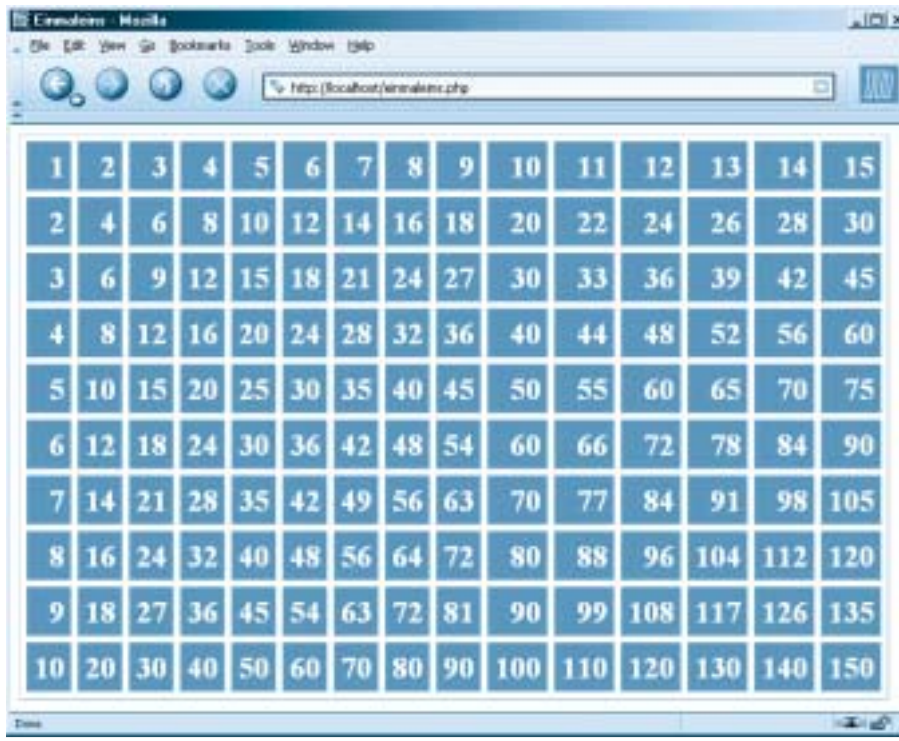
Damit ist der PHP-Teil der Aufgabe erledigt, nun kann man sich daran machen, die Ausgabe durch Optimierung des Codes noch ein wenig gefälliger zu gestalten, etwa durch Stylesheets den Zellen Hintergrundfarben zu geben und die Zelleninhalte rechtsbündig auszurichten.

Natürlich können Sie mit dieser Schleife nicht nur das kleine Einmaleins, sondern praktisch jede gewünschte Multiplikationstabelle erzeugen. Dafür müssen Sie lediglich die Schleifenwerte entsprechend anpassen. Dabei können Sie nicht nur die Start- und Endwerte der Schleife verändern, sondern auch die Schrittweite und etwa nur jeden zweiten Wert berechnen lassen:

```
<?
for ($s = 1; $s <= 10; $s = $s + 2)
?>
```

Selbstverständlich können Sie für die Schrittweite `$s = $s + 2` auch die verkürzte Darstellung benutzen:

```
<?
for ($s = 1; $s <= 10; $s += 2)
?>
```



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60
5	10	15	20	25	30	35	40	45	50	55	60	65	70	75
6	12	18	24	30	36	42	48	54	60	66	72	78	84	90
7	14	21	28	35	42	49	56	63	70	77	84	91	98	105
8	16	24	32	40	48	56	64	72	80	88	96	104	112	120
9	18	27	36	45	54	63	72	81	90	99	108	117	126	135
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150

Durch Anpassung der Schleifenwerte können Sie beliebige Multiplikationstabellen erzeugen. Dabei ist PHP für die Inhalte der Tabelle zuständig, während ihre Gestaltung wie gewohnt in HTML erfolgt.

Funktionen

Eine Funktion bezeichnet in PHP gewissermaßen ein Programm innerhalb des Programms zur Lösung häufig wiederkehrender Aufgaben oder Anforderungen. PHP besitzt bereits von Haus aus mehr als 1.000 Funktionen zur

Erledigung der unterschiedlichsten Aufgaben, ein paar davon wie `isset()`, `htmlentities()` oder `date()` haben wir bereits kennen gelernt. Außerdem können Sie beliebige Funktionen definieren, die auf Ihre individuellen Programmerfordernisse zugeschnitten sind.

Eine Funktion wird durch ihren Namen und normalerweise mit einem Parameter in Klammern aufgerufen, der innerhalb der Funktion verarbeitet wird. Das Ergebnis wird anschließend von der Funktion an das aufrufende Script zurückgegeben. Dort wird es einer Variablen zugewiesen und kann so im Script weiter verarbeitet werden.

Das klingt jetzt vielleicht etwas umständlich, aber die Beschreibung ist länger als der PHP-Code. So liefert die Funktion `sqrt()` die Wurzel eines übergebenen Wertes zurück, `sqrt(4)` ergibt also den Wert 2. Damit mit diesem Ergebnis aber weiter gearbeitet werden kann, muss man es einer Variablen zuweisen, als Text ausgeben oder als Bestandteil einer weiteren Berechnung benutzen:

```
<?
$a = $_GET['a'];
echo "Die Wurzel aus $a ist: ",sqrt($a);
?>
```

Funktionen können natürlich auch mehr als einen Parameter besitzen, die man mit Komma getrennt an die Funktion übergibt. So ersetzt etwa die Funktion `str_replace()` in einer Zeichenkette (einem so genannten »String«) beliebige Zeichen durch andere Zeichen. Dabei werden der Funktion folgende Parameter übergeben: das (oder die) Zeichen, die ersetzt, das (oder die) Zeichen, die eingefügt und schließlich der String, in dem die Zeichen ausgetauscht werden sollen.

Mit `str_replace("e", "x", "Internet")`werden folglich alle Buchstaben »e« im Wort »Internet« durch ein »x« ersetzt. Zugegeben, das ist jetzt kein sehr sinnvolles Beispiel, aber diese Funktion wird uns später noch oft und in sinnvolleren Kontexten begegnen.

Möchten Sie zur Lösung einer bestimmten Aufgabe eine Funktion selbst definieren, so können Sie dies über die Anweisung `function` tun. Sie definieren die Funktion, indem Sie ihr einen Namen geben, die benötigten Parameter nennen und, natürlich, die Anweisungen definieren, die innerhalb der Funktion ausgeführt werden sollen. Diese Anweisungen stehen immer in Mengenklammern.

So ist es problemlos möglich, den arithmetischen Befehlsvorrat von PHP durch eine Quadrat-Funktion zu ergänzen (PHP besitzt zwar eine Funktion,

um die Quadratwurzel eines Wertes zu ermitteln, aber keine, um eine Zahl zu quadrieren). Die Funktion soll »square()« heißen, mit einem Wert aufgerufen werden und dessen Quadrat zurückgeben:

```
<?
function square($zahl) {
  return $zahl * $zahl;
}
?>
```

Mit der Anweisung `return` gibt man einen Wert von der Funktion an das aufrufende Script zurück. Ein einfaches Beispiel soll den Einsatz selbstgemachter Funktionen verdeutlichen.

Gültigkeit von Variablen

Variablen, die innerhalb einer Funktion benutzt werden, sind auch nur innerhalb der Funktion gültig. Eine Variable `$zahl`*, die im Script benutzt wird, und eine Variable* `$zahl` *innerhalb einer Funktion sind für PHP zwei grundsätzlich verschiedene Werte.*

Die Funktions-Definition wird üblicherweise an den Kopf des Dokuments gesetzt, im Script selbst kann die Funktion dann so benutzt werden wie jede andere Funktion auch. Das Beispielscript macht nun nichts anderes, als via `get` einen Wert entgegenzunehmen, ihn mit der Funktion `square()` zu quadrieren und die Berechnung auszugeben. Dabei wird die numerische Entity `&#215;` benutzt, mit der sich ein Multiplikationszeichen darstellen lässt (was deutlich besser lesbar ist als der Einsatz von `x` oder `*`):

```
<?
  function square($zahl) {
    return $zahl * $zahl;
  }
?>
<html>
<head><title>Quadratfunktion</title></head>
<body>
<?
  $a = $_GET['a'];
  echo "$a &#215; $a = ",square($a);
?>
</body>
</html>
```

Wenn Sie eine Fließkommazahl übergeben möchten, müssen Sie statt des gewohnten Kommas einen Punkt notieren, also »12.1« statt »12,1«. Andernfalls wird der übergebene Wert als Zeichenkette, nicht als Zahl erkannt.

Variablentypen überprüfen

Die selbstgeschriebene Funktion `square()` arbeitet wie erwartet, aber es gibt noch einen Schönheitsfehler: Wenn die mit `get` übergebene Variable `$a` kein numerischer Wert, sondern ein Text ist – etwa bei einem Aufruf mit

http://localhost/funktion.php?a=test

dann wird das Quadrat von »test« gebildet. Das Script verarbeitet diese Eingabe zwar klaglos und wird als Ergebnis den Text »test × test = 0« ausgeben, aber sonderlich sinnvoll ist das nicht. Wir sollten also vor dem Funktionsaufruf überprüfen, ob ein numerischer Wert vorliegt.

1 Hier hilft uns die Funktion `is_numeric()`, die den Wert `true` zurückgibt, wenn die zu überprüfende Variable ein numerischer Wert ist, und `false` ergibt, wenn dies nicht der Fall ist. Damit können wir die Funktion als Bedingung in einer `if`-Konstruktion einsetzen, deren Anweisung ausgeführt wird, wenn die Variable numerisch ist, andernfalls – `else` – wird ein kurzer Hinweis angezeigt:

```
<?
  $a = $_GET['a'];
  if (is_numeric($a)) echo "$a &#215; $a = ",square($a);
  else echo "Bitte einen numerischen Wert eintragen!";
?>
```

2 Wenn wir schon dabei sind, mögliche Eingabefehler abzufangen, dann können wir auch gleich mit `isset($a)`überprüfen, ob überhaupt ein Wert übergeben wurde. Wenn ja, dann wird die numerische Prüfung durchgeführt, wenn nicht, wird zur Eingabe eines Wertes aufgefordert. Das könnte dann so aussehen:

```
<?
$a = $_GET['a'];
if (isset($a)) {
```

```
  if (is_numeric($a)) echo "$a &#215; $a = ",square($a);
  else echo "<b>$a</b> ist kein numerische Wert.";
} else echo "Bitte einen numerischen Wert f&uuml;r <b>a</b>
angeben.";
?>
```

Die wichtigsten Variablen-Funktionen

Funktion	Erläuterung
`empty($a)`	`true`, falls `$a` leer ist. Vorsicht, »0« gilt als »leer«!
`isset($a)`	`true`, falls `$a` existiert, wobei `$a` auch leer sein kann.
`is_numeric($a)`	`true`, falls `$a` ein numerischer Wert ist.
`is_int($a)`	`true`, falls `$a` ganzzahlig ist.
`is_bool($a)`	`true`, falls `$a` ein logischer Wert (boolean) ist.
`is_float($a)`	`true`, falls `$a` eine Fließkommazahl ist.
`gettype($a)`	Ermittelt den Typ der Variablen `$a`. Mögliche Rückgabewerte sind u. a.: `boolean`, `integer`, `double` und `string`.
`settype($a,Typ)`	Setzt den Typ der Variablen `$a`. Mit `$a="1 Dutzend"` ist `$a` vom Typ `string`. Nach `settype($a,"integer")` wird `$a` zu einer Integer-Variable mit dem Wert 1. Mögliche Werte für Typ sind u. a.: `boolean`, `integer`, `double` und `string`.

Menschen, nicht Maschinen: Brian Behlendorf

Brian Behlendorf aus dem sonnigen Südkalifornien schuf mit »Apache« den einflussreichsten und wichtigsten Webserver im Internet. Behlendorf studierte Anfang der neunziger Jahre an der Universität von Berkeley, Kalifornien, und gründete bereits 1993 – zwei Jahre vor dem großen Internetboom – zusammen mit Jonathan Nelson eine Webdesign-Firma. 1994 entwickelte er »Hotwired«, die Website für das Kult-Magazin »Wired«. Bei der Arbeit an Hotwired stellte man fest, dass die bisherigen Webserver nicht leistungsfähig genug waren und begann mit der Arbeit an einem eigenen Server, der auf den Namen Apache getauft und zu dessen professionellen Entwicklung im Februar 1995 die Apache Group gegründet wurde. Brian Behlendorfs private Homepage finden Sie hier: http://brian.behlendorf.com/