Verwendete Befehle:

array()
print_r inkl. echo <“pre“>
var_dump()
foreach()
count
rand()

Übung: Zufallsbilder, Preisliste, Umfrage

Arrays

Die Typen von Variablen, die bisher besprochen wurden, speichern **genau einen Wert**. Manchmal möchte man aber gleichzeitig mit mehreren Werten arbeiten, beispielsweise mit einer Liste von möglichen Farben, einer Liste von Gästen, einer Liste von zur Verfügung stehenden Versionen oder Sprachen, einer Liste von Preisen oder Produkten etc. Genau dafür sind Arrays gedacht, die mitunter auch „Feldervariablen“ genannt werden.

Bei **Arrays werden einer Variablen mehrere Werte** zugeordnet. Wenn man in einer Variablen mehrere Werte speichert, stehen viele nützliche Möglichkeiten offen: Die Werte lassen sich sortieren und neu ausgeben, man kann auf einzelne gezielt zugreifen, sie vergleichen, zählen, weitere ergänzen und wieder ausgeben lassen.

Beispiel: Array der Wochentag

Index/Schlüssel	Wert
0	Montag
1	Dienstag
2	Mittwoch
3	Donnerstag
4	Freitag
5	Samstag
6	Sonntag

Statt mit vielen Variablennamen arbeiten zu müssen, kann man mit Hilfe von Arrays einen bestimmten Wert in einem Array durch seinen Index abrufen (ausgeben), den man in eckigen Klammern hinter den Namen des Arrays setzt:

```
echo $tag[2];
```

Dies entspricht in dem Beispiel: Mittwoch

1)Numerisches Array

Dieses Beispiel zeigt ein sogenanntes „**numerisches Array**“. Dabei greift man auf die einzelnen Werte über einen numerischen Index (0,1,2,3…) als Schlüssel zugreift.
Später folgt noch das „assoziative Array“.

Arrays erstellen

Erstelle zuerst eine neue php-Datei „1.arrays.php“.
Um ein Array zu erstellen, verwendet man das **Schlüsselwort array().** Hier einmal ein Beispiel für ein einfaches Array mit drei Elementen, getrennt durch Beistriche und keine Strichpunkte!

```
$antworten = array("nie", "manchmal", "oft");
```

In den Klammern hinter array() führt man bei der Definition eines Arrays die einzelnen Werte durch Komma getrennt auf. Wenn es Strings sind, schreibt man sie wie gewohnt in Anführungszeichen. Zahlen notiert man ohne:

```
$werte = array(42, 66, 3.5, 55, 7);
```

Innerhalb eines Arrays können auch verschiedene Typen kombiniert werden:

```
$antworten = array("nie", "manchmal", "oft", 42);
```

Bei der **Namensgebung** für Arrays gelten die gleichen Regeln wie bei Variablen. Erlaubt sind Buchstaben, Ziffern und der Unterstrich, und der Name darf nicht mit einer Ziffer beginnen.

Kleine Arrays lassen sich problemlos in einer Zeile definieren. Sobald man aber mehrere Werte speichern will, wird es schnell unübersichtlich. Daher sollte man mit Hilfe von Einrückungen und mehreren Zeilen für Übersicht sorgen:

```
<?php
        $tag = array (
                "Montag",
                "Dienstag",
                "Mittwoch"
                );
?>
```

Die einzelnen Elemente werden von PHP automatisch durchnummeriert.
Die **Nummerierung beginnt dabei – das ist wichtig – bei 0**.
Das ist der sogenannte Index. Um ein einzelnes Element auszulesen, schreiben Sie den Namen des Arrays und in eckigen Klammern den Index:

```
echo $antworten[0]; /* nie */
echo "<br />\n";
echo $antworten[2]; /* oft */
```

```
 8  <body>
 9  <?php
10
11  $antworten = array("nie", "manchmal", "oft");
12  $werte = array(42, 66, 3.5, 55, 7);
13  echo $antworten[0]; /* nie */
14  echo "<br />\n";
15  echo $antworten[2]; /* oft */
16
17
18  ?>
```

Man kann Arrays auch problemlos im Nachhinein mit weiteren Elementen ergänzen.
Nehmen wir noch einmal das bestehende Array:

```
$antworten = array("nie", "manchmal", "oft", 42);
```

Wenn man versucht, das Array als Ganzes per echo auszugeben, sieht das Ergebnis nicht wie gewünscht aus:

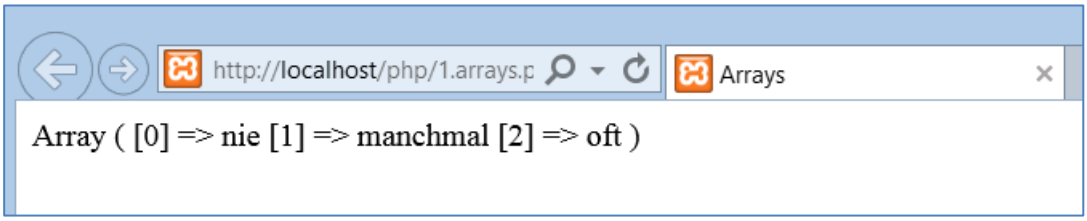
```
echo $antworten;
```

Das schreibt einfach »Array« auf den Bildschirm.

Um sich schnell einen Überblick über die Inhalte zu verschaffen, ist die **PHP-Funktion print_r()** praktisch.

```
print_r($antworten);
```

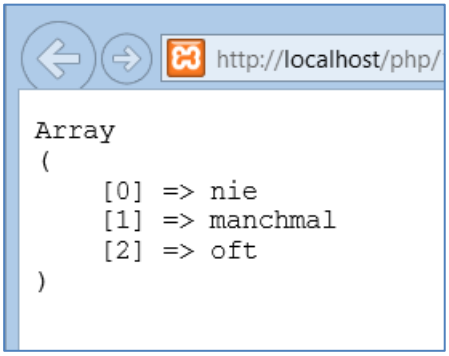
Ergebnis:



Will man den Browser dazu bringen, die Inhalte wie im Quellcode anzuzeigen, inklusive aller Leerzeichen, kann man das ansonsten selten verwendete **HTML-Element pre** benutzen und die Ausgabe von print_r() innerhalb der Start- und Endtags von pre schreiben:

```
echo "<pre>";
print_r($antworten);
echo "</pre>";
```

Ergebnis:



Noch ausführlichere Informationen über das Array erhält man, wenn man anstelle von print_r() die Funktion var_dump() benutzt:

```
echo "<pre>";
var_dump($antworten);
echo "</pre>";
```

Man sieht dann gleichzeitig, um welchen Datentyp es sich handelt, und bei Strings auch ihre Länge.

Ergebnis:

Übung „obst.php“
Erstelle ein neues php-File und darin das Array $obst aus Apfel, Birne, Kirsche, Kiwi und Banane. Gib danach alle mit Hilfe von var_dump() aus.

Arrays durchlaufen mit foreach

Die Ausgabe mit var_dump() oder print_r() ist nur geeignet, um sich bei der Programmierung einen schnellen Überblick über den Inhalt zu verschaffen – man könnte diese Ausgabe nicht einem normalen Benutzer zumuten. Dafür gibt es andere Wege: Speziell für die Ausgabe oder sonstige Bearbeitung aller Elemente eines Arrays existiert die **Schleife foreach**.

Bei foreach werden Schritt für Schritt die einzelnen Elemente des Arrays durchlaufen und die festgelegten Anweisungen für jedes Element ausgeführt. Man muss foreach nicht sagen, wie oft es das durchführen soll, denn foreach wird durch die Anzahl der Arrayelemente selbst begrenzt.

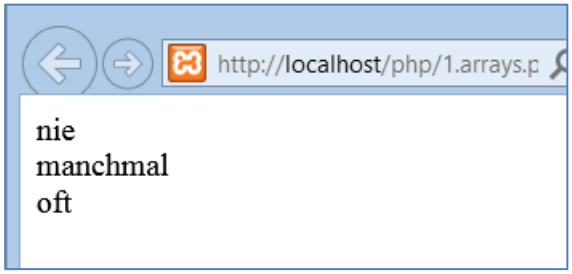
In **runden Klammern hinter foreach gibt man zuerst das Array** an, das man durchlaufen möchte. Danach folgt das Schlüsselwort as und danach der Name einer temporären Variablen, die den Wert der einzelnen Elemente zwischenspeichert. Der Name der Variable ist frei wählbar. In geschweiften Klammern steht der Code, der für jedes Element ausgeführt werden soll. Um jedes Element auszugeben, verwende den Namen, den man für die temporäre Variable eingesetzt hat.

Durch folgenden Code wird jedes Element des $antworten-Arrays ausgegeben – gefolgt jeweils von einem Zeilenumbruch:

```
foreach ($antworten as $aw) {
  echo "$aw <br />";
 }
```

```
 8  <body>
 9  <?php
10
11  $antworten = array("nie", "manchmal", "oft");
12  $werte = array(42, 66, 3.5, 55, 7);
13
14  foreach($antworten as $aw) {
15      echo "$aw <br>";
16  }
17
18  ?>
```

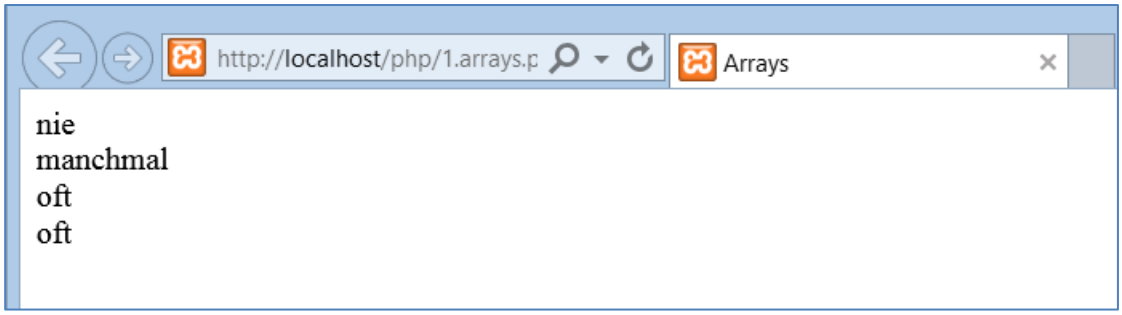
Ergebnis:



Wenn man außerhalb von foreach noch einmal auf die Variable $aw zugreift, erhält man den zuletzt dort gespeicherten Array-Wert:

```
foreach ($antworten as $aw) {
  echo "$aw <br />";
}
echo $aw;
```

Ergebnis:



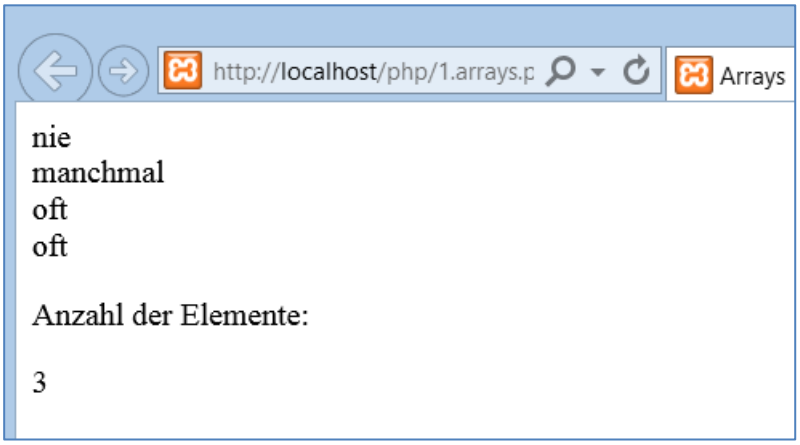
Um die Anzahl der Elemente eines Arrays zu ermitteln, kann man die **Funktion count()** einsetzen. Bei count() notiert man in runden Klammern das Array, dessen Elemente man zählen möchte. Als Rückgabewert erhält man die Anzahl der Elemente:

```
$anzahl = count($antworten);
echo $anzahl;
```

Gib einen HTML-Code ebenfalls an:

```
<body>
<?php

$antworten = array("nie", "manchmal", "oft");
$werte = array(42, 66, 3.5, 55, 7);

foreach($antworten as $aw) {
    echo "$aw <br>";
}

echo $aw;
?>
<p>Anzahl der Elemente: </p>

<?php
$anzahl = count($antworten);
echo $anzahl;

?>
</body>
</html>
```

Ergebnis:



Übung: 1.orte.php

Erstellen Sie ein Array mit fünf Orten. Lassen Sie dann alle Orte in einer foreach-Schleife ausgeben, wobei nach jedem Ort immer ein Zeilenumbruch
 eingefügt werden soll.

Beispiel: Zufällig ein Bild anzeigen lassen

Jetzt ein kleines Beispiel für die Verwendung von Arrays. Es soll zufällig eines von mehreren Bildern ausgegeben werden. Die Pfade zu den Bildern werden dafür in einem Array gespeichert.

Außerdem benötigen wir eine Funktion, die eine zufällige Zahl ermittelt. Genau dafür gibt es rand(). rand() erwartet in runden Klammern zwei Werte: Der eine bestimmt den minimalen Wert der Zufallszahl, der andere gibt den höchsten möglichen Wert an:

```
$zufallszahl = rand(0, 4);
```

Damit ist eine Zahl von 0 bis einschließlich 4 in $zufallszahl gespeichert.
Kommen wir zur zufälligen Ausgabe von Bildern:

```
01   <!DOCTYPE html>
02   <html>
03    <head>
04     <meta charset="UTF-8" />
05     <title>Zufallsbilder</title>
06   </head>
07   <body>
08   <?php
09   $bilder = array("barbara.jpg", "edin.jpg",
10                   "dennis.jpg", "julian.jpg",
11                   "alex.jpg");
12   $max = count($bilder) - 1;


13   $zufallszahl = rand(0, $max);
14   echo "<img src='$bilder[$zufallszahl]' height='200' width='150' />";
15   ?>
16   </body>
17   </html>
```

Speichern unter „zufallsbilder.php“.

In Zeile 9 wird ein Array namens $bilder angelegt. Es beinhaltet die Pfade zu den Bildern, die sich in demselben Ordner befinden wie das PHP-Skript selbst.

Zeile 12 ermittelt die Anzahl der Elemente des Arrays und zieht 1 davon ab. Damit haben wir in $max den höchsten Index des Arrays. Im Beispiel enthält das Array 5 Elemente. Der letzte Index ist aber 4 – da beim Index mit 0 zu zählen begonnen wird –, also eins weniger.

Zeile 13 ruft die Funktion rand() auf. Sie soll eine Zahl zwischen 0 und dem in $maxgespeicherten höchsten Index generieren. Diese wird in der Variablen $zufallszahlgespeichert.

In Zeile 14 erfolgt die Ausgabe des Zufallsbildes über das hierfür benötigte img-Element, das beim Attribut src den Pfad zur Datei erwartet. Hier wird auf das Array $bilder zurückgegriffen und als Index die Variable $zufallszahl benutzt, die ja einen Wert zwischen 0 und dem letzten Index enthält. Damit wird immer ein anderes Bild aus dem Bilderarray ausgelesen.

Wenn man das Skript testet, klicke mehrmals auf den Reload-Button: Welche Bilder angezeigt werden, wird zufällig bestimmt.

Übung:

Ändere das Beispiel *zufallsbilder.php* so ab, dass zufällig einer von mehreren Texten angezeigt wird. Dafür muss man natürlich zuerst ein Array mit mehreren Strings definieren!

2)Assoziative Arrays

Bei assoziativen Arrays steht keine fortlaufende Zahlenreihe als Index, sondern anstelle des Index ein String, der eine bestimmte Bedeutung hat. Bisher haben wir die einzelnen Elemente über Nummern angesprochen. Manchmal möchte man aber die Arrayelemente über Namen ansprechen. Solche Schlüssel-Wert-Paare kann man einsetzen, wenn man beispielsweise

- Farbnamen den entsprechenden in HTML/CSS üblichen hexadezimalen Farbbezeichnungen zuordnen möchten. Oder um
- Vorwahlnummern Städten zuzuordnen,
- Produktklassen zu Mehrwertsteuersätzen usw.

Beispiele:

Schlüssel	Wert
USB-Stick	6,99 €
Rapid	3 Tore
Partei 1	35%

Vordefinierte assoziative Array in PHP:

Es gibt auch viele in PHP vordefinierte assoziative Arrays. So kann man über

$_SERVER["PHP_SELF"] auf den Pfad zum aktuellen Skript zugreifen oder über
$_GET["name"] oder
$_POST["name"] auf den Inhalt von Formulardaten.

Zur Erstellung eines assoziativen Arrays verwende wieder array(), schreibe aber in runde Klammern immer die Schlüssel-Wert-Paare, die durch => verknüpft werden,

d.h. die Zuweisung erfolgt mit dem Operator =>

Das sind zwei direkt aufeinanderfolgende Zeichen, das Gleichheits- und das Größer-als-Zeichen.

Übung Farben:

Öffne wieder die Datei „1.arrays.php“ zum Weiterarbeiten:

```
$farben = array ("rot"  => "#FF0000",
                 "grün" => "#00FF00",
                 "blau" => "#0000FF");
```

Ein assoziatives Array bearbeiten:
Falls man die Funktion array() nicht einsetzen will, kann man die Elemente eines assoziativen Arrays auch **einzeln definieren**:

```
$farben["rot"]  = "#FF0000";
$farben["grün"] = "#00FF00";
```

Auf diese Art lassen sich auch nachträglich weitere Elemente **ergänzen**:

```
$farben["schwarz"] = "#000000";
```

Die einzelnen Werte kann man mit mehreren Varianten **ausgeben**:

z.B.
```
echo $farben["rot"];
echo "<p>Wir haben den Farbecode für rot $farben[rot] gefunden.</p>";
echo "<p>Wir haben den Farbecode für rot ", $farben["rot"], "gefunden.</p>";
```

Die Varianten für das echo:

- Man setzt den Schlüssel (hier rot) in Anführungszeichen und in eckige Klammern hinter den Variablennamen.
- Man verwendet bei zusätzlichen Texten die Anführungszeichen am Beginn und am Ende des Strings, verzichtet aber auf Anführungszeichen innerhalb des Strings.
- Man hängt eine Array-Variable mit einem Komma an einen String an.

Code:

```
26  $farben = array ("rot"  => "#FF0000",
27                   "grün" => "#00FF00",
28                   "blau" => "#0000FF");
29
30
31  echo $farben["rot"];
32      echo "<p>Wir haben den Farbecode für rot $farben[rot] gefunden.</p>";
33      echo "<p>Wir haben den Farbecode für rot ", $farben["rot"], " gefunden.</p>";
34
```

Ergebnis:

3#FF0000

Wir haben den Farbecode für rot #FF0000 gefunden.

Wir haben den Farbecode für rot #FF0000 gefunden.

Einen schnellen Überblick über den Inhalt eines Arrays verschafft man sich wiederum mit **print_r() oder var_dump():**

Beispiel: var_dump($farben);

```
25
26  $farben = array ("rot"  => "#FF0000",
27                   "grün" => "#00FF00",
28                   "blau" => "#0000FF");
29
30  var_dump($farben);
31
```

Ergebnis:

```
array(3) { ["rot"]=> string(7) "#FF0000" ["grün"]=> string(7) "#00FF00" ["blau"]=> string(7) "#0000FF" }
```

Foreach-Schleife

Um ein assoziatives Array der Reihe nach zu durchlaufen, benötigt man eine foreach-Schleife. Um die Inhalte ansprechender auszugeben, braucht man **foreach**. In runden Klammern gib zuerst den Namen des Arrays an, das durchlaufen werden soll. Dann folgen das Schlüsselwort as und zwei Variablen, die als temporäre Speicher für jeweils den Schlüssel und den dazugehörigen Wert dienen und durch => getrennt werden.
Für die beiden Variablen kann man eigene Bezeichnungen wählen, z.B. $key und $value.

```
foreach ($farben as $k => $v){
  echo "Schlüssel: $k, Wert: $v<br />\n";
  }
```

```
26  $farben = array ("rot"  => "#FF0000",
27                   "grün" => "#00FF00",
28                   "blau" => "#0000FF");
29
30  foreach ($farben as $k => $v) {
31      echo "Schlüssel: $k, Wert: $v <br />";
32      }
```

Ergebnis:

```
Schlüssel: rot, Wert: #FF0000
Schlüssel: grün, Wert: #00FF00
Schlüssel: blau, Wert: #0000FF
```

Übung Preisliste:

Erstelle eine Datei „preiseliste.php“ für ein assoziatives Array mit folgenden Getränken: Rotwein 2,- €; Spritzer 2,- €; Fanta 2,- €; Bier 3,- €; Schnaps 3,- €.
Gib die Liste mit foreach aus.

Übung Beruf:

Erstelle folgernde Datei „beruf.php“

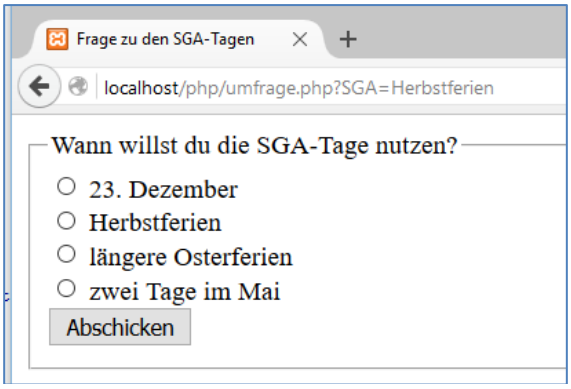
```
7
8   <body>
9   <?php
10  $beruf = array("Agent", "Rennfahrer", "Tänzer", "Koch");
11  foreach($beruf as $element)
12  {
13      echo $element . "<br>";
14  }
15
16  ?>
17
```

Beispiel: Formulare über Arrays gestalten

Mit Array lassen sich Auswahlfelder in Formularen besonders schnell erstellen.
Normalerweise muss man bei Optionen, Checkboxen oder Dropdown-Feldern jeden einzelnen Eintrag mit <input> einfügen. Legt man die einzelnen Werte aber in einem Array ab, tut man sich leichter.
Diese Möglichkeiten innerhalb des Array werden über eine foreach-Schleife ausgegeben.

Erstelle eine neue php-Datei mit dem Namen „umfrage.php“

Ergebnis:



Hier wurde vor dem Start der HTML-Info bereits per PHP der Inhalt des Formulars angegeben. Dies geschah mit Hilfe eines Arrays.
Im <body> wird dann mit foreach darauf zugegriffen.

```php
<?php
    $optionen = array (
        "Herbstferien",
        "23. Dezember",
        "längere Osterferien",
        "zwei Tage im Mai",
    );
    sort($optionen);
?>

<!doctype html>
<html>
<head>
<meta charset="utf-8">
<title>Frage zu den SGA-Tagen</title>
</head>
<body>
<form>
    <fieldset>
        <legend>Wann willst du die SGA-Tage nutzen?</legend>
        <?php
            foreach($optionen as $value) {
                echo "<input type='radio' name='SGA' value='$value'> $value <br>";
            }
        ?>
      <input type="submit" value="Abschicken">
    </fieldset>
</form>
</body>
</html>
```

Quelle:

Gisbert Damaschke in PHP & MySQL, Wiley-Vch Verlag, 2015, S. 169-205