PHP lernen

http://PHP.lernenhoch2.de/lernen/

---

1. PHP Einleitung

1.1. Motivation

Für eine dynamische Webseite ist HTML nicht ausreichend, da muss eine knackige Skriptsprache wie PHP her. Die Grundzüge von PHP sind schnell erlernt und dennoch bietet diese Skriptsprache genügend Potential, mit allen Alternativen (z.b. ASP, Java, Ruby, etc.) mitzuhalten.

PHP ist kostenlos, Server die PHP unterstützen findet man wie Sand am Meer und eine riesige Community unterstützt die Weiterentwicklung. Zudem hat die Beliebtheit für PHP etliche **Frameworks** und andere Systeme hervorgebracht, die ebenfalls kostenlos sind. So basiert z.b. das beliebteste Blog-System **Wordpress** auf PHP.

PHP nur für kleine Webseiten?

Ob klein oder groß, PHP wird durchweg genutzt:

> *As of April 2007, over 20 million Internet domains had web services hosted on servers with PHP installed and mod_php was recorded as the most popular Apache HTTP Server module.[34] Significant websites are written in PHP including the user-facing portion of Facebook,[35] Wikipedia (MediaWiki),[36] Yahoo!,[37] MyYearbook,[38] Digg, Joomla, WordPress, YouTube in its early stages, Drupal, Tagged[39] and Moodle [40].*

Quelle

Also Facebook, Digg, Youtube und andere große Seiten sind sich nicht zu Schade, PHP einzusetzen, warum sollte es deinen Ansprüchen nicht genügen? ;)

- **The PHP web sites all PHP developers ought to know about**
- **Which big sites use PHP?**

Ich wünsche dir nun viel Spass beim **lernen von PHP**

1.2. Wie funktioniert PHP

Quelle

Die Funktionsweise von PHP ist garnicht so schwer: Ein Besucher ruft über seinen Browser ein PHP-Skript auf deinem Server auf, z.b.: **http://domain.de/mein-skript.php**. Über das Internet wird diese Anfrage an deinen Server gestellt, auf dem das Skript liegt (wenn es dort kein Skript gibt, bekommt der Besucher eine Fehlermeldung).

Der Webserver sucht nun auf der Festplatte nach der PHP-Datei, wenn er sie gefunden hat, wird sie an den **PHP Interpreter** übergeben. Der PHP Interpreter verarbeitet dein PHP-Skript und übergibt das Ergebnis an den Besucher. Das Ergebnis ist in der Regel HTML, kann aber auch CSS, Javascript und sogar Bilder sein
(ja, PHP kann auch Bilder erzeugen :)).

Wann setze ich PHP ein?

PHP kann sehr viele Formate erstellen (CSS, Javascript, Bilder), doch meist wird es für die Ausgabe von HTML genutzt. Nun kommt die Frage auf "Warum soll ich PHP überhaupt verwenden, wenn ich meine HTML-Datei direkt erzeugen kann?". Das ist leicht beantwortet, eine HTML-Datei ist statisch, dass heißt, wenn du eine Tabelle mit 10 Einträgen in HTML schreibst, können nur diese 10 angezeigt werden. PHP ist dynamisch und kann dir eine Tabelle mit einer Zeile oder mit tausend Zeilen erzeugen, je nach Bedarf.

Alles was irgendwie dynamisch sein soll auf deiner Seite, muss mit PHP gemacht werden, denn HTML ist nur statisch:

- Ein Besuchercounter (bei jedem Besuch +1)
- Ein Kontaktformular (das Formular selbst wird rein in HTML erzeugt, aber die Verarbeitung der Formular-Daten erfolgt mit PHP)
- ein Gästebuch (wir wollen ja nicht bei jedem neuen Eintrag die HTML Datei aktualisieren müssen, dass soll PHP automatisch und in Echtzeit machen)
- Anzeige welches Datum heute ist
- usw.

*Wenn dich PHP interessiert, dann schau dir mal das **PHP Tutorial** an.*

1.3. Dein eigener Server

Der erste Schritt zum **lernen von PHP** und **MySQL** ist es, auf seinem Rechner einen Apache Server zu installieren. Damit kann man seine PHP-Skripte direkt testen bzw eine Datenbank anlegen und mit MySQL Befehlen rumspielen. Die einfachste Möglichkeit dafür, die ich finden konnte, ist **Apache Friends – XAMPP**

Wähle einfach die für dein Betriebssystem passende Version von XAMPP aus (als bsp.: **XAMPP für Windows**), lade die passende Datei herunter (entweder XAMPP oder XAMPP lite) und installiere XAMPP (ich empfehle die selbstentpackende .EXE, da man so nichts falsch machen kann).

Den Server starten

Nach der Installation ist der Server nun bereit, aber um ihn benutzen zu können, muss er erst noch über das "XAMPP Control Panel" gestartet werden. Schau dir dazu auch den Bereich **XAMPP Server starten, stoppen und testen** an.
Bedenke das der Server bei jedem Neustart deines Rechners erst wieder gestartet werden muss. Wenn du also über localhost (*htt://localhost/*) eine PHP-Datei aufrufst, dir aber nichts ausgegeben wird (bzw. ein Fehler), dann prüfe erstmal ob der Server auch läuft.

Bei Fragen und Problemen mit dem XAMPP Server ist die **XAMPP Friends Website** dein Freund. Dort gibt es viele Anleitungen und Tipps die dich schnell zur Lösung führen werden.

1.4. PHP Editoren

Ein wirklich guter Editor, für Webprogrammierung allgemein, ist **Aptana**. Der Aptana Editor basiert auf Eclipse, ist kostenlos, unterstützt alle nötigen Web-Sprachen (CSS, HTML, XML, PHP, etc.) und beinhaltet viele weitere Features. Ich nutze Aptana schon seit über einem Jahr und er ist meiner Meinung nach, einer der besten Editoren für PHP.

Für PHP Anfänger ist die Funktionsvielfalt in der Regel erdrückend, zumal man gerade anfangs auch problemlos mit einem simplen Texteditor seine Skripte schreiben kann. Doch wer das Anfänger Tutorial erstmal durchgearbeitet hat, der ist mit Aptana auf der sicheren Seite.

Hier noch eine Liste mit anderen PHP Editoren: **PHP-Editor, PHP-Tools und andere PHP-Software**

---

2. PHP Anfänger

Wenn du deinen eigenen Server schon gestartet hast und bereit bist, deine ersten Zeilen in PHP zu schreiben, bist du hier genau richtig. Ich setze für das PHP Anfänger Tutorial keinerlei Wissen in PHP oder anderen Skriptsprachen voraus. Es wäre praktisch, wenn du Grundkenntnisse in HTML hast, aber zwingend nötig ist auch das nicht.

Du solltest die Lektionen von oben nach unten durcharbeiten, da die jeweils folgende Lektion auf dem Wissen der vorigen aufbaut. Ich wünsche dir viel Spass und Erfolg beim **lernen von PHP**.

2.1. Das erste PHP Skript – Ausgaben mit dem “echo” Befehl

Überprüfe als erstes, ob **dein Server** läuft bzw. ob auf deinem Web-Server PHP zur Verfügung steht (wovon man heutzutage eigentlich ausgehen kann). Der Server muss laufen, damit unser PHP-Skript vom Server verarbeitet werden kann (=> sprich: Damit wir was ausgegeben bekommen).

Hallo Welt

Erstelle eine Datei mit dem Namen “index.php” und trage folgenden Code ein:

```
1 <?php
2     echo 'Hallo Welt';
3 ?>
```

Wenn du die Datei nun in deinem Browser aufrufst, sollte folgender Text ausgegeben werden:

```
1 Hallo Welt
```

Erklärung

Jeglichen PHP-Code den du schreibst, musst du innerhalb von **<?php … ?>** schreiben, damit PHP erkennt, wo der Code anfängt und wo er endet. Mit dem “echo”-Befehl kannst du Text auf dem Bildschirm ausgeben. Diesen Befehl wirst du sehr häufig benutzen, unter anderem auch um HTML-Text auszugeben:

```
1 <?php
2     echo '<h2>Dies ist HTML-Text</h2>';
3 ?>
```

2.1.1. echo mit Anführungszeichen ” und Hochkomma ‘

Den echo-Befehl kann man mit Anführungszeichen und mit Hochkommata nutzen. Also beides ist möglich:

```
1 <?php
2     echo "Hallo Welt";
3     echo 'Hallo Welt';
4 ?>
```

Unterschied zu Variante 1 (Anführungszeichen “) und Variante 2 (Hochkomma ‘)

Unterschied 1 – mit Backslash Zeichen entwerten

Möchtest du in Variante 1 innerhalb deiner Ausgabe ein Anführungszeichen verwenden, musst du dieses extra “escapen” mit dem Backslash-Zeichen:

```php
<?php
    echo "...und er sagte es wäre \"wenig\" Arbeit";
?>
```

Da PHP in dieser Variante die “-Zeichen dazu benutzt, den String einzugrenzen, müssen wir die Zeichen innerhalb des Strings mit einem Backslash “entwerten”, ansonsten würden wir einen Fehler erhalten. In Variante 2 ist das nicht nötig, da zum eingrenzen das Hochkomma genutzt wird. Möchten wir aber innerhalb des Strings Hochkomma ausgeben, müssen wir die wiederum entwerten. Nach meiner Erfahrung braucht man aber häufiger das Anführungszeichen als das Hochkomma, deshalb nutze ich Variante 2.

Unterschied 2 – Variablen im String

Du kannst Variablen innerhalb des Strings in Variante 1 packen und PHP wird automatisch den Wert einfügen:

```php
<?php
    $name = "Marie";

    echo "Ich heiße $name";
?>
```

In Variante 2 würdest du mit dieser Methode folgende Ausgabe erhalten: **Ich heiße $name**
Ich finde es allerdings unsauber, Variablen direkt in den String zu packen. Meiner Meinung nach sollte man Variablen mit dem String konkatenieren, dass macht den Code deutlich lesbarer und verständlicher:

```php
<?php
    $name = "Marie";

    echo 'Ich heiße '.$name;
?>
```

Unterschied 3 – Steuerzeichen

Folgende Ausgabe wird auf dem Monitor in einer Zeile ausgegeben:

```php
<?php
    echo "Hallo Zeile 1";
    echo "Hallo Zeile 2";
?>
```

Mittels Steuerzeichen kannst du einen Zeilenumbruch erzwingen:

```php
<?php
    echo "Hallo Zeile 1 \n";
    echo "Hallo Zeile 2";
?>
```

\n ist das Steuerzeichen für den Zeilenumbruch. Du kannst es direkt in den String schreiben und es wird “erkannt”. Mit der Hochkomma-Variante funktioniert das nicht.

Fazit

Ich bevorzuge die Hochkomma-Variante, da ich sie sauberer finde, aber es ist geschmackssache. Meine gründe dafür sind:

- Ich schreibe Variablen eh nicht direkt in den String (weil ich es für “unsauberen” Code halte)
- Ich muss in einem String die Anführungszeichen mit Backslash entwerten
- Ich verwende selten Steuerzeichen wie Zeilenumbruch (\n) oder Tabulator (\t) direkt im String, sondern versuche es mittels HTML umzusetzen

Andererseits kommt es auch auf den Einzelfall an. Du wirst in den folgenden Kapiteln häufig sehen, dass ich anstelle von Hochkommata auf Anführungszeichen zurückgreife. Es ist halt geschmackssache ;)

2.2. Variablen

Im vorigen Teil hast du den “echo”-Befehl kennengelernt und herausgefunden, wie man Text ausgibt. Nun kommt ein weiterer, sehr wichtiger Teil von PHP und zwar die **Variablen**. Sollte dir etwas unklar sein, stell deine Frage über einen Klick auf die Sprechblase auf der linken Seite.

Variablen – Einführung

Erstmal ein kleines Beispiel:

```
1 <?php
2     $zustand = "gut";
3     echo "Mir geht es ".$zustand;
4 ?>
```

Erzeuge ein neues PHP-Skript mit dem Namen “variablen1.php”, trage den Code darin ein und führe es in deinem Browser aus. Als Ergebnis sollte dir folgendes angezeigt werden:

```
1 Mir geht es gut
```

Zeile 1 und 4 kennst du schon aus dem vorigen Teil (**das erste PHP Skript**). Auch den “echo”-Befehl hast du schon gesehen und weißt, dass damit Text ausgegeben wird. Nun kommt mit Zeile 2 etwas neues hinzu und zwar eine **Variable**.

Was sind Variablen und wofür gibt es die in PHP?

Variablen sind Behälter, die einen Wert haben. Der Wert kann

- eine Zahl sein, z.b. 5 oder 14 oder auch 139283454627236 (jede beliebige Zahl halt)
- ein Text, z.b. “Wieviel Uhr ist es?”, “Wie geht es dir?”, “Mir ist langweilig”
- ein Buchstabe, z.b. “a”, “r”, “x”

und noch vieles mehr. Doch erstmal bleiben wir bei Zahlen und Texten (bzw. Buchstaben). Ausserdem kannst du den Wert einer Variable ausgeben (z.b. mit dem “echo”-Befehl) oder verändern. Eine Variable beginnt immer mit dem Dollarzeichen, gefolgt von Buchstaben, Zahlen und Unterstrichen, die den Variablen-Namen definieren.

So, du weißt also nun grob was eine Variable ist (wenn nicht frag einfach nach). Du weißt auch, dass eine Variable einen Wert besitzt, doch wie erhält sie den Wert?

```
1 <?php
2     $alter = 25;
3 ?>
```

Um einer Variable einen Wert zuzuweisen, schreibst du erstmal die Variable. Dann folgt ein “=” was soviel heißt wie “packe den Wert auf der rechten Seite vom “=” in meine Variable auf der linken Seite”. Merke dir: **Die Variable die Links vom “=” steht, bekommt immer den Wert der rechts vom “=” steht.**

Variablen haben also einen Wert und diesen Wert können wir ausgeben:

```
1 <?php
2     $zahl = 12;
3     echo $zahl;
4 ?>
```

Wir legen eine Variable mit der Bezeichnung “zahl” an und weisen ihr den Wert 12 zu. Teste das Skript und du wirst sehen, dass auf dem Bildschirm “12″ ausgegeben wird. Ändere nun den Wert der Variable Zahl auf einen beliebigen anderen Wert und schau dir das Ergebnis an.

Nun schauen wir uns nochmal das Beispiel vom anfang an:

```
1 <?php
```

```
    $zustand = "gut";
    echo "Mir geht es ".$zustand;
?>
```

Die Variable “Zustand” hat in diesem Fall keinen Wert vom Typ Zahl (**Int** für **Integer**), sondern einen Wert vom Typ Text (**String**). Lies hier nach was für **Variablen Typen** es in PHP gibt.

Also, $zustand hat den Wert “gut”, dass heißt, wenn wir $zustand ausgeben, wird uns “gut” angezeigt. Und genau das wollen wir in Zeile 3 erreichen, indem wir zuerst “Mir geht es” normal mit echo ausgeben, aber dann noch das “gut” dranhängen. Um einen Wert “dranzuhängen” bzw zu **konkatenieren**, gibt es den “.”
Schau dir die Ausgabe des Skripts an und ändere den Wert von “zustand” um das Prinzip von Variablen und dem konkatenieren von Werten zu verstehen.

Variablen-Wert während der Laufzeit ändern

Nun wollen wir mal sehen, was passiert, wenn wir den Wert einer Variablen während der Laufzeit des Skripts ändern:

```
<?php
    $zahl = 1;
    echo "Mein Wert ist ".$zahl." - ";

    $zahl = 2;
    echo "jetzt ist er ".$zahl;
?>
```

Ausgabe:

Mein Wert ist 1 – jetzt ist er 2

Der Wert einer Variable ist also nicht fest, wir können ihn beliebig verändern auch während das Skript läuft.

Variablen Zusammenfassung

Hier nochmal eine kurze Zusammenfassung zu Variablen:

- Eine Variable ist ein Behälter
- Hat einen Wert (Zahl oder Text)
- du kannst Variablen in deinem PHP-Skript anlegen
- Variablen bestehen aus **$** + **Bezeichnung** (bsp.: $a, $hallo, $uhrzeit, $aswdds)
- Ein gültiger Variablen-Name fängt mit einem Buchstaben (a-z) oder einem Unterstrich (_) an und kann dann eine beliebige Zahl von Buchstaben, Zahlen oder Unterstrichen haben (unten findest du eine Liste mit gültigen und ungültigen Variablen-Namen)

2.2.1. weitere Variablen Beispiele

Variable einer anderen Variable zuweisen

Wenn wir einer Variable einen Wert zuweisen, muss das nicht direkt eine Zahl oder ein Text sein. Wir können einer Variable auch einfach den Wert einer anderen Variable zuweisen:

```
<?php
    $a = 12;
    $z = $a;

    echo $z;
?>
```

Beide Variablen, also $a und $z, haben ab Zeile 4 den Wert “12″.

Ein einfacher(?) Tausch

Nun was kniffliges: Wir haben zwei Variablen, $a und $b und die Werte werden am Ende des Skripts ausgegeben.

```php
<?php
    $a = 100;
    $b = 200;

    //hier wird getauscht

    echo $a." - ".$b;
?>
```

Ausgabe:

100 – 200

Das ist soweit klar, bis auf Zeile 5. Dabei handelt es sich um einen Kommentar, die komplette Zeile wird von PHP ignoriert. Mit Kommentaren kannst du dir Notizen in deine Skripte schreiben, aber dazu später mehr.
Nun möchte ich, dass $a den Wert von $b erhält und $b den Wert von $a ohne an den Zeilen 2,3 und 7 was zu ändern. Ein direkter Tausch funktioniert nicht:

```php
<?php
    $a = 100;
    $b = 200;

    //hier wird getauscht
    $a = $b;
    $b = $a;

    echo $a." - ".$b;
?>
```

Ausgabe:

200 – 200

In Zeile 6 erhält $a den Wert von $b (200) und überschreibt damit den Wert 100. Was wir brauchen ist eine dritte Variable, die temporär den Wert von $a festhält:

```php
<?php
    $a = 100;
    $b = 200;

    //hier wird getauscht
    $temp = $a;
    $a    = $b;
    $b    = $temp;

    echo $a." - ".$b;
?>
```

Zeile 6: $temp erhält den Wert 100
Zeile 7: $a erhält den Wert 200
Zeile 8: den vorigen Wert von $a haben wir in $temp festgehalten und können ihn jetzt $b zuweisen; Der Tausch ist vollzogen.

Variablen miteinander addieren

Wir können Variablen Werte nicht nur ausgeben, sondern auch manipulieren. Dazu ein kleines Beispiel:

```php
<?php
    $zahl1 = 12;
    $zahl2 = 8;
    $ergebnis = $zahl1 + $zahl2;

    echo $ergebnis;
?>
```

Zeile 2 und 3 sind leicht, es werden zwei Variablen angelegt und jeweils ein Wert zugewiesen, doch was passiert in Zeile 4?

1. es wird eine Variable “ergebnis” angelegt
2. der Variable “ergebnis” wird ein Wert zugewiesen (alles was auf der rechten Seite vom “=” steht)

Doch anstelle einer Zahl oder eines Texts, stehen dort zwei Variablen und ein + Zeichen. Vielleich hast du es dir schon gedacht: Auf der rechten Seite werden die beiden Werte der Variablen zahl1 und zahl2 miteinander addiert und dieses Ergebnis wird dann der Variablen “ergebnis” zugewiesen. Eigentlich ganz einfach, teste es mal mit anderen Zahlen und mit anderen Rechenoperationen wie subtrahieren (-), multiplizieren (*) und dividieren (/) aus.

2.2.2. Beispiele für gültige und ungültige Variablen Namen

Der Name einer Variablen unterliegt ein paar Regeln, hier findest du gültige und ungültige Kombinationen.

Beispiele für gültige Variablen Namen

Denke daran, gültige Variablen-Namen fangen mit einem Buchstaben (a-z) oder einem Unterstrich (_) an und können dann eine beliebige Zahl von Buchstaben, Zahlen oder Unterstrichen haben.

- $a
- $a123
- $variable
- $platzhalter
- $uhrzeit
- $alter
- $name
- $_hallo
- $_123

Beispiele für ungültige Variablen Namen

Denke daran, Sonderzeichen etc sind nicht erlaubt im Variablen Namen. Lediglich Buchstaben, Zahlen und Unterstriche.

- $123
- $achtung!
- variable

2.2.3. Variablen Typen

Eine Variable kann unterschiedliche Werte aufnehmen, um Beispiel Zahlen oder Text. Allerdings ist es einer PHP Variable egal, welchen Wert du ihr gibst. So ist folgendes Beispiel problemlos möglich:

```
1 <?php
2     $zahl = 12;
3     echo $zahl;
4
5     $zahl = "dies ist ein Text";
6     echo $zahl;
7 ?>
```

Als erstes weisen wir der Variable “zahl” den Integer Wert 12 zu. Danach bekommt sie einen String zugewiesen und gibt den auch brav aus. PHP ist es also egal, welche Werte du in eine Variable packst, aber damit du weißt, welche Typen es gibt und worin sie sich unterscheiden, findest du hier eine Liste mit allen Wert-Typen für Variablen in PHP:

Integer

Bezeichnungen: int, integer, Ganzzahl
Beispiele:

```
1 <?php
```

```php
    $zahl1 = 2;
    $zahl2 = 15;
    $zahl3 = 4432552;
    $zahl4 = 0;
    $zahl5 = 234;
    $zahl6 = -5;
    $zahl7 = -39928;
?>
```

Ganze Zahlen ohne Nachkommastellen können mit dem Typ Integer gespeichert werden.

Float

Bezeichnungen: float, double, Fließkommazahl
Beispiele:

```php
<?php
    $float_zahl1 = 1.423;
    $float_zahl2 = 166343.23;
    $float_zahl3 = 0.234345;
?>
```

Anders als beim Typ Integer, können bei Float Werte mit Nachkommastellen gespeichert werden (und nur dafür sollte man float bzw. double nutzen, ansonsten integer)

String

Bezeichnungen: String, Zeichenkette, Text, Zeichen
Beispiele:

```php
<?php
    $string1 = "Hallo Welt";
    $string2 = "test";
    $string3 = "Klaus";
    $string4 = "a";
    $string5 = "xyz";
    $string6 = "123";
    $string7 = "1. Wahl";
?>
```

Boolean

Bezeichnungen: boolean, bool, Wahrheitswert
Beispiele:

```php
<?php
    $eingeloggt = true;
    $eingeloggt = false;
?>
```

Boolean ist mitunter der einfachste Typ, denn er kann nur zwei Werte haben: TRUE oder FALSE
Variablen vom Typ boolean werden z.b. für folgende Abfrage eingesetzt:

- ist der Benutzer eingeloggt? TRUE = Ja, er ist eingeloggt
- wurde die Taste “p” gerückt? FALSE = Nein, sie wurde nicht gedrückt
- hat sich der Benutzer für den Newsletter angemeldet? TRUE = Ja, hat er

Array

Bezeichnungen: array
Beispiel:

```php
<?php
    $zahlen = (1, 14, 82, 1002);
```

```php
    $namen = ("Peter", "David", "Henning", "Gerhard", "Norman");
?>
```

Ein Array ist eine Variable, die mehrere Werte in sich speichern kann. Mehr über den Typ Array findest du hier: Arrays.

Ist das alles?

Es gibt noch weitere, aber dies sind die wichtigsten und mit denen wirst du die meiste Zeit arbeiten.

2.3. Kommentare

Kommentare sind Zeilen in deinem Skript, die der PHP Interpreter einfach ignoriert. Dadurch kannst du dir super Notizen machen oder erklären was dein Code macht, damit du auch später noch dein Skript verstehst.

PHP Kommentare

In PHP gibt es zwei Möglichkeiten Kommentare zu erstellen:

Einzeilige Kommentare

```php
<?php
    echo "Hallo Welt";

    //Dies ist ein Kommentar

    echo "Hier wird wieder was ausgegeben";
?>
```

Mehrzeilige Kommentare

```php
<?php
    echo "Hallo Welt";

    /*
    Wenn man mal was mehr zu sagen hat und
    eine Zeile dafür nicht ausreicht, kann man leicht
    zu mehrzeiligen Kommentare greifen
    */

    //oder man macht es so
    //indem man jede Zeile auskommentiert
    //besser ist aber dann, direkt einen mehrzeiligen
    //Kommentar zu machen

    echo "Hier wird wieder was ausgegeben";
?>
```

2.4. Bedingungen

Bislang können wir zwar wie wild Variablen anlegen und ihnen Werte zuweisen, aber mehr als hier und da die Variablen auszugeben, ist nicht drin. Was machst du z.b. wenn du abhängig vom Wert einer Variablen etwas anderes ausgeben möchtest? Oder wenn du erstmal prüfen möchtest, ob eine Variable einen Wert hat, bevor du sie ausgibst? Lies weiter um zu erfahren, wie das geht.

Einfache Bedingung

```php
<?php
    $zahl = 5;

    if($zahl == 5)
        echo "Die Variable hat den Wert 5";
?>
```

Das Schlüsselwort “if” leitet unsere Bedingung ein. Der Ausdruck, der in der Klammer steht, muss **wahr** sein, damit Zeile 5 ausgeführt wird und damit auf dem Bildschirm “Die Variable hat den Wert 5″ erscheint.

Versteh ich nicht, bitte einfacher!

Eine Bedingung fängt immer mit dem Wort “if” an, gefolgt von einer öffnenden und einer schließenden Klammer:

```
1 <?php
2     if()
3 ?>
```

So alleine funktioniert das noch nicht, es fehlt nämlich noch der Ausdruck, der in der Klammer steht. Das kann vieles sein, zum Beispiel kann man zwei Werte miteinander vergleichen ($zahl == 5), oder einfach nur eine Variable reinschreiben ($eingeloggt) oder noch einfacher, man schreibt folgendes:

```
1 <?php
2     if(true)
3         echo "Die Bedingung ist wahr, deshalb siehst du diesen Text";
4 ?>
```

Oben hab ich schonmal angedeutet, dass der Ausdruck, der in der Klammer steht, wahr sein muss, damit die nächste Zeile ausgeführt wird. Das ist für Anfänger meist erstmal nicht so einfach zu verstehen (*bitte frag nach, wenn irgendwas unklar ist!*). Wann ist ein Ausdruck **wahr**? Wann ist er **falsch**? Wenn du also in deine Klammer “true” reinschreibst, ist der Ausdruck immer wahr. Das heißt, die Zeile, die der Bedingung folgt, wird immer ausgeführt. Das ist aber nur für Testfälle interessant, denn was würde es uns bringen, wenn eine Bedingung immer wahr ist, dann können wir die Bedingung auch weglassen, weil es immer ausgeführt wird (verstehst du?).

Ein Ausdruck ist wahr oder falsch

Unser Gerüst für eine Bedingung sieht also folgendermaßen aus:

```
1 <?php
2     if(Ausdruck)
3         echo "gib hier irgendwas aus";
4 ?>
```

Der Ausdruck muss also wahr sein, damit PHP die Zeile mit “echo” ausführt. Wenn der Ausdruck also falsch ist, wird die Zeile 3 nicht ausgeführt, ganz einfach.

```
01 <?php
02     $zahl = 12;
03     $name = "Hans";
04 
05     if($zahl == 11)
06         echo "Die Variable zahl hat den Wert 11";
07 
08     if($name == "Hans")
09         echo "Die Variable name hat den Wert Hans";
10 ?>
```

Die erste Bedingung ist falsch, weil unsere Variable “zahl” nunmal nicht den Wert 11, sondern den Wert 12 hat. Die zweite Bedingung hingegen ist wahr, denn unsere Variable “name” hat den Wert “Hans”. Achte darauf, dass ein Vergleich von zwei Werten immer mit dem doppelten Gleichheitszeichen erfolgt und eine Wertzuweisung mit einem einfachen Gleichheitszeichen.

Bedingung mit mehrzeiliger Anweisung

Bislang kannst du nur Bedingungen erstellen, die, wenn sie wahr sind, eine Zeile ausgeben. Doch was ist, wenn du mehrzeilige Anweisungen hast? Folgendes funktioniert nämlich nicht:

```
1 <?php
2     $name = "Hans";
3 
4     if($name == "Hans")
5         echo "Hallo Hans";
```

```
        echo "schön dich wiederzusehen";
?>
```

Wenn die Bedingung wahr ist, wird beides ausgegeben. Doch wenn du den Wert der Variable name änderst, wird nur noch das zweite echo ausgegeben. Woran liegt das? Ganz einfach, PHP schließt in die Anweisung nur die erste Zeile nach der Bedingung ein, also **echo "Hallo Hans";**
Die darauffolgende Zeile wird ganz normal ausgeführt, unabhängig von der Bedingung. Normalerweise sehe der Code auch so aus (ich habe es extra eingerückt, um es deutlich zu machen):

```
<?php
    $name = "Hans";

    if($name == "Hans")
        echo "Hallo Hans";

    echo "schön dich wiederzusehen";
?>
```

Damit obiges Beispiel korrekt funktioniert, musst du PHP klarmachen, dass du eine mehrzeilige Anweisung hast:

```
<?php
    $name = "Hans";

    if($name == "Hans")
    {
        echo "Hallo Hans";
        echo "schön dich wiederzusehen";
    }
?>
```

Alles was innerhalb der geschweiften Klammern steht, wird ausgeführt wenn die Bedingung wahr ist. Du solltest deine Zeilen ebenfalls so einrücken, durch gut strukurierten Code, wirst du Fehler schneller finden und dein Skript auch nach Monaten leicht verstehen.

Fazit

Bedingungen sind sehr wichtig, doch für Anfänger nicht gerade leicht zu verstehen. Gerade der Punkt, dass Ausdrücke wahr oder falsch sein können, wirft meist viele Fragen auf (frag nach!). Ich hab dieses Kapitel in mehrere Teile aufgesplittet, im nächsten Punkt wirst du **Bedingungen mit Alternativen** kennenlernen, also was passiert, wenn ein Ausdruck nicht wahr ist. Schau es dir an!

2.4.1. Bedingung mit Alternative – If, else

Bislang weißt du nur, wie man einfache Bedingungen erstellt, doch was ist wenn du eine Alternative haben möchtest? Also wenn die Bedingung nicht wahr ist, dann mache xyz?

Else – Bedingungen mit Alternative

Angenommen du hast ein Login-System auf deiner Webseite und möchtest prüfen, ob ein Benutzer eingeloggt ist und ihm eine Nachricht ausgeben:

```
<?php
    $eingeloggt = true;

    if($eingeloggt == true)
        echo 'Willkommen zurück!';
?>
```

Nun möchtest du dem Benutzer aber auch eine Nachricht ausgeben, wenn er nicht eingeloggt ist. Dafür gibt es das Schlüsselwort "else":

```
<?php
    $eingeloggt = true;

    if($eingeloggt == true)
```

```php
        echo 'Willkommen zurück!';
    else
        echo 'Bitte melde dich an.';
?>
```

Else funktioniert nur in Kombination mit dem Schlüsselwort “if” und muss unmittelbar nach der einzeiligen Anweisung erfolgen, bzw direkt nach einer abschließenden geschweiften Klammer:

```php
<?php
    $eingeloggt = true;

    if($eingeloggt == true)
    {
        echo 'Willkommen zurück!';
    }
    else
    {
        echo 'Bitte melde dich an.';
    }
?>
```

Auch “else” arbeitet mit einzeiligen und mehrzeiligen Anweisungen und braucht für zweiteres ebenfalls die geschweiften Klammern. Zur besseren Übersichtlichkeit und um Fehler zu vermeiden, empfehle ich Anfängern immer mit geschweiften Klammern zu arbeiten, auch bei einzeiligen Anweisungen.

Im nächsten Teil erfährst du, dass man mehr als eine Alternative pro If-Bedingung angeben kann.

2.4.2. Noch mehr Möglichkeiten – If, else if, else

Manchmal reicht eine Alternative einfach nicht aus. Deshalb können wir in PHP nicht nur “if” und “else” nutzen, sondern auch “if else” und davon soviele wie wir möchten. Sieh dir dazu erstmal folgendes Beispiel an:

```php
<?php
    $zahl = 1;

    if($zahl == 0)
        echo 'Die Variable "zahl" hat den Wert 0';
    else if($zahl == 1)
        echo 'Die Variable "zahl" hat den Wert 1';
    else
        echo 'Die Variable "zahl" ist ungleich 0 und 1';
?>
```

Wir können also mit “else if” bzw. “elseif” eine weitere Bedingung abfragen. Wenn also der Ausdruck der ersten if-Bedingung “falsch” zurückgibt, wird die Bedingung von “else if” geprüft. Gibt auch diese “falsch” zurück, nutzt die Bedingung die “else”-Anweisung und führt die Befehle darin aus.

Natürlich kann man “if” und “else if” auch ohne “else” nutzen. Ausserdem kann man beliebig viele “else if”-Abfragen nach dem ersten “if” setzen, doch dass das in den meisten Fällen besser mit **switch** gelöst wird, sehen wir später.

2.4.3. Vergleichsoperatoren

Bislang können wir nur Werte direkt miteinander vergleichen, also hat die Variable “zahl” den Wert “10″ oder hat die Variable “name” den Wert “Max”. Doch häufig reicht das nicht aus und man möchte z.b. wissen, ob die Variable einen Wert größer 10 hat oder einen Wert der zwischen 100 und 200 liegt. Dafür gibt es dann weitere Vergleichsoperatoren:

- **$a == $b** – prüft ob beide Werte gleich sind
- **$a != $b** – prüft ob die Werte ungleich sind
- **$a < $b** – prüft ob der linke Wert kleiner als der rechte Wert ist
- **$a > $b** – prüft ob der linke Wert größer als der rechte Wert ist
- **$a <= $b** – prüft ob der linke Wert kleiner oder gleich ist vom rechten Wert
- **$a >= $b** – prüft ob der linke Wert größer oder gleich ist vom rechten Wert

- **$a <> $b** – prüft ob die Werte ungleich sind (funktioniert genau wie **$a != $b**)
- **$a === $b** – prüft ob beide Werte gleich sind und ob beide vom selben Typ sind
- **$a !== $b** – prüft ob die Werte ungleich sind und ob beide einen unterschiedlichen Typ haben

Quelle

$a == $b

Das doppelte Gleichheitszeichen kennst du schon, damit können wir zwei Werte vergleichen, und wenn sie den gleichen Wert haben, dann ist der Ausdruck **wahr**.

Beispiele:

```php
<?php
    if(5 == 5)
        echo 'Der Ausdruck ist wahr';
?>
```

```php
<?php
    $a = 5;

    if($a == 5)
        echo 'Der Ausdruck ist wahr';
?>
```

```php
<?php
    $a = 5;
    $b = 5;

    if($a == $b)
        echo 'Der Ausdruck ist wahr';
?>
```

$a != $b

Das Gegenteil vom doppelten Gleichheitszeichen, prüft ob die beiden Werte ungleich sind. Wenn beide Werte ungleich sind, ist der Ausdruck **wahr**

Beispiele:

```php
<?php
    if(5 != 4)
        echo 'Der Ausdruck ist wahr, denn 5 ist ungleich 4';
?>
```

```php
<?php
    $a = 5;

    if($a != 4)
        echo 'Der Ausdruck ist wahr, denn 5 ist ungleich 4';
?>
```

$a < $b

Ist der Wert der Variable “a” kleiner als der Wert der Variable “b”, dann ist der Ausdruck wahr.

```php
<?php
    $a = 5;

    if($a < 10)
        echo 'Der Ausdruck ist wahr, denn 5 ist kleiner 10';
?>
```

```php
<?php
```

```php
    $a = 5;

    if($a < 6)
        echo 'Der Ausdruck ist wahr, denn 5 ist kleiner 6';
?>
```

$a > $b

Ist der Wert der Variable “a” größer als der Wert der Variable “b”, dann ist der Ausdruck wahr.

```php
<?php
    $a = 5;

    if($a > 4)
        echo 'Der Ausdruck ist wahr, denn 5 ist größer 4';
?>
```

$a <= $b

Ist der Wert der Variable “a” kleiner oder gleich als der Wert der Variable “b”, dann ist der Ausdruck wahr.

```php
<?php
    $a = 5;

    if($a <= 4)
        echo 'Der Ausdruck ist wahr, denn 5 ist größer 4';
?>
```

```php
<?php
    $a = 5;

    if($a <= 5)
        echo 'Der Ausdruck ist wahr, denn 5 ist gleich 5';
?>
```

$a >= $b

Ist der Wert der Variable “a” größer oder gleich als der Wert der Variable “b”, dann ist der Ausdruck wahr.

```php
<?php
    $a = 5;

    if($a >= 4)
        echo 'Der Ausdruck ist wahr, denn 5 ist größer 4';
?>
```

```php
<?php
    $a = 5;

    if($a >= 5)
        echo 'Der Ausdruck ist wahr, denn 5 ist gleich 5';
?>
```

$a <> $b

Ist der Wert der Variable “a” ungleich dem Wert der Variable “b”, dann ist der Ausdruck wahr. Genau wie der Ausdruck **$a != $b**

```php
<?php
    $a = 5;

    if($a <> 4)
        echo 'Der Ausdruck ist wahr, denn 5 ist ungleich 4';
?>
```

```php
<?php
```

```php
    $a = 10;
    $b = 1;

    if($a <> $b)
        echo 'Der Ausdruck ist wahr, denn 10 ist ungleich 1';
?>
```

$a === $b

Das dreifache Gleichheitszeichen funktioniert wie das doppelte Gleichheitszeichen und prüft den Wert beider Variablen. Doch zusätzlich prüft das dreifache Gleichheitszeichen noch, ob die beiden Variablen vom gleichen Typ sind. Dazu ein Beispiel:

```php
<?php
    $a = 0;
    if($a == false)
        echo 'Der Ausdruck ist wahr, denn der Wert 0 ist gleich false';
?>
```

Der Ausdruck in obigem Beispiel ergibt **wahr** denn der Wert integer Wert “0″ ist gleich dem boolischen Wert “false”. Der Ausdruck in folgendem Beispiel hingegen funktioniert nicht:

```php
<?php
    $a = 0;
    if($a === false)
        echo 'Dieses "echo" wird nicht ausgeführt.';
?>
```

Der Ausdruck ist falsch, denn auch wenn die Werte “0″ und “false” gleich sind, so sind die beiden Typen Integer (0) und Boolean (false) ungleich.

$a !== $b

Geprüft wird, ob die Werte der beiden Variablen unterschiedlich sind oder ob die Variablen einen unterschiedlichen Typen haben.

```php
<?php
    $a = 0;
    if($a !== false)
        echo 'Der Ausdruck ist wahr, denn der Wert "0" ist zwar gleich "false", aber die beiden Typen Integer und Boolean sind unterschiedlich.';
?>
```

2.4.4. Wahrheitswerte kombinieren

Bislang haben wir innerhalb der Bedingung nur für einen Wert geprüft, ob der Wert wahr oder falsch ist. Doch man kann mehrere Wahrheitswerte kombinieren:

AND (&&)

Häufig ist eine Bedingung von zwei Werten abhängig, also nur wenn beide Werte WAHR sind, ist die Bedingung erfüllt. Ein gutes Beispiel ist ein Kommentarsystem. Du möchtest deinen Benutzern erlauben, auf deiner Seite Kommentare zu schreiben, aber nur wenn der Benutzer seinen Namen und eine Email-Adresse angibt.

```php
<?php
    $name = "Klaus";
    $email = "klaus63552@fakemail.de"

    if($name != '')
    {
        if($email != '')
        {
            echo 'Dein Kommentar wurde gespeichert';
        }
    }
?>
```

Hier siehst du erstmal, dass man Bedingungen auch ineinander verschachteln kann, also erst wird geprüft, ob der Benutzer einen Namen eingegeben und danach ob er auch eine EMail Adresse angegeben hat. Wenn beide Bedingungen wahr sind, wird der Kommentar gespeichert. Das kann man aber auch kürzer schreiben:

```php
<?php
    $name = "Klaus";
    $email = "klaus63552@fakemail.de"

    if(($name != '') && ($email != ''))
    {
        echo 'Dein Kommentar wurde gespeichert';
    }
?>
```

Das &&-Zeichen steht für AND und heißt soviel wie, die erste Bedingung muss wahr ergeben und die zweite Bedingung ebenfalls, damit der gesamte Ausdruck **wahr** ist. Also, dass "if"-Statement möchte nur **einen** Wert aus der Klammer erhalten, WAHR oder FALSCH. Wir können aber innerhalb der Klammer mehrere Wahrheitswerte kombinieren und mit dem &&-Operator sagen, dass alle Bedingungen wahr sein sollen.

OR (||)

Zusätzlich zum &&-Operator gibt es noch den ||-Operator, den ODER-Operator. Dieser sagt, dass nur eine Bedingung wahr sein muss, damit der gesamte Ausdruck wahr ist. Wenn wir obiges Beispiel ein wenig anpassen:

```php
<?php
    $name = "Klaus";
    $email = ""

    if(($name != '') || ($email != ''))
    {
        echo 'Dein Kommentar wurde gespeichert';
    }
?>
```

Jetzt reicht es schon, wenn der Besucher entweder seinen Namen angibt oder seine Email-Adressen. Er kann aber auch weiterhin beides angeben, damit das if-Statement wahr ist. Der einzige Fall wo es nicht klappt ist, wenn er weder den Namen, noch die Email Adresse angibt.

Drei und mehr Bedingungen kombinieren

Natürlich ist man nicht auf zwei Bedingungen beschränkt, man kann soviele Bedingungen miteinander kombinieren wie man mag:

```php
<?php
    if(($wert1 != 0) && ($wert2 > 100) && ($wert2 <= 200))
    {
        //WAHR, nur wenn alle 3 Bedingungen wahr sind
    }
?>
```

```php
<?php
    if(($wert1 != 0) || ($wert2 > 100) || ($wert2 <= 200))
    {
        //WAHR, wenn einer der 3 Bedingungen wahr sind
    }
?>
```

```php
<?php
    if((($wert1 != 0) || ($wert2 > 100)) && ($wert2 <= 200))
    {
        //WAHR, wenn entweder Bedingung 1 oder Bedingung 2 wahr ist UND Bedingung 3 wahr ist
        //bei solchen Konstruktionen auf die Klammersetzung achten!
    }
?>
```

```php
<?php
    if((($wert1 != 0) && ($wert2 > 100)) || ($wert2 <= 200))
```

```
    {
        //WAHR, wenn entweder Bedingung 1 und Bedingung 2 wahr sind ODER Bedingung 3 wahr ist
        //bei solchen Konstruktionen auf die Klammersetzung achten!
    }
?>
```

Fazit

Dies ist das Kombinieren von Wahrheitswerten. Als Ergebnis erhält man immer nur WAHR oder FALSCH doch kann man dadurch mehrere Fälle zusammen abfragen. Umso mehr Bedingungen man miteinander kombiniert, desto komplexer wird die Abfrage und damit schwerer für andere zu verstehen. Daher macht es manchmal Sinn auf die Kombination von Wahrheitswerten zu verzichten und stattdessen die einzelnen Abfragen in if-Verschachtelungen abzufragen.

2.4.5. Kurzform für If then else

Das “if else”-Konstrukt wird in PHP sehr häufig verwendet, daher haben die Entwickler eine Kurzform für **if else** herausgebracht. Auch wenn sie für Anfänger häufig nicht so leicht verstanden wird, kann sie in manchen Fällen Platz und Codezeilen sparen. Das ganze nennt sich der **ternärer Operator** und sieht folgendermaßen aus:

Die Syntax für die Kurzform von if else sieht folgendermaßen aus:

```
<?php
    (wenn Ausdruck wahr) ? (mache das hier) : (ansonsten das hier)
?>
```

Nun ein paar Beispiele:

```
<?php
    if($alter < 18)
        echo 'Du bist nicht volljährig';
    else
        echo 'Du bist volljährig';

    //als Kurzform
    echo ($alter < 18) ? 'Du bist nicht volljährig' : 'Du bist volljährig';
?>
```

```
<?php
    if($a > $b)
        $b = $a;
    else
        $a = $b;

    //als Kurzform
    ($a > $b) ? $b = $a : $a = $b;
?>
```

```
<?php
    if($a > $b)
        $c = $a;
    else
        $c = $b;

    //als Kurzform
    $c = ($a > $b) ? $b : $a;
?>
```

Angenommen wir haben eine Anzeige die dir die Stunden ausgibt, seitdem du das letzte mal eingeloggt warst. Also ist die Ausgabe “zuletzt eingeloggt vor x Stunden”, doch für die Stunde 1, müsste die Ausgabe folgendermaßen lauten: “zuletzt eingeloggt vor 1 Stunde”, also ohne das **n**.

Normal:

```
<?php
```

```php
    if($stunden == 1)
        echo 'zuletzt eingeloggt vor 1 Stunde';
    else
        echo 'zuletzt eingeloggt vor '.$stunden.' Stunden';
?>
```

If Else Kurzform

```php
<?php
    echo 'zuletzt eingeloggt vor '.$stunde.' Stunde'.(($stunde==1) ? "" : "n");
?>
```

Natürlich kann man das auch immer mit dem kompletten “if/else”-Konstrukt machen, meistens ist das sogar verständlicher, doch in manchen Fällen ist eine Kurzform einfach nützlich.

2.5. Switch

Im Kapitel zu if, else if, else habe ich schon angesprochen, dass man statt einer Reihe von “else if”-Abfragen besser ein Switch-Konstrukt nutzt.

```php
<?php
    switch($zahl) {
        case 0:
            echo 'Durch 0 darf man nicht teilen';
            break;
        case 1:
            echo 'Die Zahl 1 ist sehr klein';
            break;
        case 99999:
            echo 'Das ist eine sehr große Zahl';
            echo 'kurz vor hunderttausend';
            break;
    }
?>
```

Switch ist dafür gedacht, eine Variable auf mehrere Werte zu prüfen und für den jeweiligen Wert, einen unterschiedlichen Code auszuführen. Das selbe könnte man auch mit mehreren “else if”-Abfragen bewerkstelligen, doch switch ist dafür aufgrund seiner Struktur übersichtlicher. Der Code ist schnell erklärt:

- switch erwartet eine Variable und prüft dann den Wert
- für jeden Wert kann ein “case” angelegt werden
- das “break;” nach jeder Anweisung besagt, dass die Switch-Abfrage beendet wird, wenn ein Wert gefunden wurde
- nach einem “case” können ein- und mehrzeilige Anweisungen folgen

“default” ersetzt “else” in der Switch Anweisung

Auch für die Switch Anweisung gibt es eine “else” Bedingung doch hier heißt sie “default”:

```php
<?php
    switch($zahl) {
        case 0:
            echo 'Zahl 0';
            break;
        case 1:
            echo 'Zahl 1';
            break;
        default:
            echo 'Eine Zahl ungleich 0 oder 1';
    }
?>
```

Default steht am Ende nach allen “case”-Anweisungen und muss auch nicht mit einem break geschlossen werden, da die Switch-Anweisung an dieser Stelle sowieso zuende ist.

Natürlich kann man nicht nur Zahlen sondern auch Strings prüfen:

```php
<?php
    switch($name) {
        case 'Lisa':
            echo 'Lisa ist ein schöner Name';
            break;
        case 'Laura':
            echo 'Hey Laura, ebenfalls ein schöner Name';
            break;
        default:
            echo 'Auch dein Name ist schön, '.$name;
    }
?>
```

Warum die Switch Anweisung verwenden?

Es gibt eigentlich nur 2 Gründe:

1. bei vielen “case”-Anweisungen kann switch schneller sein als mehrere “else if”-Anweisungen
2. in manchen Fällen ist es übersichtlicher, gerade wenn man viele “case”-Anweisungen hat, bzw eine Variable auf viele unterschiedliche Fälle prüfen möchte (z.b. welche Taste wurde gedrückt: “oben”, “unten”, “links”, “rechts”, “space”, “enter”, etc.)

2.6. Arrays

Ein Array ist ebenfalls eine Variable, die allerdings anstatt nur einem Wert, viele Werte speichern kann. Stell dir vor du hast 5 Namen, die du in Variablen speichern möchtest. Mit deinem bisherigen Wissen müsstest du das folgendermaßen machen:

```php
<?php
    $name1 = "David";
    $name2 = "Norman";
    $name3 = "Gerhard";
    $name4 = "Henning";
    $name5 = "Peter";
?>
```

Das ist doof, umständlich und problematisch. Einfacher geht es mit einem Array:

```php
<?php
    $namen[] = "David";
    $namen[] = "Norman";
    $namen[] = "Gerhard";
    $namen[] = "Henning";
    $namen[] = "Peter";
?>
```

Uh? Was ist denn hier jetzt passiert?
Als erstes siehst du die Variable “$namen”, das ist unser Array. Doch die Variable hat noch so eine eckige-Klammer “[]” vor dem Namen. Damit sagen wir dem Array “jetzt kommt ein Wert den wir speichern wollen, leg also im Array einen neuen Eintrag an”. Nicht verstanden? Dann jetzt mal Schritt für Schritt.

Arrays Schritt für Schritt erklärt

Wir können ein Array in PHP mit folgender Anweisung anlegen:

```php
<?php
    $namen = array();
?>
```

Wir müssen das nicht so explizit machen (im ersten Beispiel habe ich auch drauf verzichtet), aber es schadet auch nicht. Das Array hat noch keinen Wert, deshalb fügen wir ihm jetzt einen hinzu:

```php
<?php
    $namen = array();
    $namen[] = "David";

    echo $namen;
?>
```

Wenn du das Skript ausführst, wirst du als Ausgabe “Array” erhalten. Das ist auch korrekt, denn Arrays arbeiten ein bisschen anders als Variablen. Um alle Werte in einem Array zu erhalten, gibt es die Funktion “print_r()”:

```php
<?php
    $namen = array();
    $namen[] = "David";

    print_r($namen);
?>
```

Ausgabe:

```
Array ( [0] => David )
```

Ok, also der Name wurde gespeichert und befindet sich an Position “0″ im Array. Moment…. **Position 0**???
Ja also.. öhm… das ist zwar etwas seltsam, aber in Arrays startet der erste Wert nicht an Position 1, sondern an Position 0. Das wird in den kommenden Wochen für dich auch immer wieder eine Fehlerquelle sein, aber so ist es nunmal, Arrays starten bei 0, Arrays starten bei Position 0, Position 0 == 1. Wert

Du wirst dazu gleich noch weitere Beispiele sehen. Als erstes wollen wir unser Namens-Array mit weiteren Namen füttern:

```php
<?php
    $namen = array();
    $namen[] = "David";
    $namen[] = "Norman";
    $namen[] = "Gerhard";
    $namen[] = "Henning";
    $namen[] = "Peter";

    print_r($namen);
?>
```

Dies ist nur eine Variante ein Array mit Werten zu erstellen, eine andere Variante ist folgende:

Nun sieht die Ausgabe schon interessanter aus:

```
Array ( [0] => David [1] => Norman [2] => Gerhard [3] => Henning [4] => Peter )
```

Wenn du noch eine kleine Formatierung machst, wird das Array auch übersichtlicher angezeigt:

```php
<?php
    $namen = array("David", "Norman", "Gerhard", "Henning", "Peter");

    echo '<pre>';
    print_r($namen);
    echo '</pre>';
?>
```

Wie du siehst, werden im Array die Plätze 0, 1, 2, 3 und 4 belegt für 5 Namen.
Du weißt nun, dass man mit “print_r()” ein Array komplett ausgibt und mit den HTML-Tags “pre”, die Ausgabe schön formatieren kann. Aber normalerweise möchte man ja nicht das komplette Array ausgeben (fürs debugging/fehler finden sehr nützlich), sondern einzelne Werte bearbeiten und anzeigen.

Array Werte hinzufügen, ausgeben und löschen

Wie man Werte einem Array hinzufügt, hast du schon gesehen. Einfach der Name des Arrays + []-Klammern = Wert:

```php
<?php
    $namen = array("David", "Norman", "Gerhard", "Henning", "Peter");
?>
```

Der Versuch “echo $namen;” führte zu nichts, denn das gesamte Array können wir nicht ausgeben. Wir müssen auf die einzelnen Container im Array zugreifen. Möchtest du z.b. den Namen “Gerhard” ausgeben, führt dich folgender Code zum Ziel:

```php
<?php
    $namen = array("David", "Norman", "Gerhard", "Henning", "Peter");

    echo $namen[2];
?>
```

Um einen bestimmten Array-Wert auszugeben, musst du einfach nur die Container-Nummer angeben. **Achte darauf**, dass wir hier **Container-Nummer 2** angegeben haben, denn an *Position 0* ist “David” an *Position 1* ist “Norman” und erst an *Position 2* ist “Gerhard”. Vergiss niemals die Position 0 in einem Array zu berücksichtigen!

Gut, jetzt können wir Array-Werte hinzufügen und ausgeben, doch was ist wenn wir einen Array Wert löschen wollen?
Wir haben ein kleines Spiel in PHP programmiert und oben ist unser Array mit allen 5 Spielern. Nun hat der Spieler “Henning” verloren und wir wollen ihn aus unserem Array löschen. Dazu hilft uns die Funktion “unset()”.

```php
<?php
    $namen = array("David", "Norman", "Gerhard", "Henning", "Peter");

    echo "Anzahl Spieler: ".count($namen)."<br />";
    unset($namen[3]);
    echo "Anzahl Spieler: ".count($namen)."<br />";

        echo '<pre>';
    print_r($namen);
    echo '</pre>';
?>
```

In Zeile 7 geben wir erstmal aus, wieviele Spieler wir derzeit haben. Die Funktion “count()” gibt die Anzahl aller Elemente eines Arrays aus. In Zeile 9 löschen wir mit “unset()” den 3. Container, also den Spieler “Henning” (0. Container beachten!). In Zeile 11 prüfen wir nun nochmal, wieviele Spieler vorhanden sind und stellen fest, dass nun nur noch 4 Spieler dabei sind.

Zum Schluss geben wir nochmal das Array formatiert aus und stellen fest, dass zwar der Spieler gelöscht wurde, aber die Array-Indizes (also die Container-Zahlen) lückenhaft sind (die 3 fehlt). Im Kapitel Schleifen wirst du sehen, dass das nicht so toll ist und fortlaufende Indizes weniger Probleme machen. Daher müssen wir, nachdem der Spieler “Henning” gelöscht wurde, noch schnell das Array neu auflisten mit der PHP Funktion “array_values($namen);”:

```php
<?php
    $namen = array("David", "Norman", "Gerhard", "Henning", "Peter");

    echo "Anzahl Spieler: ".count($namen)."<br />";
    unset($namen[3]);
    echo "Anzahl Spieler: ".count($namen)."<br />";

    echo '<pre>';
    print_r($namen);
    echo '</pre>';

    $namen = array_values($namen);

    echo '<pre>';
    print_r($namen);
    echo '</pre>';
?>
```

Ausgabe:

```
Anzahl Spieler: 5
Anzahl Spieler: 4

Array
```

```
(
    [0] => David
    [1] => Norman
    [2] => Gerhard
    [4] => Peter
)

Array
(
    [0] => David
    [1] => Norman
    [2] => Gerhard
    [3] => Peter
)
```

Array – Fazit

Damit haben wir gerade mal an der Oberfläche von Arrays gekratzt. Arrays werden ständig benötigt und machen die Programmierung mit PHP erst so richtig dynamisch. Wenn dir etwas unklar ist oder du noch Fragen hast, zögere nicht und stell sie (Feedback-Button auf rechter Seite).

2.6.1. Assoziative Arrays

2.6.2. Mehrdimensionales Array

Wenn man ein Array benötigt, welches pro Container mehr als einen Wert speichern soll, greift man auf mehrdimensionale Arrays zurück. Stell dir vor, du hast ein Schachbrett und möchtest alle Felder in einem Array speichern. 8 Reihen x 8 Zeilen = 64 Felder => 64 Elemente in unserem Array:

```php
<?php
    $schachbrett[0] = "A1";
    $schachbrett[1] = "A2";
    $schachbrett[2] = "A3";
    $schachbrett[3] = "A4";
    ...
    $schachbrett[8] = "B1";
    $schachbrett[9] = "B2";
    $schachbrett[10] = "B3";
    ...
    $schachbrett[62] = "H7";
    $schachbrett[63] = "H8";
?>
```

Damit hätten wir ein ein-dimensionales Array mit 64 Feldern. Das ist natürlich möglich, man kann das Schachbrett aber auch als zwei-dimensionales Array erzeugen:

```php
<?php
    $schachbrett[0][0] = "A1";
    $schachbrett[0][1] = "A2";
    $schachbrett[0][2] = "A3";
    $schachbrett[0][3] = "A4";
    $schachbrett[0][4] = "A5";
    $schachbrett[0][5] = "A6";
    $schachbrett[0][6] = "A7";
    $schachbrett[0][7] = "A8";
    $schachbrett[1][0] = "B1";
    $schachbrett[1][1] = "B2";
    $schachbrett[1][2] = "B3";
    $schachbrett[1][3] = "B4";
    $schachbrett[1][4] = "B5";
    ...
    $schachbrett[7][6] = "H7";
    $schachbrett[7][7] = "H8";

    echo $schachbrett[0][4]; //Ausgabe => A5
    echo $schachbrett[1][2]; //Ausgabe => B3
?>
```

Man kann aber noch mehr Werte in sein Array packen, also entweder die Anzahl der Dimensionen erhöhen und oder das ganze mit einem

assoziativen Array kombinieren:

```php
<?php
    $schachbrett[0][0] = array("Feld" => "A1", "Farbe" => "schwarz");
    $schachbrett[0][1] = array("Feld" => "A2", "Farbe" => "weiß");
    $schachbrett[0][2] = array("Feld" => "A3", "Farbe" => "schwarz");
    $schachbrett[0][3] = array("Feld" => "A4", "Farbe" => "weiß");

    echo '<pre>';
    print_r($schachbrett);
    echo '</pre>';

    echo $schachbrett[0][2]["Farbe"]; // Ausgabe => schwarz
?>
```

Ausgabe:

```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [Feld] => A1
                    [Farbe] => schwarz
                )

            [1] => Array
                (
                    [Feld] => A2
                    [Farbe] => weiß
                )

            [2] => Array
                (
                    [Feld] => A3
                    [Farbe] => schwarz
                )

            [3] => Array
                (
                    [Feld] => A4
                    [Farbe] => weiß
                )

        )

)
```

2.6.3. Array Werte sortieren

- sortieren erklären + beispiel
- sortier funktionen in php

mehrdimensionales Array sortieren

Array sortieren nach Spalte

Array alphabetisch sortieren

Array nach Datum sortieren

Array sortieren und Umlaute berücksichtigen

Array nach Zufall sortieren

2.7. Schleifen

Schleifen sind ebenfalls ein Konstrukt, was du ständig brauchen wirst. Gerade wenn du einen bestimmten Algorithmus häufig wiederholen willst oder wenn du z.b. ein Array durchlaufen möchtest, sind Schleifen die richtige Wahl.

In PHP gibt es drei Schleifen-Varianten:

- For-Schleifen
- While-Schleifen
- Do-While-Schleifen
- Foreach-Schleifen

Es kommt auf den jeweiligen Fall an, welche Variante am besten geeignet ist. Aus meiner Erfahrung kann ich sagen, dass du die For-Schleife und die While-Schleife deutlich häufiger brauchen wirst, als die Do-While-Schleife, doch versuche sie alle zu verstehen. Wenn du etwas in den folgenden 3 Teilen nicht verstehst oder Fragen hast, zögere nicht den Feedback-Button auf der rechten Seite zu verwenden.

2.7.1. For-Schleife

Die **For-Schleife** ist die erste Schleife, die wir kennenlernen. Der typische Aufbau sieht folgendermaßen aus:

```
1 <?php
2     for($i=0; $i < 10; $i++)
3     {
4         echo $i."<br />";
5     }
6 ?>
```

In Zeile 2 sind alle Bedingungen der For-Schleife angegeben. Mit dem Schlüsselwort “for” wird PHP mitgeteilt, dass die For-Schleife hier beginnt. Innerhalb der Klammer sind 3 Blöcke, getrennt durch Semikolons “;”:

- **$i=0** – Im ersten Block erhält unsere Zählvariable ihren Startwert. Wir nennen unsere Zählvariable “i”, sie kann aber auch jeden anderen Namen haben, etwa “zaehler”, “counter”, “sdfaawerwe”, etc.
- **$i < 10** – Nun folgt die Abbruchbedingung der Schleife. Solange der Wert der Zählvariable “i” kleiner als 10 ist, wird die Schleife ausgeführt
- **$i++** – Bei jedem Schleifendurchlauf erhöht sich die Zählvariable um den Wert 1. Dadurch ist gewährleistet, dass unsere Schleife nach 10 Durchläufen beendet wird. Natürlich kann man auch den Wert um 2 oder mehr erhöhen: “$i = $i + 2″

Wenn du das Beispiel ausführst, solltest du eine vertikal-verlaufende Zahlenreihe von 0-9 sehen. Die Zählvariable ist das Herzstück der Schleife, denn nur durch ihren Wert weiß die Schleife, wann sie beendet werden soll.

Array mit Schleife durchlaufen

Im Abschnitt Arrays haben wir ein Array mit Namen angelegt. Dieses wollen wir jetzt mal mit einer for-Schleife ausgeben:

```
1 <?php
2     $namen = array("David", "Norman", "Gerhard", "Henning", "Peter");
3 
4     for($i=0; $i < 5; $i++)
5     {
6         echo $namen[$i]."<br />";
7     }
8 ?>
```

Anstelle also die Container manuell aufzurufen ($namen[0], $namen[1], $namen[2], etc.) nutzen wir eine Schleife, die für uns alle Container anzeigt. Die Schleife startet bei “0″ (weil das 0. Array Element unser 1. Wert ist) und endet sobald $i den Wert 5 hat. Wenn die Schleife feststellt, dass ihre Zählvariable den Wert 5 hat, geht sie nicht mehr in den Ausführungsbereich.

Mit einer Schleife können wir also Programmierzeilen zusammenfassen, bzw eine Operation die wir x-mal schreiben müssten in einer Schleife zusammenfassen. Für 5 Werte macht das noch nicht wirklich Sinn, aber stell dir mal vor, unser Array hätte 2000 oder mehr Namen.

Ist das soweit verständlich?

Endlosschleifen

Ich habe schon darauf hingewiesen, dass eine Schleife eine Abbruchbedingung braucht. Denn wenn eine Schleife nicht abbricht, läuft sie endlos => Endlosschleife. Schau dir dazu folgendes Beispiel an:

```
1 <?php
```

```php
    for($i=0; $i < 5; $i++)
    {
        echo $i."<br />";
        $i--;
    }
?>
```

Wir haben zwar eine Abbruchbedingung und die Zählvariable wird nach jedem Durchlauf um eins hochgezählt, doch innerhalb der Schleife wird sie immer um 1 reduziert. PHP prüft sowas natürlich nicht, das liegt an uns sowas zu vermeiden/verhindern. Dies ist noch ein einfaches Beispiel, doch manchmal schleichen sich Fehler ein, die nicht so schnell zu finden sind. Andererseits kommt man Endlosschleifen schnell auf die Schliche, denn wenn man sein Skript testet und es nicht fertig wird, liegt das in der Regel an einer Endlosschleife.

For-Schleife Zusammenfassung

Die For-Schleife ist essentiell in PHP und mit ihrer Zählvariable kann man Arrays einfach durchlaufen. **Nach** jedem Durchlauf kann der Wert der Zählvariable verändert werden, dies ist nötig damit die Abbruchbedingung der Schleife erreicht wird und keine Endlosschleife entsteht.

2.7.2. While-Schleife

Der Aufbau der While-Schleife ist anders als bei der For-Schleife:

```php
<?php
    $i = 0;
    while($i < 10)
    {
        echo $i."<br />";
        $i++;
    }
?>
```

aber dennoch finden sich auch hier die Initialisierung der Zählvariable (Zeile 2), die Abbruchbedingung (Zeile 3) und die Erhöhung der Zählvariable (Zeile 6) wieder. Die **While-Schleife** läuft solange, bis die Abbruchbedingung erfüllt ist. Die While-Schleife prüft erstmal ob das Statement, also die Bedingung innerhalb der Klammer, **TRUE** ergibt. Wenn dies zutrifft, wird der Schleifeninhalt ausgeführt, wenn nicht, wird die Schleife nicht einmal ausgeführt.

Zur While-Schleife gibt es eigentlich nicht viel mehr zu sagen, sie unterscheidet sich ein wenig von der for-Schleife, aber nur minimal. Ob man die for- oder die while-Schleife nutzt, kommt auf den jeweiligen Anwendungsfall an.

2.7.3. Do-While-Schleife

Die do-while-Schleife arbeitet nahezu identisch wie die while-Schleife, doch mit einem entscheidenden Unterschied: Die Prüfung der Abbruchbedingung erfolgt am Ende der Schleife was garantiert, dass **die Schleife mindestens einmal ausgeführt wird**:

```php
<?php
    $i = 0;

    do {
        echo $i."<br />";
        $i++;
    } while ($i < 10);
?>
```

2.7.4. Foreach-Schleifen

Die Foreach-Schleife ist eine Schleife zum durchlaufen von Arrays:

```php
<?php
    $tage = array("Montag", "Dienstag", "Mittwoch", "Donnerstag", "Freitag", "Samstag", "Sonntag");

    foreach($tage as $tag)
```

```
    {
        echo $tag."<br />";
    }
?>
```

Das Beispiel zeigt den typischen Einsatz der Foreach-Schleife: ein Array mit allen Wochentagen soll durchlaufen werden. Der Schleifenkopf nimmt das Array “$tage” und holt sich das 1. Element und weist es der Variablen “$tag” zu. Das tolle daran ist, dass wir innerhalb der Schleife mit den Werten des Arrays arbeiten können. In diesem Fall ist es eine einfache Variable, doch wenn es ein multidimensionales Array wäre, könnten wir auf alle Felder zugreifen ($tag[0][0], $tag[0][1], etc.). Und auch für assoziative Arrays ist die foreach-Schleife wie gemacht:

```
<?php
    //Jeder Spieler hat eine bestimmte Punktzahl

    $spieler["Maurice"] = 4233;
    $spieler["Anna"] = 3872;
    $spieler["Sarah"] = 1497;

    foreach($spieler as $name => $punkte)
    {
        echo $name." hat ".$punkte." erreicht <br />";
    }
?>
```

Die foreach-Schleife funktioniert unabhängig von den Indizes. Also damit eine for-Schleife ein Array durchlaufen kann, müssen die Indizes lückenlos sortiert sein ($tage[0],
$tage[1], $tage[2], usw.), einer foreach-Schleife ist das egal, die durchläuft einfach alle Array-Elemente:

```
<?php
    $werte[rand(999, 9999)] = "Wert 1";
    $werte[rand(999, 9999)] = "Wert 2";
    $werte[rand(999, 9999)] = "Wert 3";

    foreach($werte as $index => $wert)
    {
        echo $wert." - Index: ".$index."<br />";
    }
?>
```

Wir erzeugen 3 Container im Array “$werte” und die Indizes sind jeweils zufällig zwischen 999 und 9999. Dennoch durchläuft das Array problemlos alle 3 Container und gibt uns den Wert der jeweiligen Index-Zahl aus.

2.7.5. Schleifen Fazit

Jede Schleife in PHP hat ihre Vor-und Nachteile gegenüber einer anderen und es kommt immer auf das jeweilige Ziel des Algorithmus an, welche Schleife zu bevorzugen ist.

Beim durchlaufen von Arrays nutze ich so gut wie immer die foreach-Schleifen, für alles andere die for- und while-Schleifen (je nach Anforderung). Die Do-While-Schleife verwende ich so gut wie nie, auch wenn es hin und wieder mehr Sinn machen würde, eine Do-While-Schleife einer While-Schleife vorzuziehen.

2.8. Konstanten

Konstanten sind nahezu identisch mit Variablen, haben allerdings einen entscheidenden Unterschied: Ihr Wert kann sich nicht ändern, ihr Wert ist konstant. Hat man in seinem Skript einen Wert, der mehrmals benutzt wird, aber immer gleich bleibt, so sollte man eine Konstante statt einer Variablen bevorzugen (dazu gleich mehr).

Konstanten definieren und benutzen

```
<?php
    define("BREITE", "500");
    define("MEHRWERTSTEUER", "19");
```

```
    define("VERSION", "1.0.3");
?>
```

Wie du siehst, starten Konstanten nicht mit einem $-Zeichen, sondern müssen über die Funktion “define” definiert werden. Ausserdem werden sie komplett großgeschrieben, dass hat auch den Vorteil, dass du Konstanten in deinem Skript schnell identifizieren kannst (die fallen auf).

Der Sinn von Konstanten

Oben hast du schon die Konstante MEHRWERTSTEUER gesehen. Angenommen wir haben ein Skript, was an mehreren Stellen mit dem Wert der Mehrwertsteuer rechnet, so macht es Sinn diese als Konstante zu definieren. Stell dir vor dein Skript ist mehrere hundert Zeilen lang und plötzlich wird vom Staat der Betrag der Mehrwertsteuer von 19 auf 21% geändert. Ohne Konstante müsstest du nun dein Skript durchsuchen und jedesmal die 19 auf eine 21 ändern. Mit einer Konstante, die du am anfang deines Skripts angelegt hast, musst du nur **an einer Stelle** den Betrag ändern und dein Skript ist aktualisiert.

Regeln für Konstanten Namen

Die Regeln für den Namen einer Konstante sind identisch mit denen für Variablen: der Name startet mit einem Buchstaben oder Unterstrich und danach kann er eine beliebige Anzahl an Buchstaben, Zahlen oder Unterstrichen haben.

2.9. Include

Die Include-Anweisung von PHP ermöglicht es, PHP Skripte in anderen PHP-Dateien einzubinden und die Variablen, Funktionen und Anweisungen auszuführen. Dazu mal ein einfaches Beispiel:

index.php

```
<?php
    echo "Hallo Welt (index.php)";
?>
```

Dies ist unser Skript “index.php”, welches wir aufrufen. Doch bislang passiert darin noch nicht viel, lediglich “Hallo Welt (index.php)” wird ausgegeben. Jetzt legen wir ein zweites Skript an:

frage.php

```
<?php
    echo "Wie geht es dir? (frage.php)";
?>
```

Jetzt wollen wir, dass die Ausgabe von “frage.php” unterhalb der Ausgabe in der index.php erfolgt, dazu nehmen wir eine kleine Änderung an der index.php vor:

```
<?php
    echo "Hallo Welt (index.php)";
    include 'frage.php';
?>
```

Die Anweisung “include ‘frage.php’;” sagt PHP, “*suche im gleichen Ordner, indem die Datei ‘index.php’ liegt, nach der Datei ‘frage.php’ und füge sie an dieser Stelle ein*“.

Zugegeben, so ist das noch recht unspektakulär, aber bedenke, du kannst mit include jede PHP Datei in eine andere integrieren. Das wiederum heißt, wenn du dir eine PHP Webseite baust, brauchst du nicht eine riesige index.php zu haben, in der alles steht, sondern kannst deine Bereiche in einzelnen PHP-Dateien auslagern.

Variablen auslagern und mit include einbinden

Folgendes Beispiel soll den Include-Befehl noch etwas verständlicher machen:

index.php

```php
<?php
    include 'variable.php';
    include 'berechnen.php';

    echo 'Die Variable "$zahl" hat den Wert :'.$zahl;
?>
```

Die index.php soll nur die beiden Dateien “variable.php” und “berechnen.php” einbinden und den Wert der Variable “$zahl” ausgeben.

variable.php

```php
<?php
    $zahl = 100;
?>
```

Hier wird unsere Variable erzeugt und ein Wert zugewiesen.

berechnen.php

```php
<?php
    $zahl = $zahl - 50;
    $zahl = $zahl * rand(2,5);
?>
```

berechnen.php subtrahiert von der Variable Zahl den Wert 50 und multipliziert die Variable dann mit einem zufälligen Wert zwischen 2 und 5. Wenn man nun die index.php aufruft, sollte einer der Ausgaben erfolgen:

```
Die Variable "$zahl" hat den Wert :100
Die Variable "$zahl" hat den Wert :150
Die Variable "$zahl" hat den Wert :200
Die Variable "$zahl" hat den Wert :250
```

Natürlich kann man das auch alles in einer Datei machen, aber dieses Beispiel soll verdeutlichen, dass man Code in mehreren Dateien auslagern kann, um dadurch sein Projekt logisch zu strukturieren und übersichtlicher zu machen (stell dir eine index.php Datei mit 10.000 Code-Zeilen vor ;)).

Pfad beachten

Immer wenn du eine Datei mit “include()” in einer anderen Datei einbindest, musst du auf den korrekt Pfad achten (**Datei 1** bindet **Datei 2** ein):

```php
<?php
    //Datei 1 liegt im gleichen Ordner wie Datei 2
    include 'datei2.php';
?>
```

```php
<?php
    //Datei 1 liegt im root-Ordner und Datei 2 liegt dadrunter im Ordner "Dateien"
    include 'dateien/datei2.php';
?>
```

```php
<?php
    //Datei 1 liegt einen Ordner unterhalb von Datei 2
    include '../datei2.php';
?>
```

2.10. Formulare erstellen und auswerten

Ein Formular besteht eigentlich nur aus HTML *(eventuell noch Javascript)* und hat somit erstmal nichts mit PHP zu tun. Wenn wir aber das Formular auswerten möchten, brauchen wir PHP.

Ein typisches Formular

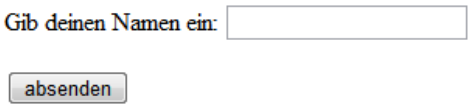
```html
<form action="formular_verarbeiten.php" method="get">
    <p>Gib deinen Namen ein: <input type="text" name="benutzername" /></p>
    <input type="submit" value="absenden" />
</form>
```

Hier siehst du ein typisches HTML-Formular. Unser Formular hat zwei Attribute:

- **action** – das action-Attribut hat den Wert “formular_verarbeiten.php”. Nachdem der Besucher auf den Absenden Button des Formulars klickt, wird diese Datei aufgerufen und die Formular-Werte übermittelt.
- **method** – Die Methode mit der die Variablen verschickt werden sollen. Es gibt die Wahl zwischen **get** und **post**. Wir benutzen hier erstmal die Methode GET

Der Rest ist Standard HTML: Ein Input-Feld mit dem Feldnamen “benutzername” und einen Submit-Button, um das Formular abzuschicken.

Erstelle jetzt die Datei “index.html” und füge obigen HTML Code ein. Wenn du die Datei jetzt aufrufst, solltest du folgende Ausgabe sehen:



Das ist unser HTML-Formular, und viel passiert noch nicht. Wenn du deinen Namen einträgst und auf absenden klickst, versucht der Browser die Datei aufzurufen, die im action-Attribut angegeben ist. Da die Datei noch nicht existiert, erscheint eine Fehlermeldung.

Formular Daten mit PHP verarbeiten

Um die Fehlermeldung zu beseitigen, erstellen wir jetzt die Datei “formular_verarbeiten.php” im selben Ordner wie die “index.html”:

```php
<?php
    echo 'Hallo ';
?>
```

Bislang alles wie gewohnt, aber nun wollen wir auf die Formular-Eingaben zugreifen. Die Formular-Werte werden in die PHP-Variablen “$_GET” und “$_POST” gespeichert, je nachdem, welche Methode wir oben im Formular angegeben haben. Da wir uns erstmal für “get” entschieden haben, sind da auch unsere Werte drin. Da “$_GET” ein Array ist, wollen wir uns das mal anschauen:

```php
<?php
    echo 'Hallo ';

    echo '<pre>';
    print_r($_GET);
    echo '</pre>';
?>
```

```
Hallo
Array
(
    [benutzername] => Christian
)
```

Die Formular-Werte werden also als assoziatives Array in der $_GET-Variable abgelegt. Der Wert des “name”-Attributs ist der Index und der Wert ist die Eingabe, die der User ins Feld geschrieben hat. Das wollen wir nun direkt mal nutzen und passen die Datei “formular_verarbeiten.php” an:

```php
<?php
    echo 'Hallo '.$_GET['benutzername'];
?>
```

Ausgabe: Hallo Christian *(bzw. welchen Namen du eingibst)*

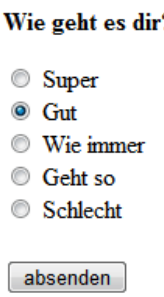
Ein weiteres Beispiel

Man kann alle Formular-Werte an PHP schicken, man muss den Eingabe-Feldern nur einen Namen geben:

index.html

```
<form name="eingabe" action="formular_verarbeiten.php" method="get">
    <p><strong>Wie geht es dir?</strong></p>
    <input type="radio" name="zustand" value="1" /> Super <br />
    <input type="radio" name="zustand" value="2" /> Gut <br />
    <input type="radio" name="zustand" value="3" /> Wie immer <br />
    <input type="radio" name="zustand" value="4" /> Geht so <br />
    <input type="radio" name="zustand" value="5" /> Schlecht <br />

    <br />
    <input type="submit" value="absenden" />
</form>
```



formular_verarbeiten.php

```
<?php
    switch($_GET['zustand']) {
        case 1:
            echo 'Geht mir genau so, ein super Tag!';
            break;

        case 2:
            echo 'Toll, das freut mich für dich :)';
            break;

        case 3:
            echo 'Besser als sich schlecht zu fühlen ;)';
            break;

        case 4:
            echo 'Kopf hoch, das wird schon wieder.';
            break;

        case 5:
            echo 'Denk dran, nach jedem Tief kommt ein Hoch.';
            break;

        default:
            echo 'Du hast vergessen deinen Zustand auszuwählen.';
    }
?>
```

Der Besucher kann seinen Gemüts-Zustand auswählen und wir geben ihm eine dazu passende Antwort. Wenn der Besucher vergisst etwas auszuwählen, nutzen wir die “default”-Aktion von **Switch**.

2.10.1. Unterschied zwischen GET und POST

Beim Formular-Attribut “method” haben wir die Wahl zwischen “get” und “post”, je nachdem welches wir wählen, müssen wir im PHP Script auf die Variable “$_GET” oder “$_POST” zugreifen. Doch was ist der Unterschied zwischen den beiden?

GET wird in der URL angezeigt

Wenn wir ein Formular mit “get” verschicken, wird in der URL-Zeile des Browser die Variablen + ihrem Wert angezeigt:

```
/formular_verarbeiten.php?zustand=2
```

Diesen Wert kann der Benutzer manipulieren, allerdings ist $_POST kein Schutz dagegen, aber es sieht in den meisten Fällen “schöner” aus, da die URL-Zeile nicht mit Variablen vollgepumpt ist.

GET ist begrenzt, POST nicht

Gerade wenn man ein Formularfeld hat, indem sehr sehr viele Daten reinkommen können (z.b. ein Blog-Artikel, ein Forumbeitrag, ein Wikipedia-Artikel) muss man POST verwenden, da die get-Methode begrenzt ist. Bzw eigentlich ist nicht “get” begrenzt, sondern die Url-Zeile des Browsers. Je nach Browser hat man eine maximale Länge um 1024 Zeichen und für Artikel ist das in der Regel nicht ausreichend *(für Twitter-Nachrichten schon ;))*

Vor- und Nachteile von post und get

Hier nochmal die Vor- und Nachteile zusammengefasst:

- Bei GET sieht der User, welche Daten übergeben werden (kann man als Vorteil oder als Nachteil sehen)
- Bei GET ist die Länge begrenzt, bei POST nicht
- Die Ergebnisseite eines GET-Formulars kann man bookmarken, da alle nötigen Informationen in der URL enthalten sind
- Die Ergebnisseite eines POST-Formulars kann man weder bookmarken noch im Browser aktualisieren, da die Daten nicht mehr zur Verfügung stehen
- File-Upload ist nur mit POST möglich

Quelle

2.11. Fehler finden und beheben für Anfänger

Als PHP Anfänger macht man in der Regel folgende Fehler am häufigsten:

- Semikolon vergessen
- normale/geschweifte Klammer nicht geschlossen
- Array Größe überschritten (z.b. in einer Schleife)
- Vergessen, dass das 1. Array-Element an der 0. Stelle beginnt
- Dollar-Zeichen am Anfang einer Variable vergessen (statt “$zahl” => “zahl”)
- Logische Fehler (Endlosschleife, falsche Programmlogik, etc.)

Wie kann man solche Fehler vermeiden?

Leider ist hier fast der einzige Weg Praxis. Wer häufig den selben Fehler macht, wird über kurz oder lang diesen Fehler so gut wie nicht mehr machen. Neben Praxis hilft aber auch ein gut strukturierter Quellcode:

```php
<?php
    //unübersichtlich
    if($zahl < 100) { $zahl++; $zahl += $zahl2 - 14; } else { $zahl = 0; }
?>
```

```php
<?php
    //besser
    if($zahl < 100)
    {
        $zahl++;
        $zahl += $zahl2 - 14;
    }
    else
    {
        $zahl = 0;
    }
?>
```

Zusätzlich zu einem gut strukturierten Quellcode empfiehlt sich aufjedenfall der Einsatz eines Editors mit Syntaxhighlighting. Dieser macht den Quellcode nochmals leichter zu überblicken und man kann durch farbliche Hervorhebung schnell erkennen, wenn man vergessen hat eine Klammer oder einen String zu schließen.

Einfaches Debugging in PHP

Semikolon und vergessene Klammern lassen sich recht leicht finden, da PHP die Datei und die Zeilennummer angibt, inder der Fehler vorkommt. Doch manchmal läßt sich der Fehler trotz der Angabe nicht finden oder es handelt sich um einen logischen Fehler (dein Code arbeitet nicht so, wie du es gerne hättest).

Logische Fehler können z.b. Endlosschleifen sein, falsche/fehlende Array-Werte oder dein Quellcode spuckt einfach etwas aus, was du so nicht erwartet hast. Nun heißt es den Code Zeile für Zeile durchzugehen und an den jeweiligen Stellen die Variablen auszugeben (um zu schauen, welchen Wert sie haben). Ein Array kannst schön formatiert folgendermaßen ausgeben:

```php
<?php
    echo '<pre>';
    print_r($array);
    echo '</pre>';
?>
```

Als letzter Tipp, bei Fehlermeldungen die du nicht direkt verstehst, empfiehlt sich nach dem **Fehler zu googlen**: Wenn PHP eine Fehlermeldung ausspuckt, einfach komplett kopieren (ohne Pfadangaben, Zeilenangaben, weil zu spezifisch) und in Google mit Anführungszeichen einfügen. Meist erhält man dadurch genügend Links, die das Problem schon behandelt haben und eine Lösung anbieten.

---

3. PHP Fortgeschritten

3.1. PHP Version herausfinden

Im Kapitel PHP Fortgeschritten gibt es einige Bereiche, die nur mit PHP 5 funktionieren. Daher solltest du dir, mit diesem kleinen Skript, deine PHP Version anzeigen lassen:

info.php

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

Als Ergebnis solltest du folgendes erhalten:

Oben siehst du die PHP Version, in meinem Fall 5.2.6
Wenn du weiter runterscrollst, findest du noch etliche weitere Daten, welche Module für diese Version installiert sind, die Apache Konfigurations-Werte, etc.
Auch wenn du nur eine 4er PHP-Version hast, kannst du die wichtigsten Teile in diesem Kapitel angehen, nur beim Bereich “Klassen und Objekte” könnte es Probleme geben, dazu aber im jeweiligen Kapitel nochmal ein extra Hinweis.

3.2. Funktionen

Funktionen sind dafür da, Programmlogik zu bündeln, die man immer wieder braucht. Dadurch vermeidet man Redundanz, somit auch Fehlerquellen und muss bei einer Änderung nur an einer Stelle eingreifen. Der Aufbau einer Funktion sieht so aus:

```php
<?php
    function NAME_DER_FUNKTION (parameter1, parameter2, ...)
    {
        //auszuführender Code
    }
?>
```

Nun wollen wir mit einer Funktion zwei Zahlen addieren und die Rechnung + Ergebnis ausgeben:

```php
<?php
    function addieren($zahl1, $zahl2)
    {
        echo $zahl1.' + '.$zahl2.' = '.($zahl1+$zahl2).'<br />';
    }

    addieren(5, 10);
?>
```

Unsere Funktion nennen wir sinngemäß “addieren” und erwarten zwei Parameter, “$zahl1″ und “$zahl2″. Parameter sind Möglichkeiten, Werte an unsere Funktion zu übergeben. Man kann auch Funktionen programmieren, die keinen Parameter erwarten (leere Klammer), aber da wir hier zwei Zahlen addieren wollen, brauchen wir zwei Parameter.

Der Code innerhalb der Funktion sollte selbsterklärend sein.
Nun können wir beliebige Zahlen miteinander addieren und zudem so häufig wir wollen:

```php
<?php
    function addieren($zahl1, $zahl2)
    {
        echo $zahl1.' + '.$zahl2.' = '.($zahl1+$zahl2).'<br />';
    }

    addieren(5, 10);
    echo '<br />';
    addieren(2732, 19883);
    echo '<br />';
    addieren(1, 1);
    echo '<br />';

    for($i=1; $i < 30; $i++)
    {
        addieren($i, $i*10);
        echo '<br />';
    }
?>
```

Ergebnis:

7 + 70 = 77

8 + 80 = 88

9 + 90 = 99

10 + 100 = 110

11 + 110 = 121

Wert mit return zurückgeben

Häufig möchte man, dass die Funktion einen Wert zurückgibt, also zuerst wird etwas berechnet und abhängig vom Ergebnis dieser Berechnung, gibt die Funktion einen Wert zurück. Im folgenden Beispiel möchten wir, dass unsere Funktion uns nur das Ergebnis der Addition zurückgibt:

```php
<?php
    function addieren($zahl1, $zahl2)
    {
        $ergebnis = $zahl1 + $zahl2;

        return $ergebnis;
    }

    for($i=1; $i < 10; $i++)
    {
        echo $zahl1.' + '.$zahl2.' = '.addieren($i, $i);
    }
?>
```

1 + 1 = 2
2 + 2 = 4
3 + 3 = 6

4 + 4 = 8
5 + 5 = 10
6 + 6 = 12
7 + 7 = 14
8 + 8 = 16
9 + 9 = 18

Optionale Parameter

Manchmal möchte man eine Funktion haben, die einen optionalen Parameter beinhaltet:

```php
<?php
    function addieren($zahl1, $zahl2=5)
    {
        echo $ergebnis = $zahl1 + $zahl2;
    }

    addieren(10,12);
    echo '<br />';
    addieren(10);
?>
```

Wenn beide Werte angegeben sind, läuft alles wie gewohnt. Wird aber nur ein Parameter befüllt, erhält der zweite Parameter automatisch den Wert “5″.

Funktionen sind ein mächtiges Werkzeug und ohne kommt man als PHP Programmierer einfach nicht aus. Da aber bestimmte Funktionen von so gut wie allen PHP Entwicklern benötigt werden, werden diese von PHP direkt zur Verfügung gestellt. Schau dir dazu den Unterartikel **PHP Standard Funktionen** an.

3.2.1. Referenz Parameter

Normalerweise übergibt man einer Funktion ein paar Variablen, mit dieser arbeitet die Funktion dann und gibt ein Ergebnis zurück, womit das PHP Skript weiterarbeiten kann. Bei der Übergabe werden Kopien der Variable angelegt und mit diesen Kopien gearbeitet, sprich die Ursprungs-Variable wird nicht verändert:

```php
<?php
    function verkleinern($zahl)
    {
        $zahl = $zahl - 10;
    }

    $zahl = 20;
    verkleinern($zahl);

    echo $zahl;
?>
```

Als Ergebnis erhält man den Wert “20″, denn auch wenn wir innerhalb der Funktion den Wert der Variablen “$zahl” um 10 reduzieren, handelt es sich dabei nur um die Kopie der Variablen.

Möchte man nun aber, dass die Original-Variable verändert wird, also keine Kopie der Variable angelegt wird, muss man mit Referenz Parametern arbeiten. Hört sich schlimmer an als es ist, denn im Grunde setzt man lediglich ein kaufmännisches Und (&) vor die Variable und macht sie dadurch zu einer Referenz:

```php
<?php
    function verkleinern(&$zahl)
    {
        $zahl = $zahl - 10;
    }

    $zahl = 20;
    verkleinern($zahl);

    echo $zahl;
```

```
11 ?>
```

Nun erhalten wir als Ergebnis tatsächlich den Wert “10″ denn es wird keine Kopie der Variable angelegt, sondern die Variable direkt manipuliert.

3.2.2. PHP Funktionen und Dokumentation

PHP kommt mit einer Fülle an nützlichen Funktionen, die man ständig benötigt und warum sollte man das Rad neuerfinden? Eine Übersicht über die Standard PHP Funktionen findest du hier: **PHP Funktionsreferenz**

Da das für Anfänger recht unübersichtlich ist, hier die Referenzen, welche von Anfängern am häufigsten benötigt werden:

- **Mathematische Funktionen**
- **String Funktionen**
- **Array Funktionen**
- **Variablenbehandlung Funktionen**
- **Datum und Uhrzeit Funktionen**
- **MySQL Funktionen**
- **Session Funktionen**

PHP bietet dir also schon viele Funktionen an, doch wie verwendest du die jetzt? Und wie findest du die richtige für dein Problem?

PHP Dokumentation verstehen

In Foren werden Anfänger bei Fragen und Problemen häufig auf die PHP Dokumentation verwiesen. An sich ist das auch absolut legitim, denn die Dokumentation ist vollständig, aktuell und bietet zudem sehr viele Beispiele zu den einzelnen Funktionen. Das Problem ist allerdings, die meisten Anfänger wissen einfach nicht wie die Dokumentation zu lesen ist und wie sie ans Ziel kommen. Daher hier mal eine kleine Annäherung:

1. Funktionen finden
Auf was bezieht sich dein Problem, auf Arrays, auf Strings, auf Mathematik? Identifziere als erstes den Bereich und suche dann in der **Funktionsreferenz** nach diesem Bereich (tricky: “Strings” sind hier unter “Zeichenketten” zu finden).

2. Funktionsbeschreibung lesen und verstehen
Wenn man nun seine Funktion gefunden hat (oder glaubt sie gefunden zu haben), steht in der Beschreibung wie man diese Funktion anwenden muss. Als erstes mal ein einfaches Beispiel, **strlen** gibt die Länge einer Zeichenkette zurück:

strlen

(PHP 4, PHP 5)

strlen — Ermitteln der String-Länge

Beschreibung Report a bug

int **strlen** (string *$string*)

Gibt die Länge der Zeichenkette *string* zurück.

Erklärung:
Das “int” vor “strlen” besagt, dass die Funktion einen Integer-Wert zurückgibt, in diesem Fall die Länge des Strings. Die Funktion erwartet einen Parameter, dieser ist vom Typ “string”.

Nun ein etwas komplexeres Beispiel, dafür nehmen wir die Funktion **str_replace** welches einen Teil eines Strings durch einen anderen ersetzt:

str_replace

(PHP 4, PHP 5)

str_replace — Ersetzt alle Vorkommen des Suchstrings durch einen anderen String

Beschreibung Report a bug

mixed **str_replace** (mixed *$search* , mixed *$replace* , mixed *$subject* [, int *&$count*])

Diese Funktion gibt einen String oder ein Array zurück, in dem alle Vorkommen von *search* innerhalb von *subject* durch den angegebenen *replace* -Wert ersetzt wurden.

Wenn Sie keine ausgefallenen Ersetzungsregeln (wie Reguläre Ausdrücke) benötigen, sollten Sie immer diese Funktion anstelle von ereg_replace() oder preg_replace() verwenden.

Das “**mixed**” steht für Arrays und Variablen und “**mixed str_replace**” bedeutet, dass die Funktion eine Variable oder ein Array zurückgibt, jenachdem mit welchen Werten man die Funktion aufruft. Nun folgen die Parameter: “**mixed $search**” ist ein Pflichtparameter, also da muss entweder eine Variable oder ein Array rein. Gleiches gilt für “**mixed $replace**” und “**mixed $subject**“, hingegen ist der vierte Parameter **int &$count** optional (eckige Klammern). Die PHP Dokumentation ist wirklich gut, so findet man unterhalb der Beschreibung eine Parameter-Liste, die erklärt was die Parameter machen und welche Auswirkungen sie haben.

Wer aber nicht so gerne mit der Beschreibung arbeitet, kann sich stattdessen unten auch die Beispiele anschauen. Diese sind wirklich sehr gut gewählt und zeigen die Variationsmöglichkeiten der Funktionen und ihre Ausgaben.

3.3. Klassen und Objekte (Einführung)

Funktionen waren der erste Schritt, komplexe PHP Skripte übersichtlicher zu machen und Code wiederzufinden. Der nächste Schritt sind Klassen und Objekte. PHP wurde lange Zeit von Programmierern mit einem lächeln abgetan, doch spätestens seit dem Einzug von OOP *(Objekt-orientierte Programmierung sprich: Klassen und Objekte)* wird PHP weitgehend ernst genommen.

Was sind Klassen und Objekte

Klassen kann man grob als “Bauplan” ansehen, mit dem Bauplan selbst kann man nichts machen, aber man kann den Bauplan nutzen, um etwas herzustellen. Also unsere Klassen können wir direkt nicht benutzen, aber aus den Klassen können wir Objekte erzeugen. Stell dir vor du hast eine Klasse “Auto” *(einen Bauplan für ein Auto)*. Nun kannst du mit der Klasse ein Objekt erzeugen, in diesem Fall ein Auto. Also könntest du z.b. sagen: Klasse “Auto”, gib mir einen “Opel Insignia”.

Von einer Klasse kannst du beliebig viele Objekte erzeugen, also kannst du dir von der Klasse “Auto” einen ganzen Fahrzeughof erschaffen ;)

Bevor wir jetzt von dieser abstrakten Erklärung zum konkreten Beispiel springen noch einen Hinweis: Häufig liest man von Objekten und Instanzen und fragt sich nach dem Unterschied. Objekte und Instanzen sind das gleiche, denn: **Ein Objekt ist die Instanz einer Klasse**. Also nicht verwirren lassen, es gibt Klassen und Objekte/Instanzen.

Zusammenfassung von Klassen und Objekten

- Wir brauchen Klassen (Baupläne) um Objekte (Gegenstände) zu erzeugen
- Mit Klassen selbst können wir nichts machen, nur mit Objekten
- Von einer Klasse können wir beliebig viele Objekte erstellen
- Objekte werden auch als Instanzen bezeichnet

Im nächsten Teil erzeugen wir unsere erste PHP Klasse und damit wird die ganze Erklärung etwas verständlicher.

3.4. Fehler finden und beheben für Fortgeschrittene
