



CODERS.**BAY**

PHP 7

# WAS IST PHP?

PHP ist die Abkürzung für „PHP: Hypertext Preprocessor“

PHP ist eine weitverbreitete Open Source Skriptsprache

PHP-Skripte auf dem Server ausgeführt

# WAS KANN PHP?

- PHP kann dynamischen Seiteninhalt generiert
- PHP kann erstellen, öffnen, lesen, schreiben, löschen und Schließen von Dateien auf dem Server
- PHP können Formulardaten sammeln
- PHP können Cookies senden und empfangen
- PHP kann hinzufügen, löschen, ändern Daten in Ihrer Datenbank
- PHP kann verwendet werden, um Benutzerzugriff zu steuern
- PHP kann Daten verschlüsseln

## WARUM PHP?

- PHP läuft auf verschiedenen Plattformen (Windows, Linux, Unix, Mac OS X, etc.)
- PHP ist kompatibel mit fast allen Servern die in Verwendung sind (Apache, IIS, etc.)
- PHP unterstützt eine Vielzahl von Datenbanken
- PHP ist kostenlos.
- PHP ist leicht zu erlernen läuft auf der Server-Seite

# WAS IST EINE PHP-DATEI?

- PHP-Dateien können Text, HTML, CSS, JavaScript und PHP-Code enthalten
- PHP-Code auf dem Server ausgeführt, und das Ergebnis wird an den Browser als einfache HTML zurückgegeben
- PHP - Dateien haben Extension „.php“

- Ein PHP-Skript kann beliebige Stelle im Dokument platziert werden.
- Eine PHP - Skript beginnt mit `<?php` und endet mit `?>`
- Die Standarddateierweiterung für PHP - Dateien ist „ .php“.
- Eine PHP-Datei enthält normalerweise HTML-Tags, und einige PHP Script-Code.

```
<?php  
    // PHP code goes here  
    echo "Hello World!";  
?>
```

# KOMMENTARE IN PHP

```
<?php  
    // This is a single-line comment  
    # This is also a single-line comment  
    /*  
        This is a multiple-lines comment block  
        that spans over multiple lines  
    */  
  
    // You can also use comments to leave out parts of a code l  
    ine  
    $x = 5 /* + 15 */ + 5;  
    echo $x;  
?>
```

# EINFÜHRUNG

- Weitverbreitete Sprache zur Entwicklung **dynamischer Internetanwendungen**
- Abkürzung für: Hypertext Preprocessor
- **Dynamisch:** Inhalte können sich aufgrund von Aktionen (zB des Benutzers oder neuer Basisinformationen in Datenbank) ändern
- Unterstützt einfache Auswertung von Formularen
- Ermöglicht Zusammenarbeit mit vielen verschiedenen Datenbanksystemen

# EINFÜHRUNG

## › Vorzüge

- ▶ Erschaffen für Entwicklung von Internetanwendungen
- ▶ Ermöglicht einfache Entwicklung von Programmen
- ▶ Unterstützt verschiedene Plattformen
- ▶ Arbeitet sehr gut mit Apache Webservern zusammen
- ▶ Ist flexibel

## › Erlernbarkeit

- ▶ Relativ leicht erlernbar

## › Einsatzbereich

- ▶ Wird von vielen Typen von Webservern einheitlich unterstützt
- ▶ Andere Sprachen kommen nur auf bestimmten Servertypen zum Einsatz

## › Preis

- ▶ Kostet nichts
- ▶ Kann unter anderem mit frei verfügbaren Apache Webservern unter verschiedenen Betriebssystemen eingesetzt werden

## › Systemvoraussetzung

- ▶ PHP-fähiger Webserver (zB Apache)
- ▶ PHP selbst
- ▶ Datenbanksystem wie MySQL

# EINBETTUNG VON PHP IN HTML

- <?php leitet eine einzelne PHP-Anweisung oder einen Block ein
- Werden bis zur Markierung ?> bearbeitet, es stellt das Ende des Blocks dar
- PHP Blöcke können im gesamten Dokument untergebracht werden
- Code wird von oben nach unten abgearbeitet
- Dateien mit PHP enden auf .php
- echo
  - ▶ gibt den angegebenen Text auf dem Bildschirm aus
  - ▶ Muss in doppelten Anführungszeichen oder einfachen Hochkommata geschrieben werden

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head>
<body>
<p>
    Die erste Zeile in HTML<br>
    <?php echo "Die zweite Zeile in PHP<br>"; ?>
    Die 3. Zeile in HTML<br>
    <?php
        echo "Die 4. Zeile in PHP<br>";
        echo "die 5. Zeile in PHP";
    ?>
</p>
</body>
</html>
```

## BEISPIEL

- Starte den Apache Webserver (XAMPP)
- Speichere das vorherige Beispiel als test.php ab
  - ▶ Vervollständige den <head> Bereich!
  - ▶ Die Datei muss in einem Ordner im *localhost* Verzeichnis von XAMPP liegen
- Gib im Browser ein  
*http://localhost/test.php* ein

# KOMMENTARE IN PHP

- Man unterscheidet zwischen einzeiligen und mehrzeiligen Kommentaren
  - ▶ Einzeilige Kommentare beginnen mit den Zeichen `//`  
sie enden am Schluss der Zeile
  - ▶ Mehrzeilige Kommentare beginnen mit dem Zeichen `/*` und enden mit dem Zeichen `*/`

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

    echo "das ist der Anfang"; //Kommentar und sonstiges

    /* ein Kommentar
       in mehreren Zeilen
       ist ganz einfach */

    echo " und hier ist das Ende des Programms.';

?>

</body></html>
```



CODERS.BAY

# VARIABLEN, DATENTYPEN UND OPERATOREN

- Informationen können zur **späteren Verwendung in Variablen** gespeichert werden
- PHP unterstützt Datentypen für:
  - ▶ Ganze Zahlen
  - ▶ Zahlen mit Nachkommastellen (Fließkommazahlen)
  - ▶ Zeichenketten (Strings)
  - ▶ Felder (ein- und mehrdimensionale Felder von Variablen)
  - ▶ Objekte
- Datentyp richtet sich nach dem **Zusammenhang**, in dem die Variable verwendet wird
- Variable kann ihren Datentyp innerhalb eines Programms wechseln
- Es gibt **keine Variablendeklaration**
- Variable kann nach erscheinen **sofort benutzt** werden
- Variablen fangen mit dem `$`-Zeichen an ( `$ichBinEineVariable` )

# NAMEN

## ➤ Regeln für Variablen

- ▶ Er muss mit einem **Dollarzeichen** beginnen
- ▶ Er darf **keine Leerzeichen** enthalten
- ▶ Er darf **nur aus Buchstaben und Ziffern** bestehen, wobei das **erste Zeichen** ein **Buchstabe** sein muss
- ▶ Groß- und Kleinbuchstaben sind erlaubt, zwischen denen jedoch unterschieden wird (`$HokusPokus` ist nicht das Gleiche wie `$hokusPokus`)
- ▶ **Keine Umlaute, kein ß**
- ▶ Keine **Sonderzeichen** (nur Unterstrich `_`)
- ▶ Darf nicht mit einem **reservierten Wort** identisch sein
- ▶ Sollten **selbsterklärende Namen** besitzen

## VARIABLEN FÜR ZAHLEN

- Variable `$liter` wird deklariert und der Wert 14 wird ihm zugewiesen  
Variable wird zu einer ganzen Zahl
- Variable `$preis` wird deklariert und der Wert 1.35 wird zugewiesen  
Variable wird zu einer Zahl mit Nachkommastellen  
Achtung: Dezimaltrennzeichen ist der Punkt!!
- Variable `$zahlung` wird deklariert  
die Variablen `$liter` und `$preis` werden miteinander multipliziert; das Ergebnis wird der Variable `$zahlung` zugewiesen. Damit wird `$zahlung` ebenfalls zu einer Variablen für eine Zahl mit Nachkommastellen
- Der Wert `$zahlung` wird mit einem `echo` ausgegeben

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php
    $liter = 14;
    $preis = 1.35;
    $zahlung = $liter * $preis;
    echo $zahlung;
?>

</body></html>
```

Operator	Bedeutung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo-Operation: der Rest bei einer ganzzahligen Division. So ergibt zum Beispiel <code>7 % 3</code> den Wert 1, denn 7 dividiert durch 3 ergibt 2, Rest 1.
**	Potenzieren mithilfe des Exponentialoperators Bsp.: <code>2**3</code> , gesprochen 2 hoch 3

- `$x = 5; $x += 3;` (`$x` hat den Wert 8)
- `$x = 5; $x -= 3;` (`$x` hat den Wert 2)
- Multiplikation und Division kommen vor Addition und Subtraktion
- Eine (mathematisch nicht erlaubte) Division einer positiven oder negativen Zahl durch 0 führt zum Ergebnis `INF` bzw. `-INF`  
`INF` = infinity (deutsch: unendlich)

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>
<?php
$liter1 = 16;
$liter2 = 23;
$liter3 = 34;
$preis = 1.35;
$gesamtzahlung = ($liter1 + $liter2 + $liter3) * $preis;
echo $gesamtzahlung;
?>
</body></html>
```

- Zeichenketten (**Strings**) müssen in doppelten (" ") oder einfachen (' ') Hochkommata eingeschlossen werden
- Das Zeichen . (Punkt) dient zur Verkettung von Zeichenketten bzw. von Zahlen und Zeichenketten  
Bsp.: kommentierte Ergebnisausgabe
- Operator .= (Punkt gleich) kann zur Vergrößerung einer Zeichenkette eingesetzt werden

## BEISPIEL

```
<!DOCTYPE html><html><head><meta charset="utf-8"> </head><body>

<?php

$liter = 14;
$preis = 1.35;
$zahlung = $liter * $preis;
$einheit1 = "Liter";
$einheit2 = 'Euro';

$gesamt = "<p>" . $liter . " " . $einheit1;
$gesamt .= " kosten " . $zahlung . " " . $einheit2 . "</p>";
echo 'ausgabe 1' . $gesamt;
echo "<p>Ausgabe 2: $liter $einheit1 kosten $zahlung $einheit2</p>";
echo '<p>Ausgabe 3: $liter $einheit1 kosten $zahlung $einheit2</p>';

?>

</body></html>
```

- Im ersten Teil findet die Berechnung des Preises statt
- Den Variablen `$einheit1` und `$einheit2` werden Zeichenketten zugewiesen - in doppelten bzw einfach Hochkommata
- Der Variablen `$gesamt` wird eine Zeichenkette zugewiesen, die sich aus einzelnen Zeichenketten, Zahlen- und Zeichenkettenvariablen sowie aus HTML-Code zusammensetzt
  
- Zeichenkette `$gesamt` wird verlängert (Operator `.=`)
  
- Zeichenkette `$gesamt` wird ausgegeben
- Falls Variable innerhalb einer Zeichenkette in einfachen Hochkommata steht, wird nur der Name nicht aber der Wert der Variable ausgegeben

# KONSTANTEN

- Dienen zur Speicherung von **unveränderlichen Werten**
- Schlüsselwort `const` werden Zahlenkonstanten und Zeichenkettenkonstanten definiert
- Kein `$`-Zeichen vor dem Namen notwendig
- Konstanten können **nicht innerhalb von Zeichenketten ausgegeben** werden (können nicht vom restlichen Text unterschieden werden)

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php
const pi = 3.1415926;
const gruss = "Guten Morgen";
echo pi . " <br>";
//gruss = "Hallo";
?>

</body></html>
```

## REFERENZEN

- Man kann auf eine vorhandene Variable eine sogenannte Referenz einrichten
- Möglichkeit, auf diese Variable **über einen weiteren Namen zuzugreifen**
- Vor allem im Zusammenhang mit Funktionen und Methoden interessant

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php
$orig = 12.3;
echo "$orig<br>";
$refe = &$orig;
$refe = 5.8;
echo "$orig<br>";
?>
</body></html>
```

- Die Variable `$orig` wird erzeugt. Es wird ihr ein Wert zugewiesen und ausgegeben. Danach wird mithilfe des Operators `&` die Referenz `$refe` auf die Variable `$orig` eingerichtet. Eine anschließende Änderung des Werts über die Referenz entspricht einer Änderung des Werts der Variablen



CODERS.BAY

# EINFACHE FORMULARAUSWERTUNG

# EINFACHE FORMULARAUSWERTUNG

## - BEISPIEL 1

- Attribut **action** verweist hier auf die Datei **eingabe.php** mit dem PHP-Auswertungsprogramm
- Attribut **method**, verweist auf die Übermittlungsmethode zum Webserver

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>  
<p>Bitte tragen Sie Ihren Vornamen und Ihren Nachnamen ein.<br>  
Senden Sie anschließend das Formular ab</p>  
<form action = "eingabe.php" method = "post">  
<p><input name = "vor"> Vorname</p>  
<p><input name = "nach"> Nachname</p>  
<input type = "submit">  
<input type = "reset">  
</form>  
</body></html>
```

# EINFACHE FORMULARAUSWERTUNG

## - BEISPIEL 1

➤ Die Ausgabe sieht wie folgt aus

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php
echo "Guten Tag, " . $_POST["vor"] . " " . $_POST["nach"];
?>

</body></html>
```

➤ `$_POST`

- ▶ assoziatives Feld
- ▶ Vordefinierte Variable
- ▶ Gehört zu den superglobalen Feldern (später mehr dazu)
- ▶ Für die Übermittlungsmethode post
- ▶ Elemente eines assoziativen Felds lassen sich nicht in einer Zeichenkette innerhalb von Hochkommata ausgeben

- Texteingabefeld eines Formulars nimmt **Zeichenkette** auf
- Zeichenketten werden nach folgenden Regeln implizit konvertiert bzw. umgewandelt
  - ▶ Zeichenketten, die nach einem optionalen Vorzeichen nur Ziffern enthalten, werden in ganze Zahlen umgewandelt  
(Bsp.: "42", "-42" oder "+42")
  - ▶ Zeichenketten, die zusätzlich einen Dezimalpunkt oder am Ende nach einem optionalen Vorzeichen einen Exponenten nach einem e oder E enthalten, werden in Zahlen mit Nachkommastellen umgewandelt, also in Fließkommazahlen.  
(Bsp.: "4.2", "-4.2", "42e3", "4.2e3", "4.2e-3" oder "-4.2E3")
  - ▶ Andere Zeichen sollten vermieden werden
  - ▶ Falls innerhalb einer Zeichenkette andere Zeichen stehen, wird nur der vordere Teil der Zeichenkette bis zum Beginn der anderen Zeichen umgewandelt. Allerdings erfolgt eine Warnung, dass es sich nicht um einen wohlgeformten numerischen Wert handelt.  
Bsp.: "42abc23" (Zahlenwert: 42) oder "4.2 Liter" (Zahlenwert 4.2)
  - ▶ Falls am Beginn einer Zeichenkette andere Zeichen stehen, ergibt sich der Zahlenwert 0 und es erfolgt ebenfalls die oben genannte Warnung  
Bsp.: "abc42" oder "Summe 4.2"

# EXPLIZITE UMWANDLUNG

- **doubleval()**
  - ▶ Für Zahlen mit Nachkommastellen
- **intval()**
  - ▶ Für ganze Zahlen

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php
$a = 435;
echo intval($a) . "<br>";

$b = "22.6";
echo doubleval($b) . "<br>";
echo intval($b) . "<br>";
?>

</body></html>
```

## BEISPIEL

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<form action = "eingabe_zahl.php" method = "post">

<p>Wert 1: <input name = "w1"></p>
<p> Wert 2: <input name = "w2"> </p>
<input type = "submit">
<input type = "reset">
</form>

</body></html>
```

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

    $w1 = doubleval($_POST["w1"]);
    $w2 = doubleval($_POST["w2"]);
    $erg = $w1 + $w2;
    echo "Die Summe von $w1 und $w2 ist $erg";
?>

</body></html>
```



CODERS.BAY

# VERZWEIGUNGEN

- Einzelne Anweisungen werden in diesem Fall nur in bestimmten Situationen ausgeführt
- Die Ausführung dieser Anweisungen wird in solchen Fällen von einer oder von mehreren Bedingungen (`if`-Anweisungen) abhängig gemacht
- Bei Nichterfüllung der Bedingung können alternative Anweisungen bearbeitet werden
- Bedingungen werden mithilfe von Wahrheitswerten (wahr oder falsch) und Vergleichsoperatoren erstellt

Operator	Bedeutung	Geltungsbereich
<code>==</code>	gleich	Zahlen und Zeichenketten
<code>!=</code>	ungleich	Zahlen und Zeichenketten
<code>&gt;</code>	größer als	Zahlen
<code>&lt;</code>	kleiner als	Zahlen
<code>&gt;=</code>	größer als oder gleich	Zahlen
<code>&lt;=</code>	kleiner als oder gleich	Zahlen

## IF-Anweisung

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>
<?php
$preis = 0.98;
if ($preis < 1) echo "Der Preis liegt unter 1 &euro;";
?>
</body></html>
```

- Falls `$preis` kleiner als `1` ist, wird der entsprechende Text in das Dokument geschrieben, falls nicht geschieht nichts.

- Falls mehrere Anweisungen ausgeführt werden sollen, müssen diese innerhalb von geschweiften Klammern `{ }` stehen. Das nennt man **Anweisungsblock**

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>
<?php
$preis = 0.98;
if ($preis < 1){
    echo "Der Preis liegt unter 1 &euro;";
    echo "Das ist billig";
}
?>
</body></html>
```

# IF/ELSE-Anweisung

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php
$preis = 0.98;
if ($preis < 1){
    echo "Der Preis liegt unter 1 &euro; <br> ";
    echo "Das ist billig";
} else {
    echo "Der Preis liegt bei 1 &euro; oder mehr<br>";
    echo "Langsam wird es teuer.";
}
?>
</body></html>
```

- Falls die Bedingung hinter `if` nicht zutrifft, werden die Anweisungen hinter `else` ausgeführt.

## BEISPIEL

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<form action = "ifelse_zugang.php" method = "post">

<p>Passwort: <input name = "pw"></p>

<input type = "submit">

<input type = "reset">

</form>

</body></html>
```

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

if ($_POST["pw"] == "bingo")
    echo "Zugang gestattet";
else
    echo "Zugang verweigert";

?>

</body></html>
```

# LOGISCHE OPERATOREN

## ➤ logisches Oder (Zeichenfolge || )

- ▶ wird verwendet, wenn nur eine von mehreren Bedingungen zutreffen muss
- ▶ jede Bedingung muss vollständig formuliert werden, sonst kommt es zu einer Fehlermeldung

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php
if ($_GET["pw"] == "bingo" || $_GET["pw"] == "kuckuck")
    echo "Zugang gestattet";
else
    echo "Zugang verweigert";
?>

</body></html>
```

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>
<form action = "ifelse_zugang.php" method = "post">
<p>Passwort: <input name = "bname"> Name</p>
<p>Passwort: <input name = "pw"></p>
<input type = "submit">
<input type = "reset">
</form>
</body></html>
```

- logisches Und (Zeichenfolge `&&`)
  - ▶ wird verwendet, wenn alle Bedingungen zutreffen müssen
  - ▶ jede Bedingung muss vollständig formuliert werden, sonst kommt es zu einer Fehlermeldung
  - ▶ wenn nur eine der beiden Bedingungen nicht erfüllt wird kommt es zur Ausgabe in der `else` Verzweigung

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>
<?php
if ($_GET["bname"] == "Mair" && $_GET["pw"] == "kuckuck")
    echo "Zugang gestattet";
else
    echo "Zugang verweigert";
?>
</body></html>
```

## LOGISCHES NICHT

- logisches Nicht (Zeichenfolge ! )
  - ▶ wird der Wahrheitswert von Bedingungen umgekehrt
  - ▶ kann bei komplexen logischen Verknüpfungen hilfreich sein

- Ausdrücke mit mehreren Operatoren werden von links nach rechts aufgelöst - unter Beachtung der Rangordnung
- Die Tabelle beginnt mit der höchsten Stelle der Rangordnung
- Klammern stehen innerhalb der Rangordnung an erster Stelle. Mit ihrer Hilfe können Ausdrücke in einer gewünschten Reihenfolge verarbeiten lassen Reihenfolge

Operator	Bedeutung
( )	Klammern
! -	logisches Nicht, negatives Vorzeichen
* / %	Multiplikation, Division, Modulo-Operation
+ -	Addition, Subtraktion
< <= > >=	kleiner, kleiner oder gleich, größer, größer oder gleich
== !=	gleich, ungleich
&&	logisches Und
	logisches Oder
=	Zuweisung

# MEHRFACHE VERZWEIGUNG

- Verzweigungen mit `if` und `else` lassen sich verschachteln, sodass eine mehrfache Verzweigung möglich ist

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php
$preis = 1.12,
    if ($preis < 1) {
        echo "Unter 1 &euro; <br>";
        echo "Das ist billig";
    } else {
        if ($preis <= 1.2) {
            echo "Zwischen 1 &euro; und 1.20 &euro;<br>";
            echo "Langsam wird es teuer";
        } else {
            echo "Mehr als 1.2 &euro; <br>";
            echo "Das ist viel zu teuer";
        }
    }
?>
</body></html>
```

## SWITCH/CASE-ANWEISUNG

- alternative Schreibweise für einen bestimmten Typ von mehrfachen Verzweigungen
- kann eingesetzt werden, wenn eine bestimmte Variable auf mehrere feste Werte hin geprüft werden soll
- kann übersichtlicher sein als eine verschachtelte Verzweigung (falls viele unterschiedliche Fälle vorliegen)
- innerhalb des **switch-Blocks** wird der Wert von `$_POST["sorte"]` untersucht und alle vorhandenen Fälle (**cases**) werden der Reihe nach mit diesem Wert verglichen bis einer der Fälle zutrifft
- Die Anweisung **break** unterbricht dann alle anderen
- **default** wird benutzt, wenn keiner der genannten Fälle zutrifft

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

$liter = doubleval($_POST["liter"]);

switch($_POST["sorte"]) {

    case "N":

        $zahlung = $liter * 1.35;

        echo "$liter Liter Normal kosten $zahlung &euro;";

        break;

    case "S":

        $zahlung = $liter * 4;

        echo "$liter Liter Super kosten $zahlung &euro;";

        break;

    case "D":

        $zahlung = $liter * 1;

        echo "$liter Liter Diesel kosten $zahlung &euro;";

        break;

    default:

        echo "Als Sorte nur N, S oder D eingeben!";

}

?>

</body></html>
```



CODERS.BAY

MEHR ÜBER  
VERZWEIGUNGEN

- Falls innerhalb einer einfachen oder mehrfachen Verzweigung jeweils nur reiner HTML-Code ohne PHP-Variablen ausgegeben werden muss, ist eine gemischte Schreibweise mit PHP und HTML recht nützlich

- Ablauf wird auf mehrere PHP-Blöcke verteilt:
  - ▶ Nach der Bedingung `if($preis < 1)` kommt ein Doppelpunkt. Bedeutet: Verzweigung ist noch "offen". Anschließende HTML-Code wird nur ausgeführt wenn Bedingung `true` ist
  - ▶ es folgt die `else`-Anweisung ebenfalls mit Doppelpunkt
  - ▶ Anweisung wird mit `endif` abgeschlossen
- Dazwischen kann HTML-Code ohne `echo`, Anführungszeichen, Semikolon usw. notiert werden
- Schreibweise kann für andere Formen der Verzweigung und anderen Kontrollstrukturen benutzt werden

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php
    $preis = 1.02;
    if ($preis < 1):
?
>
    Der Preis liegt unter 1 &euro; <br>
    Das ist billig

<?php else: ?>
    Mehr als 1.2 &euro; <br>
    Das ist viel zu teuer

<?php endif; ?>

</body></html>
```

# WAHRHEITSWERTE

- Wahrheitswerte können in eigene Variablen zwischengespeichert werden
- Datentyp **boolean**
  - ▶ in diesen Datentyp wird entweder `true` oder `false` gespeichert
  - ▶ auch genannt als *boolesche Variablen*
- Zahlen, Zeichenketten und Variablen besitzen ebenfalls einen Wahrheitswert der genutzt werden kann
  - ▶ kann implizit erfolgen (durch automatische Umwandlung)
  - ▶ explizit mithilfe der Funktion `boolval()`
- mit Vergleichsoperatoren `==` und `!=` kann festgestellt werden ob zwei Werte übereinstimmen **und** denselben Datentyp haben

# BEISPIEL

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

$ww = 5 > 3;
echo "Wahrheitswert: $ww<br>";
if($ww) echo "Dieser Wert ist wahr<br><br>";

echo "Impliziert: 5 > 3: " . (5>3) . ", 5<3: " . (5<3) . "<br>";
echo "Explizit: boolval(5>3): " . boolval(5>3) . ", boolval(5<3): " . boolval(5<3) . "<br><br>";

echo "TURE: " . TRUE . ", true: " . true . "<br>";
echo "FALSE: " . FALSE . ", false: " . false . "<br>";

echo "boolval(1): " . boolval(1) . ", boolval(0): " . boolval(0) . ", boolval(-1): " . boolval(-1) . "<br>";
echo "boolval(0.0): " . boolval(0.0) . ", boolval(0.000000001): " . boolval(0.000000001) . "<br>";
echo "boolval(''): " . boolval('') . ", boolval(' '): " . boolval(' ') . "<br>";
echo "boolval('0'): " . boolval('0') . "<br>";

?>

</body></html>
```

## BEISPIEL

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

$zahl = 42;
$text = '42';
if($zahl == $text) echo "==<br>";
if($zahl != $text) echo "!=<br>";
if($zahl === $text) echo "===<br>";
if($zahl !== $text) echo "!==<br>";

?>

</body></html>
```

# TERNÄRER OPERATOR ?:

- in vielen Fällen eine Schreibabkürzung bei Verzweigungen
- der *ternäre Operator* arbeitet mit drei Operanden
  - ▶ vor dem Zeichen `?` steht Bedingung
  - ▶ zwischen dem Zeichen `?` und `:` steht das Ergebnis wenn es zu trifft
  - ▶ wenn es nicht zutrifft wird das Ergebnis nach dem Zeichen `:` ausgeliefert

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

    $preis = 42;
    echo($preis < 1 ? "Das ist billig" : "Langsam wird es teuer") . "<br>";
?>

</body></html>
```

- mit PHP 7.0 eingeführt
- Vergleichsoperator
- Ergebnisse:
  - ▶ den Wert **1**, falls der erste Wert größer ist
  - ▶ oder den Wert **-1**, falls der zweite Wert größer ist
  - ▶ oder den Wert **0**, falls beide Werte übereinstimmen
- seinen Namen hat er in Anlehnung an frühere textbasierte Computerspiele, in denen ein Raumschiff mithilfe der Zeichenfolge **<=>** angezeigt wurde

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

    echo "Erster Wert: " . (12 <=> 5) . "<br>";
    echo "Zweiter Wert: " . (5 <=> 12) . "<br>";
    echo "Werte sind gleich: " . ( 5 <=> 5) . "<br>";

?>

</body></html>
```

## EXISTENZ EINER VARIABLE

- mit Funktion `isset()` Überprüfung möglich ob eine Variable existiert
- Bsp.:
  - ▶ feststellen ob bestimmte Werte aus Formular gesendet werden
- liefert einen Wahrheitswert, daher kommt sie meist innerhalb einer Verzweigung zum Einsatz
  
- Erläuterung des Beispieles
  - 1. Vor der Zuweisung eines Werts existiert die Variable nicht
  - 2. Nach der Zuweisung eines Werts existiert die Variable
  - 3. Nach Löschung mithilfe von `unset()` existiert sie nicht mehr
  - 4. Nach erneuter Zuweisung existiert sie wieder
  - 5. Nach Löschung durch Zuweisung von `null` existiert sie nicht mehr
  
- Achtung: Viele Funktionen liefern im Fehlerfall den Wert `null` zurück

## BEISPIEL

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>    $preis = 1.02;  
<?php  
  
if (isset($preis)) echo "1: $preis<br>";  
else      echo "1: Nicht vorhanden<br>";  
  
$preis = 1.02;  
if(isset($preis)) echo "2: $preis<br>";  
else      echo "2: Nicht vorhanden<br>";  
  
unset($preis);  
if(isset($preis)) echo "3: $preis<br>";  
else      echo "3: Nicht vorhanden<br>";  
  
    if(isset($preis)) echo "4: $preis<br>";  
    else      echo "4: Nicht vorhanden<br>";  
  
    $preis = 0;  
    if(isset($preis)) echo "5: $preis<br>";  
    else      echo "5: Nicht vorhanden<br>";  
?  
</body></html>
```

## TYP PRÜFEN

- Es gibt eine Reihe von Prüffunktionen, mit deren Hilfe man den Typ einer Variablen oder eines Werts feststellen kann
  - ▶ `is_int()` prüft, ob es sich um eine ganze Zahl handelt
  - ▶ `is_float()` prüft, ob es sich um eine Zahl mit Nachkommastellen, also um eine Fließkommazahl handelt
  - ▶ `is_string()` prüft, ob es sich um eine gültige Zeichenkette handelt
  - ▶ `is_numeric()` prüft, ob es sich um einen gültigen Zahlenwert handelt oder um eine Zeichenkette, die einen gültigen Zahlenwert beinhaltet
  - ▶ `is_bool()` prüft, ob es sich um einen Wahrheitswert handelt

# BEISPIEL

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>  
<?php  
if(is_int(42))  
    echo "42 ist eine ganze Zahl<br>";  
if(!is_int(42.0))  
    echo "42.0 ist keine ganze Zahl<br>";  
if(is_float(42.0))  
    echo "42.0 ist eine Fliesskommazahl<br>";  
if(!is_float(42))  
    echo "42 ist keine Fliesskommazahl<br>";  
if(is_string("42"))  
    echo "\"42\" ist eine Zeichenkette<br>";  
if(is_string('42'))  
    echo "'42' ist eine Zeichenkette<br>";  
if(!is_string(42))  
    echo "42 ist keine Zeichenkette<br>";  
  
        if(is_numeric("42"))  
            echo "\"42\" ist numerisch <br>";  
        if(is_numeric("42.0"))  
            echo "\"42.0\" ist numerisch <br>";  
        if(is_numeric("-4.2e-3"))  
            echo "\"-4.2e-3\" ist numerisch <br>";  
        if(!is_numeric("42a"))  
            echo "\"42a\" ist nicht numerisch <br>";  
        if(is_bool(true))  
            echo "true ist boolean <br>";  
        if(is_bool(5>3 && 7<12))  
            echo "5>3 && 7<12 is boolean <br>";  
        if(!is_bool("true"))  
            echo "\"true\" ist nicht boolean <br>";  
    ?>  
</body></html>
```

## TYP PRÜFEN

- Sobald eine Zahl einen Dezimalpunkt mit einer Nachkommastelle hat, wird sie von der Funktion `is_int()` nicht mehr als ganze Zahl erkannt, sie wird aber von der Funktion `is_float()` als Fließkommazahl erkannt, selbst wenn die Nachkommastelle den Wert `0` hat
- Alles innerhalb von einfachen oder doppelten Hochkommata wird von der Funktion `is_string()` als Zeichenkette erkannt
- Ganze Zahlen, Zahlen mit Nachkommastellen oder Exponentialzahlen werden auch innerhalb einer Zeichenkette von der Funktion `is_numeric()` als gültige Zahlenwerte erkannt. Ebenso trifft das zu, wenn sie ein negatives Vorzeichen oder einen negativen Exponenten besitzen. Sobald innerhalb der Zeichenkette ein ungültiges Zeichen vorkommt, werden sie nicht mehr als gültige Zahlenwerte erkannt
- Die Werte `true` und `false` werden von der Funktion `is_bool()` als Wahrheitswerte erkannt. Das trifft auch für Bedingungen zu, die mithilfe von Vergleichsoperatoren und logischen Operatoren gebildet werden. Inhalte von Zeichenketten werden nicht als Wahrheitswerte erkannt.

# KOALESZENZOPERATOR ??

- mit PHP 7.0 eingeführt
- verschmilzt Arbeitsweise der Funktion `isset()` mit der des ternären Operators `?:` in stark verkürzter Form
- wird auch *isset ternary* Operator genannt
- trägt seinen Namen in Anlehnung an die Verschmelzung von unterschiedlichen Flüssigkeiten

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

    echo ($preis ?? "Nicht vorhanden") . "<br>";
    $preis = 1.02;
    echo ($preis ?? "Nicht vorhanden") . "<br>";

?>

</body></html>
```



CODERS.BAY

SCHLEIFEN

# EINFÜHRUNG

- Du verwendest die `for`-Schleife, wenn du die Anzahl der Wiederholungen kennst oder diese sich eindeutig im Verlauf des Programms vor der Schleife ergibt (*Zählschleife*)
- Du verwendest die `while`-Schleife oder die `do-while`-Schleife, wenn du die Anzahl der Wiederholungen nicht kennst und diese sich nicht eindeutig im Verlauf des Programms vor der Schleife ergibt. Die Wiederholung oder der Abbruch der Schleife ergibt sich erst zur Laufzeit des Programms (*bedingungsgesteuerte Schleife*)

# FOR-SCHLEIFE

- wird verwendet, um eine feste Anzahl an Wiederholungen zu erzeugen
- mithilfe des Programms werden fünf Zeilen in das Dokument geschrieben, jeweils mit dem Inhalt **Zeile: <Nummer>**

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

    for ($i=1; $i<=5; $i++) {
        echo "Zeile: $i<br>";
    }

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

## FOR-SCHLEIFE

- besteht aus: einem Kopf und einem Rumpf
- Kopf besteht aus drei Teilen, die durch Semikola voneinander getrennt sind:
  - ▶ Startwert
  - ▶ Bedingung zur Wiederholung
  - ▶ Veränderung der Schleifenvariable
- Im Beispiel ist `$i` die Schleifenvariable
- mit ihrer Hilfe wird die Schleife gesteuert
- Schleife wird so oft durchlaufen bis die Bedingung zur Wiederholung nicht mehr zutrifft
- Zahlen mit Nachkommastellen sind nicht mathematisch genau
- **Achtung!**
  - ▶ Darauf achten, dass du keine Endlosschleife erzeugst wie zum Beispiel  
`for ($i = 3; $i > 2; $i=$i+3)`  
das Programm hängt sich dabei auf!!

# BEISPIELE

Kopf der for-Schleife	Zur Verfügung stehende Werte
for ( \$i = 10; \$i <= 15; \$i++ )	10, 11, 12, 13, 14, 15
for ( \$i = 10; \$i < 15; \$i++ )	10, 11, 12, 13, 14
for ( \$i = 10; \$i >= 5; \$i-- )	10, 9, 8, 7, 6, 5
for ( \$i = 10; \$i > 15; \$i-- )	10, 9, 8, 7, 6
for ( \$i = 3; \$i <= 22; \$i=\$i+3 )	3, 6, 9, 12, 15, 18, 21
for ( \$i = 32; \$i > 12; \$i=\$i-4 )	32, 28, 24, 20, 16
for ( \$i = 12; \$i < 12.9; \$i=\$i+0.2 )	12.0, 12.2, 12.4, 12.6, 12.8
\$a = 6; \$b = 16; \$c = 2; for ( \$i = \$a; \$i < \$b; \$i=\$i+\$c )	6, 8, 10, 12, 14

- Schleifen können verschachtelt werden
- es befindet sich eine Schleife innerhalb einer anderen Schleife (*Schachtelung*)
- für später: Bearbeitung einer zweidimensionaler Struktur wird möglich
  - ▶ bsp.: eine Tabelle oder ein zweidimensionales Feld
- Beispiel:
  - ▶ die erste (äußere) Schleife wird fünfmal durchlaufen
  - ▶ innerhalb ist eine Schleife die bei jedem Durchlauf der äußeren dreimal durchlaufen wird
  - ▶ es ergeben sich insgesamt  $5 \times 3 = 15$  Wiederholungen

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

    for ($z=1; $z<=5; $z=$z+1) {

        for ($s = 1; $s <= 3; $s=$s+1) {

            echo "Ze$z/Sp$s ";

            echo "<br>";

        }

    }

?>

</body></html>
```

# SCHLEIFEN UND TABELLEN

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>

<table>
<?php
for ($i=8; $i <= 13; $i++) {
    echo "<tr>";
    echo "<td>Zeile</td>";
    echo "<td align='right'>$i</td>";
    echo "</tr>";
}
?>
</table>
</body></html>
```

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>

<table>
<?php
for ($i=8; $i <= 13; $i=$i+1) {
    echo "<tr>";
    for ($s=1; $s <=5; $s=$s+1)
        echo "<td align='right'>$i/$s</td>";
    echo "</tr>";
}
?>
</table>
</body></html>
```

- Die Ausrichtung der Zellen (`align='right'`) muss innerhalb der Zeichenkette (die zwischen doppelten Hochkommata steht) in einfachen Hochkommata angegeben werden, da in PHP ansonsten die Zeichenkette zu früh beendet würde.

## WHILE-SCHLEIFE

- wird dazu benutzt, eine unbestimmte Anzahl an Wiederholungen zu erzeugen
- das Ende der Wiederholungen wird bei einem der Schleifendurchläufe erreicht
- wird häufig für Datenbankabfragen eingesetzt
- im folgenden Beispiel wird gewürfelt
  - ▶ die gewürfelten Zahlen werden addiert
  - ▶ dies wird solange wiederholt, bis die Summe der gewürfelten Zahlen 25 beträgt oder darüber liegt
  - ▶ zum Erzeugen der "zufälligen" Würfelergebnisse wird ein einfacher Zufallszahlengenerator verwenden: `rand()`
  - ▶ Zufallszahlengenerator muss zunächst initialisiert werden, damit er tatsächlich zufällige Ergebnisse produziert  
    → `srand()`
  - ▶ Funktion `microtime()` liefert die aktuelle Systemzeit
- Bedingung zur Wiederholung muss in Klammer stehen

# BEISPIEL

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

    srand((double)microtime()*1000000);

    $summe = 0;

    while ($summe < 25) {
        $zufallszahl = rand(1,6);
        $summe = $summe + $zufallszahl;
        echo "Zahl $zufallszahl, Summe $summe<br>";
    }
?>

</body></html>
```

## DO-WHILE-SCHLEIFE

- arbeitet wie die `while`-Schleife
- Unterschied: die Prüfung für die Wiederholung wird erst am Ende der Schleife durchgeführt.

```
<!DOCTYPE html><html><head><meta charset="utf-8"></head><body>

<?php

    srand((double)microtime()*1000000);

    $summe = 0;

    do {
        $zufallszahl = rand(1,6);
        $summe = $summe + $zufallszahl;
        echo "Zahl $zufallszahl, Summe $summe<br>.";
    } while ($summe < 25);

?>

</body></html>
```

# SCHLEIFENABBRUCH BREAK

- Mithilfe der Anweisung `break` kann eine Schleife vorzeitig beendet werden
- zusätzliche Möglichkeit für eine Schleifensteuerung
- **Hinweis**
  - ▶ Eine `break`-Anweisung, die nicht in einem `switch`-Block steht, aber innerhalb einer Schleife, wird immer gemeinsam mit einer Bedingung auftreten, da der vorzeitige Abbruch einer Schleife nur in einem "Sonderfall" erfolgen sollte
- Beispiel
  - ▶ Die `$zaehler` Variable wird immer um eines erhöht bis die Zahl **6** erreicht bzw. überschritten ist
    - Schleife bricht unmittelbar ab
  - ▶ Der Vergleich `if($zaehler == 6)` hätte auch zu einem Abbruch geführt, allerdings nur bei einer Erhöhung um 1.

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>

<?php

srand((double)microtime()*1000000);

$summe = 0;
$zaehler = 0;

while ($summe < 25) {

    $zufallszahl = rand(1,6);

    $summe = $summe + $zufallszahl;
    $zaehler = $zaehler + 1;

    echo "Nr. $zaehler, Zahl $zufallszahl, ";
    echo "Summe $summe<br>.";

    if ($zaehler >= 6) break;

}

?>

</body></html>
```

- Anweisung `continue` sofort für den Abbruch des aktuellen Schleifendurchlaufs
- Schleife wird anschließen unmittelbar mit dem nächsten Durchlauf fortgesetzt
- Beispiel: Für die Werte `5` bis `12` wird keine Ausgabe vorgenommen

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>

<?php
    for ($i = 1; $i <= 15; $i++) {
        if ($i >= 5 && $i <= 12)
            continue;
        echo "Zeile $i<br>";
    }
?>
</body></html>
```



CODERS.BAY

FUNKTIONEN

# FUNKTIONEN

- es gibt zahlreiche vordefinierte Funktionen
- eigene Funktionen, benutzerdefinierte Funktionen
  - ▶ gleiche oder ähnliche Vorgänge müssen nur einmal beschrieben und können beliebig oft ausgeführt werden
  - ▶ Programme können modularisiert werden können in kleinere Bestandteile zerlegt werden, die übersichtlicher sind und einfach gewartet werden können
  - ▶ werden mithilfe von **function ... () { ... }** definiert
  - ▶ Name der Funktion folgt nach dem Schlüsselwort **function**
  - ▶ in den Runden Klammern folgen die Parameter, sofern welche vorhanden sind
  - ▶ in den geschweiften Klammern folgt der eigentliche Funktionsrumpf

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head>
<body>
<?php
    function trennstrich() {
        echo "<br>";
        for ($i=1; $i<=40; $i++) {
            echo "-";
        }
    }
    trennstrich();
    echo "Dies ist ein Programm, ...";
    trennstrich();
?>
</body>
</html>
```

- es können nur Funktionen aufgerufen werden, die das Programm schon kennt, deshalb müssen es also:
  - ▶ entweder vordefinierte Funktionen sein
  - ▶ oder im Dokument definiert sein
  - ▶ oder aus einer eigenen externen Datei stammen
- Für Funktionen mit Parametern bzw. Rückgabewert gilt folgendes:
  - ▶ Eine Funktion ohne Parameter führt bei jedem Aufruf immer genau die gleiche Aufgabe aus
  - ▶ Eine Funktion mit einem oder mehreren Parametern führt bei jedem Aufruf in Abhängigkeit von den Parametern ähnliche Aufgaben aus
  - ▶ Eine Funktion mit einem Rückgabewert führt gleiche oder ähnliche Aufgaben aus und liefert ein Ergebnis an die aufrufende Stelle zurück
- Für den Namen einer Funktion gelten die gleichen Regeln wie für den Namen einer Variablen
  - ▶ Ausnahme: darf nicht mit einem Dollarzeichen anfangen

# FUNKTIONEN MIT EINEM PARAMETER

- eine Funktion mit einem Parameter führt bei jedem Aufruf in Abhängigkeit vom Parameterwert ähnliche Aufgaben aus
- Funktion **trennstrich()** wird insgesamt 4x aufgerufen, jedes mal mit einem anderen Wert innerhalb der Klammer hinter dem Funktionsnamen = Parameter
  - ▶ es kann sich dabei um eine Zahl, eine Variable oder das Ergebnis einer Berechnung handeln
- Parameter wird an Funktion übergeben
- dort wird Wert in der Variable **\$anzahl** gespeichert
- **\$anzahl** steuert die Ausführung der **for**-Schleife

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>

<?php

function trennstrich($anzahl) {
    echo "<br>";
    for ($i=1; $i<=$anzahl; $i++) {
        echo "-";
    }
}

trennstrich(30);
echo "Dies ist ein Programm, ...";
trennstrich(40);
echo "sind die Trennstriche";
$x = 20;
trennstrich($x);
echo "unterschiedlich lang";
trennstrich($x * 3);

?>

</body></html>
```

- › bei Übergabe von mehreren Parametern ist die Anzahl, der Datentyp (Zahl oder Zeichenkette) und die Reihenfolge der Parameter wichtig
  - ▶ erster Wert wird an den ersten Parameter übergeben
  - zweiter Wert an den zweiten Parameter, usw.
- › beim Aufruf der Funktion `flexloop()` müssen jeweils drei Parameter übergeben werden, und war durch Kommata voneinander getrennt
- › diese werden in der vorliegenden Reihenfolge den Variablen `$von`, `$bis`, und `$schritt` zugeordnet
- › Variablen werden zur Steuerung der for-Schleife in der Funktion verwendet
- › Parameter werden bei der Übergabe durch Kommata getrennt

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>
<?php
function flexloop($von, $bis, $schritt) {
    echo "Eine Schleife von $von bis $bis mit"
    . " der Schrittweite $schritt<br>";
    for ($i=$von; $i<=$bis; $i=$i+$schritt) {
        echo "$i ";
    }
}
echo "<p>Nummer 1:<br>";
flexloop(5,27,3) . "</p>";
echo "<p>Nummer 2:<br>";
flexloop(10,10,4) . "</p>";
echo "<p>Nummer 3:<br>";
$x = 100;
$y = 200;
$z = 10;
flexloop($x, $y, ($y-$x)/8+$z);
echo "</p>";
?>
</body></html>
```

# RÜCKGABEWERT EINER FUNKTION

- viele eigene und vordefinierte Funktionen arbeiten auf die folgenden Weisen:
  - ▶ innerhalb der Funktion wird ein Ergebnis ermittelt und mithilfe eines sogenannten Rückgabewerts an die aufrufende Stelle zurückgeliefert
  - ▶ dieser Wert muss entweder in einer Variablen gespeichert oder direkt ausgegeben werden, andernfalls geht er verloren
  - ▶ Anweisung **return** wird der Wert an die aufrufende Stelle zurückgeliefert und kann dort weiterverarbeitet werden
  - ▶ Aufruf einer Funktion darf nicht innerhalb einer Zeichenkette stehen
  - ▶ mit Anweisung **return** kann Funktion auch vorzeitig verlassen werden

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>
<?php
    function add($z1, $z2) {
        $summe = $z1 + $z2;
        return $summe;
    }
$c = add(3,4);/* Aufruf und Zuweisung */
echo "Summe: $c<br>";

$x = 5;
$c = add($x, 12);
echo "Summe: $c<br>";

/* Aufruf innerhalb der Ausgabe */
echo "Summe: " . add(13,2) . "<br>";
?>
</body></html>
```

- Variablen werden nach ihrem **Gültigkeitsbereich** unterschieden  
ist der Bereich, in dem die betreffende Variable mit ihrem Wert bekannt ist  
man unterscheidet
  - ▶ **lokale Variablen**
    - diese werden innerhalb einer Funktion definiert und stehen nur innerhalb dieser Funktion zur Verfügung
  - ▶ **globale Variable**
    - diese werden außerhalb einer Funktion definiert und stehen nur außerhalb derselben zur Verfügung
  - ▶ **superglobale Variablen**
    - bei diesen Variablen handelt es sich um PHP-Systemvariablen. Sie stehen sowohl innerhalb als auch außerhalb von Funktionen zur Verfügung. Zu ihnen zählt das assoziative Feld `$_POST`
- **Regeln**
  - ▶ Ein Parameter, der als Kopie an eine Funktion übergeben wird, ist dort lokal

- ▶ lokale Variablen gleichen Namens in unterschiedlichen Funktionen haben nichts miteinander und auch nichts mit einer globalen Variablen gleichen Namens zu tun
- ▶ falls eine globale Variable innerhalb einer Funktion benutzt wird, muss sie dort entweder mit dem Schlüsselwort `global` bekannt gemacht oder als Parameter übergeben werden

## ➤ Hinweis

- ▶ die Variable eines Programms sollte immer "so lokal wie möglich" und "so wenig global wie möglich" sein  
Vorteile:
  - **Modularisierung** des Programms wird verbessert, das heißt die Zerlegung eines Programms in übersichtliche Programmteile mit klar definierten Schnittstellen zwischen den Teilen
  - **Wiederverwendbarkeit** der Funktion für andere Programme wird erleichtert
  - **Variablen** können nicht so leicht aus Versehen an weit voneinander entfernten Stellen verändert werden

## BEISPIEL

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>
<?php
function summiere() {
    echo "Variable z: $z<br>";
    global $x;
    $y = 35;
    $z = $x + $y;
    echo "Variable z: $z<br>";
}

$x = 6; $y = 62;
$z = $x + $y;
summiere();
echo "Variable z: $z<br>";
?>
</body></html>
```



CODERS.BAY

BEHANDLUNG VON  
FEHLERN

# FEHLERBEHANDLUNG

- › Konzept: ***Exception Handling***
- › bietet die Möglichkeit, bestimmte Fehler abzufangen
- › Programm kann weiterlaufen, obwohl Fehler aufgetreten ist
- › Exception Handling
  - ▶ dabei wird der Codebereich, in dem ein Fehler auftreten kann, in einen sogenannten **try**-Block eingeschlossen
  - ▶ falls ein definierter Fehler auftritt, wird ein Objekt der Klasse **Exception** durch die Anweisung **throw** erzeugt
  - ▶ anschließend wird statt des restlichen Codes im **try**-Block der Code im dazugehörigen catch-Block ausgeführt; der Fehler wird somit "abgefangen"

## BEISPIEL

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>
<?php
/* Variable unbekannt */
try {
    if(!isset($x))
        throw new Exception("Variable unbekannt");
    echo "Variable: $x<br>";
} catch(Exception $e) {
    echo $e->getMessage() . "<br>";
} finally {
    echo "Ende, Variable unbekannt<br>";
}

/* Division durch 0 */
$x = 42;
$y = 0;
$z = $x / $y;
echo "Division: $x / $y = $z<br>";

/* Division durch 0 */
$x = 42; $y = 0;
try {
    if($y == 0)
        throw new Exception("Division durch 0");
    $z = $x / $y;
} catch(Exception $e) {
    echo $e->getMessage() . "<br>";
}
?>
</body></html>
```



CODERS.BAY

FELDER

## FELDER - ARRAY

- zur Speicherung einer größeren Menge von zusammengehörigen Daten kann ein Feld von Variablen mit einem einheitlichen Namen genutzt werden
- Felder bieten eine schnelle und komfortable Verarbeitung
  - ▶ **Numerische indizierte Felder**
    - Die einzelnen Variablen in einem numerisch indizierten Feld werden über eine laufende Nummer innerhalb des Felds angesprochen
  - ▶ **Assoziative Felder** (auch **Hash-Tabelle** genannt)
    - die einzelnen Variablen in einem assoziativen Feld werden über eine eindeutige Bezeichnung innerhalb des Felds angesprochen

- › der Wert innerhalb eckiger Klammern bei der Ausgabe eines Arrays nennt man Index
- › alternative Schreibweise zur Erzeugung eines numerischen indizierten Feldes  
`$tp = [17.5, 19.2, 21.8, 21.6, 17.5];`

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>
<?php
$tp = array(17.5, 19.2, 21.8, 21.6, 17.5);
$tp[5] = 20.2;
$tp[6] = 16.6;
/* Achtung Endlosschleife */
for($i=0; $i<=6; $i+1)
    echo "Temperatur $i: $tp[$i]<br>";
?>
</body></html>
```

## ASSOZIATIVE FELDER

- Elemente eines assoziativen Felds werden über eine Schlüsselbezeichnung (key) identifiziert
- möglich, Feldname eindeutige Begriffe zuzuordnen -> Suche wird vereinfacht
- alternative Schreibweise zur Erzeugung eines numerischen indizierten Feldes  
`$tp = [17.5, 19.2, 21.8, 21.6, 17.5];`
  
- mithilfe der Funktion `array()` wird die Variable `$tp` zu einem Feld mit drei Elementen. Diese Elemente haben eindeutige Schlüsselbezeichnungen (Keys) und dazugehörige Werte (Values). Diese Paare werden einander mit dem Operator => zugeordnet. Der Key muss dabei zwischen doppelten Hochkommata geschrieben werden
- Felder können auch einfach durch die Zuweisung einzelner Elemente erzeugt oder vergrößert werden. Dies ist hier mit den Zuweisungen in der Form `$tp["Samstag"] = 20.2;` usw.. geschehen.
- Anders als bei numerisch indizierten Feldern sollten die Ausgaben von einzelnen Elementen assoziativer Felder nicht innerhalb von Zeichenketten vorgenommen werden. Ist zwar möglich, führt aber häufig zu Fehlinterpretationen und zu Fehlern.

## BEISPIEL

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>
<?php
$tp = array(
    "Montag"=>17.5,
    "Dienstag"=>19.2,
    "Mittwoch"=>21.8
);
$tp["Donnerstag"] = 20.2;
$tp["Freitag"] = 16.6;
$tp["Samstag"] = 20.2;
$tp["Sonntag"] = 21.6;

// ein bestimmtes Element
echo "<p>" . $tp["Montag"] . "</p>";

// Alle Keys und Values aus dem Feld
echo "<p>";
foreach($tp as $name=>$wert)
    echo "$name, $wert<br>";
echo "</p>";

/* Nur alle Values aus dem Feld zum Berechnen d
es
Mittelwerts */
$summe = 0;
foreach($tp as $wert)
    $summe = $summe + $wert;
$mittelwert = $summe / 7;
echo "<p>Mittelwert: $mittelwert</p>";

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

## FOREACH

- die **foreach**-Schleife bietet eine Möglichkeit, alle Elemente eines assoziativen Felds auszugeben
  - ▶ in der ersten Schleife sorgt **foreach(\$tp as \$name=>\$wert)** dafür, dass bei jedem Schleifendurchlauf jeweils ein einzelnes Key-Value-Paar in den Variablen **\$name** und **\$wert** bereitgestellt wird
  - ▶ in der zweiten Schleife **sort foreach(\$tp as \$wert)** dafür, dass bei jedem Schleifendurchlauf jeweils nur der Value jedes Element in der Variablen **\$wert** bereitgestellt wird
  - ▶ wie bei allen Verzweigungen und Schleifen gilt:
    - ➔ sollen mehrere Anweisungen ausgeführt werden, müssen sie innerhalb von geschweiften Klammern in einem Anweisungsblock notiert werden
  - ▶ innerhalb einer **foreach**-Schleife wird jeweils nur mit einer Kopie eines Feldelements gearbeitet. Eine Veränderung dieser Kopie hat keine Auswirkung auf die Inhalte des Felds
- alternative Schreibweise
  - ▶ **\$tp = ["Montag"=>17.5, "Dienstag"=>19.2, ...];**

# REKURSIVE FUNKTIONEN

- Funktionen können jederzeit andere Funktionen aufrufen = verschachtelten Aufrufen
- Funktionen können sich auch selbst aufrufen = Rekursion
- eine rekursive Funktion muss eine Verzweigung beinhalten, die die Rekursion wieder beendet, da es sonst zu einer endlosen Kette von Selbstaufufen kommt.

```
<!DOCTYPE html><html>
<head><meta charset="utf-8"></head>
<body>
<?php
/* Schleife */
$x = 1.5;
echo "x: $x<br>";
while($x > 0.1) {
    $x = $x / 2;
    echo "x: $x<br>";
}
echo "<br>";

/* Rekursion */
function halbieren(&$z) {
    $z = $z / 2;
    if($z > 0.1) {
        echo "z: $z<br>";
        halbieren($z);
    }
}
$x = 1.5;
echo "x: $x<br>";
halbieren($x);
echo "x: $x<br>";
?>
</body></html>
```

## INCLUDE-Anweisung

- benutzerdefinierte Funktionen die in mehreren Programmen zum Einsatz kommt, können in externe Dateien ausgelagert werden
- mit der **include**-Anweisung wird der Inhalt dieser Dateien in dasjenige Programm eingebunden, das sie benötigt
- Empfehlung:
  - ▶ aus geltender Konvention und aus Gründen der Sicherheit empfiehlt es sich der Datei die Endung **.inc.php** zu geben

```
<?php
function maxi($x, $y) {
    if($x > $y) {
        $erg = $x;
        return $erg;
    } else {
        $erg = $y;
        return $erg;
    }
}
?>
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>
<?php
    include "function_einbinden.inc.php";
    $a = 2;
    $b = 6;
    $c = maxi($a, $b);
    echo "Das Maximum von $a und $b ist $c";
?>
</body></html>
```

# GENERATOREN

- gibt es seit PHP 5.5
- besonderer Typ von Funktionen, da sie nicht nur einen einzelnen Wert liefert, sondern eine ganze Reihe von Werten
- dabei steht zu jedem Zeitpunkt nur ein Wert im Arbeitsspeicher. Es müssen nicht alle Werte der Reihe gleichzeitig im Arbeitsspeicher aufbewahrt werden. Auf diese Weise wird die Nutzung des Arbeitsspeichers optimiert. Als Generator wird eine Funktion definiert, aus der die Werte mithilfe des Schlüsselworts **yield** geliefert werden
- Im Unterschied zu einer normalen Funktion, die nur einen Wert liefern kann, liefert eine Generatorfunktion mehrere Werte. Wird sie innerhalb einer **foreach**-Schleife aufgerufen, endet die Generatorfunktion nicht nach einem Aufruf, sondern wird beim nächsten Aufruf fortgesetzt. Auf diese Weise werden alle generierten Werte nacheinander geliefert

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>
<?php
    srand((int)microtime()*1000000);
    function wuerfelwertGenerator() {
        for($i=1; $i<=10; $i++)
            yield rand(1,6);
    }
    foreach(wuerfelwertGenerator() as $wert)
        echo "$wert ";
    ?>
</body></html>
```



CODERS.BAY

BEISPIELE

Bei der Entwicklung deiner eigenen Programme solltest du Schritt für Schritt vorgehen. Stelle dir zuerst einige Überlegungen darüber an, wie das gesamte Programm aufgebaut sein sollte, und zwar auf Papier. Aus welchen Teilen sollte es nacheinander bestehen? Versuche nicht, das gesamte Programm mit all seinen komplexen Bestandteilen auf einmal zu schreiben! Dies ist der größte Fehler, den Einsteiger machen können.

Schreibe zunächst eine einfache Version des ersten Programmteils. Anschließend testest du sie. Erst nach einem erfolgreichen Test fügst du den folgenden Programmteil hinzu. Nach jeder Änderung testest du wiederum. Sollte sich ein Fehler zeigen, weißt du, dass er aufgrund der letzten Änderung aufgetreten ist. Nach dem letzten Hinzufügen hast du eine einfache Version deines gesamten Programms.

Nun änderst du einen Teil deines Programms in eine komplexere Version ab. Auf diese Weise machst du dein Programm Schritt für Schritt komplexer, bis du schließlich das gesamte Programm so erstellt hast, wie es deiner anfänglichen Überlegung auf Papier entspricht.

Manchmal ergibt sich während der praktischen Programmierung noch die eine oder andere Änderung gegenüber deinem Entwurf. Das ist kein Problem, solange

sich nicht der gesamte Aufbau ändert. Sollte das allerdings der Fall sein, kehre noch einmal kurz zum Papier zurück und überdenke den Aufbau. Das bedeutet nicht, dass du die bisherigen Programmzeilen löschen musst, aber möglicherweise musst du sie ein wenig ändern und anders anordnen.

Schreibe deine Programme übersichtlich. Falls du gerade überlegst, wie du drei, vier bestimmte Schritte deines Programms auf einmal machen kannst: Erstelle darauf einfach einzelne Anweisungen, die der Reihe nach ausgeführt werden. Dies vereinfacht eine eventuelle Fehlersuche. Falls du dein Programm später einmal ändern oder erweitern möchtest, gelingt der Einstieg in den Aufbau des Programms wesentlich schneller.

Eine typische Frage, die du dich besonders im Zusammenhang mit Verzweigungen und Schleifen immer wieder stellen solltest, lautet: Habe ich alle Klammern, die ich geöffnet habe, auch wieder geschlossen?

Du könntest die Anweisung echo natürlich auch zur Kontrolle von Werten und zur Suche von logischen Fehlern einsetzen. Außerdem kannst du einzelne Teile deines Programms in Kommentarklammern setzen, um festzustellen, welcher Teil des Programms fehlerfrei läuft und welcher Teil demnach fehlerbehaftet ist.

# FORMATIERUNG VON ZAHLEN

```
<!DOCTYPE html><html><head>
<meta charset="utf-8"></head><body>
<?php
    echo "<p><b>Zahlenformatierung:</b></p>";
    $d = 10000 / 7;
    echo "<p>Variable d: $d</p>";
    echo "<p>Mit Tausenderkomma, ohne Dezimalstellen: <br>" .
        . number_format($b) . "</p>";
    echo "<p>Mit Taussenderkomma, auf drei Stellen gerundet:<br>" .
        . number_format($b,3) . "</p>";
    echo "<p>Mit Taussenderpunkt, auf drei Stellen gerundet:<br>" .
        . number_format($b,3,",",".") . "</p>";
?
</body></html>
```



CODERS.BAY

ENDE