P H P - Dokumentation

Eine vollständige beschreibung von PHP findet man im Internet auf der seite **php.net/manual/de**

inhalt

autor: **B. Hartard**

stand: **3.4 / 25.03.2021**

1. einleitung

1.1 was ist PHP

Das kürzel **PHP** bedeutete ursprünglich **Personal Home Page**, wird heute etwas gewaltsam als **Hyper Text Preprozessor** gedeutet und ist eine script-sprache zur dynamischen erstellung von WEB-seiten. Um zu verstehen, was PHP tut, ist ein kurzer blick notwendig auf das, was geschieht, wenn eine seite geladen wird. Der browser (Internet Explorer, Firefox, Chrome und wie sie sonst noch heißen) fordert mit hilfe einer adresse eine seite an, die auf dem server gespeichert ist, auf den die adresse zeigt. Die seite wird vom server zum browser übertragen, der die **HTML**-anweisungen der seite auswertet und die seite entsprechend darstellt. Bei diesem vorgang spielen die script-sprachen (Javasript, Perl, PHP und ggf. andere) eine große rolle. Die anweisungen der script-sprachen sind in die seite **eingebettet**, sind aber in der darstellung der seite nicht sichtbar. Die anweisungen von **Javascript** werden in der regel erst ausgeführt, wenn die seite beim anwender angezeigt ist und können dann das ausehen der seite durchaus verändern. Auf einzelheiten kann hier nicht eingegangen werden. Die anweisungen von **PHP** werden dagegen serverseitig ausgeführt, d.h. **bevor** die seite zum browser übertragen wird. Dabei werden meist wesentliche teile der seite erst durch die ausführung der PHP-anweisungen erzeugt oder aus einer datenbank gelesen und in die seite eingebaut. Wenn die seite endlich zum browser übertragen wird, sind die PHP-anweisungen verschwunden.

Zu HTML und Javascript vgl. die dazu vorhandenen beschreibungen.

1.2 aufgaben von PHP

Das sind die hauptaufgaben von PHP:

- Abhängig von daten, die beim aufruf der seite zum server geschickt werden, die seite gestalten.
- Dateien und datenbanken, die auf dem server gespeichert sind, bearbeiten und die daten der dateien und datenbanken zur gestaltung der seite verwenden.
- Dateien zum server hochladen oder zum anwender herunterladen.
- Formulare erstellen und / oder auswerten.

Zur erfüllung dieser aufgaben verfügt PHP über variable, konstanten und anweisungen. Es sind ferner für verschiedene zwecke funktionen vorhanden, zudem besteht die möglichkeit, eigene funktionen zu konstruieren. Mit ausnahme der bearbeitung von datenbanken werden die möglichkeiten von PHP in dieser dokumentation beschrieben. Für die bearbeitung von datenbanken vgl. die beschreibung von MySQL

1.3 PHP-anweisungen

1.3.1 formales

Für die beschreibung der PHP-anweisungen gilt in dieser unterlage der folgende formalismus.

normale schrift	die angabe ist genau so zu schreiben
kursiv oder ***kursiv***	das ist ein platzhalter für einen wert, der hier anzugeben ist
[ ]	angaben in eckigen klammern können wahlweise gemacht werden.
. . .	die vorangehende angabe kann mehrfach wiederholt werden
|	durch einen senkrechten strich getrennte angaben sind alternativ
`Courier`	beispiel in PHP oder HTML-code

1.3.2 PHP-abschnitt

PHP-anweisungen stehen immer in einem PHP-abschnitt, der wiefolgt definiert ist:

<?php
anweisung;
. . .
anweisung;
?>

hinweise

- Eine anweisung wird mit einem strichpunkt abgeschlossen.
- PHP-abschnitte können überall in einer seite eingefügt werden.
- Eine datei, die eine HTML-seite mit PHP-abschnitten enthält, hat die dateinamenerweiterung **.php**, dadurch wird serverseitig erkannt, dass die seite vor dem absenden zum browser mit hilfe von PHP-anweisungen zu bearbeiten ist.

1.4 echo-anweisung

Das ist wohl die wichtigste anweisung, mit der das meiste erledigt wird.

echo *variable* | *konstante* | *zeichenkette* ;

Der inhalt der variablen, konstanten oder zeichenkette wird in die seite geschrieben, es wird dabei nicht geprüft, ob der browser damit etwas anfangen kann. In den meisten fällen ist der inhalt eine **HTML**-anweisung.

Die beschreibung der echo-anweisung ist hier noch ein wenig vorläufig, sie wird noch vertieft (vgl. 4.3). Trotzdem schon jetzt als beispiel ein kleiner ausschnitt aus einer seite.

So sieht das in der datei aus, die auf dem server gespeichert ist: erst eine HTML-anweisung, dann ein PHP-abschnitt, dann wieder eine HTML-anweisung

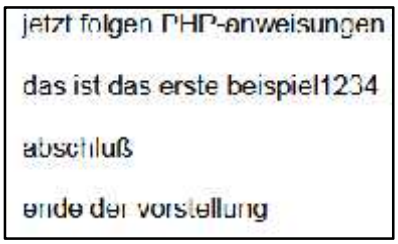
```
<p>jetzt folgen PHP-anweisungen</p>
<?php
   echo "das ist das erste beispiel";
   echo 1234;
   echo "<p>abschluß</p>";
?>
<p>ende der vorstellung</p>
```

In der seite, so wie sie nach ausführung der PHP-anweisungen beim browser landet, sieht das so aus:

```
<p>jetzt folgen PHP-anweisungen</p>
das ist das erste beispiel1234<p>abschluß</p><p>ende der vorstellung</p>
```

Wie man erkennen kann, schreibt PHP alles ohne punkt und komma in die seite, das ist sicher nicht ideal, in der praxis macht man das auch etwas anders, aber das kann erst später erklärt werden. Der browser kommt aber damit klar.

Und so wird die seite vom browser angezeigt.



1.5 kommentare

In einen PHP-abschnitt können kommentare eingefügt werden; sie haben keine funktion bei der gestaltung der seite, sind aber zur dokumentation sehr nützlich. Es gibt zwei verschiedenen formate:

einzeiliger kommentar

Ein einzeiliger kommentar beginnt mit zwei schrägstrichen und bildet entweder eine eigene zeile oder steht in einer zeile nach einer anweisung.

```
// das ist eine kommentarzeile
echo "<p>beliebiger text</p>";         //  anweisung mit einem kommentar
```

mehrzeiliger kommentar

Ein mehrzeiliger kommentar beginnt mit **/*** (schrägstrich, stern), erstreckt sich über beliebig viele zeilen und endet mit ***/** (stern, schrägstrich). Der kommentar kann auch in einer zeile nach einer anweisung beginnen.

```
/*  das ist eine kommentarzeile  */
/*  mit dieser methode kann
    ein kommentar auch über
    mehrere zeilen gehen  */
```

Den mehrzeiligen kommentar benützt man gerne, um eine folge von **PHP**-anweisungen ungültig zu machen, die man noch nicht löschen will, vielleicht braucht man sie doch mal wieder.

```
/*  echo "<p>irgend ein text, den man </p>";
    echo "<p>vielleicht doch mal braucht </p>";  */
```

1.6 hinweis zu den beispielen

In der homepage **hartard-bernhard.de** kann unter dem menüpunkt **doku / PHP** die vorliegende berschreibung interaktiv gelesent werden. Dabei werden nahezu alle beispiele auch ausgeführt und die ergebnisse angezeigt. Zudem sind die beispiele kapitelweise zusammengefaßt und können als eine seite aufgerufen werden. Die seite kann dabei auch heruntergeladen werden.

2. variable und konstanten

2.1 variable definieren

Variable haben einen namen, der mit dem zeichen $ beginnt, dem dann weitere alphanumerische zeichen folgen, aber keine geschlossene umlaute oder scharfes ß. Auch sonderzeichen sollte man vermeiden, denn einige sind zulässig und einige nicht, wer soll sich das denn merken. Außerdem ist zu beachten, dass groß- und kleinschreibung relevant ist, **$var** und **$VAR** sind unterschiedliche variable.

Eine variable wird dadurch definiert, dass ihr ein wert zugewiesen wird, der wert bestimmt den typ der variablen.

$var = *numerischer-wert* | *zeichenkette* | *variable* | *konstante*

unset (*variable*)

Mit dieser funktion wird eine variable gelöscht; gemeint ist nicht der inhalt, sondern die variable selbst, d.h. sie ist dann nicht mehr verfügbar.

2.2 numerische variable

2.2.1 variable definieren

Eine numerische variable wird dadurch definiert, dass ihr ein numerischer wert zugewiesen wird.

numerische werte

123 1.23 +123 -123 .123 0.123

20e2 das ist 20 * 10 hoch 2 = 2000

20e-2 das ist 20 * 10 hoch -2 oder 20 / 100 = 0.2

angabe einer oktal- oder hexa-zahl

oktal	**0**12	ergibt 10	achtung: es muss eine null am anfang stehen
hexa	0xFF	ergibt 255	

2.2.2 explizit umwandeln

Mit der funktion **intval** wandelt man eine dezimalzahl in eine ganzzahl um, dabei wird nicht gerundet, sondern nach dem dezimalpunkt abgeschnitten. Wenn der umzuwandelnde wert nicht numerisch ist, ist das ergebnis 0.

```
$var = 12.95;
$erg = intval($var);              ergibt 12
$erg = intval("abc");             ergibt 0
```

Mit der funktion **floatval** wird eine zeichenkette soweit wie möglich in eine dezimalzahl umgewandelt. Ist keine umwandlung möglich, ist das ergebnis 0.

zuweisung	***ergebnis***
`erg = floatval("123.456");`	123.456
`erg = floatval("123.456mist");`	123.456
`erg = floatval("123.4mist56");`	123.4
`erg = floatval("mist123.456");`	0

2.3 string- oder zeichenketten-variable

2.3.1 definition

Eine string-variable wird dadurch erzeugt, dass ihr eine zeichenkette oder eine string-variable als wert zugewiesen wird.

$charvar = "zeichen" | 'zeichen' | $var | "$var" | '$var' ;

Eine zeichenkette ist eine folge von zeichen die in anführungszeichen oder apostrophe eingeschlossen sind. Wird eine variable als wert zugewiesen, kann diese variable ebenfalls in anführungszeichen oder apostrophe eingeschlossen werden, dabei ist aber folgendes zu beachten:

$var steht in anführungszeichen	das ergebnis ist der inhalt von $var als zeichenkette
$var steht in apostrophe	das ergebnis ist der name von $var

zuweisung	***ergebnis***
`$var1 = "ein beispiel";`	ein beispiel
`$var2 = 'ein beispiel';`	ein beispiel
`$var3 = $var1;`	ein beispiel
`$var4 = "$var1";`	ein beispiel
`$var5 = '$var1';`	$var1

2.3.2 zeichenketten zusammenhängen

Zeichenketten und string-variable kann man mit einem punkt zu einer neuen zeichenkette oder string-variablen verbinden. Verbindet man einen numerischen wert mit einer zeichenkette oder string-variablen, ist das ergebnis immer eine zeichenkette.

zuweisung	***ergebnis***
`$var1 = "zeichenkette";`	zeichenkette
`$var2 = "eine " . $var1;`	eine zeichenkette
`$num  = 20;` `$var3 = $num . " zeichen";`	20 als numerischer wert 20 zeichen
`$var4 = $var2 . " mit " . $var3;`	eine zeichenkette mit 20 zeichen
`var5  = "das ist eine ";` `var5 .= "zeichenkette";`	das ist eine das ist eine zeichenkette

2.3.3 besonderheiten

Mit string-variablen kann man durchaus basteln, man hat beispielsweise in einer variablen den namen **Otto** stehen und hätte in einer anderen gerne **Ottos Alter ist** stehen. Man kann natürlich einfach eine weitere variable mit diesem inhalt erzeugen, aber es geht auch anders:

zuweisung	***ergebnis***
`$name = "Otto";`	Otto
`$var = "$name" . "s Alter ist";`	Ottos Alter ist
`$var = "$names Alter ist";`	**das ist falsch**, die variable **$names** ist nicht definiert
`$var = "{$name}s alter ist";`	Ottos Alter ist mit den geschweiften klammern isoliert man die variable von allem, was davor oder dahinter steht und in der zeichenkette steht der inhalt der variablen.

achtung

Eine string-variable ist ein kompliziertes gebilde, sie besteht nicht einfach aus **einem** wert, sondern aus einzelnen zeichen, die man per index ansprechen kann, d.h. folgende anweisungen sind möglich:

```
$kette = "ABCDE";
$char  = $kette[3];                $char enthält jetzt D
```

Außerdem gibt es eine unzahl von vordefinierten funktionen, die aber aus guten gründen noch nicht hier behandelt werden (vgl. 7.).

2.4 logische variable

Eine logische variable wird dadurch definiert, dass ihr der wert true oder false zugewiesen wird.

$varlog = true | false ;

2.5 konstanten

Konstanten haben einen namen ohne vorangestelltes $-zeichen; dem namen wird ein wert zugeordnet. Konstanten werden dann als sog. selbstdefinierende werte verwendet, sie dürfen aber weder in anführungszeichen noch in apostrophe eingeschlossen werden, auch geschweifte klammern helfen da nicht. Das macht ihre verwendung etwas mühsam.

define ("name", *numerischer-wert* | *zeichenkette*);

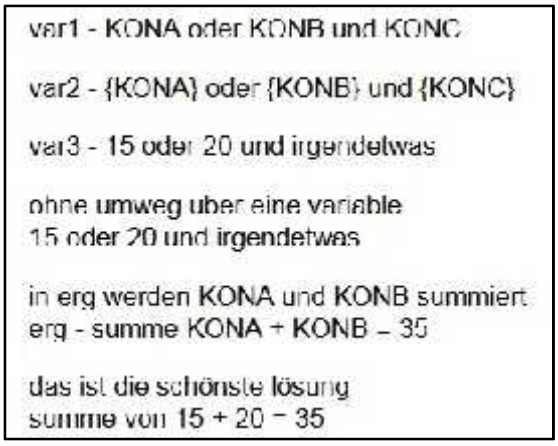
define ('name', *numerischer-wert* | *zeichenkette*);

beispiel

```
<?php
define ("KONA", 15);
define ('KONB', 20);
define ("KONC", " irgendetwas ");

$var1 = "KONA oder KONB und KONC";
echo "<p>var1 - $var1</p>";
$var2 = "{KONA} oder {KONB} und {KONC}";
echo "<p>var2 - $var2</p>";
$var3 = KONA . " oder " . KONB . " und " . KONC;
echo "<p>var3 - $var3</p>";
echo "<p>ohne umweg über eine variable<br />"
     . KONA . " oder " . KONB . " und " . KONC . "</p>";
$erg = KONA + KONB;
echo "<p>in erg werden KONA und KONB summiert<br />"
     . "erg - summe KONA + KONB = $erg</p>";
echo "<p>das ist die schönste lösung<br />"
     . "summe von " . KONA . " + " . KONB . " = "
     . (KONA + KONB) . "</p>";
?>
```

hinweise

var1	Der variablen wird eine zeichenkette zugewiesen und weil die konstanten innerhalb der anführungszeichen stehen, erscheint nicht ihr inhalt sondern ihr name in der zeichenkette.
var2	Der gleiche fall wie zuvor, die geschweiften klammern nützen hier nichts.
var3	Wenn man die zeichenkette so zusammenbastelt, erscheint tatsächllich der inhalt der konstanten in der zeichenkette
ohne umweg	Nach dem gleichen muster wie zuvor wird hier die zeichenkette direkt in der echo-anweisung gebastelt.
erg	Der numerische ausdruck **KONA + KONB** wird der variablen zugewiesen; in der zeichenkette erscheinen dann wieder nur die namen der konstanten, weil sie mal wieder in anführungs-zeichen stehen.
schönste	hier wird der wert des ausdrucks **KONA + KONB** direkt in die zeichenkette gebastelt, deshalb wird er in runde klammern eingeschlossen. Die konstanten erscheinen mit ihrem wert in der zeichenkette.



3. programiertechniken - teil 1

3.1 zuweisen und rechnen

Für rechenoperationen und das zuweisen von werten an variable stehen entsprechende operatoren zur verfügung, mit denen PHP-elemente miteinander verknüpft werden. Der einfachste zuweisungsoperator, das zeichen = wurde bereits vorgestellt, aber es gibt noch weitere operatoren, die zu dieser gattung gehören.

3.1.1 zuweisungs- und arithmetische operatoren

anweisung	***zugewiesen wird***
`$a = $b;`	$b
`$erg = $a + $b;`	$a + $b
`$erg = $a - $b;`	$a - $b
`$erg = $a * $b;`	$a mal $b
`$erg = $a / $b;`	$a dividiert durch $b
`$erg = $a % $b;`	modulo, d.h. rest von $a dividiert durch $b
`$erg = $a ** $b;`	$a hoch $b
`$erg = $a++;`	$a + 1 (increment)
`$erg = $a--;`	$a – 1 (decrement)
`$a += $b;`	$a + $b
`$a -= $b;`	$a - $b
`$a *= $b;`	$a mal $b
`$a /= $b;`	$a dividiert durch $b
`$a %= $b;`	modulo, d.h. rest von $a dividiert durch $b
`$a++;`	$a + 1 (increment)
`$a--;`	$a – 1 (decrement)

3.1.2 bitweise operatoren

Zu dieser gruppe von operatoren gehören noch die sog. bit-operatoren für leicht merkwürdige basteleien:

`$a & $b;`	in $a werden alle bits auf 1 gesetzt, die in $a **und** $b auf 1 gesetzt sind
`$a ^ $b;`	in $a werden alle bits auf 1 gesetzt, die in $a **oder** $b aber **nicht in beiden** auf 1 gesetzt sind
`$a | $b;`	in $a werden alle bits auf 1 gesetzt, die in $a **oder** $b auf 1 gesetzt sind
`~ $a;`	in $a werden alle bits umgekehrt, d.h 0 wird 1 und 1 wird 0
`$a << $b;`	in $a werden alle bits um $b stellen nach links geschoben (entspricht je stelle $a * 2)
`$a >> $b;`	in $a werden alle bits um $b stellen nach rechts geschoben (entspricht je stelle $a / 2)

3.1.3 mathematische funktionen

`$erg = sqrt($a);`	quadratwurzel aus $a
`$erg = floor($a);`	rundet eine dezimalzahl auf die nächste ganzzahl ab 3.7 ergibt 3 und -3.2 ergibt -4
`$erg = ceil/$a);`	rundet eine dezimalzahl zur nächsten ganzzahl auf 3.2 ergibt 4 und -3.6 ergibt -3
`$erg = round($a);`	rundet eine dezimalzahl kaufmännisch 3.4 ergibt 3 und 3.5 ergibt 4 -3.4 ergibt -3 und -3.5 ergibt -4
`$erg = intdiv($w1, $w2);`	ganzzahlige division, d.h. im ergebnis werden nachkommastellen abgeschnitten. Nachkommastellen in $w1 und $w2 werden vor der division abgeschnitten.
`$erg = max($w1, $w2, . . .;`	höchsten wert ermitteln
`$erg = min($w1, $w2, . . .;`	niedrigsten wert ermitteln

3.2 bedingungen

Bedingungen werden mit operatoren formuliert, es gibt vergleichs-operatoren und logische operatoren.

3.2.1 vergleichsoperatoren

$a **==** $b	$a gleich $b
$a **!=** $b	$a ungleich $b
$a **<>** $b	$a ungleich $b
$a **<** $b	$a kleiner $b
$a **>** $b	$a größer $b
$a **<=** $b	$a kleiner oder gleich $b
$a **>=** $b	$a größer oder gleich $b

3.2.2 logische operatoren

Mit logischen operatoren, werden logische werte zu einer bedingung verknüpft.

$a **and** $b	wenn $a gleich true **und** $b gleich true
$a **&&** $b	wenn $a gleich true **und** $b gleich true
$a **or** $b	wenn $a gleich true **ode**r $b gleich true
$a **||** $b	wenn $a gleich true **ode**r $b gleich true
$a **xor** $b	wenn $a gleich true **oder** $b gleich true **aber nicht** beide
! $a	wenn $a nicht gleich true, d.h. false

Mit logischen operatoren können bedingungen miteinander verknüpft werden (vgl. ablaufsteuerung 3.3).

3.2.3 ternärer operator ?

```
$erg = ($wert < 100 ? "zu wenig" : "ausreichend");
```

Wenn der inhalt von $wert kleiner 100 ist, wird an $erg zugewiesen, was zwischen fragezeichen und doppelpunkt steht, andernfalls das, was dem doppelpunkt folgt.

3.2.4 spaceship-operator <=>

```
$erg = ($w1 <=> $w2);
```
an $erg wird zugewiesen

-1	wenn $w1 kleiner $w2
0	wenn $w1 gleich $w2
1	wenn $w1 größer $w2

3.3 ablaufsteuerung - verzweigung, schleifen

3.3.1 if-anweisung - alternative

if (*bedingung*)
{ *anweisung*;
...
}
	`if ($a == $b)`
	`{ echo "<p>A ist gleich B</p>";`
	
	`}`

elseif (*bedingung*)
{ *anweisung*;
...
}
	`elseif ($a > $b)`
	`{ echo "<p>A ist größer B</p>";`
	
	`}`

else
{ *anweisung*;
...
}
	`else`
	`{ echo "<p>A ist kleiner B</p>";`
	
	`}`

hinweise

Die elseif- und die else-anweisung dürfen fehlen. Mit der if-, elseif- oder else-anweisung wird jeweils ein block von anweisugnen begonnen, der in geschweifte klammern eingeschlossen wird. Enthält ein block nur eine anweisung dürfen die klammern entfallen.

Bedingungen können durch logische operatoren verknüpft werden, auf die klammerung der bedingungen muss man dabei sehr genau achten.

```
if (($a == $b) || ($c <= $d))
if (($x > $y) && $logvar))
if ((($a == $b) || ($c <= $d)) && (($x > $y) && $logvar))
```

3.3.2 while-anweisung - abweisende schleife

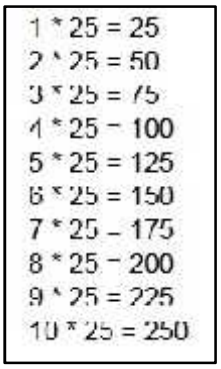
while (*bedingung*)
{ *anweisung*;
  ...
}

hinweis

Der **while**-anweisung folgt ein block von anweisungen, die durchlaufen werden, solange die *bedingung* erfüllt ist. Es kann sein, dass die schleife nicht durchlaufen wird, wenn die *bedingung* von anfang an nicht erfüllt ist. Wenn der block nur eine anweisung enthält, können die geschweiften klammern entfallen.

beispiel

```
<?php
$fakt = 0;
$op = " * 25 = ";
echo "<p><b>abweisende schleife</b><br />";
while ($fakt < 10)
{   $fakt++;
    $erg = $fakt * 25;
    echo "$fakt $op $erg<br />" ;
}
echo "</p>";
?>
```



3.3.3 do-while-anweisung - nicht abweisende schleife

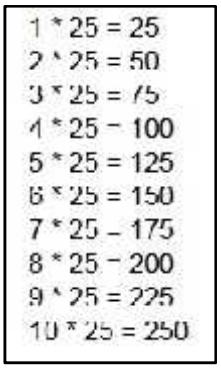
do
{ *anweisung*;
  ...
}
while (*bedingung*); **hier muss ein strichpunkt stehen**

hinweis

Der **do**-anweisung folgt ein block von anweisungen, die durchlaufen werden; wenn dann die *bedingung* der **while**-anweisung erfüllt ist, wird die schleife erneut durchlaufen, d.h. die schleife wird wenigstens einmal durchlaufen. Wenn der block nur eine anweisung enthält, können die geschweiften klammern entfallen.

beispiel

```
<?php
$fakt = 0;
$op = " * 25 = ";
echo "<p><b>nicht abweisende schleife</b><br />";
do
{   $fakt++;
    $erg = $fakt * 25;
    echo "$fakt $op $erg<br />" ;
}
while ($fakt < 10);
echo "</p>";
?>
```



3.3.4 for-anweisung - zählschleife

for (*von*; *bis*; *increm*)
{ *anweisung*;
. . .
}

von	anweisung, mit der einer sog. laufvariablen vor dem ersten durchlauf ein numericher wert als anfangswert zugewiesen wird.
bis	bedingung, mit der die laufvariable vor jedem schleifendurchlauf auf einen endewert geprüft wird. Ist der endwert überschritten, wird die schleife nicht mehr durchlaufen
increm	numerischer wert, der am ende eines schleifendurchlaufs auf die laufvariable addiert wird.

hinweise

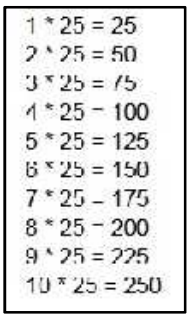
Der **for**-anweisung folgt ein block von anweisungen, die ggf. durchlaufen werden. Wenn der block nur eine anweisung enthält, können die geschweiften klammern entfallen. Anfangswert und endewert können auch als variable angegeben werden.Eine zählschleife kann auch rückwärts laufen, d.h. der endewert ist niedriger als der anfangswert, das increment muss dann aber negativ sein.

Eine zählschleife kann auch als unbedingte schleife definiert sein, d.h. *von*, *bis* und *increm* sind nicht angegeben. Dann muß es aber in der schleife eine abbruchmöglichkeit geben (vgl. break 3.4.3).

for (;;)

beispiel

```
<?php
echo "<p><b>zählschleife</b><br />";
$op = " * 25 = ";
for ($fakt=1; $fakt<=10; $fakt++)
{   $erg = $fakt * 25;
    echo "$fakt $op $erg<br />" ;
}
echo "</p>";
?>
```



3.4 ablaufsteuerung - schleifen schachteln, unterbrechen, abbrechen

3.4.1 schleifen schachteln

schleifen können beliebig geschachtelt werden.

beispiel
In eine zählschleife ist eine abweisende schleife geschachtelt.

```
<table class="font10" style="width: 600px">
<tr>
<?php
echo "<p><b>geschachtelte schleifen</b></p>";
for ($lauf=13; $lauf<20; $lauf+=2)
{   $fakt1 = 0;
    $fakt2 = $lauf;
    $op = " * " . $fakt2 . " = ";
    echo "<td>einmaleins mit $fakt2<br />";
    while ($fakt1 < 10)
    {   $fakt1++;
        $erg = $fakt1 * $fakt2;
        echo "$fakt1 $op $erg<br />" ;
    }
    echo "</td>";
}
?>
</tr>
</table>
```

einmaleins mit 13	einmaleins mit 15	einmaleins mit 17	einmaleins mit 19
1 * 13 = 13	1 * 15 = 15	1 * 17 = 17	1 * 19 = 19
2 * 13 = 26	2 * 15 = 30	2 * 17 = 34	2 * 19 = 38
3 * 13 = 39	3 * 15 = 45	3 * 17 = 51	3 * 19 = 57
4 * 13 = 52	4 * 15 = 60	4 * 17 = 68	4 * 19 = 76
5 * 13 = 65	5 * 15 = 75	5 * 17 = 85	5 * 19 = 95
6 * 13 = 78	6 * 15 = 90	6 * 17 = 102	6 * 19 = 114
7 * 13 = 91	7 * 15 = 105	7 * 17 = 119	7 * 19 = 133
8 * 13 = 104	8 * 15 = 120	8 * 17 = 136	8 * 19 = 152
9 * 13 = 117	9 * 15 = 135	9 * 17 = 153	9 * 19 = 171
10 * 13 = 130	10 * 15 = 150	10 * 17 = 170	10 * 19 = 190

3.4.2 schleife unterbrechen

Mit der anweisung **continue** kann eine schleife unterbrochen werden, d.h. wenn die anweisungen **continue** erreicht wird, werden nachfolgende anweisungen übersprungen und der nächste schleifendurchlauf wird begonnen.

beispiel

Fast das gleiche beispiel wie zuvor, aber das einmaleins mit 15 wird nicht angezeigt.

```
<table class="font10" style="width: 600px">
<tr>
<?php
echo "<p><b>schleife mit unterbrechung</b></p>";
for ($lauf=13; $lauf<20; $lauf+=2)
{   $fakt1 = 0;
    $fakt2 = $lauf;
    $op = " * " . $fakt2 . " = ";
    echo "<td>einmaleins mit $fakt2<br />";
    if ($fakt2 == 15)
    {   echo "nein, das nicht</p>";
        continue;
    }
    while ($fakt1 < 10)
    {   $fakt1++;
        $erg = $fakt1 * $fakt2;
        echo "$fakt1 $op $erg<br />" ;
    }
    echo "</td>";
}
?>
</tr>
</table>
```

einmaleins mit 13
1 * 13 = 13
2 * 13 = 26
3 * 13 = 39
4 * 13 = 52
5 * 13 = 65
6 * 13 = 78
7 * 13 = 91
8 * 13 = 104
9 * 13 = 117
10 * 13 = 130

einmaleins mit 15
nein, das nicht

einmaleins mit 17
1 * 17 = 17
2 * 17 = 34
3 * 17 = 51
4 * 17 = 68
5 * 17 = 85
6 * 17 = 102
7 * 17 = 119
8 * 17 = 136
9 * 17 = 153
10 * 17 = 170

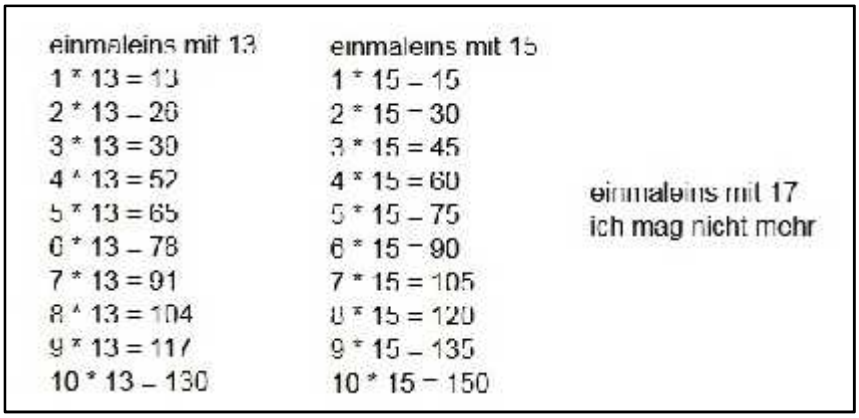
einmaleins mit 19
1 * 19 = 19
2 * 19 = 38
3 * 19 = 57
4 * 19 = 76
5 * 19 = 95
6 * 19 = 114
7 * 19 = 133
8 * 19 = 152
9 * 19 = 171
10 * 19 = 190

3.4.3 schleife abbrechen

Mit der anweisung **break** kann eine schleife abgebrochen werden.

beispiel
Nochmal das beispiel mit den geschachtelten schleifen, aber jetzt wird nach dem einmaleins mit 15 abgebrochen

```
<table class="font10" style="width: 600px">
<tr>
<?php
echo "<p><b>schleife mit abbruch</b></p>";
for ($lauf=13; $lauf<20; $lauf+=2)
{   $fakt1 = 0;
    $fakt2 = $lauf;
    $op = " * " . $fakt2 . " = ";
    echo "<td>einmaleins mit $fakt2<br />";
    if ($fakt2 > 15)
    {   echo "ich mag nicht mehr<br />";
        break;
    }
    while ($fakt1 < 10)
    {   $fakt1++;
        $erg = $fakt1 * $fakt2;
        echo "$fakt1 $op $erg<br />" ;
    }
    echo "</td>";
}
?>
</tr>
</table>
```



hinweis

Wenn schleifen mehrfach geschachtelt sind, gibt es die möglichkeit aus einer tief verschachtelten schleife gleich mehrere schleifen abzubrechen.

break *anzahl*;

anzahl ist ein numerischer wert und gibt an, wieviel schleifen abgebrochen werden sollen.

beispiel

Die innerste schleife würde genau 1000 mal durchlaufen, wenn nicht vorzeitig abgebrochen würde.

```
<table class="font10" style="width: 600px">
<tr>
<?php
$wert = 0;
echo "<td>start mit $wert<br/>";
for ($x=1; $x<=10; $x++)
{   for ($y=1; $y<=10; $y++)
    {   for($z=1; $z<=10; $z++)
        {   $wert++;
            echo "$wert    ";
            if ($wert > 32)
            {   echo "schluss mit lustig<br />";
                break 3;
            }
        }
        echo "<br />";
    }
}
echo "</td>";
?>
</tr>
</table>
```

start mit 0
1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31 32 33 schluss mit lustig

3.5 ablaufsteuerung - mehrfachverzweigung

Abhängig vom inhalt einer variablen werden unterschiedliche folgen von anweisungen durchlaufen.

```
switch ($var)
{  case wert-1:                 wenn $var den wert-1 hat werden die anschließenden
      anweisungen-1;            anweisungen bis break ausgeführt
      . . .
      break;
   case wert-2:                 wenn $var den wert-2 hat werden die anschließenden
      anweisungen-2;            anweisungen bis break ausgeführt
      . . .
      break;
   default:                     wenn keine der vorstehenden fälle eingetreten ist
      anweisung;                werden die anschließenden anweisungen ausgeführt
}
```

hinweis

Die werte in den **case**-anweisungen können direkt oder in variablen angegeben werden.

beispiel

```
<?php
for ($x=0; $x<=5; $x++)
{   switch($x)
    {   case 0:
           echo "<p>erster durchlauf</p>"
           break;
        case 1:
           echo "<p>zweiter durchlauf</p>"
           break;
        case 2:
           echo "<p>dritter durchlauf</p>"
           break;
        default:
           echo "<p>durchlauf mit $x</p>"
    }
}
?>
```

erster durchlauf

zweiter durchlauf

dritter durchlauf

durchlauf mit 3

durchlauf mit 4

durchlauf mit 5

Folgen einer **case**-anweisung keine anweisung, gelten für diesen fall die anweisungen der nächsten **case**-anweisung:

```
<?php
for ($x=0; $x<=5; $x++)
{   switch($x)
    {   case 0:
        case 1:
           echo "<p>erster oder zweiter durchlauf</p>"
           break;
        case 2:
           echo "<p>dritter durchlauf</p>"
           break;
        default:
           echo "<p>durchlauf mit $x</p>"
    }
}
?>
```

erster oder zweiter durchlauf

erster oder zweiter durchlauf

dritter durchlauf

durchlauf mit 3

durchlauf mit 4

durchlauf mit 5

4. programmiertechniken – teil 2

4.1 nützliche funktionen

4.1.1 funktionsaufruf

Die sprache PHP enthält viele vorgefertigte funktionen; eine funktion ist ein programmteil, der an jeder beliebigen stelle aufgerufen, d.h.ausgeführt werden kann. Der aufruf einer funktion hat folgendes format:

ergebnis = *funktion* ([*argumente*])

funktion	name der funktion
ergebnis	ein numerischer oder logischer wert oder eine zeichenkette; eine funktion hat fast immer ein ergebnis (sonst macht das keinen sinn)
argumente	ein oder mehrere werte (argumente), die der funktion übergeben werden und mit denen die funktion das ergebnis ermittelt. Mehrere argumente werden durch komma getrennt. Nicht jede funktion benötigt argumente.

Das ergebnis einer funktion muss keinesweg immer einer variablen zugewiesen werden, der aufruf einer funktion kann vielmehr auch in einer anweisung genau an der stelle stehen, wo dieser wert benötigt wird.

beispiele
In den folgenden beispielen liefert die funktion **strlen** die länge einer zeichenkette. Die seltsamen entwerteten anführungszeichen werden noch erklärt,

```
<?php
$text = "Karl Valentin";
$len  = strlen($text);
echo "<p>der string \"$text\" enthält $len zeichen</p>";
echo "<p>der string \"$text\" enthält " . strlen($text) . " zeichen</p>";
?>
```

der string "Karl Valentin" enthält 13 zeichen

der string "Karl Valentin" enthält 13 zeichen

Es ist nicht sinnvoll, alle vorgefertigten funktionen an einer stelle zusammenzufassen, vielmehr werden immer wieder funktionen bechrieben, die zum gerade behandelten thema passen. Nachfolgend werden nun einige funktionen beschrieben, die sehr nützlich sind, aber an anderer stelle nicht unterzubringen sind.

4.1.2 werte prüfen

Die folgenden funktionen liefern einen logischen wert (true oder false) und werden meist nach diesem muster aufgerufen:

if (*funktion*(*parameter)*)

isset($var)	prüft ob $var vorhanden ist
isset($kette[5])	prüft ob position 5 von $kette belegt ist.
empty($var)	prüft ob $var vorhanden und nicht leer ist
is_int($var)	prüft ob $var eine ganzzahl enthält
is_float($var)	prüft ob $var eine dezimalzahl enthält
is_string($var)	prüft ob $var eine zeichenkette enthält
is_bool($var)	prüft ob $var eine logische variable ist
is_array($var)	prüft ob $var ein feld ist
is_null($var)	prüft ob $var null enthält

4.1.3 zeichen codieren, verschlüsseln

`$erg = chr(65);`	die funktion **chr** wandelt eine ganzzahl in das zeichen-äquivalent des ASCII-codes um; $erg enthält **65** also den buchstaben **A**
`$y = ord("A");`	die funktion **ord** wandelt ein zeichen in den entsprechenden ganzzahl-wert um; $y enthält 65
`$erg = crc32($kette);`	liefert eine prüfsumme; damit kann man beispielsweise erkennen, dass sich zwei gleich scheinende texte dadurch unterscheiden, dass unsichtbare steuerzeichen "hineingeschmuggelt" wurden .
`$erg = crypt($kette, "xx");`	verschlüsselt
`$erg = md5($kette);`	verschlüsselt
`$erg = str_rot13($kette);`	verschlüsselt

beispiele

```
<?php
$erg = chr(65) . $ende;
echo "<p>chr erzeugt aus 65: $erg<br />" . $ende;
$erg = ord("A") . $ende;
echo "<p>ord erzeugt aus A: $erg</p>" . $ende;
$kennwort = "ABCabc123$#*" . $ende;
echo "<p>das kennwort $kennwort<br />"
     . "wird verschlüsselt mit</p>" . $ende;
$erg = crypt($kennwort, "xyz") . $ende;
echo "<p>crypt mit xyz: $erg</p>". $ende;
$erg = crypt($kennwort, "abc") . $ende;
echo "<p>crypt mit abc: $erg</p>" . $ende;
$erg = md5($kennwort) . $ende;
echo "<p>md5: $erg</p>" . $ende;
$erg = str_rot13($kennwort) . $ende;
echo "<p>str_rot13: $erg</p>" . $ende;
$text1 = "das ist ein harmloser test";
$text2 = "das ist ein" . chr(10) . "harmloser test";
$erg1 = crc32($text1);
$erg2 = crc32($text2);
echo "<p>$text1 - $erg1<br />$text2 - $erg2</p>" . $ende;
?>
```

chr erzeugt aus 65: A

ord erzeugt aus A: 65

das kennwort ABCabc123$#*
wird verschlüsselt mit

crypt mit xyz: xy7dIeDRadRkI

crypt mit abc: abUldRxXV3Go

md5: 10bc15aeb9b03ee5fb412e10bcc37da8

str_rot13: 10be15aeb9b03ee5fb412e10bcc37da8

das ist ein harmloser test - -115506910
das ist ein harmloser test - 112715258

4.1.4 zufallszahlen

Die beiden funktionen liefern eine zufallszahl, aus einem angegebenen bereich.

erg = **rand**(*gz1, gz2*);

erg = **mt_rand**(*gz1, gz2*);

mit ***gz1*** und ***gz2*** wird der ganzzahlige bereich definiert, aus dem eine zufällige zahl erzeugt und in ***erg*** gespeichert wird.

beispiel
Mit der funktion rand wird 1.000.000 mal gewürfelt, d.h. es wird eine zufallszahl zwischen 1 und 6 erzeugt und zuletzt als ergebnis angezeigt, wie oft die werte 1,2 usw. gewürfelt wurden. .

```
<?php
$z    = 1000000;
$sum  = 0;
$feld = array(0,0,0,0,0,0,0);
echo "<p>$z zufallszahlen zwischen 1 und 6</p>" . $ende;
for ($x=0; $x<$z; $x++)
{   $erg = rand(1, 6);
    if (($erg > 0) && ($erg <= 6))
        $feld[$erg] += 1;
    else
        $feld[0] += 1;
}
for ($x=1; $x<=6; $x++)
{   echo "<p>wert $x - $feld[$x]</p>" . $ende;
    $sum += $feld[$x];
}
echo "<p>brauchbar waren $sum zahlen</p>" . $ende;
if ($feld[0] > 0)
   echo "<p>unbrauchbar waren $feld[0] zahlen</p>" . $ende;
?>
```

1000000 zufallszahlen zwischen 1 und 6

wert 1 - 167102

wert 2 - 167095

wert 3 - 166201

wert 4 - 166706

wert 5 - 166824

wert 6 - 166072

brauchbar waren 1000000 zahlen

4.1.5 PHP-script abbrechen

Beim aufbau einer seite mit **PHP** kann der fall eintreten, dass auf grund irgend eines mangels die seite nicht fehlerfrei aufgebaut werden kann. Dann ist es sinnvoll, das PHP-script abzubrechen. Die seite wird aber trotzdem zum browser übertragen. Daher ist es notwendig, in die seite vor dem abbruch des scripts dem benutzer eine möglichkeit zu geben, die angezeigte seite zu verlassen.

exit (["*fehlermeldung*"]);

die (["*fehlermeldung*"]);

Beide funktionen beenden das laufende PHP-script; ggf. wird die fehlermeldung ausgegeben. In dem folgenden beispiel erwartet die aufgerufene seite die übergabe eines sog. aufruf-parameters (vgl. 6.1 daten von seite zu seite). Wenn dieser parameter fehlt, wird das PHP-script, das die seite aufbaut, abgebrochen.

beispiel

```
<html>
<head>
<title>testbruch.php - script abbrechen</title>
</head>
<body>
<p>diese seite zeigt nur, wie man ein <b>PHP-script</b><br />
   unter einer bestimmten bedingung abbricht<br />
   hier wird beispielsweise beim aufruf der seite<br />
   die übergabe des parameters <b>param</b> erwartet</p>
<?php
   $ende  = chr(13) . chr(10);
   if (!isset($_GET["param"]))
   {   echo "<p><a href='Javascript: history.back();'>"
            . "abbruch </a></p>" . $ende;
       exit ("param fehlt");
   }
   else
   {   $wert = $_GET["param"];
       echo "<p>beim aufruf wurde geliefert:</p>" . $ende;
       echo "<p><b>$wert</b></p>" . $ende;
       echo "<p>die restliche seite könnte jetzt aufgebaut "
            . "werden</p>" . $ende;
       echo "<p><a href='Javascript: history.back();'>"
            . "zurück </a></p>" . $ende;
   }
?>
</body>
</html>
```

aufruf: `<a href="doku/PHP/testbruch.php?param=HALLO">aufruf</a>`

aufruf: `<a href="doku/PHP/testbruch.php">aufruf</a>`

Der erste aufruf führt dazu, dass die seite aufgebaut und ausgeführt wird, beim zweiten aufruf wird der aufbau der seite abgebrochen, die seite kommt zwar ebenfalls zur anzeige, enthält aber eine fehlermeldung und die möglichkeit die seite abzubrechen.

4.2 anführungszeichen und apostrophe

In einer **echo**-anweisung, die **HTML**-anweisungen erzeugt, müssen diese in anführungszeichen oder apostrophe eingeschlossen sein. Diese **HTML**-anweisungen können **PHP**-elemente (variableln, konstanten) enthalten und dabei gibt es probleme, denn wenn ein PHP-element in apostrophe eingeschlossen ist, wird von der echo-anweisung nicht der inhalt, sondern der name des elements in die seite geschrieben. Kritisch ist auch, wenn die echo-anweisung in einem anzuzeigenden text ein anführungszeichen oder einen apostrophen erzeugen soll oder wenn eine **HTML**-anweisung mit attributen erzeugt wird (attribute stehen ja in anführungszeichen oder apostrophen). Klar ist: anführungszeichen in anführungszeichen oder apostroph in apostrophen geht gar nicht, apostroph in anführungszeichen oder anführungszeichen in apostrophen geht immer. Wenn das nicht passt, muss man mit entwerteten oder maskierten zeichen arbeiten. Nachstehend die regeln, die man beachten muss:

regeln

- In **HTML**-anweisungen, die in anführungszeichen eingeschlossen sind, wird von **PHP**-elementen, der inhalt ausgegeben, d.h. in die seite geschrieben (beispiel 1)
- In **HTML**-anweisungen, die in apostrophe eingeschlossen sind, wird von **PHP**-elementen, der name ausgegeben, d.h. in die seite geschrieben (beispiel 2)
- Anführungszeichen dürfen nicht in anführungszeichen und apostrophe nicht in apostrophe eingeschlossen werden (kein beispiel, weil das zu einem fehler führen würde).
- Anführungszeichen dürfen in apostrophe und apostrophe in anführungszeichen eingeschlossen werden (beispiel 3 und 4)
- Entwertete anführungszeichen dürfen in anführungszeichen und entwertete apostrophe in apostrophe eingeschlossen werden (beispiel 5 und 8).
- Entwertete anführungszeichen in apostrophen oder entwertete apostrophe in anführungszeichen liefern ein untaugliches ergebnis, d.h. sie werden als entwertetes zeichen dargestellt (also wenn man das mal braucht, ist das die lösung). Beispiel 6 und 7.
- Maskierte anführungszeichen und maskerte apostrophe können in anführungszeichen oder apostrophe eingeschlossen werden (beispiel 9-12),
- Ein zeichen wird durch einen vorangestellten backslash \\ entwertet.
- Die maskierung von zeichen wird in der HTML-doku unter sonderzeichen behandelt.

beispiele

Die folgenden beispiele verdeutlichen diese regeln.

```
<?php
$name  = "0tto";
$alter = 15;
echo "<p>anf = anführungszeichen<br />"
      . "apo = apostroph</p>";
echo "<p> 1 - in anf: $name ist $alter jahre alt<br />";
echo ' 2 - in apo: $name ist $alter jahre alt</p>';
echo "<p> 3 - apo ' in anf ist ok<br />";
echo ' 4 - anf " in apo ist ok</p>';
echo "<p>entwertete anf.zeichen oder apostrophe<br />";
echo " 5 - entw. anf \" in anf ist ok<br />";
echo " 6 - entw. apo \' in anf ist mist<br />";
echo ' 7 - entw. anf \" in apo ist mist<br />';
echo ' 8 - entw. apo \' in apo ist ok</p>';
echo "<p>maskierte anf.zeichen oder apostr. ok<br />";
echo " 9 - mask. anf &quot; in anf<br />";
echo "10 - mask. apo &apos; in anf<br />";
echo '11 - mask. anf &quot; in apo<br />';
echo '12 - mask. apo &apos; in apo</p>';
?>
```

4.3 echo-anweisung optimieren

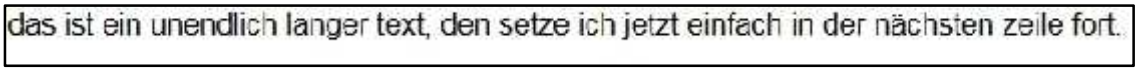
Eine **echo**-anweisung, mit der **HTML**-code erzeugt wird, kann sehr lang werden und es ist sinnvoll, sie dann auf eine oder mehrere zeilen umzubrechen. Ohne PHP ist das bei **HTML**-anweisungen recht einfach:

```
<p>das ist ein unendlich langer text,
   den setze ich jetzt einfach in der nächsten
   zeile fort. </p>
```

Den browser stört das überhaupt nicht, er kommt damit klar, aber bei **PHP** geht das nicht so einfach, da muss man auf die möglichkeit zurückgreifen, zeichenketten zusammen zu setzen.

```
echo "<p>das ist ein unendlich langer text, "
     . "den setze ich jetzt einfach in der nächsten "
     . "zeile fort. </p>";
```

Der browser liefert in beiden fällen das gleiche ergebnis:



So wird der code einer seite schon viel übersichtlicher. Jetzt bleibt noch das problem, dass die echo-anweisung alles hintereinander in die seite schreibt. Zur erinnerung nochmal ganz an den anfang (1.4), da gab es folgendes beispiel:

```
<p>jetzt folgen PHP-anweisungen</p>
<?php
   echo "das ist das erste beispiel";
   echo 1234;
   echo "<p>abschluß</p>";
?>
<p>ende der vorstellung</p>
```

In der seite, so wie sie nach ausführung der **PHP**-anweisungen beim browser landet, sieht das so aus:

```
<p>jetzt folgen PHP-anweisungen</p>
das ist das erste beispiel1234<p>abschluß</p><p>ende der vorstellung</p>
```

Das stört eigentlich nicht, weil auch damit der browser klar kommt und ein vernünftiges ergebnis ausgibt. Aber bei der entwicklung einer seite ist man doch manchmal gezwungen, sich genau anzuschauen, was beim browser angekommen ist. Wie das geht ?

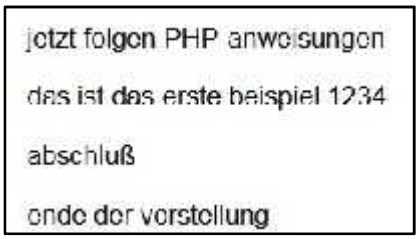
Einfach die angezeigte seite mit der rechten maustaste anklicken und **seitenquelltext anzeigen** wählen, dann wird der code angezeigt, wie er tatsächlich in der seite steht. Da wäre dann ein wenig zeilenumbruch schon angenehm. Den erreicht man mit **X'0D0A'** oder dezimal **13 10** an der stelle wo umgebrochen werden soll. Also erzeugt man am besten eine variable mit diesem inhalt und stellt sie dorthin, wo sie gebraucht wird. Das obige code-beispiel sieht dann so aus:

```
<p>jetzt folgen PHP-anweisungen</p>
<?php
   $ZE = chr(13) . chr(10);
   echo "das ist das erste beispiel" . $ZE;
   echo 1234 $ZE;
   echo "<p>abschluß</p>" . $ZE;
?>
<p>ende der vorstellung</p>
```

und das liefert dann PHP in der seite ab:

```
<p>jetzt folgen PHP-anweisungen</p>
das ist das erste beispiel
1234
<p>abschluß</p>
<p>ende der vorstellung</p>
```

und das ergebnis, das der browser abliefert entspricht hoffentlich den erwartungen.



4.4 seiten dynamisch gestalten

Wie schon an anderer stelle ausgeführt, dient PHP dazu, seiten dynamisch zu gestalten, d.h. an bedingungen und aktuelle gegebenheiten anzupassen. Dazu gibt es besonders zwei möglichkeiten.

4.4.1 PHP-datei einbinden

Man kann an jeder beliebigen stelle innerhalb eines **PHP-abschnitts** eine **PHP-datei** in eine seite einbinden. Die datei kann **HTML**-code und **PHP-abschnitte** enthalten, d.h. sie enthält den teil einer seite, der nur einmal geschrieben wird und dann entweder an mehreren stellen in eine seite oder auch in verschiedene seiten eingebaut werden kann. Der dateiname muss die erweiterung **.php** haben.

include | **require** "*pfadname*";

include_once | require_once "*pfadname*";

Die funktionen include und require binden eine datei ein, include_once und require_once binden nur ein, wenn die datei nicht schon eingebunden ist. Die anweisungen unterscheiden sich nur bei einer etwaigen fehlermeldung.

4.4.2 HTML und PHP mischen - HereDoc-syntax

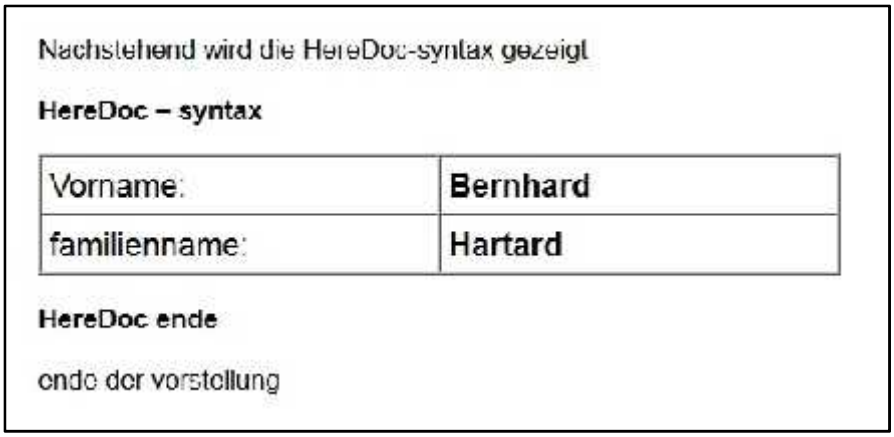
In einer seite können **HTML**-code und **PHP-abschnitte** gemischt werden, allerdings können PHP-elemente (variable, konstanten) nicht in den HTML-anweisungen verwendet werden. Werden aber HTML-anweisungen in einem PHP-abschnitt erzeugt (echo-anweisung) dürfen sie sehr wohl PHP-elemente enthalten, das ist ja gerade die aufgabe von PHP.

Mit der sog. **HereDoc-syntax** ist es aber doch möglich, in reinen HTML-anweisungen PHP-elemente zu verwenden. Im folgenden beispiel steht ein PHP-abschnitt zwischen zwei HTML-anweisungen. In dem abschnitt werden zunächst zwei variable erzeugt, die dann in HTML-anweisungen verwendet werden. Diese anweisungen sind aber in **<<<DOC . . . DOC;** eingeschlossen und das ganze wird mit der echo-anweisung ausgegeben. Der strichpunkt nach DOC ist unbedingt erforderlich.

achtung
In HTML-anweisungen innerhalb der HereDoc-syntax dürfen keine PHP-abschnitte eingeschoben werden.

beispiel

```
<p>Nachstehend wird die HereDoc-syntax gezeigt</p>
<?php
echo "<p><b>HereDoc – syntax</b></p>";
$vorname = "Bernhard";
$name = "Hartard";
echo <<<DOC
<table style="width: 400ox;" class="tbstd">
  <tr>
    <td width="50%">Vorname:</td>
    <td><b>$vorname</b></td>
  </tr>
  <tr>
    <td>familienname:</td>
    <td><b>$name</b></font></td>
  </tr>
</table>
DOC;
echo "<p><b>HereDoc ende</b></p>";
?>
<p>ende der vorstellung</p>
```



variante

Bei der variante der anwendung wird alles, was in Here-Doc-syntax geschrieben wird einer variablen zugewiesen und der inhalt der variablen dann mit der echo-anweisung in die seite geschrieben. Das ergebnis gleicht dem vorhergehenden beispiel.

```
<p>Nachstehend wird die HereDoc-syntax gezeigt</p>
<?php
echo "<p><b>HereDoc – syntax</b></p>";
$vorname = "Bernhard";
$name = "Hartard";
$ausgabe = <<<DOC
<table border="1" cellspacing="0" width="400" cellpadding="4">
. . .
</table>
DOC;
echo $ausgabe;
echo "<p><b>HereDoc ende</b></p>";
?>
<p>ende der vorstellung</p>
```

4.5 eigene funktionen

4.5.1 konstruktion

Werden PHP-abschnitte in einer seite mehrmals benötigt, ist es sinnvoll, diesen abschnitt nicht mehrmals in die seite zu schreiben, sondern als funktion zu definieren. Funktionsdefinitionen stellt man am besten an den anfang oder das ende der seite, man kann das aber auch an jeder beliebigen stelle tun, das ist dann aber nicht sehr übersichtlich. Meist liefert eine funktion ein ergebnis, ausnahmsweise tut sie das auch nicht. Häufig werden einer funktion beim aufruf werte übergeben, die sie für die ermittlung des ergebnisses benötigt. Die konstruktion ist sehr einfach:

```
<?php
function name ( [ par, par=wert, &par ])
{       variable = wert;
        global variable [, variable . . . ];
        anweisungen, mit denen das ergebnis ermittelt wird
        [ return [ ergebnis ] ] ;
}
?>
```

function	schlüsselwort, mit dem ein PHP-abschnitt beginnt, der eine funktion definiert
name	die funktion erhält wie eine variable einen namen, nur ohne das $-zeichen am anfang
par	parameter, d.h. name einer variablen, der beim aufruf der funktion ein wert als argument übergeben wird. Die parameter legen fest, wieviel und in welcher reihenfolge argumente an die funktion übergeben werden. Bei der schreibweise ist folgendes zu beachten:
par	Die variable ist **lokal** (call by value), d.h. sie kann nur innerhalb der funktion verwendet werden, änderungen haben keine auswirkung auf den wert außerhalb der funktion.
par=wert	wie zuvor; *w*enn beim aufruf kein wert übergeben wird, gilt der angegebene *wer*t.
&par	die variable ist **global** (call by referenz), es wird kein wert, sondern die adresse eines wertes übergeben, der außerhalb der funktion definiert ist. In der funktion wird mit diesem wert gearbeitet. Änderungen wirken sich auf diesen wert aus.
variable	in der funktion können variable definiert werden; sie sind lokal, d.h. sie gelten nur innerhalb der funktion.
global	wenn variable, die außerhalb der funktion definiert sind, in der funktion verwendet werden sollen oder wenn sie in der funktion definiert werden und außerhalb verwendet werden, müssen sie nach diesem schlüsselwort aufgeführt werden.
return	anweisung, mit der das ergebnis der funktion zurückgegeben wird; die anweisung darf fehlen, wenn kein ergebnis zurückgegeben wird, sie kann auch mehrfach vorkommen.
ergebnis	ausdruck oder variable mit dem ergebnis

4.5.2 variable parameterlisten

In der regel wird einer funktion für jeden parameter ein wert übergeben, das muß aber nicht so sein, wenn man in der funktion die folgenden funktionen verwendet.

$anz = **func_nums_args();**	liefert in $anz die anzahl der übergebenen argumente
$erg = **func_get_arg(3);**	liefert in $erg das dritte übergebene argument
$feld = **func_get_args()**;	liefert in $feld alle übergebenen argumente

5. felder

Ein feld ist eine zusammengehörige folge von variablen, die unter einem gemeinsamen namen zusammengefaßt werden. Die variablen eines feldes werden meist als elemente des feldes bezeichnet. Ein bestimmtes element des feldes wird mit hilfe eines index angesprochen; der index bezeichnet die position des elements innerhalb des feldes. Der index des ersten elements hat immer den wert 0 (null).

5.1 eindimensionales feld

5.1.1 definition

Bei einem eindimensionalen feld sind die elemente innerhalb des feldes in einer zeile angeordnet. Üblicherweise definiert man felder mit der funktion **array**.

name = **array** (*wert, wert*, ….);	definition und wertzuweisung
name[*index*] = *wert*;	wertzuweisung an ein element
$*erg* = *name*[*index*];	lesen eines elements

name	name des feldes; es ist eine bezeichnung wie bei einer variablen zu verwenden
index	index eines elements
wert	wert, der einem element zugewiesen wird; die angegebenen werte werden den elementen des feldes fortlaufend zugewiesen. Die anzahl der werte bestimmt die größe des feldes. Es können numerische werte, zeichenketten und logische werte (auch gemischt) zugewiesen werden.
$erg	variable, die den inhalt eines elements erhält.

```
$numfeld = array(25, 17, 1.23, 15.4, 13);
$strfeld = array("abc", "rm", "s  s", "xyz");
$mixfeld = array("abc", true, "xyz", 123.4, "123.4");
$mixfeld[2] = "geändert";
$mixfeld[5] = "angefügt";
$mixfeld[7] = "erweitert mit lücke";
```

Indem man einzelnen elementen einen wert zuweist, kann ein feld geändert oder erweitert werden. Mit **mixfeld[7]** wird gezeigt, dass beim erweitern lücken möglich sind. Beim feld **$neufeld** wird gezeigt, dass man ein feld auch ohne die funktion **array** erzeugen kann; auch dabei kann man lücken lassen.

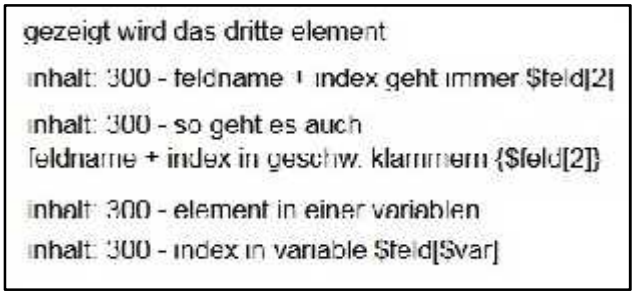
```
$neufeld[0] = 0;
$neufeld[1] = 1;
$neufeld[2] = 2;
$neufeld[5] = 2;
```

5.1.2 feldelemente anzeigen

Feldelemente können wie variable verwendet werden, neben dem feldname ist nur zusätlich, in eckige klammern eingeschlossen, der index des elemente anzugeben. Der index kann auch in einer variablen gespeichert sein.

beispiele
Merkwürdig ist nur das beispiel mit den geschweiften klammern; bei eindimensionalen feldern ist das eine zulässige, aber nicht notwendige möglichkeit.

```
<?php
$feld = array(100, 200, 300, 400, 500);
echo "<p>gezeigt wird das dritte element</p>";
echo "<p>inhalt: $feld[2] - feldname + index geht immer " . '$feld'
     . "[2]</p>";
echo "<p>inhalt: {$feld[2]} - so geht es auch<br />"
     . "feldname + index in geschw. klammern {" . '$feld[2]' . "}</p>";
$wert = $feld[2];
echo "<p>inhalt: $wert - element in einer variablen</p>";
$var = 2;
echo "<p>inhalt: $feld[$var] - index in variable " . '$feld[$var]' . "</p>";
?>
```



5.1.3 feld anzeigen

Für das anzeigen eines ganzen feldes gibt es hilfreiche anweisungen und funktionen

foreach - indices und werte anzeigen

Hier handelt es sich um eine anweisung zur ablaufsteuerung, sie ist der **for**-anweisung vergleichbar und definiert eine schleife, die abhängig von der anzahl der feldelemente durchlaufen wird.

foreach (*name* as [*key* =>] *var*)
{ *anweis*
}

name	name eines feldes
key =>	name einer variablen + zuweisungs-operand; bei jedem durchlauf wird in ***key*** der wert eines index zugewiesen. Fehlt die angabe, werden nur die inhalte der elemente bereitgestellt.
var	name einer variablen, der bei jedem durchlauf der inhalt des zu ***key*** gehörenden elements zugewiesen wird.
anweis	eine oder mehrere anweisungen, mit der/denen ***key*** und ***var*** verarbeitet werden können.

beispiel

```
<?php
$feld = array(100, 200, 300, 400, 500);
echo "<p>alle indices und zugehörigen werte<br />";
foreach($feld as $key => $wert)
{   echo "element $key enthält $wert <br />";
}
echo "</p>";

 echo "<p>alle werte<br />";
 foreach($feld as $wert)
 {   echo "$wert <br/>";
 }
 echo "</p>";
 ?>
```

```
alle indices und zugehörigen werte
element 0 enthält 100
element 1 enthält 200
element 2 enthält 300
element 3 enthält 400
element 4 enthält 500

alle werte
100
200
300
400
500
```

print_r und var_dump - feld testen

Diese beiden funktionen sind für den testbetrieb recht brauchbar, sie zeigen den gesamten inhalt eines feldes in einer oft recht langen zeile an. Wenn man die ausgabe präformatiert ausgibt, dann recht übersichtlich. Sie unterscheiden sich nur hinsichtlich der informationen, die sie anzeigen.

print_r($feld);

var_dump($feld);

```
<?php
$feld = array("abc", true, 125,
        123.4, "123.4");
echo "<pre>";
print_r($feld);
var_dump($feld);
echo "</pre>";
?>
```

Man beachte, dass bei **print_r** logische werte mit 0 (false) bzw. mit 1 (true) angezeigt werden.

```
Array
(
    [0] => abc
    [1] => 1
    [2] => 125
    [3] => 123.4
    [4] => 123.4
)
array(5) {
  [0]=>
  string(3) "abc"
  [1]=>
  bool(true)
  [2]=>
  int(125)
  [3]=>
  float(123.4)
  [4]=>
  string(5) "123.4"
}
```

5.1.4 in einem feld blättern

Mit den folgenden funktionen kann man in einem feld vorwärts und rückwärts blättern.

$wert = **current**($feld) liefert das aktuelle element, beim ersten aufruf das erste

$wert = **next**($feld) liefert das nächstes element

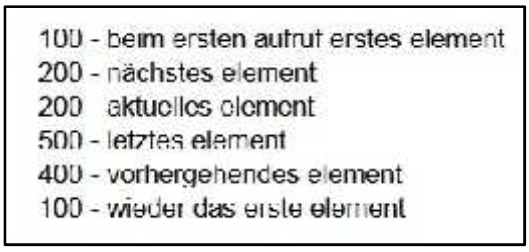
$wert = **prev**($feld) liefert das vorhergehende element

$wert = **end**($feld) liefert das letzte element

$wert = **reset**($feld) liefert das erstes element

beispiel

```
<?php
$feld = array(100, 200, 300, 400, 500);
echo "<p> " . current($feld) . " - beim ersten aufruf erstes element"
     . "<br />" . next($feld) . " - nächstes element"
     . "<br />" . current($feld) . " - aktuelles element"
     . "<br />" . end($feld) . " - letztes element"
     . "<br />" . prev($feld) . " - vorhergehendes element"
     . "<br />" . reset($feld) . " - wieder das erste element</p>"
?>
```



5.1.5 weitere funktionen für eindimensionale felder

$anz = **count**($feld);	anzahl der elemente
$anz = **sizeof**($feld);	anzahl der elemente
$erg = **max**($feld);	element mit dem höchsten wert
$erg = **min**($feld);	element mit dem niedrigsten wert
	die funktionen **max** und **min** sind nur bei numerischen feldern sinnvoll.
unset($feld[3]);	entfernt das vierte element, die übrigen elemente behalten aber ihren index
$feld[3] = "";	entfernt **nicht** das vierte element
$feld = **range**($wert1, $wert2 [, $schritt]);	erzeugt **$feld** mit allen werten zwischen **$wert1** und **$wert2** mit schrittweite **$schritt**. Fehlt $schritt gilt 1. Auch zeichen möglich
$erg = **in_array**("string", $feld);	sucht string in $feld , liefer true oder false
list($var1, $var2, …) = $feld	zerlegt $feld in var1, var2 usw, d.h. var1 = $feld[0], var2 = feld[1] . . .
[$neu =] **sort**($feld, **SORT_NUMERIC**);	aufsteigend numerisch sortieren
[$neu =] **rsort**($feld, **SORT_NUMERIC**);	absteigend numerisch sortieren
[$neu =] **sort**($feld, **SORT_STRING**);	aufsteigend nach zeichen sortieren
[$neu =] **rsort**($feld, **SORT_STRING**);	absteigend nach zeichen sortieren
[$neu =] **array_reverse**($feld);	liefert feld in umgekehrter reihenefolge
[$neu =] **array_unique**($feld);	entfernt duplikate, die übrigen elemente behalten ihren index
[$neu =] **array_push**($feld, $w1, $w2 …);	fügt **$w1** und folgene am ende von **$feld** ein.
[$neu =] **array_unshift**($feld, $w1, $w2 …);	fügt **$w1** und folgene am anfang von $feld ein.
[$erg =] **array_pop**($feld);	entfernt **letztes** element von **$feld** und speichert ggf. das element in **$erg**
[$erg =] **array_shift**($feld);	entfernt **erstes** element von **$feld** und speichert ggf. das element in **$erg**

Wenn bei den vorstehenden funktionen **$neu** angegeben ist, wird das ergebnis in dem feld $neu gespeichert und $feld bleibt unverändert, andernfalls kommt das ergebnis nach $feld.

achtung

Eine zeichenkette hat zwar eine gewisse ähnlichkeit mit einem eindimensionalen feld, beispielsweise kann man einzelne zeichen per index ansprechen, aber eine zeichenkette ist **kein** feld und deshalb kann man die im abschnitt 5.1 gezeigten funktionen nicht auf eine zeichenkette anwenden.

5.2 mehrdimensionales feld

Behandelt wird hier nur das zweidimensionale feld, in dem die elemente in zeilen und spalten angeordnet sind. Ein element hat daher zwei indices, einen zeilen- und einen spalten-index.

5.2.1 definition

name = **array** (**array** (*wert*, *wert*, . . .), **array** (*wert*, *wert*, . . .), . . .);

name [*index1*] [*index2*] = *wert*;

$*erg* = *name* [*index1*] [*index2*];

Ein zweidimensionales feld wird definiert, indem man in der funktion array für jede zeile erneut die funktion array für die spalten aufruft. Im übrigen geht alles wie beim eindimensionalen feld, d.h. ein zweidimensionales feld kann man durch angabe von elementen ändern, erweitern oder ganz neu anlegen wie ein eindimensionales, bei den elementen sind natürlich zwei indices anzugeben.

5.2.2 feldelemente anzeigen

merkwürdig ist hier, dass in einer echo-anweisung die angabe **name[index1] [index2]** nicht funktioniert, man muss die angabe in geschweifte klammern stellen, also **{ name[index1] [index2] }**

```
<?php
$feld = array (array (10, 20, 30, 40, 50),
        array ("aaa", "bbb", "ccc", "ddd", "eee"),
        array (60, 70, 80, 90, 100),
        array ("vvv", "www", "xxx", "yyy", "zzz"));
echo "<p>angezeigt wird das dritte element der zweiten zeile</p>";
echo "<p>inhalt: $feld[1][2] - feldname + indices geht nicht "
     . '$feld' . "[1][2] </p>";
echo "<p>inhalt: {$feld[1][2]} - aber in geschw. klammern {"
     . '$feld' . "[1][2]}</p>";
$var = $feld[1][2];
echo "<p>inhalt: $var - es geht in variable " . '$var = $feld'
     . "[1][2] </p>";
$x = 1;
$y = 2;
echo "<p>mit indices in variablen <br />";
echo "<p>inhalt: {$feld[$x][$y]} nur in geschw. klammern {"
     . '$feld[$x][$y]' . "}</p>";
?>
```

angezeigt wird das dritte element der zweiten zeile
inhalt: Array[2] - feldname + indices geht nicht $feld[1][2]
inhalt: ccc - aber in geschw. klammern {$feld[1][2]}
inhalt: ccc - es geht in variable $var = $feld[1][2]
mit indices in variablen
inhalt: ccc nur in geschw. klammern {$feld[$x][$y]}

5.2.3 funktionen und anweisungen für zweidimensionale felder

$anz = **count**($feld);	anzahl der zeilen
$anz = **count**($feld[1]);	anzahl der elemente der zweiten zeile
$anz = **sizeof**($feld);	anzahl der zeilen
$anz = **sizeof**($feld[1]);	anzahl der elemente der zweiten zeile

die funktionen **sort** und **rsort** liefern seltsame ergebnisse, die testfunktionen **print_r** und **var_dump** funktionieren. Was von den sonstigen funktionen brauchbar ist, wurde noch nicht getestet.

Die anweisung **foreach** ist für das gesamte feld nicht wirklich brauchbar:

foreach($feld as $key => $wert) liefert für jede zeile nur die information, dass das element einen array enthält.

foreach($feld[0] as $key => $wert) liefert dagegen für eine zeile (hier die erste) brauchbare werte.

5.2.4 felder mit höheren dimensionen

Man kann auch felder mit mehr als zwei dimensionen konstruieren, die komplikationen steigen entsprechend an und felder dieser art wurden bisher noch nicht getestet.

5.3 assoziatives feld

Hier handelt es sich um ein feld, bei dem den elementen kein ganzahliger wert, sondern ein schlüsselwort (zeichenkette) als index zugeordnet ist. Dieser index wird auch als **key** bezeichnet.

5.3.1 definition

name = **array** *(key* **=>** *wert*, *key* **=>** *wer*t, ….);	definition und wertzuweisung
name[*key*] = wert;	wertzuweisung an ein element
$erg = *name* [*key*];	lesen eines elements

name	name des feldes; es ist eine bezeichnung wie bei einer variablen zu verwenden
key	schlüsselwort, immer eine zeichenkette, also in anführungszeichen oder apostrophe
wert	wert, der einem schlüsselwort zugeordnet wird, es sind numerische werte, logische werte und zeichenketten zulässig (auch gemischt).
$erg	variable die den inhalt eines elements erhält.

Auch ein assoziatives feld kann man durch angabe einzelner elemente erweitern, ändern oder ganz neu anlegen. Das schlüsselwort kann dabei auch in einer variablen stehen.

```
$feld = array("Montag" => 20, "Dienstag" => 17.4, "Mittwoch" => 16.3);
$feld["Montag"] = 19;            //  überschreiben
$feld["Donnerstag"] = 21.3;      //  erweitern
$var = "Freitag";                //  key in variable
$feld[$var] = 23.6;              //  und erweitern
```

5.3.2 feldelemente anzeigen

Bei einem element aus einem assoziativen feld wird der **key** in anführungszeichen oder apostrophe eingeschlossen oder als variable angegeben. Die zuweisung eines elements an eine variable ist einfach, die beiden folgenden anweisungen liefern das richtige ergebnis.

```
$var1 = $feld["Dienstag"];
$var2 = $feld['Mittwoch'];
```

Probleme gibt es, wenn mit einer echo-anweisung eine HTML-anweisung erzeugt wird und dabei ein feldelement benötigt wird, denn diese anweisung muss in apostrophe oder anführungszeichen eingeschlossen werden. Aber apostrophe taugen nicht, denn dann wird bekanntlich nicht der inhalt sondern der name des elements ausgegeben. Anführungszeichen aber führen zu einem fehler, es sei denn, die angabe des elements ist in geschweifte klammern eingeschlossen. Das gilt auch, wenn als key eine variable angegeben ist.

Aus dem gesagten ergibt sich:

richtig ist:

```
echo "<p>es hat {$feld["Dienstag"]} grad </p>";
echo "<p>es hat {$feld['Dienstag']} grad </p>";
$var = "Dienstag";
echo "<p>es hat {$feld[$var]}  grad</p>";
```

untauglich ist, weil in apostrophe

```
echo '<p>es hat $feld["Dienstag"] grad</p>';
echo '<p>es hat {$feld["Dienstag"]} grad</p>';
$var = "Dienstag";
echo '<p>es hat {$feld[$var]}  grad </p>';
```

falsch ist:
weil entweder anführungszeichen in anführungszeichen, apostrophe in apostrophen oder apostrophe in anführungszeichen stehen

```
echo "<p>es hat $feld["Dienstag"] grad </p>";
echo "<p>es hat $feld['Dienstag'] grad </p>";
echo '<p>es hat $feld['Dienstag'] grad</p>';
```

mit basteln geht immer etwas

```
echo "<p>es hat  " . $feld["Dienstag"]  . " grad</p>";
echo "<p>es hat  " . $feld['Dienstag']  . " grad</p>";
$var = "Dienstag";
echo "<p>es hat  " . $feld[$var]  . " grad</p>";
```

5.3.3 anweisungen und funktionen

foreach
Die anweisung **foreach** ist gut zu verwenden, sie liefert für die key des feldes die namen der key.

funktionen

extract($feld);	liefert variable mit den namen der **key** und dem inhalt des damit assoziierten feldeelements.
$anz = **count**($feld);	anzahl der elemente
$anz = **sizeof**($feld);	anzahl der elemente
[$neu =] **asort**($feld, **SORT_NUMERIC**);	aufsteigend nach numerischem wert
[$neu =] **arsort**($feld, **SORT_NUMERIC**);	absteigend nach numerischem wert
[$neu =] **ksort**($feld, **SORT_STRING**);	aufsteigend nach key
[$neu =] **krsort**($feld, **SORT_STRING**);	absteigend nach key

Wenn bei den vorstehenden funktionen $neu angegeben ist, wird das ergebnis in dem feld $neu gespeichert und $feld bleibt unverändert.

beispiele

```
<?php
$feld = array("Montag" => 20, "Dienstag" => 17.4, Mittwoch" => 16.3,
              "Donnerstag" => 21.3, "Freitag" => 23.6);
echo '<p>inhalt von $feld<br />';
$anz = sizeof($feld);
echo "<p>es sind $anz elemente vorhanden</p>";
foreach($feld as $key => $wert)
   echo "am $key hatte es $wert grad <br />";
echo "</p>";
extract($feld);
echo "<p>am Montag hatte es $Montag grad</p>";
?>
```

inhalt von $feld
es sind 5 elemente vorhanden
am Montag hatte es 20 grad
am Dienstag hatte es 17.4 grad
am Mittwoch hatte es 16.3 grad
am Donnerstag hatte es 21.3 grad
am Freitag hatte es 23.6 grad

am Montag hatte es 20 grad

6. daten von seite zu seite

6.1 senden mit aufruf-parameter

6.1.1 seitenaufruf mit parametern

Beim aufruf einer seite können mit sog. aufruf-parametern daten an die aufgerufene seite übergeben werden. In der regel ruft eine seite eine andere seite mit einem link auf, aufruf-parameter können aber auch bei allen anderen methoden verwendet werden.

<a href="*seite*.php**?***param*=*wert***&***param*=*wert***&***param*=*wert* . . .">

seite	name der seite, die aufgerufen wird, in der regel wohl eine PHP-datei.
?	beginn der aufruf-parameter
&	trennzeichen zwischen aufruf-parametern
param	name eines parameters
wert	wert, der übergeben wird; immer eine zeichenkette, aber **nie** in anführungszeichen oder apostrophe. Kann auch eine variable sein.

6.1.2 übername der parameter

Die daten der aufruf-parameter werden mit der methode GET übertragen und in dem assoziativen feld **$_GET** bereitgestellt, dabei dienen die namen der paramerter als **key**. Für die auswertung des feldes werden entsprechende PHP-anweisungen verwendet.

erg = **$_GET**["*param*"];

erg	name einer variablen, der der wert des parameters zugewiesen wird.
param	name eines parameters

Für die aufgerufene seite ist nicht festgelegt, wieviel und welche parameter übergeben werden. Der versuch, einen nicht vorhandenen parameter zu übernehmen, führt zu einem fehler, daher ist es meist nötig, vor der auswertung zunächst zu prüfen, ob ein bestimmter parameter übergeben wurde, d.h. ob der erwartete wert in dem assoziativen feld **$_GET** vorhanden ist.

```
if isset($_GET["param"])
    erg = $_GET["param"];
```

6.1.3 übergebene werte säubern

Bei den übergebenen werten handelt es sich immer um zeichenketten, das birgt die gefahr, dass informationen übergeben werden, die für die aufgerufene seite unangenehme folgen haben. Wenn diese gefahr nicht auf der seite der aufrufenden seite ausgeschlossen werden kann, müssen die daten gesäubert werden. Dazu gibt es zwei funktionen.

$wert = **strip_tags**($wert [, "<b> <i>"]);

Die funktion entfernt aus $wert alle HTML-tags außer den in [ ] angegebenen.

$wert = **htmlspecialchars**($wert);

Die funktion maskiert in $wert gefährliche zeichen:

<	zu	<
>	zu	>
&	zu	&
"	zu	"

6.1.4 beispiele

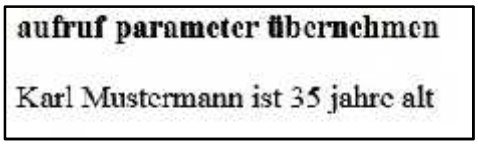
Mit dem folgenden link wird die seite **php-test.php** aufgerufen; der aufruf enthält drei aufruf-parameter.

```
<p>seite <b>php-testget.php</b> mit parametern aufrufen</p>
<p><a href="doku/PHP/php-test.php?name=Mustermann&vorn=Karl&alter=35">
   seite aufrufen</a></p>
```

Hier folgt der für das beispiel relevante teil der aufgerufenen seite; um das beispiel nicht zu überfrachten, wird nur für den parameter **name** geprüft, ob er vorhanden ist. In der praxis sollte man alle parameter prüfen.

```
<p><b>aufruf-parameter übernehmen</b></p>
<?php
   $ende  = chr(13) . chr(10);
   if (isset($_GET["name"]))
   {   $name    = $_GET["name"];
       $vorname = $_GET["vorn"];
       $alt     = $_GET["alter"];
       echo "<p>$vorname $name ist $alt jahre alt</p>" . $ende;
   }
   else
       echo "<p>kein name übergeben</p>" . $ende;
?>
```

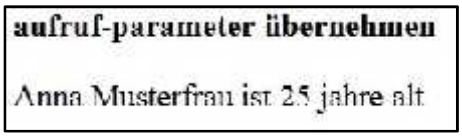
So sieht das ergebnis aus:



Als werte für die aufrufparameter kann man auch **PHP**-elemente verwenden,
der aufruf sieht dann wie folgt aus:

```
<p>seite <b>php-test.php</b> mit parametern aufrufen</p>
<?php
   $name = "Musterfrau";
   $vorname = "Anna";
   $jahre = 25;
   echo "<p><a href='doku/PHP/php-test.php"
        ."?name=$name&vorn=$vorname&alter=$jahre'>"
        . "seite aufrufen</a></p>";
?>
```

Das ergebnis ist sicher nicht überraschend



6.2 senden mit formular

Der aufbau und das senden von formularen ist in der **HTML**-dokumentation beschrieben, es ist zu beachten, dass formulareingaben in dem zeichensatz codiert werden, der bei der sendenden seite für den browser vereinbart ist (vgl. HTML-doku 2.2 und 2.5), bzw. der im form-tag bestimmt wird.. Hier wird nun dargestellt, wie ein gesendetes formular ausgewertet wird.

6.2.1 formular aufbauen und senden

Als beispiel, an dem die auswertung gezeigt wird, wird das folgende formular mit drei text-eingabefeldern verwendet.

```
<form name="fm1" action="doku/PHP/testform.php" accept-charset="utf-8"
      method="POST">
<p><input type="text" name="name" size="20" maxlength="40" /> Name</p>
<p><input type="text" name="vorn" size="20" maxlength="40" /> Vorname</p>
<p><input type="text" name="alter" size="3" maxlength="3" /> Alter</p>
<p><input type="submit" name="sender" value="abschicken" /></p>
</form>
```

name	formularname, nur notwendig, wenn an dem formular etwas mit Javascript gebastelt wird.
action	name der seite, an die das formular geschickt wird.
method	übergabemethode, hier POST; es ist auch GET möglich.
accepted-charset	zeichencode der eingabedaten.

Achtung
Alle formularelemente haben einen **namen**, das ist notwendig, denn abhängig von der übergabemethode stehen die value-werte der formularelemente auf der zielseite in den assoziativen feldern **$_POST** oder **$_GET** zur verfügung. Dabei werden als **key** die elementnamen verwendet. Meist ist es sinnvoll, auch dem submit-element einen namen zu geben; mehr dazu bei nr. 6.3.

POST oder GET
Das ist fast eine glaubensfrage, auf die hier nicht eingegangen wird. Zulässig sind beide methoden, bei formularen wird meist POST verwendet, aber bei großen datenmengen empfehlen fachleute GET.

6.2.2 formular auswerten

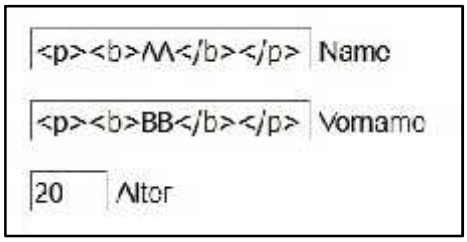
Die auswertung des übergebenen formulars entspricht der auswertung der aufruf-parameter, daher gleicht die seite **testform.php** weitgehend der seite **php-test.php**, hinzugekommen ist das säubern der werte. und die prüfung, ob für den key **alter** ein ganzzahliger wert übergeben wurde.

so sieht der für das beispiel relevante teil der aufgerufenen seite aus:

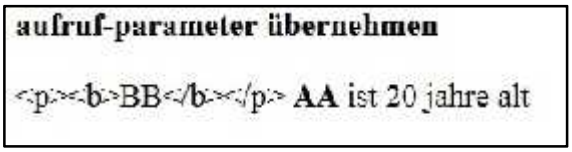
```
<p><b>formular-werte übernehmen</b></p>
<?php
   $ende  = chr(13) . chr(10);
   if (isset($_POST["name"]))
   {   $name    = strip_tags($_POST["name"], "<b> <i>" );
       $vorname = htmlspecialchars($_POST["vorn"]);
       $alt     = $_POST["alter"];
       $alt = intval($alt);
       if ((!is_int($alt)) || ($alt <= 0))
           $alt = "??";
       echo "<p>$vorname $name ist $alt jahre alt</p>" . $ende;
   }
   else
       echo "<p>kein name übergeben</p>" . $ende;
?>
```

beispiel

formular anzeigen



ergebnis anzeigen



In das formular wurde bei **Name** und **Vorname** HTML-code eingegeben. Das ergebnis zeigt, bei **Name** wird durch die funktion **strip_tags** das p-tag eliminiert, das b-tag bleibt erhalten und ist wirksam. Bei **Vorname** werden durch die funktion **htmlspecialchars** die zeichen < und > maskiert und damit die tags unwirksam

6.2.3 aufruf-parameter im formular

Im **form**-tag können auch aufruf-parameter angegeben werden, die dann in jedem fall in dem assoziativen feld **$_GET** bereitgestellt werden. Das beispiel könnte wie folgt geändert werden:

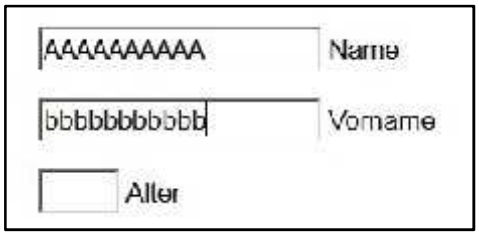
```
<form action="doku/PHP/testform.php?typ=Test&autor=Hartard" method="POST">
```

und die seite **testform.php** wird dann um die folgenden anweisungen erweitert.

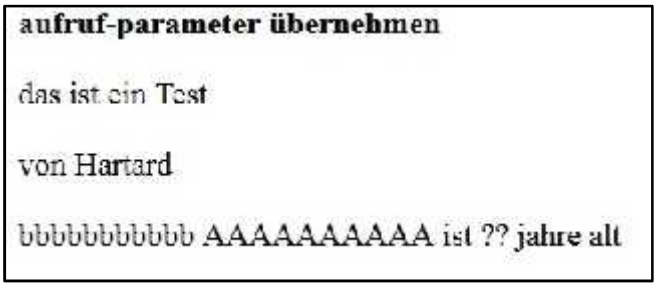
```
if (isset($_GET["typ"]))
{   $text = $_GET["typ"];
    echo "<p>das ist ein $text</p>" . $ende;
}
if (isset($_GET["autor"]))
{   $autor = $_GET["autor"];
    echo "<p>von $autor</p>" . $ende;
}
```

beispiel

formular anzeigen



ergebnis anzeigen



Das angezeigte formular unterscheidet sich nicht vom vorhergehenden beispiel, d.h. es ist nicht zu erkennen, dass beim aufruf der seite **testform.php** für **typ** und **autor** werte übergeben werden, die dann im ergebnis angezeigt werden.

6.3 rekursive formular-übergabe

Damit ist gemeint, dass eine seite ein formular aufbaut und dann an sich selbst schickt, d.h. sich selbst aufruft und dabei das formular übergibt. Das bedeutet, beim aufruf der seite muss geprüft werden, ob der aufruf mit oder ohne formular erfolgte; im einen fall kann das formular ausgewertet werden, im andern fall muss das formular aufgebaut werden. Da ein formular in der regel ein **submit**-element enthält, gibt man dem element einen namen und prüft in der aufgerufenen seite, ob im feld $_POST oder $_GET das element vorhanden ist. Natürlich kann man auch einen parameter übergeben, der anzeigt, ob der aufruf mit oder ohne formular erfolgt ist. Das bisher behandelte beispiel wurde für rekursive formular-übergabe umgebaut.

erster aufruf der seite

```
<p>seite <b>rekurs.php</b> aufrufen</p>
<p><a href="doku/PHP/rekurs.php">aufrufen</a></p>
```

so sieht die seite aus

```
<p><b>formular rekursiv senden</b></p>
<?php
   $ende  = chr(13) . chr(10);
   if (!isset($_POST["sender"]))
   {   echo "<p><b>formular aufbauen</b></p>" . $ende;
echo <<<DOC
       <form name="fm1" action="rekurs.php" accept-charset="utf-8" method="POST">
       <p><input type="text" name="name" size="20" maxlength="40" /> Name</p>
       <p><input type="text" name="vorn" size="20" maxlength="40" /> Vorname</p>
       <p><input type="text" name="alter" size="3" maxlength="3" /> Alter</p>
       <p><input type="submit" name="sender" value="abschicken" /></p>
       </form>
DOC;
       echo "<p><a href='Javascript: history.back();'>abbrechen </a></p>" . $ende;
   }
   else
   {   echo "<p>formular auswerten</p>" . $ende;
       if (isset($_POST["name"]))
       {   $name    = strip_tags($_POST["name"], "<b> <i>" );
           $vorname = htmlspecialchars($_POST["vorn"]);
           $alt     = $_POST["alter"];
           $alt = intval($alt);
           if ((!is_int($alt)) || ($alt <= 0))
               $alt = "??";
           echo "<p>$vorname $name ist $alt jahre alt</p>" . $ende;
       }
       else
           echo "<p>kein name übergeben</p>" . $ende;
       echo "<p><a href='Javascript: history.back();'>zurück </a></p>" . $ende;
   }
?>
```

hinweis
Es ist ein wenig unbefriedigend, dass man nach der auswertung mit **zurück** wieder zu dem ausgefüllten formular kommt und dann nur mit **abbrechen** das ganze zu ende bringt. Das geht auch anders, aber das beispiel sollte nicht mit komplikationen belastet werden. In der praxis sind solche rekursiven spielchen oft sehr kompliziert, weil beispielsweise das formular zur fehlerkorrektur zurückgegeben wird, dabei aber gültige werte nicht mehr verändert werden sollen und ähnliches. Machbar ist da fast alles.

6.4 komplexe formulare

Die meisten formulare sind einfach auszuwerten, nur bei zwei typen von formularelementen wird es gelegentlich etwas schwieriger.

6.4.1 checkbox / kontrollkästchen

Hier gibt es das problem, dass in einem formular oft mehrere checkboxen vorhanden sind und beim absenden nicht unbedingt klar ist, welche und wieviele boxen angeklickt wurden. Bei der auswertung bleibt da nichts anderes übrig, als alle boxen abzufragen, das kann man umständlich, aber auch elegant lösen. Voraussetzung für eine gute lösung ist, dass die elementnamen mit einem sinnvollen system vergeben werden.

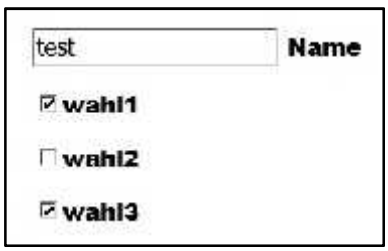
beispiel

Das formular enthält drei checkboxen und ein text-eingabefeld (nur zur verwirrung). Man achte auf die durchnumerierten namen der boxen.

```
<form action="doku/PHP/testforma.php?typ=box" accept-charset="utf-8"
      method="POST">
<p><input type="text" name="name" size="20" maxlength="40" /> Name</p>
<p><input type="checkbox" name="box1" value="ABCD" checked="checked" />wahl1<p>
<p><input type="checkbox" name="box2" value="XYZ" />wahl2</p>
<p><input type="checkbox" name="box3" value="mist" />wahl3</p>
<p><input type="submit" name="sender" value="abschicken" /></p>
</form>
```

Auf der aufgerufenen seite werden die boxen nicht einzeln abgefragt, sondern in einer schleife wird aus dem begriff **box** und einer fortlaufenden nummer der name einer box erstellt und dann geprüft ob diese box vorhanden ist.

```
<?php
   $ende  = chr(13) . chr(10);
   $typ = $_GET["typ"];
   if ($typ == "box")
   {   echo "<p><b>checkboxen auswerten</b></p>" . $ende;
       $name = $_POST["name"];    //  text-eingabe
       echo "<p> gewählt hat: $name<br />" . $ende;
       $z = 0;                    //  anzahl ausgew. boxen
       $c = 1;
       $anz = sizeof($_POST);     //  anzahl formularelemente
       for ($x=1; $x<=$anz; $x++)
       {   $key = "box" . $c;
           if  (isset($_POST[$key]))
           {   $erg = $_POST[$key];
               echo "auswahl: $erg<br />" . $ende;
               $z++;
               $c++;
           }
           else
               $c++;
       }
       if ($z > 0)
           echo "<p>$z auswahlen</p></p>" . $ende;
       else
           echo "<p>es wurde nichts gewählt</p>" . $ende;
   }
?>
```



6.4.2 mehrfach-auswahl-liste

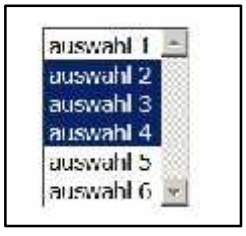
Hier ergibt sich ein kleines problem daraus, dass mehrere auswahlen möglich sind und die ausgewählten werte als feld übergeben werden, d.h. in dem assoziativen feld **$_POST** oder **$_GET** ist das element mit dem namen des select multible-tags als key selbst wieder ein feld. Das folgende beispiel zeigt, dass die auswertung gar nicht so schwierig ist.

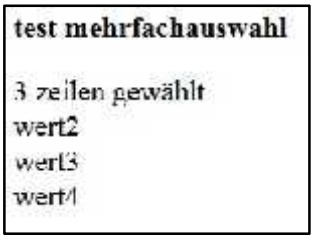
beispiel

```
<form action="doku/PHP/testforma.php?typ=opt" accept-charset="utf-8"
      method="POST">
<p><select multiple name="auswahl[]" size="6">
<option value="wert1">auswahl 1</option>
<option value="wert2">auswahl 2</option>
<option value="wert3">auswahl 3</option>
<option value="wert4">auswahl 4</option>
<option value="wert5">auswahl 5</option>
<option value="wert6">auswahl 6</option>
</select></p>
<p><input type="submit" name="sender" value="senden" /> </p>
</form>
```

Die auswertung in der aufgerufenen seite ist eigentlich einfach: in dem assoziativen feld **$_POST** wird mit dem key **auswahl** ein **feld** übergeben. Man stellt mit der funktion **sizeof** fest, wieviel elemente das feld enthält und arbeitet die elemente der reihe nach ab. Weil aber nicht sicher ist, dass überhaupt eine auswahl getroffen wurde, muss zuvor festgestellt werden, ob **auswahl** überhaupt übergeben wurde.

```
<?php
   $ende  = chr(13) . chr(10);
   $typ = $_GET["typ"];
   if ($typ == "opt")
   {   echo "<p><b>test mehrfachauswahl</b></p>" . $ende;
       if (isset($_POST["auswahl"]))
       {   $zahl = sizeof ($_POST["auswahl"]);
           echo "<p>$zahl zeilen gewählt<br />" . $ende;

           // damit funktioniert es, die routine ist
           // aber stillgelegt
    /*     for ($i=0; $i<$zahl; $i++)
           {   if (isset ($_POST["auswahl"] [$i]))
               {   $wert = $_POST["auswahl"] [$i];
                   echo "$wert<br />" . $ende;
               }
           }
           echo "</p>" . $ende; */

           // aber das ist eleganter
           foreach ($_POST["auswahl"] as  $wert)
               echo "$wert<br />" . $ende;
           echo "</p>" . $ende;
       }
       else
           echo "<p>es wurde nichts gewählt</p>" . $ende;
   }
?>
```





6.5 assoziative felder $_POST und $_GET

Bei den beispielen des abschnitts 6 wurde bei der auswertung dieser felder geprüft, ob bestimmte werte in dem feld vorhanden sind. Dazu musste man die namen der übergebenen parameter bzw. der feld-elemente kennen. Es geht aber auch ganz anders: die felder **$_POST** und **$_GET** stehen bei jedem aufruf einer seite zur verfügung, auch wenn gar nichts übergeben wurde, dann sind sie eben leer. Das bedeutet, man kann beim aufruf einer seite prüfen, ob die felder einen inhalt haben und ggf. dann die felder auswerten, am besten mit **foreach**. Für das feld $_GET wird das nachstehend gezeigt.

```
<p><b>hier ist die seite testget.php</b></p>
<?php
   $ende = chr(13) . chr(10);
   $anz  = count($_GET);
   if ($anz <= 0)
       echo "<p>es wurde nichts übergeben</p>" . $ende;
   else
   {   echo "<p>übergeben wurde:<br />" . $ende;
        foreach($_GET as $key => $wert)
           echo "parameter <b>$key</b> liefert: $wert<br />" . $ende;
       echo "</p>" . $ende;
   }
?>
```

aufruf der seite testget.php, einmal mit und einmal ohne parameter

```
<p><a href="doku/PHP/testget.php?name=Rembremerdeng&vorn=Wrdlbrmft&alter=99">
   seite aufrufen</a></p>
<p><a href="doku/PHP/testget.php">seite aufrufen</a></p>
```

Der erste aufruf liefert das folgende ergebnis, der zweite einen hinweis, dass nichts übergeben wurde

übergeben wurde:
parameter **name** liefert: Rembremerdeng
parameter **vorn** liefert: Wrdlbrmft
parameter **alter** liefert: 99

7. zeichenketten bearbeiten

7.1 zeichenketten bearbeiten

Wenn seiten mit PHP aufgebaut werden, ist es sehr häufig notwendig, zeichenketten aufzubereiten. Dafür gibt es viele funktionen. Soweit bei den funktionen als argumente zeichenketten anzugeben sind, können diese auch in variablen angegeben werden.

$erg = **strlen**($kette);	Mit dieser funktion wird die länge einer zeichenkette ermittelt.
$erg = **str_repeat**("abc", nn)	zeichenkette erzeugen mit **nn**-mal "**abc**"

7.1.1 zeichenkette umformen

$erg = **strtolower**($kette);	alles in kleinbuchstaben
$erg = **strtoupper**($kette);	alles in großbuchstaben
$erg = **ucfirst**($kette);	erstes zeichen großbuchstabe
$erg = **ucwords**($kette);	erstes zeichen aller worte als großbuchstabe
$erg = **strrev**($kette);	zeichenkette umdrehen
$erg = **strtr**($kette, "a", "A");	in der zeichenkette alle **a** zu **A**
$erg = **strtr**($kette, "xy", "AB");	in der zeichenkette alle **x** oder **y** zu **A** oder **B**
$erg = **str_replace**("abc", "xy", $kette);	in der zeichenkette alle **abc** zu **xy**
$erg = **trim**($kette);	entfernt ZWR am anfang und ende von $kette
$erg = **ltrim**($kette);	entfernt ZWR links, d.h. am anfang von $kette
$erg = **rtrim**($kette);	entfernt ZWR rechts, d.h. am ende von $kette

7.1.2 zeichenkette zerlegen / zusammenfügen

$kette = "das;ist;ein;test";

$feld = **explode**(";", $kette);	zerlegt in elemente von **$kette**, trennzeichen ist **;** (strichpunkt) die teile von $kette werden im feld $feld abgelegt, das ergibt hier $feld[0] enthält "das" $feld[1] enthält "ist" usw.
$kette = **implode**(".", $feld);	fügt die elemente von **$feld** mit trennzeichen punkt zu **$kette** zusammen. Ergebnis: "das.ist.ein.test"
$feld = **str_split**($kette, 5)	zerlegt $kette in jeweils 5 zeichen lange teile und speichert diese als elemente in $feld.

7.1.3 teil-zeichenketten bilden

$erg = **substr**($kette, 3 [, 5]);	von dem zeichen an position **3** bis zum ende von $kette oder maximal **5** zeichen werden in $erg gespeichert.
$erg = **substr**($kette, -8 [, 5]);	von dem zeichen an position **8** bis zum anfang von $kette oder maximal **5** zeichen werden in $erg gespeichert.
$erg = **strstr**($kette, "Test" [, true] | false]);	sucht **erstes** auftreten von "**Test**" und liefert als ergebnis ohne zusatz oder mit false: "Test" + rest von $kette mit true: alles von $kette vor "Test", ohne "Test"
$erg = **stristr**($kette, "Test" [, true] | false]);	wie **strstr**, aber groß/klein-schreibung **nicht** beachten
$erg = **strchr**($kette, "X");	sucht das **erste** zeichen **X** und liefert das zeichen + rest
$erg = **strrchr**($kette, "X");	sucht das **letzte** zeichen **X** und liefert das zeichen + rest
$erg = $kette[5];	das zeichen an position **5,** das geht ohne funktion, einfach durch zuweisung

7.1.4 position ermitteln

$erg = **strpos**($kette, "Test" [, 3]);	position von **"Test"** in $kette, suche von anfang an oder ab position **3**. Geliefert wird das **erste** ergebnis.
$erg = **strpos**($kette, "Test" , -20);	position von **"Test"** in $kette, suche ab position **20** in richtung anfang. Geliefert wird das **erste** ergebnis.
$erg = **stripos**($kette, $such [, $pos]);	wie **strpos**, aber groß/klein-schreibung **nicht** beachten
$erg = **strrpos**($kette, "Test" [, 3]);	position von **"Test"** in $kette, suche von anfang an oder ab position **3**. Geliefert wird das **letzte** ergebnis
$erg = **strrpos**($kette, "Test" , -20);	position von **"Test"** in $kette, suche ab position **20** in richtung anfang. Geliefert wird das **letzte** ergebnis
$erg = **strripos**($kette, $such [, $pos]);	wie **strrpos**, aber groß/klein-schreibung **nicht** beachten

hinweis

Bei den funktionen **teilzeichenketten bilden** (außer **substr**) und **position ermitteln** enthält das ergebnis eine zeichenkette, einen numerischen wert oder **false,** wenn kein brauchbares ergebnis geliefert wird. Sichere prüfung des ergebnis mit:

```
if ($erg === false)     ergebnis nicht brauchbar
if ($erg !== false)     ergebnis brauchbar
```

7.1.5 wordwrap - zeichenkette zerlegen

Diese etwas sonderbare funktion ist im anhang (12.1) beschrieben

7.2 zeichenketten vergleichen

$erg = **strcmp**($kette1, $kette2);	$kette1 mit $kette2 vergleichen
$erg = **strcasecmp**($kette1, $kette2);	beim vergleich groß-/klein-schreibung nicht beachten
$erg = **strnatcmp**($kette1, $kette2);	vergleich von strings mit numerischem inhalt
$erg = **strnatcasecmp**($kette1, $kette2);	vergleich von strings mit numerischem inhalt und zeichen
$erg = **similar_text**($kette1, $kette2);	anzahl gleicher zeichen

hinweise

Man muss beachten, dass im ASCII-code kleinbuchstaben "größer" sind als großbuchstaben und das bedeutet: bei der funktion **strcmp** ist **hallo** größer als **Hallo**. Die funktion **strcasecmp** erkennt das als gleich. Werden zeichenketten mit numerischem inhalt verglichen, muss man die funktion **strnatcmp** verwenden, andernfalls ist **"-100"** größer als **"100".** Ein wert wie **"+100"** liefert immer ein falsches ergebnis. Wenn gar werte wie **"A 100"** und **"a 100"** numerisch verglichen werden sollen, muss man die funktion **strnatcasecmp** verwenden.

Bei allen vergleichs-funktionen wird nicht die länge der zeichenketten verglichen, sondern der inhalt. Als ergebnis des vergleichs wird nicht, wie eigentlich zu erwarten, ein logischer, sondern ein numerischer wert geliefert. Es bedeutet:

0	$kette1 = $kette2
< 0	$kette1 < $kette2
> 0	$kette1 > $kette2

Wenn es nur darum geht, zu prüfen, ob zwei zeichenketten den gleichen inhalt haben, genügt auch eine einfache vergleichoperation:

```
if ($kette1 == $kette2)
```

7.3 daten aufbereiten

7.3.1 sprintf - daten formatieren

Die funktion bereitet ein oder mehrere datenelemente mit hilfe eines formats auf und schreibt das ergebnis in eine zeichenkette.

$zeile = **sprintf**("*format*", *wert* [, *wert*] …);

$zeile	zeichenvariable, in der das ergebnis der aufbereitung gespeichert wird.
format	zeichenkette oder variable, die das format enthält. Ein format wird wie folgt vereinbart:
	"[*text*] *fmanweis* [[*text*] *fmanweis*] …"
text	beliebige zeichen, die unverändert ausgegeben werden; auch ZWR zulässig.
fmanweis	formatanweisung, mit der die aufbereitung gesteuert wird
wert	wert, der mit der formatanweisung aufbereitet wird; numerisch, zeichen oder entsprechende variable

formatanweisung

%[*vz*][*fz*][**-**][*breite*][.*dez*]*typ*

%	smbol für eine formatanweisung; die angabe ist notwendig
vz	vorzeichen: + erzwingt bei positiven werten die ausgabe des plus-zeichens, bei negativen werten wird auch ohne diese angabe immer das minus-zeichen ausgegeben.
fz	füllzeichen; damit wird die ausgabe bis zur angegebenen *breite* aufgefüllt. Zulässig ist 0 oder ZWR oder jedes beliebige zeichen, dem aber ein apostroph vorangestellt werden muß. Das füllzeichen kann wahlweise angegeben werden.
breite	breite der ausgabe; wahlweise anzugeben. Fehlt die angabe, ist die breite so groß, wie zur aufbereitung des werts nötig ist.
-	ausgabe erfolgt linksbündig, fehlt diese angabe, erfolgt die ausgabe rechtsbündig.
.dez	anzahl der dezimalstellen; angabe wahlweise und nur für die ausgabe eines numerischen wertes zulässig.
typ	formattyp; die angabe ist notwendig

formattypen

d - ausgabe als ganzzahl

%**d**	ganze zahl volle länge
%5**d**	ganze zahl, mindestlänge 5
%05**d**	wie zuvor, mit führenden nullen

f - ausgabe mit dezimalpunkt

%**f**	in voller länge
%.2**f**	mit zwei stellen nach punkt, gerundet
%9.2**f**	wie zuvor, 9 zeichen lang
%09.2**f**	wie zuvor mit führenden nullen

weitere formattypen

%**s**	zeichen rechtsbündig	%-s	zeichen linksbündig
%%	prozentzeichen	%**b**	binärzahl
%**c**	dem wert entspr. ASCII -zeichen	%**o**	oktalzahl
%**u**	pos. vorzeichenlose ganzzahl	%**e**	wissenschaftl. schreibweise
%**x**	hexa mit kleinbuchstaben	%**X**	hexa mit großbuchstaben

beispiele

```
<?php
$Z1 = 125;
$Z2 = 123.456;
$TX = "ABCD";
$CH = 125;
$zeile = sprintf("int: %+d - float: %+f - string: %s", $Z1, $Z2, $TX);
echo "<pre>";
echo "<p>$zeile";
$zeile = sprintf("int: % 5d - float: % 10.2f - string: % 10s", $Z1, $Z2, $TX);
echo "<br />$zeile";
$zeile = sprintf("int: % -5d - float: % -10.2f - string: % -10s", $Z1, $Z2, $TX);
echo "<br />$zeile";
$zeile = sprintf("int: %05d - float: %010.2f - string: %'*10s", $Z1, $Z2, $TX);
echo "<br />$zeile";
$zeile = sprintf("int: %0-5d - float: %0-10.2f - string: %'*-10s", $Z1, $Z2, $TX);
echo "<br />$zeile";
$zeile = sprintf("zeichen: %c - oktal: %o - hexa: %X oder %x - binär: %b",
         $CH, $CH, $CH, $CH, $CH);
echo "<br />$zeile</p>";
echo "</pre>";
?>
```

```
int: +125 - float: +123.456000 - string: ABCD
int:   125 - float:     123.46 - string:       ABCD
int: 125   - float: 123.46     - string: ABCD
int: 00125 - float: 0000123.46 - string: ******ABCD
int: 125   - float: 123.460000 - string: ABCD******
zeichen: } - oktal: 175 - hexa: 7D oder 7d - binär: 1111101
```

erklärung des beispiels

Ohne erklärung ist das beispiel nur schwer zu verstehen. Es werden in fünf zeilen jeweils die variablen $Z1, $Z2 und $TX mit verschiedenen formaten aufbereitet und dabei die texte "int: ", "float: " und "string: " eingefügt.

$Z1 = 125; aufbereitung als ganzzahl

%+d	ausgabe ohne breitenangabe und füllzeichen, aber mit vorzeichen ergibt linksbündig **+125**
% 5d	ausgabe mir breite 5 und füllzeichen zwischenraum (ZWR) ergibt rechtsbündig **125** mit zwei ZWR am anfang
% -5d	ausgabe mit breite 5 und füllzeichen ZWR ergibt linksbündig **125** und dann 2 ZWR
%05d	ausgabe mit breite 5 und füllzeichen 0 ergibt rechtsbündig **00125**
%0-5d	ausgabe mit breite 5 und füllzeichen 0 ergibt linksbündig **125** und dann 2 ZWR, keinesfalls 0, das wäre unsinn

$Z2 = 123.456; aufbereitung als dezimalzahl

%+f	ausgabe ohne breitenangabe und füllzeichen, aber mit vorzeichen ergibt **+123.456000** - wieso hier drei schleppende nullen ??
% 10.2f	ausgabe mit breite 10, füllzeichen ZWR und zwei dezimalstellen ergibt rechtsbündig **123.46** und vier ZWR voraus. Es wurde gerundet.
% -10.2f	ausgabe mit breite 10, füllzeichen ZWR und zwei dezimalstellen ergibt linksbündig **123.46** und vier ZWR hinterher. Es wurde gerundet
%010.2f	ausgabe mit breite 10, füllzeichen 0 und zwei dezimalstellen ergibt rechtsbündig **0000123.46**, es wurde gerundet
%0-10.2f	ausgabe mit breite 10, füllzeichen 0 und zwei dezimalstellen ergibt linksbündig **123.460000**, es wurde gerundet

$TX = "ABCD" aufbereitung als zeichenkette

%s	ausgabe ohne breitenangabe ergibt linksbündig **ABCD**
% 10s	ausgabe mit breite 10 und füllzeichen ZWR ergibt rechtsbündig 6 ZWR und **ABCD**
% -10s	ausgabe mit breite 10 und füllzeichen ZWR ergibt linksbündig **ABCD** und 6 ZWR
%'*10s	ausgabe mit breite 10 und füllzeichen * ergibt rechtsbündig ********ABCD** Man beachte das zeichen apostroph vor dem zeichen * im format
%'*-10s	ausgabe mit breite 10 und füllzeichen * ergibt linksbündig **ABCD******** Man beachte das zeichen apostroph vor dem zeichen * im format

$CH = 125; aufbereitung eines numerischen wertes

%c	ergibt das zeichen **}**
%o	ergibt die oktalzahl **175**, im format steht der buchstabe o, nicht null
%X	ergibt die hexadezimalzahl **7D**
%x	ergibt die hexadezimalzahl **7d**
%b	ergibt die binärzahl 1111101

7.3.2 number_format - komplexe aufbereitung

Die funktion bereitet einen numerischen wert mit dezimalpunkt und tausender-komma auf

$erg = **number_format**(*numwert*, *nks*, [,*tnk* [, *tts*]] ;

$erg	variable für das ergebnis der aufbereitung
numwert	numerischer wert, der aufbereitet wird
nks	anzahl der nachkommastellen
tnk	trenner nachkommastellen (in anführungszeichen), standard ist punkt
tts	trenner tausender (in anführungszeichen), standard ist komma

anweisungen	***ergebnis***
`$var = 1.2367;` `$var = number_format($var, 2)`	`1.24`
`$var = 1234.5678` `$var = number_format($var, 2)`	`1,234.57`
`$var = 1234.5678` `$var = number_format($var, 2, "," , ".")`	`1.234,57`
`$var = 1234567` `$var = number_format($var, 2)`	`1,234,567.00`

7.3.3 sscanf - zeichenkette formatiert zerlegen

Die funktion zerlegt mit hilfe eines formats eine zeichenkette in mehrere variable. Das format wird wie bei der funktion sprintf aufgebaut. Enthält das format zeichen, müssen diese zeichen an der entsprechenden stelle in der zeichenketten enthalten sein. Praktisch kann **sscanf** nur in verbindung mit der funktion **list** eingesetzt werden.

sscanf ($zeile, "format")

```
<?php
$kette = "125, 123.45 EURO 20";
list ($anz, $preis, $nr) = sscanf ($kette, "%d, %f EURO %d");
echo "<p>$kette</p>";
echo "<p>$anz Artikel Nummer $nr kosten $preis €</p>";
?>
```

125, 123.45 EURO 20

125 Artikel Nummer 20 kosten 123.45 €

Nicht aufbereitet wurden die zeichenfolgen ", " und " EURO ", die zeichenfolgen "125", "123.45" und "20" wurden den variablen $anz, $preis und $nr zugewiesen.

7.4 datum und uhrzeit

7.4.1 datum und uhrzeit aufbereiten

time - zeitstempel lesen

Die funktion liefert einen sog. UNIX-zeitstempel, d.h. einen ganzzahligen wert, der angibt, wieviel sekunden seit dem 1. Januar 1970, 00:00:00 uhr bis zum aufruf der funktion vergangen sind.

$zeit = **time**( );

7.4.2 strftime, date - zeitstempel aufbereiten

Für diese beiden funktionen benötigt man einen zeitstempel der funktion **time.** Beide funktionen werten den zeistempel aus und liefern das ergebnis als zeichenkette.

$erg = **strftime**("*par par* …", $zeit);

$erg = **date**("*par par* ..", $zeit);

$erg enthält datum und/oder uhrzeit als zeichenkette aufbereitet.

par parameter, mit denen die aufbereitung gesteuert wird, zwischen den parametern dürfen beliebige zeichen stehen, die unverändert ausgegeben werden.

$zeit variable, die einen zeitstempel enthält.

parameter

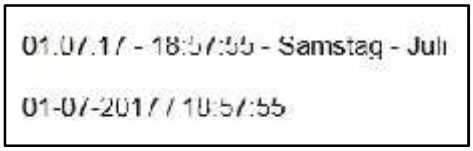
strftime	**date**	**ergebnis**
%d	d	tag 00 - 31
%m	m	monat 01 - 12
%Y	Y	jahr vierstellig
%y	y	jahr zweistellig
%j	z	tag des jahres 001 - 366 bei **strftime** 0 - 365 bei **date** (also immer +1)
	w	tag der woche 0 ist Sonntag usw.
%H	H	stunde 00 – 23
%M	i	minute 00 - 59
%S	s	sekunde 00 - 59
%A		name wochentag *)
%B		monatsname *)

*) das funktioniert nur, wenn zuvor aufgerufen wird:

setlocale(LC_ALL, "german");

beispiel

```
<?php
$zeit = time();
setlocale(LC_ALL, "german");
$erg = strftime("%d.%m.%y - %H:%M:%S - %A - %B", $zeit);
echo "<p>$erg</p>";
$erg = date("d-m-Y / H:i:s", $zeit);
echo "<p>$erg</p>";
?>
```



hinweis
Wenn das ergebnis der aufbereitung für rechenoperationen oder ähnliches verwendet werden soll, ist zu berücksichtigen, daß das ergebnis eine zeichenkette ist und deshalb in einen numerischen wert umgewandelt werden muß:

```
<?php
$wtag  = array("Sonntag","Montag","Dienstag",
               "Mittwoch","Donnerstag","Freitag","Samstag");
$jetzt  = time();                          //  datum abholen
$tag   = date("w", $jetzt);                //  nr des wochentags
$x     = intval($tag);                     //  umwandeln in ganzzahl
echo "<p>heute ist $wtag[$x]</p>";         //  als index für $wtag verwenden
?>
```

heute ist Samstag

7.4.3 checkdate - datum prüfen

[$bool =] **checkdate**($monat, $tag, $jahr)

7.4.4 strtotime - zeitstempel erzeugen

$var = **strtotime**("*monat tag jahr*");

$var	der zeitstempel wird in dieser variablen gespeichert
monat	monatsname (englisch)
tag	tag des monats (numerisch)
jahr	jahr (numerisch)

Es gibt noch viele weitere möglichkeiten, die hier nicht aufgeführt werden, weil das zu unendlichen basteleien führt. Eine gewisse bedeutung hat die funktion, weil man damit rechenoperationen durchführen kann.

beispiel

```
<?php
$heute = strtotime("January 28 2013");
$wichtig = strtotime("March 31 2013");
$tage = ($wichtig - $heute) / 60 / 60 / 24;
$dat1 = date("d.m.y", $heute);
$dat2 = date("d.m.y", $wichtig);
echo "<p>zwischen $dat1 und $dat2 liegen $tage tage</p>"
?>
```

zwischen 28.01.13 und 31.03.13 liegen 62 tage

8. ordner und dateien

8.1 aktuellen ordner ermitteln / ordner wechseln

8.1.1 getcwd - aktuellen ordner ermitteln

$verz = **getcwd( )**;

Die funktion liefert in $verz den pfadnamen des ordner (auf dem server), in dem sich die aufrufende seite befindet.

8.1.2 chdir - ordner wechseln

Mit dieser funktion wechselt man in einen anderen ordner des servers.

[*bool*] = **chdir**(*dir*);

dir zeichenkette oder variable mit dem namen des ordners, in den gewechselt werden soll. Als angabe ist möglich

name name eines unterordners des ordners, in dem sich die aufrufende seite befindet.

.. / *name* [/ *name*] . . .
name eines beliebigen ordners; man muß dabei unter umständen über mehrere stufen zum root-verzeichnis und dann wieder nach oben. Ausgangspunkt ist der ordner, in dem sich die aufrufende seite befindet.

bool ergebnis **true** oder **false** in einer variablen

8.2 informationen über einen ordner

8.2.1 opendir - ordner öffnen

$*handle* = **opendir**(*dir*);

dir zeichenkette oder variable mit dem namen eines ordners (wie oben)

$handle die funktion liefert in einer variablen als ergebnis einen sog. **handle,** mit dem in anderen funktionen der ordner angesprochen wird.

hinweis
Die angabe des ordners funktioniert so wie hier beschrieben nicht zuverlässig, sicherer ist es mit **chdir** in den gewünschten ordner zu wechseln, den namen mit **getcwd** zu lesen und damit den ordner zu öffnen:

```
$erg = chdir ("../test/daten")
if ($erg)
{    $name = getcwd();
     $ordner = opendir($name);
     weitere anweisungen
}
```

8.2.2 readdir - ordner-einträge lesen

Die funktion liest beim ersten aufruf einen eintrag aus einem ordner und schaltet zum nächsten eintrag weiter, der dann beim nächsten aufruf gelesen wird.

$eintrag = **readdir**($handle)

$eintrag	liefert jeweils den nächsten eintrag des ordners. Die beiden ersten einträge sind "." und ".."
$handle	variable, die den **handle** eines ordners enthält.

Nach dem lesen eines eintrags sind folgende funktionen sinnvoll verwendbar:

is_file($eintrag)	prüfen ob der eintrag eine datei ist
is_dir($eintrag)	prüfen ob der eintrag ein ordner ist
is_readable($eintrag)	prüfen ob der eintrag lesbar ist
is_writeable($eintrag)	prüfen ob der eintrag beschreibbar ist
$info = **stat**($eintrag)	weitere informationen, falls es sich um eine datei handelt (vgl. 8.3)

8.2.3 close - ordner schließen

close($handle)

beispiel

In dem folgenden, einfachen beispiel wird vom aktuellen ordner in den unterordner **doku/php** gewechselt und dann gezählt, wieviel unterordner und dateien sich in dem ordner befinden.

```
<?php
   $ende = chr(13) . chr(10);
   $zdir = 0;
   $zfil = 0;
   chdir("doku/php");                  //  in unterordner wechseln
   $verz = getcwd();                   //  aktuellen ordner ermitteln
   $handle = opendir($verz);           //  aktuellen ordner öffnen
   echo "<p class='font10b'>inhalt $verz</p>" . $ende;
   while($name = readdir($handle))     //  alle einträge lesen
   {   if (($name == ".") || ($name == ".."))
           $x = 0;
       else
       {   if (is_dir($name))          //  dateiverzeichnis
               $zdir++;
           else if (is_file($name))    //  datei
               $zfil++;
       }
   }
   closedir($handle);                  //  ordner schließen
   echo "<p class='font10'>der ordner enthält<br />"
        . "$zdir unterordner und $zfil dateien</p>" . $ende;
?>
```

8.3 stat - informationen über eine datei

Die funktion **stat** liefert zu einem dateinamen informationen über die datei.

$info = **stat**(*name*);

name	pfadname einer datei, befindet sich die datei im aktuellen ordner nur der dateiname
$info	feld mit den datei-informationen

inhalt des info-feldes
Das feld kann als eindimensionales oder assoziatives feld behandelt werden. Es wird nur aufgeführt, was unter Windows sinnvoll belegt ist.

index	***assoziativ***	***inhalt***
0	dev	gerätenummer
2	mode	dateiattribute
3	nlink	anzahl der links (??)
6	rdev	gerätetyp (??)
7	size	größe der datei in bytes
8	atime	datum des letzten zugriffs (UNIX-zeitstempel)
9	mtime	datum der letzten änderung (UNIX-zeitstempel)
10	ctime	datum der letzten attribut-änderung (UNIX-zeitstempel)

beispiel ordner-baum

Das beispiel zeigt den inhalt eines ordners und seiner unterordner

```
<?php
function dirbaum( )
{   global $ende;                       //  außerhalb der funktion definiert
    $verz   = getcwd( );                //  aktuellen ordner ermitteln
    $handle = opendir($verz);           //  handle für aktuellen ordner
    while ($dname = readdir($handle))   //  alle einträge lesen
    {   if ($dname != "." && $dname != "..")
        {   if (is_dir($dname))         //  wenn unterordner
            {   chdir($dname);          //  in unterordner
                dirbaum( );             //  funktion rekursiv aufrufen
                chdir("..");            //  wieder nach oben
            }
            else                        //  wenn datei
            {   $info = stat($dname);           //  datei-info lesen
                $dat  = date("d.m.Y", $info[8]);
                $wert = $info[2];               //  datei-attribute
                $okt  = sprintf("%o", $wert);   //  oktal aufbereiten
                $attr = substr($okt, -4);       //  die letzten vier stellen
                echo "<tr><td>$verz</td><td>$dname</td>"
                     . "<td>$dat</td><td>$attr</td></tr>" . $ende;
            }
        }
    }
    closedir($handle);                  //  handle schließen
}
```

```
$ende = chr(13) . chr(10);
chdir("doku");                          //  im unterordner doku beginnen
$verz = getcwd();                       //  aktuellen ordner zeigen
echo "<p class='font10'>start in <b>$verz</b></p>" . $ende;
echo "<table class='font10' style='width: 600px'>" . $ende;
echo "<tr><td style='width: 50%;'><b>ordner</b></td>"
     . "<td style='width: 25%;'><b>datei</b></td>"
     . "<td style='width: 15%;'><b>datum</b></td>"
     . "<td><b>attr.</b></td></tr>" . $ende;
dirbaum( );                             //  erster aufruf der funktion
echo "</table>" . $ende;
?>
```

Es wird hier nur ein ausschnitt aus dem ergebnis gezeigt.

start in **/home/www/public_html/doku/PHP**

ordner	datei	datum	attr.
/home/www/public_html/doku/PHP	downtest.txt	19.04.2020	0644
/home/www/public_html/doku/PHP/im	phpdok10-23.jpg	19.04.2020	0644
viele weitere einträge			
/home/www/public_html/doku/PHP	php-test.php	19.04.2020	0644
/home/www/public_html/doku/PHP	phpdoku1-1-inc.php	21.04.2020	0644
/home/www/public_html/doku/PHP	phpdoku1-3-inc.php	20.04.2020	0644
/home/www/public_html/doku/PHP	phpdoku1-inc.php	21.04.2020	0644
/home/www/public_html/doku/PHP	phpdoku10-1-inc.php	21.04.2020	0644
/home/www/public_html/doku/PHP	phpdoku10-2-inc.php	22.04.2020	0644

8.4 datei öffnen, schließen und sonstige funktionen

8.4.1 fopen - datei öffnen

Die funktion **fopen** liefert einen sog. dateizeiger (file-pointer). Bei allen weiteren funktionen wird die datei mit diesem dateizeiger angesprochen.

$fp = **fopen**("dateiname", "modus");

dateiname	name oder verzeichnis / name
modus	open-modus
	r lesen
	r+ lesen und schreiben
	w schreiben
	w+ schreiben und lesen
	a anhängen
	a+ anhängen und lesen

8.4.2 weitere funktionen

fclose($fp);	datei schließen
rewind($fp);	auf dateianfang setzen
if (**feof**($fp)) ..	abfrage auf dateiende, üblicherweise vor einer leseoperation
if (**file_exists**("dateiname")) ..	prüfen ob die datei vorhanden
$lang = **filesize**("dateiname");	liefert die länge der datei
[$erg] = **unlink**("dateiname");	datei löschen, $erg enthält true oder false

8.5 datei schreiben

Die beiden funktionen **fputs** und **fwrite** sind funktionsgleich.

fputs($fp, *kette* [, *nn*]);

fwrite($fp, *kette* [, *nn*]);

$fp	variable, die einen **dateizeiger** enthält
kette	zeichenkette, aus der zeichen in die datei geschrieben werden; die angabe einer variablen, die eine zeichenkette enthält, ist möglich.
nn	anzahl der zeichen, die geschrieben werden, fehlt die angabe, wird die gesamte zeichenkette in die datei geschrieben.

hinweise

- Die ausgabe-daten werden in dem zeichensatz codiert, der für den browser vereinbart ist. Einzelheiten dazu vgl. 2.2 und 2.5 der **HTML**-doku.
- Die zeichenkette kann zeilenende-zeichen enthalten, die mit in die datei geschrieben werden. Es gibt zwei möglichkeiten, das zeilenende zu definieren:

$ende = "\n" ;	ergibt X'0A'	also ein zeichen
$ende = chr(13) . chr(10);	ergibt X"0D0A"	also zwei zeichen

8.6 datei lesen

$kette = **fgets**($fp, *nn*);

$fp	variable, die einen **dateizeiger** enthält
nn	anzahl der zeichen, die gelesen werden sollen
$kette	variable mit dem ergebnis der leseoperation.

Die funktion **fgets** liest maximal ***nn*-1** zeichen, aber höchstens bis zu einem zeilenende-zeichen oder dateiende. Das zeilenende-zeichen erscheint nicht im ergebnis $kette. Üblicherweise verwendet man diese funktion, um eine datei zu lesen, die durch zeilenende-zeichen strukturiert ist.

$kette = **fread**($fp, *nn*);

$fp	variable, die einen **dateizeiger** enthält
nn	anzahl der zeichen, die gelesen werden sollen
$kette	variable mit dem ergebnis der leseoperation.

Die funktion **fread** liest ***nn*** zeichen oder bis zum dateiende, zeilenende-zeichen werden gelesen aber in ZWR umcodiert. Die funktion wird verwendet, um eine datei zu lesen, die nicht durch zeilenende-zeichen strukturiert ist oder wenn diese struktur ignoriert werden soll.

achtung

Bei der verarbeitung von dateien, die geschlossene umlaute oder bestimmte sonderzeichen (€, §, °, ², ³, µ) enthalten, gibt es unter umständen probleme. Siehe dazu ziffer 8.9.

8.7 beispiele

8.7.1 beispiel 1

Es werden fünf zeilen in eine datei geschrieben, dabei wird jeder zeile das zeilenende-zeichen angefügt. Beim lesen sollte man die größe so wählen, dass auch die längste zeile mit **einer** operation gelesen werden kann, sonst wird die zeile gnadenlos zerhackt (hier die zweite zeile)

```
<?php
   $text = array ( "Valentin", "Rembremerdeng",
          "Karlstadt", "Otto", "Wrdlbrmft" );
   $ZE = "\n";                          //  zeilenende-zeichen
   $anz   = sizeof($text);
   $fname = "testdaten.txt";
   $fp = fopen($fname, "w");            //  datei zum schreiben öffnen
   echo "<p class='cour10'><b>geschrieben wird:</b><br />" . $ende;
   for ($x=0; $x<$anz; $x++)
   {   $nr = $x + 1;
       $zeile = "eintrag nr. " . $nr . " $text[$x]";
       echo "$zeile<br />" . $ende;
       $zeile = $zeile . $ZE;           //  zeilenende anfügen
       fputs($fp, $zeile);              //  zeile in datei
   }
   fclose($fp);                         //  datei schließen
   if (file_exists($fname))             //  datei vorhanden ?
   {   $fp = fopen ($fname, "r");       //  datei zum lesen öffnen
       echo "<p class='cour10'><b>gelesen wird<b><br />" . $ende;
       while (!feof($fp))               //  solange nicht dateiende
       {   $zeile = fgets($fp, 25);     //  maximal 25 zeichen lesen
           echo "$zeile<br />";
       }
   }
   fclose($fp);                         //  datei schließen
   unlink ($fname);                     //  datei löschen
?>
```

geschrieben wird:
eintrag nr. 1 Valentin
eintrag nr. 2 Rembremerdeng
eintrag nr. 3 Karlstadt
eintrag nr. 4 Otto
eintrag nr. 5 Wrdlbrmft

gelesen wird
eintrag nr. 1 Valentin
eintrag nr. 2 Rembremerd
eng
eintrag nr. 3 Karlstadt
eintrag nr. 4 Otto
eintrag nr. 5 Wrdlbrmft

8.7.2 beispiel 2

Es wird die gleiche datei wie zuvor geschrieben (wird nicht gezeigt), aber dann mit der funktion fread gelesen. Das ergebnis ist nicht wirklich brauchbar.

```
<?php
   //   anfang wie bei beispiel 1

   $fp = fopen ($fname, "r");          //  datei zum lesen öffnen
   echo "<p class='cour10'><b>gelesen wird</b><br />" . $ende;
   while (!feof($fp))                  //  solange nicht dateiende
   {   $zeile = fread($fp, 25);        //  25 zeichen lesen
       echo "$zeile<br />";
   }
   fclose($fp);                        //  datei schließen
   unlink ($fname);                    //  datei löschen
?>
```

```
gelesen wird
cintrag nr. 1 Valentin ci
ntrag nr. 2 Rembremerdeng
cintrag nr. 3 Karlstadt
cintrag nr. 4 Otto cintra
g nr. 5 Wrdlbrmft
```

achtung

Die möglichen schwierigkeiten mit geschlossenen umlauten und einigen sonderzeichen wird bei ziffer 8.9 behandelt.

8.8 fseek - datei positionieren

Die funktion positioniert eine datei vor einer lese- oder schreiboperation auf eine beliebige position

fseek($fp, *nn*, SEEK_SET | SEEK_CUR | SEEK_END);

$fp	variable, die einen **dateizeiger** enthält
nn	anzahl der zeichen, um die ab einer position versetzt wird
SEEK_SET	versetzen ab dateianfang, ***nn*** ist positiv
SEEK_CUR	versetzen ab der aktuellen position, ***nn*** ist positiv oder negativ, d.h. es kann vorwärts oder zurück positioniert werden
SEEK_END	versetzen ab dateiende, ***nn*** ist negativ

ftell - position ermitteln

Die funktion liefert die aktuelle position als ganzzahligen wert.

$pos = **ftell**($fp)

beispiel

Wirklich praktikabel ist das positionieren eigentlich nur, wenn die datei eine gleichmäßige struktur von immer gleich langen teilen hat, wie in dem folgenden beispiel. Hier wird die datei mit "sätzen" von je 40 zeichen erstellt, dabei wird auf das zeilenende-zeichen ganz verzichtet. Satzweise lesen kann man die datei nur, wenn man immer nur teilstücke von 40 zeichen liest, genau das tut die funktion LESEN.

Im beispiel geschieht folgendes: eine datei wird geschrieben und mit der funktion LESEN gelesen und angezeigt. Dann wird die datei zum lesen **und** schreiben geöffnet, von der position 3 wird ein satz gelesen und an die position 4 geschrieben. Dabei ist zu beachten, dass nach dem lesen oder schreiben eines satzes, die datei auf den nächsten satz positioniert ist. Zuletzt wird die datei noch einmal gelesen, angezeigt und dann gelöscht.

```
datei wurde geschrieben

das steht in der datei:
eintrag nr. 00001 ************Valentin//
eintrag nr. 00002 *******Rembremerdeng//
eintrag nr. 00003 ***********Karlstadt//
eintrag nr. 00004 ****************Otto//
eintrag nr. 00005 ***********Wrd|brmtt//


satz von position 3 lesen
eintrag nr. 00003 ***********Karlstadt//

an nächste (4.) position schreiben

das steht jetzt an position 4
eintrag nr. 00003 ***********Karlstadt//

das steht in der datei:
eintrag nr. 00001 ************Valentin//
eintrag nr. 00002 *******Rembremerdeng//
eintrag nr. 00003 ***********Karlstadt//
eintrag nr. 00003 ***********Karlstadt//
eintrag nr. 00005 ***********Wrd|brmtt//
```

```php
<?php
   $ende = chr(13) . chr(10);

   //
   //   funktion LESEN
   //
   function LESEN ($par)
   {   global $ende;
       $fpa = fopen($par, "r");     //  zum lesen öffnen
       echo "<p class='cour10'><b>das steht in der "
            . "datei:</b><br />" . $ende;
       while (!feof($fpa))          //  nicht dateiende
       {   $ber = fread($fpa, 40);  //  lesen
           echo "$ber <br />" . $ende;
       }
       echo "</p>" . $ende;
       fclose($fpa);                //  datei schließen
   }
```

```
   //
   //   verarbeitung
   //
   $text = array ( "Valentin", "Rembremerdeng",
          "Karlstadt", "Otto", "Wrdlbrmft" );
   $anz   = sizeof($text);
   $fname = "testdaten.txt";
   $fp = fopen($fname, "w");      //  öffnen zum schreiben
   for ($x=0; $x<$anz; $x++)
   {   $nr = $x + 1;              //  $satz mit länge 40
       $satz = sprintf("eintrag nr. %05d %'*20s//", $nr, $text[$x]);
       fwrite($fp, $satz);        //  schreiben
   }
   fclose($fp);                   //  datei schließen
   echo "<p class='cour10'><b>datei wurde geschrieben</b>"
        . "<br/>" . $ende;
   LESEN($fname);
   $fp = fopen ($fname, "r+");    //  lesen und schreiben
   echo "<p class='cour10'><b>satz von position 3 lesen</b>"
        . "<br />" . $ende;
   fseek($fp, 2*40, SEEK_SET);    //  position 3. satz
   $satz = fread($fp, 40);        //  3. satz lesen
   echo "$satz<br />";
   echo "<p class='cour10'><b>an nächste (4.) position "
        . "schreiben</b><br />" . $ende;
   fwrite($fp, $satz);            //  in nächsten satz schreiben
   echo "<p class='cour10'><b>das steht jetzt an position 4"
        . "</b><br />" . $ende;
   fseek($fp, -80, SEEK_END);     //  position 4. satz
   $satz = fread($fp, 40);        //  40 zeichen lesen
   echo "$satz<br />";
   fclose($fp);                   //  datei schließen
   echo "</p>" . $ende;
   LESEN($fname);
   unlink ($fname);               //  datei löschen
?>
```

8.9 zeichensatz für datei-ein/ausgaben

Bei der darstellung von geschlossenen umlauten und einigen sonderzeichen (€, §, °, ², ³, µ) gibt es oft probleme. Im sog. Unicode (utf-8) sind diese zeichen mit **zwei bytes** verschlüsselt. Man vermeidet probleme mit diesen zeichen, wenn man sicherstellt, dass der zeichensatz mit dem eine seite erstellt wurde der gleiche ist, der im header der seite vereinbart wird. Die **HTML**-beschreibung enthält dazu details (siehe dort ziffer 2.5). Wenn es sich um den zeichensatz **Unicode** handelt, wird eine ausgabe-datei mit diesem zeichensatz erstellt und eine eingabe-datei wird problemlos verarbeitet, wenn sie ebenfalls im **Unicode** erstellt wurde. Wurde die eingabe-datei mit **Ansicode** erstellt, gibt es probleme. Wenn konsequent überall mit **Ansicode** gearbeitet wird, sollte es eigentlich keine probleme geben, aber leider stimmt das nicht (mehr), weil viele provider, bei denen man eine homepage hostet, diesen code sehr häßlich behandeln.

Das nachstehende beispiel zeigt die darstellung von zwei inhaltlich gleichen eingabe-dateien, von denen die erste im **Ansicode** und die zweite im **Unicode** erstellt wurde. Die dateien sind einfache textdateien mit zeilenende-zeichen und werden mit **fgets** eingelesen. Bei der Ansicode-datei können die geschlossenen umlaute und die o.g. sonderzeichen nicht dargestellt werden. Das beispiel zeigt noch mehr: maskierte zeichen werden immer richtig dargestellt und das tabulator-zeichen wird durch zwischenraum ersetzt, bewirkt aber sonst nichts. Das gilt für alle steuerzeichen.

inhalt der datei

```
diese datei ist mit dem zeichensatz Ansicode erstellt
und enth◆lt sonderzeichen und umlaute
! § % & / ( ) = ? @ | - #
◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆ ◆
maskiert: Ä ä €
mit tabulator: tab ◆◆◆◆◆◆ ABC XYZ
```

inhalt der datei

```
diese datei ist mit dem zeichensatz Unicode erstellt
und enthält sonderzeichen und umlaute
! § % & / ( ) = ? @ | - #
Ä Ö Ü ä ö ü ß ‰ ° ² ³ µ
maskiert: Ä ä €
mit tabulator: ÄÖÜäöü ABC XYZ
```

Im einfachsten fall behebt man das problem, indem man die eingabedatei mit hilfe eines geeigneten editors von Ansicode in Unicode umcodiert. Aber wenn das nicht möglich ist oder die seite aus welchen gründen auch immer eine seite eingaben sowohl in Unicode wie in Ansicode verabeiten soll, dann beginnt eine arge bastelei. Das folgende beispiel zeigt das.

beispiel

Die dateien werden wie zuvor mit fgets eingelesen und dann zeilenweise mit der funktion UMSETZ umcodiert. Wenn umlaute und die o.g. sonderzeichen im Ansicode codiert sind werden sie durch die entsprechenden 2-byte-zeichen des Unicode ersetzt. Liegen die zeichen im Unicode vor, wird das erste byte abgeschnitten und dann das passende zeichen im Unicode eingefügt (das scheint umständlich, vereinfacht aber die prozedur). Alle steuerzeichen werden unterdrückt, mit ausnahme des tabulator-zeichens, das so durch geschützte leerzeichen ersetzt wird, dass nachfolgender text auf die positionen 8, 16, 24 usw. kommt.

```
function UMSETZ($line)
{
   $umt1 = array( 132, 150, 156, 164, 182, 188, 196, 214, 220,
                  228, 246, 252, 159, 223, 128, 130, 167,
                  176, 178, 179, 181 );
   $umt2 = array( "Ä", "Ö", "Ü", 'ä', 'ö', 'ü', 'Ä', 'Ö', 'Ü',
                  'ä', 'ö', 'ü', 'ß', 'ß', '€', '€', '§',
                  '°', '²', '³', 'µ' );
   $zwr = " ";                    //  geschützter zwr
   $l = strlen($line);
   $text = "";                         //  ergebnis-zeile
   $pos  = 0;                          //  position in zeile
   for ($x=0; $x<$l; $x++)             //  zeichenweise
   {   $od = ord($line[$x]);           //  in dezimalwert
       if ($od < 32)                   //  steuerzeichen
       {   if ($od == 9)               //  tabulator ersetzen
           {   $mod = $pos % 8;        //  je nach position
               $zz  = 8 - $mod;        //  durch 1-8 zwr
               $char = $zwr;
               for ($y=0; $y<$zz; $y++)
                   $char = $char . $zwr;
               $pos += $zz;
           }
           else                        //  nicht darstellbar
              $char = "";              //  unterdrücken
       }
```

```
        else                            //  geschl. umlaute
        {   if ($od > 127)              //  und sonderzeichen
            {   $utf = false;           //  umcodieren
                for ($z=0; $z<21; $z++)
                {   if ($od == $umt1[$z])
                    {   $char = $umt2[$z];
                        $pos += 2;
                        $utf = true;
                        break;
                    }
                }
                if (!$utf)              //  nicht darstellbar
                    $char = "";         //  unterdrücken
            }
            else                        //  darstellbare
            {   $char = $line[$x];      //  zeichen
                $pos++;                 //  übernehmen
            }
        }
        $text = $text . $char;          //  an ergebniszeile
    }
    return $text;
}
```

Vom ergebnis wird nur die anzeige der Ansicode-datei gezeigt, weil die anzeige der Unicode-datei völlig gleich ist. Man beachte auch, die wirkung des tabulator-zeichens.

inhalt der datei

```
diese datei ist mit dem zeichensatz Ansicode erstellt
und enthält sonderzeichen und umlaute
! § % & / ( ) = ? @ | - #
Ä Ö Ü ä ö ü € ‰ ° ² ³ µ
maskiert: Ä ä €
mit tabulator:   tab      ÄÖÜäöü     ABC      XYZ
```

9. spezielle datei-operationen

9.1 datei hochladen (upload)

Mit einem speziellen formular kann man von einem PC eine datei auf den server hochladen. Die konstruktion von formularen wird in der **HTML-dokumentation** erklärt, hier wird nur auf die besonderheiten des speziellen formulars eingegangen.

<form **enctype="multipart/form-data"** action="seite.php" method="**POST**">

[<input type="**hidden**" name="**MAX_FILE_SIZE**" value="***wert***" />]

<input name="***fname***" type=**"file"** />

Das **form**-tag benötigt das **enctype**-attribut mit der angabe **multipart/form-data** und die methode **POST.**
Es ist ein **input**-tag mit dem typ **file** nötig; das tag muss ein **name**-attribut haben, der hier verwendete, frei wählbare name ***fname*** wird in der aufgerufenen seite verwendet. Wahlweise kann ein **input**-tag mit dem typ **hidden** angegeben werden. Das tag benötigt das **name**-attribut mit der angabe **MAX_FILE_SIZE** und einen numerisch ***wert*** als **value-**attribut. Damit wird für die hochzuladende datei eine maximale größe festgelegt. Das tag muss **vo**r dem input-tag mit dem typ **file** stehen. Es wurde nicht getestet, was geschieht, wenn wenn die maximale größe überschritten ist.

In der **aufrufenden** seite bietet das formular die möglichkeit, eine datei für das hochladen auszuwählen. Die datei wird zunächst in eine temporäre datei des servers hochgeladen. Der aufgerufenen seite wird das zweidimensionale, assoziative feld **$_FILES** mit folgenden werten zur verfügung gestellt mit denen in der **aufgerufenen** seite das hochladen zum abschluß gebracht werden kann.

$_FILES["*fname*"]["name"]	original-dateiname
$_FILES["*fname*"]["size"]	dateigöße in bytes
$_FILES["*fname*"]["type"]	dateityp (stimmt nur bedingt)
$_FILES["*fname*"]["tmp_name"]	temporäre datei auf dem server
$_FILES["*fname*"]["error"]	fehlercode (0 = ok)

Man kann diese werte nur lesen, besser gesagt, es wurde nicht getestet, was geschieht, wenn man die werte ändert.

prüfen
Es sollte zunächst geprüft werden, ob die datei als temporäre datei hochgeladen wurde, dazu gibt es zwei möglichkeiten:

```
if ($_FILES["fname"]["size"] > 0)
```

bessere variante

```
if (isset($_FILES["fname"]) AND (!$_FILES["fname"]["error"]))
```

Nur nach erfolgreicher prüfung ist es sinnvoll, das hochladen fortzusetzen.

kopieren
Dann kann die temporäre datei an ihren endgültigen platz kopiert werden, wird dabei nur ***neuer-name*** angegeben, wird die datei in den ordner der aufgerufenen seite kopiert, es ist aber auch die angabe eines pfadnamens (*dir* / *neuer-name* u.ä.) möglich.

copy ($_FILES["*fname*"]["tmp_name"], "*neuer-name*"); oder bessere varianten

move_uploaded_file ($_FILES["*fname*"]["tmp_name"], "*neuer-name*")

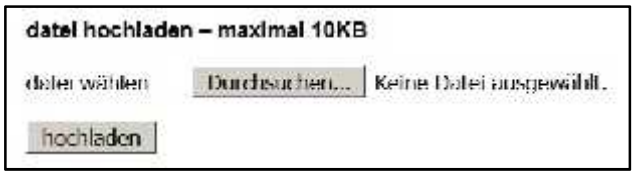
move_uploaded_file ($_FILES["*fname*"]["tmp_name"], **basename**($_FILES["*fname*"]["name"]) *)

*) damit behält die hochgeladene datei ihren ursprünglichen namen. Wenn das gewünscht wird, sollte man unbedingt diese variante verwenden, weil dabei der eventuell mitgelieferte pfadnamen abgeschnitten wird. Soll die datei nicht in den ordner der aufgerufenen seite kopiert werden, muss man schreiben **"*dir*/" . basename**

beispiel

definition des formulars

```
<p><b>datei hochladen – maximal 10KB</b></p>
<form enctype="multipart/form-data" action="doku/PHP/testladen.php"
      method="POST">
<input type="hidden" name="MAX_FILE_SIZE" value="10240" />
</p>datei wählen:     
    <input name="datei" type="file" />     </p>
<p><input name="sender" type="submit" value="hochladen" /></p>
</form>
```

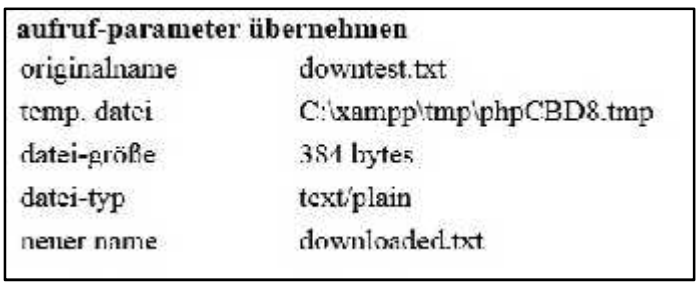


Nach der auswahl einer datei wird neben dem button **Durchsuchen** der dateiname angezeigt. Nach dem betätigen des buttons **hochladen** wird die in **action** angegebene seite **testladen.php** geladen und das formular an die seite übergeben.

datei testladen.php

Die seite wertet das feld **$_FILES** aus, extrahiert das suffix des dateinamens und rechnet die größe der datei von bytes auf kilobytes um.

```
<?php
   $ende  = chr(13) . chr(10);
   if (isset($_FILES["datei"]) AND (! $_FILES["datei"]["error"]))
   {   $orig = $_FILES["datei"]["name"];         //  originalname
       $temp = $_FILES["datei"]["tmp_name"];     //  temporäre datei
       $lang = $_FILES["datei"]["size"];         //  dateigröße
       $art  = $_FILES["datei"]["type"];         //  dateityp
       //   suffix des dateinamens extrahieren
       $feld = explode(".", $_FILES["datei"]["name"]);
       $len  = count($feld);
       $ext  = $feld[$len - 1];
       $neu  = "downloaded." . $ext;             //  neuer name
       //   temporäre datei kopieren
       //   move_uploaded_file($_FILES["datei"]["tmp_name"], $neu);
       //   anzeige
       echo "<table style='width: 500px'>" . $ende;
       echo "<tr><td style='width:30%;'>originalname</td><td>2
            . "$orig</td></tr>" . $ende;
       echo "<tr><td>temp. datei</td><td>$temp</td></tr>" . $ende;
       echo "<tr><td>datei-größe</td><td>$lang bytes</td></tr>" . $ende;
       echo "<tr><td>datei-typ</td><td>$art</td></tr>" . $ende;
       echo "<tr><td>neuer name</td><td>$neu</td></tr>" . $ende;
       echo "</table>";
   }
   else
       echo "<p>ERR: datei nicht hochgeladen</p>" . $ende;
?>
```



9.2 datei herunterladen (download)

Es ist möglich von einer seite aus eine datei vom server herunter zu laden. Allerdings kann man sich als benutzer diese seite nicht aussuchen, sondern man muss nehmen, was der verfasser der seite anbietet. Die zum herunterladen aufgerufene seite muss das mit einer sehr speziellen funktion tun. Die funktion benötigt den namen des ordners, in dem sich die datei befindet, den namen der datei, den sog. mime-typ der datei und den namen, den die heruntergeladene datei erhalten soll. Beim herunterladen kann man sich die datei anzeigen lassen oder tatsächlich herunterladen. Dabei kann man den zielordner bestimmen und der datei einen anderen namen geben.

mögliche mime-typen (auswahl)

css	text/css	gif	image/gif
htm	text/htm	jpe	image/jpeg
html	text/html	jpg	image/jpeg
txt	text/plain	jpeg	image/jpeg
php	text/php	png	image/png
doc	application/msword	pdf	application/pdf
exe	application/octet-stream	pps	application/mspowerpoint
js	application/x-javascript	zip	application/zip

seite testdown.php

```
<?php
   $ende  = chr(13) . chr(10);
   $ord   = "../../doku/PHP/";    //  ordner der datei
   $datei = "downtest.txt";       //  dateiname
   $pfad  = $ord . $datei;        //  pfadname der datei
   $mime  = "text/plain";
   $dldat = "downloaded.txt";     //  heruntergeladene datei

   //  $datei darf nicht leer sein
   //  darf keinen schrägstrich enthalten
   //  muss vorhanden sein
   if (!empty($datei) && !preg_match('=/=', $datei))
   {   if (file_exists ($datei))
           down($datei, $ord, $mime, $dldat);
       else
           echo "<p>datei $pfad nicht gefunden</p>" . $ende;
   }
   else
       echo "<p>irgendwie quatsch</p>" . $ende;
   //
   //   funktion down - datei herunterladen
   //   übergeben wird für
   //   $file - $datei  -  datei die heruntergeladen wird
   //   $dir  - $ord    -  ordner in dem die datei ist
   //   $type - $mime   -  mime-typ der datei
   //   $dat  - $dldat  -  name heruntergeladene datei
   //
   //   die funktion überträgt den mime-typ und den namen
   //   für die heruntergeladene datei zum browser,
   //   liest die datei ein und übergibt sie dem browser
   //
   function down($file, $dir, $type, $dat)
   {
      header("Content-Type: $type");
      header("Content-Disposition: attachment; filename=\"$dat\"");
      readfile($dir.$file);
   }
?>
```

9.3 email senden

Mit der funktion **mail** kann man von einer seite aus eine email verschicken.

[erg] = **mail**("*adresse*", "*betreff*", "*nachricht*" [, "**FROM**: *absender*"]);

adresse	email.adresse des empfängers in der form aaa@irgendwo.de aaa@irgendwo.de**\r\nCc:** nnn@web.de Es muss eine gültige email-adresse angegeben werden. Die zweite form der angabe (mit Cc:) wurde noch nicht getestet.
betreff	wahlweise angabe, erscheint im betreff der email
nachricht	enthält die nachricht, siehe hinweise;
absender	Es sollte eine gültige email-adresse angegeben werden, damit der empfänger die nachricht unmittelbar beantworten kann. Wird keine gültige email-adresse oder ein beliebiger text angegeben, wird beim empfänger als absender eine recht seltsame adresse angegeben, zu der keine antwort geschickt werden kann.
erg	variable für das ergebnis; Die funktion liefert true, wenn die mail abgeschickt wurde.

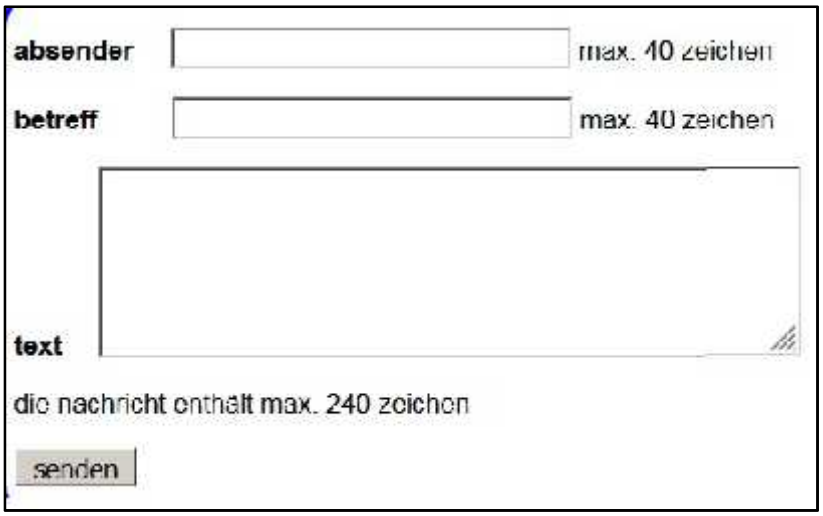
Alle angaben können auch in variablen gemacht werden.

hinweise

- Aus sicherheitsgründen sollten die angaben zu **absender, betreff** und **nachrich**t mit der funktion **htmlspecialchars** gesäubert werden (siehe 6.1).
- Die nachricht sollte nicht wesentlich mehr als **500 zeichen** enthalten, sonst macht die funktion irgendwann schwierigkeiten. Die maximalgröße konnte bisher nicht festgestellt werden.
- Im betreff und dem text der nachricht sollte man die zeichen Ä, Ö, Ü, ä, ö, ü, €, §, & vermeiden, sie kommen beim empfänger als schmierzeichen an.
- Auch mit dem zeilenumbruch gibt es manchmal probleme, also darauf verzichten oder sehr sparsam sein
- Die zeilen der nachricht sollten maximal **70 zeichen** enthalten. Sicherheitshalber beschränkt man in der nachricht die wortlänge auf diese oder eine geringere länge. Das macht man mit der funktion **wordwrap** (siehe anhang)..
- Die funktion **mail** kann nur im serverbetrieb verwendet werden, also nicht beim testen unter XAMPP.

beispiel

Mit dem folgenden formular wird die seite **testmail.php** aufgerufen, die eine email an den autor dieser doku sendet.



code des formulars

```
<form name="form1" action="doku/PHP/testmail.php" method="POST">
<p><b>absender</b>    
   <input type="text" name="absender" size="30"
          maxlength="40" /> max. 40 zeichen</p>
<p><b>betreff</b>      
   <input type="text" name="betr" size="30"
          maxlength="40" /> max. 40 zeichen</p>
<p><b>text</b>    
   <textarea name="text" cols="70" rows="5"
   maxlength="240"></textarea></p>
<p>die nachricht enthält max. 240 zeichen</p>
<p><input type="submit" name="sender" value="senden" /> </p>
</form>
```

testmail.php

Die länge der nachricht ist auf 240 zeichen, die zeilenlänge auf 70 zeichen begrenzt. Vorsichtshalber werden die zeilen der nachricht mit hilfe der funktion **wordwrap** (siehe 7.1) auf 50 zeichen umgebrochen

```
<?php
     $ende  = chr(13) . chr(10);
     if (isset($_POST["absender"]))            //  absender
          $name = htmlspecialchars($_POST["absender"]);
     else
          $name = "anonym";
     if (isset($_POST["betr"]))                //  betreff
          $betr = htmlspecialchars($_POST["betr"]);
     else
          $betr = "ohne betreff";
     if (isset($_POST["text"]))                //  nachricht
          $text = htmlspecialchars($_POST["text"]);
     else
          $text = "keine nachricht";
     $text = wordwrap($text, 50, " ", true);   //  zeilenlänge
     $adr = "familie.hartard@t-online.de";     //  empfänger
     $abs = "FROM: " . $name;
     echo "<p>$abs<br />$betr<br />$text</p>" . $ende;
     $erg = mail($adr, $betr, $text, $abs);    //  senden
     if ($erg)
          echo "<p>mail an $adr verschickt</p>" . $ende;
     else
          echo "<p>da ist was schief gegangen</p>" . $ende;
?>
```

9.4 funktion header - seite aufrufen

Mit dieser funktion wird der browser veranlaßt, eine andere seite aufzurufen, aber auf eine etwas merkwürdige art: die funktion muß aufgerufen werden bevor auf der aktuellen seite irgendetwas ausgegeben wird, das beispiel in **testheader.php** ist sehr genau zu beachten. Durch die funktion wird eine neue seite aufgerufen, aber das PHP-skript auf der aufrufenden seite wird weiter ausgeführt. Das ist meist nicht sinnvoll, d.h. man sollte nach dem aufruf das skript beenden.

header("Location: ***url***");

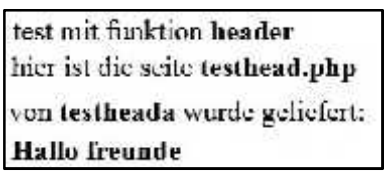
url	adresse der seite, die aufgerufen wird; dafür gibt es folgende möglichkeiten:
seite.php	aufruf einer seite, im gleichen ordner, wie die aufrufende seite
unterordner/seite.php	seite in einem unterordner
../ordner/seite.php	seite in einem übergeordneten ordner .
http://irgendwo.de/home/seite.php");	seite irgendwo im Internet (nicht getestet)

testhead.php
Die seite ruft entweder die seite **testheada.php** auf oder baut eine seite auf und verwendet dabei daten, die von **testheada.php** geliefert werden.

```
<?php
   $ende  = chr(13) . chr(10);
   if (!isset($_GET["param"]))            //  wenn param nicht ges.
   {   header("Location: testheada.php"); //  seite aufrufen
       exit();                            //  script beenden
   }
   else
   {   echo "<html>" . $ende;             //  seite aufbauen
       echo "<body>" . $ende;
       $wert = $_GET["param"];            //  param auswerten
       echo "<p>test mit funktion <b>header</b></p>" . $ende;
       echo "<p>hier ist die seite <b>testhead.php</b></p>" . $ende;
       echo "<p>von <b>testheada</b> wurde geliefert:</p>" . $ende;
       echo "<p><b>$wert</b></p>" . $ende;
       echo "<p><a href='Javascript: history.back();'>"
            . "zurück </a></p>" . $ende;
       echo "</body>" . $ende;
       echo "</html>" . $ende;
   }
?>
```

testheada.php
Die seite ruft wieder die seite **testhead.php** auf und übergibt dabei einen kurzen text. Dass man das ringelspiel mit **abbruch** beenden muss, ist unschön, aber eine bessere lösung wäre etwas unübersichtlich.

```
<?php
   $ende = chr(13) . chr(10);
   $info = "Hallo freunde";
   echo "<p>test der funktion <b>header</b></p>" . $ende;
   echo "<p>hier ist die seite <b>testheada.php</b><br />" . $ende;
   echo "aufgerufen von der seite <b>testhead.php</b></p>" . $ende;
   echo "<p class='font10b'><a href='testhead.php?param=$info'>"
            . "zurück</a> zu <b>testhead.php</b></p>" . $ende;
?>
```



9.5 systemkommando ausführen

Die beiden folgenden funktionen führen ein systemkommando auf dem PC des anwenders aus; sie können auch dazu verwendet werden, ein programm zu starten, Wenn das im hintergrund läuft, müssen die ausgaben des programms in eine datei umgelenkt werden, weil sonst PHP hängt, bis das programm zu ende ist. Die ausgaben von systemkommandos sind meist nicht besonders gut geeignet, um direkt vom browser ausgegeben zu werden, d.h. man leitet sie am besten in eine datei um.

9.5.1 funktion system

Die funktion führt ein systemkommando aus; wenn mit dem kommando ein programm gestartet wird, das im hintergrund abläuft, müssen die ausgaben des programms in eine datei umgelenkt werden, weil sonst PHP hängt, bis das programm zu ende ist.

$zeile = **system** ("*kdo*" [, $wert]);

kdo	kommando, das ausgeführt werden soll, kann auch in einer variablen stehen. Die ausgaben des kommandos sind meist nicht dafür geeignet, direkt im browser ausgegeben zu werden, d.h. man leitet sie am besten in eine datei um (vgl. beispiel).
$zeile	variable mit der letzte zeile der kommando-ausgabe
$wert	variable mit dem rückgabewert des kommandos

9.5.2 funktion exec

Die funktion führt ein systemkommando aus; wenn mit dem komando ein programm gestartet wird, das im hintergrund abläuft, müssen die ausgaben des programms umgelenkt werden, weil sonst PHP hängt, bis das programm zu ende ist.

$zeile = **exec** ("*kdo*" [, $output] [, $wert]);

kdo	kommando, das ausgeführt werden soll, kann auch in einer variablen stehen. Die ausgaben des kommandos sind meist nicht dafür geeignet, direkt im browser ausgegeben zu werden, d.h. man leitet sie am besten in eine variable oder in eine datei um (vgl. beispiel).
$zeile	variable mit der letzte zeile der kommando-ausgabe; wenn die ausgaben umgelenkt werden, wird $zeile nicht versorgt.
$wert	rückgabewert des kommandos
$output	array, in den alle ausgaben des kommandos gespeichert werden. Enthält der array bereits daten sollte man sie vorher mit **unset** löschen. Der inhalt des array eignet sich nicht besonders gut für eine direkte ausgabe im browser.

beispiele
Bei den drei beispielen wird das systemkommando **dir** ausgeführt, mit dem der inhalt eines ordners angezeigt wird. Bei der angabe des ordners ist nicht der schrägstrich sondern backslash zu verwenden. Beim ersten und dritten beispiel werden die ausgaben des kommandos in eine datei umgelenkt, beim zweiten in eine variable. Der inhalt der variable (s.u) ist eher unbrauchbar, während die beiden textdateien ein sehr ordentliches ergebnis zeigen.

```
<?php
    system("dir doku > ergebnis1.txt", $wert);
    echo "<p class='font10'>rückgabe kommando <b>system</b>: \$wert = "
         . "$wert<br />ergebnis in datei ergebnis1.txt</p>" . $ende;
    exec ("dir doku", $ergebnis, $wert);
    echo "<p class='font10'>kommando <b>exec</b> liefert in variable "
         . "\$ergebnis</p>" . $ende;
    print_r($ergebnis);
    echo "<p class='font10'>rückgabe kommando <b>exec</b>: \$wert = "
         . "$wert</p>" . $ende;
    exec ("dir doku > ergebnis2.txt");
    echo "<p class='font10'>kommando <b>exec</b> liefert in datei "
         . "ergebnis2.txt</p>" . $ende;
?>
```

Besonders das ergebnis des ersten exec-kommandos in der variablen **$ergebnis** ist kurios:

Der inhalt der dateien ergebnis1.txt bzw ergebnis2.txt sieht da schon besser aus

```
Datentr„ger in Laufwerk C: ist OS
 Volumeseriennummer: 281F-5AEF

 Verzeichnis von C:\xampp\htdocs\homepage\doku

13.02.2019  18:14    <DIR>          .
13.02.2019  18:14    <DIR>          ..
01.05.2019  14:45    <DIR>          CSS
01.10.2018  17:03             3.000 doku-inc.php
16.03.2017  17:58            15.886 doku-inhalt.docx
29.07.2016  20:28            20.676 homepage-dru.docx
08.05.2017  11:18            44.684 homepage.docx
29.07.2016  20:13            12.716 homepage.psc
15.04.2019  15:39    <DIR>          HTML
02.05.2019  12:32    <DIR>          JAVASCR
15.04.2019  15:39    <DIR>          MYSQL
12.05.2019  17:38    <DIR>          PHP
               5 Datei(en),         96.962 Bytes
               7 Verzeichnis(se), 868.458.815.488 Bytes frei
```

10. muster verwenden

10.1 muster

Bei PHP gibt es die möglichkeit, **muster** in zeichenketten zu erkennen oder zu ersetzen. Ein muster ist eine zeichenkette, die einen **regulären ausdruck** enthält, der in begrenzungszeichen (delimiter) eingeschlossen ist. Ein regulärer ausdruck besteht aus **operatoren** und/oder **literalen**. Als delimiter ist mit ausnahme des backslash ("\") jedes zeichen zulässig, das dann aber im ausdruck nicht vorkommen kann. In den beispielen wird der schrägstrich als delimiter verwendet.

Der einfachste reguläre ausdruck ist ein literal, d.h. ein selbstdefinierender wert. Mit dem folgenden muster wird in einer zeichenkette die teilzeichenkette Test gesucht.

```
$muster = "/Test/";
```

Folgt dem ausdruck der buchstabe "i", wird die groß- kleinschreibung nicht berücksichtigt. Die folgenden muster sind daher gleichwertig.

```
$muster1 = "/ABCD/i";
$muster2 = "/abcd/i";
$muster3 = "/Abcd/i";
```

10.2 funktionen

10.2.1 preg_match - muster vergleichen

[$bool =] **preg_match** ("*muster*", *zeichenkette*);

Die funktion vergleicht eine ***zeichenkette*** mit einem ***muster*** und liefert als ergebnis die werte true oder false. ***muster*** und ***zeichenkette*** können in variablen stehen.

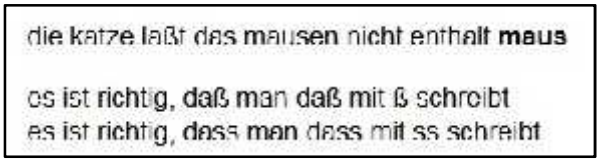
10.2.2 preg_replace - muster ersetzen

[*ergebnis* = **]** **preg_replace** ("*muster*", *ersatz*, *zeichenkette*);

Die funktion ersetzt in einer ***zeichenkette*** alle ***muster*** durch die zeichenkette ***ersatz*** und liefert das ***ergebnis*** in einer zeichenkette. Ist kein ***ergebnis*** angegeben wird die ursprüngliche ***zeichenkette*** geändert. ***muster***, ***ersatz*** und ***zeichenkette*** können in variablen stehen

10.2.3 beispiel

```
<?php
   $zeile = "die katze läßt das mausen nicht";
   $muster = "/maus/";
   if (preg_match($muster, $zeile))
       echo "<p>$zeile enthält <b>maus</b></p>" . $ende;
   $zeile = "es ist richtig, daß man daß mit ß schreibt";
   $muster = "/ß/";
   $erg = preg_replace($muster, "ss", $zeile);
   echo "<p>$zeile<br />$erg</p>";
?>
```



10.3 regulärer ausdruck

Ein regulärer ausdruck besteht aus literalen, d.h. beliebigen zeichenfolgen und/oder operatoren. Durch [ ] werden mehrere zeichen zu einem element zusammengefaßt.

operator	bedeutung	muster	beispiel	treffer
^	die folgenden zeichen / elemente stehen am beginn der zeichenkette	/^K/	Katze die Katze	ja nein
$	die vorangehenden zeichen / elemente stehen am ende einer zeichenkette	/k$/ /k$/i	es ist ok es ist OK es ist OK	ja nein ja
.	beliebiges zeichen	/**../**	AB ABC X	ja ja nein, es müssen mind. 2 zeichen sein
?	vorhergehende zeichen / elemente können vorkommen oder nicht, aber nichts anderes	/Bra?ut/	der zeichenfolge **ut** kann **Bra** oder nichts vorhergehen, aber nichts anderes Braut Brautkranz Braten utriusque	 ja nein **Bra** folgt nicht **ut** ja **ut** geht nichts voran
[ ]	liste optionaler zeichen oder elemente	/M[üö]ller/	**M** folgt **ü** oder **ö** und dann **ller** Müller Möller Mueller Müll	 ja ja nein nein
-	alle zeichen zwischen zwei zeichen oder elementen	/d[a-c]d/	einem **d** folgt **a, b** oder **c** und dann wieder **d** dad dcd ddd	 ja ja nein einem **d** folgt nicht **a**, **b** oder **c**
+	eines oder mehrere der vorhergehenden zeichen oder elemente	/.+m+/	einem oder mehreren beliebigen zeichen folgt ein **m** oder mehrere **m** Dom dumm Don	 ja ja es gibt kein **m**
		/Tor[1-3]+/	**Tor** folgen eine oder mehrere ziffern 1, 2 oder 3 Tor1 Tor32 Tor1213 Tor Tor324	 ja ja ja nein folgt nicht 1, 2, 3 nein 4 nicht zulässig
*	keines, eines oder mehrere der vorhergehenden zeichen oder elemente	/Tor[1-3]*/	**To**r folgt eine, mehrere oder keine ziffer 1, 2 oder 3 Tor1 Tor3211 Tor Tor324	 ja ja ja nein 4 nicht zulässig

operator	bedeutung	muster	beispiel	treffer
( )	zeichenfolge, die genauso vorhanden sein muß	/(ei)/	Leib Weiber Liebe	ja ja nein
(| |)	liste alternativer zeichen, literale oder elemente	/(üll|uell|ill)/	Müller Miller Mueller Mühller Billett	ja ja ja nein ja
\	gültigkeit sonderzeichen aufheben	/3\+2/	das zeichen **+** hat nicht die wirkung eines operators 3+2 32	 ja nein zwischen 3 und 2 fehlt **+**

operator wiederholung

{s, e } dieser operator ist etwas kompliziert, mit **s** kann ein startwert und mit **e** ein endewert angegegeben werden. Damit wird bestimmt, wie oft die vor der geschweiften klammer stehenden zeichen oder elemente wiederholt werden muss bzw. kann. Wichtig ist, dass vor und nach der wiederholung im muster eindeutig definiert ist, was vorhergeht bzw. folgt.

/^A{2,5}$/ mindestens zwei, höchstens fünf **A** in folge, vorher nichts und nacher nichts.
treffer AA, AAA, AAAA, AAAAA
kein treffer A, AAAA**AA**, AAA**XY**, **XY**AAAA nur ein **A,** zu viele **A**, unzulässige zeichen

/^A{2,}$/ kein endewert angegeben, d.h.beliebig viele, aber mindestens zwei **A** in folge, vorher nichts und nachher nichts
treffer AA, AAAAAAAA
kein treffer A. AA**XY**, **XY**AAA nur ein **A**, unzulässige zeichen

/^A{2,5}.*/ am anfang einer zeichenkette mindestens zwei und höchstens fünf **A** dann können beliebig viele beliebige zeichen folgen.
treffer AA, AAAAA, AAAAAAAAA, AAAXYZ
kein treffer **XYZ**AAA am anfang unzulässige zeichen

/A{2,5}.*/ fast wie zuvor, aber jetzt müssen mindestens zwei **A** in folge irgendwo in der zeichenkette stehen, d.h. davor oder danach dürfen beliebige zeichen stehen.
treffer AA, testAAAXY
kein treffer A, mistAXYAZ nur alleinstehende **A**

/^A{1}.*/ nur startwert, zeichenkette muss mit **A** beginnen, dann beliebige zeichen
treffer A, AAA, ABCD
kein treffer **X**AA am anfang kein **A**

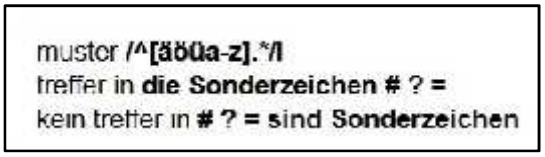
/AA{1}.*/ nur startwert, irgendwo in der zeichenkette müssen wenigstens zwei **A** stehen
treffer AA, AAA, testAABCXY
kein treffer XYZABC nr ein **A** vorhanden

/^A{,5}$/ anfangswert fehlt, da können noch so viele A am anfang, am ende oder
/A{,5}/ irgendwo stehen, beide muster ergeben nie einen treffer, sind also unsinnig.

10.4 beispiele

10.4.1 beispiel 1 - buchstaben, umlaute suchen

```
<?php
   //
   //   "/^[ÄÖÜA-Z].*/"          am anfang großbuchstabe, dann wegen .* beliebig
   //                            viele beliebige zeichen oder nichts
   //   "/^[äöüÄÖÜA-Za-z].*/"    am anfang groß- oder kleinbuchstaben dann
   //                            wegen .* beliebig viele beliebige zeichen
   //                            oder nichts
   //   "/^[äöüa-z].*/i"         wie zuvor, man beachte das "i" nach dem muster
   //   "/^[äöüa-z]*/i"          jetzt fehlt der punkt, d.h. dem ersten buchstaben
   //                            dürfen nur buchstaben folgen.
   //   "/^[äöüa-z ]*$/i"        am anfang und am ende muss ein buchstabe oder
   //                            zwischenraum stehen und dazwischen auch nichts
   //                            anderes.
   //   "/^[äöüa-z ]{1,10}$/i"   wie zuvor, aber höchstens 10 zeichen
   //
   $muster = "/^[äöüa-z].*/i";
   $zeile1 = "die Sonderzeichen # ? =";
   $zeile2 = "# ? = sind Sonderzeichen";
   echo "<p>muster <b>$muster</b><br />" . $ende;
   if (preg_match($muster, $zeile1))
       echo "treffer in <b>$zeile1</b><br />" . $ende;
   else
       echo "kein treffer in <b>$zeile1</b><br />" . $ende;
   if (preg_match($muster, $zeile2))
       echo "treffer in <b>$zeile2</b><br />" . $ende;
   else
       echo "kein treffer in <b>$zeile2</b><br />" . $ende;
   echo "</p>" . $ende;
?>
```



10.4.2 beispiel 2 - postleitzahl suchen

```
<?php
   //
   //   "/^(D-)?[0-9]{5}$/"       deutsche postleitzahl prüfen
   //        ^(D-)?               text beginnt mit "D-" oder, bedingt durch "?",
   //                             gleich mit den ziffern
   //        [0-9]                gefolgt von ziffer 0 bis 9
   //        {5}$                 es müssen 5 ziffern folgen, die ziffernfolge
   //                             steht am ende des textes, d.h. der text enthält
   //                             nur eine postleitzahl
   //
   $muster = "/^(D-)?[0-9]{5}$/";
   $zeile1 = "D-81737";
   $zeile2 = "81737";
   echo "<p>muster <b>$muster</b><br />" . $ende;
   if (preg_match($muster, $zeile1))
       echo "treffer in <b>$zeile1</b><br />" . $ende;
   else
       echo "kein treffer in <b>$zeile1</b><br />" . $ende;
   if (preg_match($muster, $zeile2))
       echo "treffer in <b>$zeile2</b><br />" . $ende;
   else
       echo "kein treffer in <b>$zeile2</b><br />" . $ende;
   $zeile1 = "D 81737";
   $zeile2 = "D-1234";
   if (preg_match($muster, $zeile1))
       echo "treffer in <b>$zeile1</b><br />" . $ende;
   else
       echo "kein treffer in <b>$zeile1</b><br />" . $ende;
   if (preg_match($muster, $zeile2))
       echo "treffer in <b>$zeile2<b><br />" . $ende;
   else
       echo "kein treffer in <b>$zeile2</b><br />" . $ende;
   echo "</p>" . $ende;
?>
```

muster **/^(D-)?[0-9]{5}$/**
treffer in **D-81737**
treffer in **81737**
kein treffer in **D 81737**
kein treffer in **D-1234**

10.4.3 beispiel 3 - email-adresse suchen

```
<?php
   //
   //   email-adresse prüfen
   //   "/^[a-z][a-z_\.-]+@{1}[a-z\._-]+(\.de|\.com)$/i"
   //   ^[a-z]                    am anfang muss ein buchstabe stehen
   //   [a-z_\.-]+                wegen + müssen eines oder mehrere zeichen aus
   //                             der eckigen klammer folgen (buchstabe,
   //                             unterstrich, punkt, bindestrich)
   //   @{1}                      es folgt ein zeichen @
   //   [a-z_\.-]+                dann wieder ein oder mehrere zeichen aus der
   //                             eckigen klammer (die gleichen wie zuvor)
   //   (\.de|\.com)$             am ende muss stehen  .de oder .com
   //   i                         wegen des i nach dem ausdruck sind groß und
   //                             kleinbuchstaben erlaubt.
   //
   $muster = "/^[a-z][a-z_\.-]+@{1}[a-z\._-]+(\.de|\.com)$/i";
   $zeile1 = "info_alle@hans-mayer.muenchen.de";
   $zeile2 = "info_alle@hans-mayer.muenchen.eu";
   echo "<p>muster <b>$muster</b><br />" . $ende;
   if (preg_match($muster, $zeile1))
       echo "treffer in <b>$zeile1</b><br />" . $ende;
   else
       echo "kein treffer in <b>$zeile1</b><br />" . $ende;
   if (preg_match($muster, $zeile2))
       echo "treffer in <b>$zeile2</b><br />" . $ende;
   else
       echo "kein treffer in <b>$zeile2</b><br />" . $ende;
   $zeile1 = "info#alle@hans-mayer.muenchen.de";
   $zeile2 = "info_alle@@hans-mayer.muenchen.com";
   if (preg_match($muster, $zeile1))
       echo "treffer in <b>$zeile1</b><br />" . $ende;
   else
       echo "kein treffer in <b>$zeile1</b><br />" . $ende;
   if (preg_match($muster, $zeile2))
       echo "treffer in <b>$zeile2</b><br />" . $ende;
   else
       echo "kein treffer in <b>$zeile2</b><br />" . $ende;
?>
```

hinweis
Es werden nicht alle zulässigen email-adressen akzeptiert, z.b. werden umlaute und sonderzeichen außer punkt, bindestrich und unterstrich nicht akzeptiert, das muster muss also ggf erweitert werden.

muster **/^[a-z][a-z_\.-]+@{1}[a-z\._-]+(\.de|\.com)$/i**
treffer in **info_alle@hans-mayer.muenchen.de**
kein treffer in **info_alle@hans-mayer.muenchen.eu**
kein treffer in **info#alle@hans-mayer.muenchen.de**
kein treffer in **info_alle@@hans-mayer.muenchen.com**

11. OOP - objekt orientierte programmierung

11.1 grundbegriffe

Die grundkenntnisse über die objekt-orientierte programmierung werden vorausgesetzt, hier wird nur gezeigt, wie man das auch mit PHP machen kann.

11.1.1 klasse definieren

Für eine klasse werden eigenschaften und meist auch methoden definiert.

class *klasse*
{ eigenschaften definieren

[methoden definieren]
}

klasse name einer klasse, eine bezeichnung wie der name einer variablen, aber ohne $-zeichen am anfang

11.1.2 eigenschaft definieren

typ eigenschaft [= *wert*];

eigenschaft name einer variablen, die hier üblicherweise als eigenschaft bezeichnet wird.

wert anfangswert der eigenschaft; der wert wird beim erzeugen eines objekts der klasse zugewiesen. Die möglichkeit wird eher selten verwendet, weil man das meist mit dem **konstruktor** erledigt (vgl. 11.2.1).

typ typ der eigenschaft
- private die eigenschaft ist nur innerhalb der klassendefinition verfügbar
- protected die eigenschaft ist in der klassendefinition und in übergeordneten oder abgeleiteten klassen verfügbar
- public die eigenschaft ist überall verfügbar. Wird nur ausnahmsweise verwendet, weil es eigentlich dem sinn der objekt-orientierten programmierung widerspricht.

11.1.3 methode definieren

Eine methode wird wie eine funktion definiert und enthält anweisungen, mit denen eigenschaften der klasse bearbeitet werden.

[*typ*] *name* (*[$par* [= *wert*] , . . .])
{ **$this->***eigenschaft* = *wert* | *$par*;

$var = **$this->**eigenschaft;
. . .
[return *ergebnis*;]
}

name name der funktion, d.h. name der methode

typ typ der methode
- private die methode ist nur innerhalb der klassendefinition verfügbar
- protected die methode ist in der klassendefinition und in übergeordneten oder abgeleiteten klassen verfügbar
- public die methode ist überall verfügbar, standard, d.h. die angabe darf fehlen

$par [= wert] parameter, für den beim aufruf der methode ein argument übergeben wird; der wert gilt, wenn kein argument übergeben wird. Als argument kann auch eine referenz auf ein objekt (vgl. 11.1.4) übergeben werden.

$this-> bezug auf eine eigenschaft, die in der klasse definiert ist oder in einer anderen klasse definiert und dort als **public** oder **protected** deklariert ist. Der name der eigenschaft, wird **ohne** das $-zeichen angegeben.

11.1.4 objekt einer klasse verwenden

Außerhalb der klassendefinition gibt es verschiedene möglichkeiten, objekte einer klasse zu verwenden.

$*refer* = **new** *klasse*();

Es wird eine instanz der ***klasse*** erzeugt, d.h. es gibt nun ein objekt der *klasse*, dabei ist **$*refer*** keine gewöhnliche variable, sondern eine **referenz** (zeiger) auf das objekt,

$var = *$refer->eigenschaft*;

Der variablen $var wird eine ***eigenschaft*** des objekts zugewiesen, auf das ***$refer*** zeigt. Die eigenschaft muss den typ **public** haben

$refer->eigenschaft = *wert*;

Einer ***eigenschaft*** des objekts, auf das ***$refer*** zeigt, wird ein wert zugewiesen. Die eigenschaft muss den typ **public** oder **protected** haben

$refer->methode(*[wert*, . . .])

Für das objekt, auf das ***$refer*** zeigt, wird die ***methode*** (funktion) ausgeführt. Methoden haben als standard den typ **public**

achtung

In einer methode kann auf eine methode oder eigenschaft einer anderen klasse nur zugegriffen werden, wenn diese als **public und static** deklariert sind (vgl. 11.3)

11.1.5 beispiel

Die klassendefinition **Test** enthält zwei eigenschaften mit dem anfangswert null und zwei methoden. Mit der methode **aendern** werden die beiden eigenschaften geändert. Die methode **anzeige** zeigt die beiden eigenschaften an.

In der PHP-routine werden zwei objekte erzeugt und mit der methode **anzeige** angezeigt. Dann werden mit der methode **aendern** die eigenschaften der objekte geändert und die objekte erneut angezeigt. Dann wird für ein objekt eine eigenschaft geändert, ohne dass dazu die methode **aendern** verwendet wird. Das ist so möglich, weil die eigenschaft den typ **public** hat. Zuletzt wird das objekt angezeigt.

```
<?php
   $ende  = chr(13) . chr(10);                //  zeilenende

//
//   klassendefintion
//
class Test
{   public  $abteil  = 0;
    private $beitrag = 0;

    function aendern($abt, $bei = 10.50)
    {   $this->abteil = $abt;
        $this->beitrag = $bei;
    }

    function anzeige()
    {   global $ende;
        echo "abteilung: $this->abteil - "
             . "beitrag: $this->beitrag<br />" . $ende;
    }
}
```

```
   //
   //   test-routine
   //   zwei objekte erzeugen
   //
   echo "<p><b>OOP - grundbegriffe</b></p>" . $ende;
   $fall1 = new Test();
   $fall2 = new Test();

   //   die beiden objekte anzeigen
   echo "<p>erzeugte objekte<br />" . $ende;
   echo "fall1: ";
   $fall1->anzeige();
   echo "fall2: ";
   $fall2->anzeige();
   echo "</p>";

   //   eigenschaften der objekte ändern
   //   und beide objekte anzeigen
   $fall1->aendern(3, 15.00);
   $fall2->aendern(1);
   echo "<p>geänderte objekte<br />" . $ende;
   echo "fall1: ";
   $fall1->anzeige();
   echo "fall2: ";
   $fall2->anzeige();
   echo "</p>";

   //   eigenschaft eines objekts ändern
   //   und objekt anzeigen
   $fall1->abteil = 2;               //  abteil ist public
   //   $fall2->beitrag = 20.00;     //  beitrag ist private
   echo "<p>eigenschaft von fall1 geändert<br />" . $ende;
   echo "fall1: ";
   $fall1->anzeige();
   echo "</p>";
?>
```

OOP - grundbegriffe

erzeugte objekte
fall1: abteilung: 0 - beitrag: 0
fall2: abteilung: 0 - beitrag: 0

geänderte objekte
fall1: abteilung: 3 beitrag: 15
fall2: abteilung: 1 beitrag: 10.5

eigenschaft von fall1 geändert
fall1: abteilung: 2 - beitrag: 15

11.2 standard-methoden

11.2.1 konstruktor

Der konstruktor ist eine standard-methode, mit der objekte erzeugt werden. Die methode kann individuell für die bedürfnisse einer klasse gestaltet werden, sie dient meist dazu, eigenschaften einen anfangswert zuzuweisen. Wenn ein objekt einer klasse eingerichtet wird (11.1.3), wird der konstruktor ausgeführt. Der konstruktor wird wie jede andere methode definiert, kann daher auch mit parametern versorgt werden, eine return-anweisung ist nicht erforderlich. Der konstruktor hat immer den namen **__construct** , zu beachten sind dabei die beiden unterstriche am anfang. Wenn für eine klasse kein konstruktor definiert ist, erhalten beim einrichten eines objekts die eigenschaften die werte wie sie bei der definition der eigenschaften angegeben sind.

```
function __construct( [ $par [ = wert ] , . . . ] )
{
        this->eigenschaft = wert | $par;
        . . .   weitere anweisungen
}
```

11.2.2 string-ausgabe

Auch hier handelt es sich um eine standard-methode, die bei der ausführung eine textzeile zurückgibt. Die zeile ist frei gestaltbar. Parameter für die übergabe von argumenten gibt es nicht. Die aufgebaute zeile wird mit return zurück-gegeben. Die methode hat immer den namen **__toString**, zu beachten sind dabei die beiden unterstriche am anfang. Die methode ist die einfachste möglichkeit, ein objekt anzuzeigen. Der aufruf der methode ist etwas seltsam, man schreibt einfach den namen des objekts an die stelle, an der eine zeichenkette benötigt wird und erhält dort die zeichenkette, die von der methode erzeugt wird.

```
function __toString( )
{
        $zeile = " . . . ";
        . . .   weitere anweisungen
        return $zeile;
}
```

11.2.3 destruktor

Das ist eine weitere standard-methode, mit der speicherplatz von objekten freigegeben wird. Wenn sie vorhanden ist, wird sie automatisch ausgeführt, wenn PHP zu ende ist, d.h. bevor die seite vom server zum anwender übertragen wird. Es ist auch möglich, den destruktor für ein bestimmtes objekt aufzurufen. Fehlt der destruktor, wird ein standard-destruktor ausgeführt, d.h. der speicherplatz in jedem fall freigegeben. Die methode ist daher nahezu überflüssig. Die methode hat immer den namen **__destruct**, zu beachten sind dabei die beiden unterstriche am anfang.

```
function __destruct()
{
        beliebige anweisungen
}
```

11.2.4 beispiel

Die definition der klasse **Person** enthält eigenschaften und die standard-methoden **construct**, **destruct** und **toString.** Die methoden **ändern** dient wie beim vorigen beispiel (11.1.4) dazu, eigenschaften zu ändern.

In der PHP-routine werden zwei objekte erzeugt; dabei wird automatisch die standard-methode **construct** verwendet. Die objekte werden mit der methode **toString** angezeigt (man beachte den leicht unterschiedlichen aufbau der echo-anweisung, das ergebnis ist aber gleich). Dann werden eigenschaften eines objekts geändert und das objekt erneut angezeigt. Zuletzt wird für das objekt **fall2** mit **destruct** der speicherplatz freigegeben. Am ende wird automatisch mit **destruct** der speicherplatz für alle objekte freigegeben.

```
<?php
   $ende  = chr(13) . chr(10);

//   klassendefinition
class Person
{   private $name;
    private $vorname;
    private $abteil;
    private $beitrag;

    function __construct($nam, $vor)
    {   global $ende;
        $this->name    = $nam;
        $this->vorname = $vor;
        $this->abteil  = 0;
        $this->beitrag = 0;
        echo "konstruktor für $nam $vor<br />" . $ende;
    }

    function __toString()
    {   $wert   = sprintf("beitrag:  % 5.2f", $this->beitrag);
        $zeile  = "mitglied: <b>$this->name</b> $this->vorname - "
                . "abteilung: $this->abteil - $wert";
        return $zeile;
    }

    function __destruct()
    {
        echo "destruktor ";
    }

    function aendern($abt, $bei = 10.50)
    {   $this->abteil  = $abt;
        $this->beitrag = $bei;
    }
}
```

```
   //   objekte erzeugen
   echo "<p>" . $ende;
   $fall1 = new Person("Valentin", "Karl");
   $fall2 = new Person("Karlstadt", "Liesel");
   echo "</p>" . $ende;

   //   objekte mit methode toString anzeigen
   echo "<p>anzeige mit toString<br />" . $ende;
   echo "$fall1 <br />" . $ende;
   echo $fall2 . "<br />" . $ende;

   //   eigenschaften ändern
   $fall1->aendern(3, 15.80);
   echo "objekt fall1 geändert<br />" . $ende;
   echo "$fall1" . "</p>" . $ende;

   echo "<p class='font10'>destruktor für \$fall2<br />" . $ende;
   $fall2->__destruct();     //  destruktor für fall2
   echo "</p>" . $ende;
?>
```

11.3 spezielle elemente

11.3.1 klassen-konstante

Für eine klasse können konstanten (klassen-konstanten) vereinbart werden, die immer den typ **public** haben. Eine klassen-konstante kann überall verwendet werden, sie ist nicht an ein bestimmtes objekt gebunden, d.h. sie ist auch verfügbar, wenn für eine klasse keine objekte erzeugt sind.

const *name* = *wert*;

const	schlüsselwort für die definition einer konstanten
name	name der konstanten; der name darf **nicht** mit dem $-zeichen beginnen.
wert	zugewiesener wert
self::*name*	so wird eine klassen-konstante innerhalb der klassendefinition angesprochen Man beachte den doppelten doppelpunkt.
klasse::*name*	so wird eine klassen-konstante außerhalb der klassendefinition angesprochen

11.3.2 statische eigenschaften

Eine eigenschaften kann als statisch deklariert werden; sie erhält meist bei der definition einen wert, der dann mit methoden der klasse oder, wenn sie den typ **public** oder **protected** hat, auch außerhalb einer methode verändert werden kann. Eine statische eigenschaft steht mit ihrem aktuellen wert immer zur verfügung. Die eigenschaft ist nicht an ein bestimmtes objekt gebunden, d.h. sie ist auch verfügbar, wenn für eine klasse keine objekte erzeugt sind.

[*typ*] **static** *eigenschaft* [= *wert*];

static	schlüsselwort zur definition eines statischen elements
eigenschaft	name der eigenschaft, beginnt mit dem $-zeichen
wert	anfangswert der eigenschaft
typ	typ der eigenschaft, möglich ist private, protected, public (vgl. 11.1.1)
self::eigenschaft	so wird eine statische eigenschaft innerhalb der klassendefinition angesprochen
klasse::*name*	so wird eine statische eigenschaft außerhalb der klassendefinition angesprochen (nur wenn sie den typ **public** oder **protected** hat)

11.3.3 statische methode

Auch eine methode kann als statisch deklariert werden, dazu wird dem namen der methode das schlüsselwort **static** vorangestellt, im übrigen vgl.11.1.2. Eine statische methode ist einer klasse zugeordnet, wird aber nicht für objekte ausgeführt. Sie wird verwendet, um operationen auszuführen, die nicht einem bestimmten objekt gelten, allerdings kann als parameter eine referenz auf ein objekt übergeben werden.

self::methode	aufruf der methode innerhalb der klassen-definition
klasse::methode	aufruf der methode innerhalb und außerhalb der klassen-defintion

11.3.4 beispiel

Die definition der klasse **Person** enthält u.a. die klassen-konstante **verein** und die statische eigenschaft **$nr**. Beide werden im konstruktor und außerhalb der klassen-definition verwendet ($nr ist public). Die eigenschaft $nr erhält den anfangswert 0 und wird im konstruktor beim erzeugen eines objekts um 1 erhöht, d.h. die eigenschaft dient als zähler für die vorhandenen objekte. Außerdem wird $nr verwendet, um jedem objekt eine eindeutige identifikation zu geben.

Bei der verarbeitung werden drei objekte erzeugt und angezeigt; dazu wird die methode **anzeige** verwendet, mit der die anzeige etwas aufwendiger gestaltet wird. Die klassenkonstante und die statische eigenschaft werden auch außerhalb der klassenmethoden verwendet.

```
<?php
   $ende  = chr(13) . chr(10);   //  zeilenende

//  klassendefinition
class Person
{   private $id;
    private $name;
    private $vorname;
    private $abteil;
    private $beitrag;

    public static $nr = 0;        //  statische eigenschaft
    const verein = "FC Mau";
```

```
    function __construct($nam, $vor)
    {   self::$nr = self::$nr + 1;
        $this->id = 1000 + self::$nr;
        $this->name    = $nam;
        $this->vorname = $vor;
        $this->org     = self::verein;
        $this->abteil  = 1;
        $this->beitrag = 10.00;
    }

    function anzeige()
    {   global $ende;

        $wert = sprintf("% 5.2f", $this->beitrag);
        echo "<tr><td>$this->id</td><td>$this->name</td>"
             . "<td>$this->vorname</td>"
             . "<td>$this->org - $this->abteil</td>"
             . "<td>$wert</td></tr>" . $ende;
    }
}

   //  verarbeitung
   $fall1 = new Person("Valentin", "Karl");
   $fall2 = new Person("Karlstadt", "Liesel");
   $fall3 = new Person("Rembremerdeng", "Wrdlbrmft");

   echo "<p>im <b>" . Person::verein . "</b> gibt es "
        . Person::$nr . " mitglieder</p>" . $ende;
   echo "<table style='width: 600px' class='tbohne'>" . $ende;
   echo "<thead>" . $ende;
   echo "<tr><th style='width: 10%;'>nr</th>"
        . "<th style='width: 30%;'>name</th>"
        . "<th style='width: 30%;';>vorname</th>"
        . "<th>abteilung</th>"
        . "<th style='width: 15%;'>beitrag</th></tr>" . $ende;
   echo "</thead>" . $ende;
   echo "tbody>" . $ende;
   $fall1->anzeige();
   $fall2->anzeige();
   $fall3->anzeige();
   echo "</tbody>" . $ende;
   echo "</table>" . $ende;
?>
```

im **FC Mau** gibt es 3 mitglieder

nr	name	vorname	abteilung	beitrag
1001	Valentin	Karl	FC Mau - 1	10.00
1002	Karlstadt	Liesel	FC Mau - 1	10.00
1003	Rembremerdeng	Wrdlbrmft	FC Mau - 1	10.00

11.4 umgang mit objekten

11.4.1 referenzieren

Beim erzeugen eines objekts erhält man in einer variablen eine referenz (zeiger) auf das objekt. Man kann diese variable kopieren, die kopie enthält dann keine kopie des objekts, sondern wiederum nur eine referenz auf das objekt. Gewonnen hat man eine zweite variable, mit der man auf das objekt zugreifen kann.

11.4.2 kopieren

Man kann keine eigene methode bauen, mit der man von einem objekt eine kopie mit einem anderen namen erzeugen kann. Aber es gibt einen notbehelf: man schreibt eine methode, mit der ein neues objekt erzeugt wird und überträgt dann die eigenschaften eines anderen objekts ganz oder teilweise auf das neue objekt. Die methode muss als **static** deklariert werden (vgl beispiel).

11.4.3 klonen

Mit der standard-methode **clone** kann man tatsächlich eine völlig identische kopie, einen echten klon eines objekts erzeugen. Die methode hat den namen **__clone** wird aber einfach mit **clone** aufgerufen und die variable mit der referenz des zu klonenden objekts wird nicht als parameter übergeben (in runden klammern) sondern einfach hinter den aufruf geschrieben (sehr merkwürdig).

$neu = **clone** $alt*;*

$neu name einer variablen der eine referenz auf das geklonte objekt zugewiesen wird.

$*alt* name einer variablen, die eine referenz auf ein vorhandenes objekt enthält

Wenn die klassen-definition die methode **__clone** enthält, wird beim klonen diese methode ausgeführt und man kann dabei an dem geklonten objekt noch änderungen vornehmen (vgl. beispiel). Wenn die methode nicht in der klassen-definition definiert ist, wird trotzdem geklont.

11.4.4 beispiel

Die definition der klasse **Person** enthält eine klassen-konstante und die statische eigenschaft, die verwendet werden, wie bei 11.3.4. Wichtig ist, dass beim erzeugen eines objekts die statische variable **nr** um 1 erhöht wird und davon die ID des objekts abgeleitet. wird.. Auch die funktionen **construct** und **aendern** zeigen nichts neues. Der statischen methode **kopieren** werden als parameter eine variable mit einer referenz und zwei zeichenketten übergeben. Mit den zeichenketten wird ein neues objekt erzeugt, das dann zum teil eigenschaft des objekts erhält, auf die die referenz verweist. Da das objekt mit dem konstruktor erzeugt wid, wird auch hier die statische variable **nr** erhöht und damit die ID des objekts gebildet. Mit der standard-methode **clone** wird ein objekt geklont. Da dabei das neue objekt nicht mit hilfe des konstruktors erzeugt wird, muss in der methode **clone** die statische variable **nr** erhöht und damit die ID des neuen objekts gebildet werden. Die eigenschaft **name** des neuen objekts wird geändert. Zu beachten ist, dass die anzeige der objekte vollständig in die klassen-definition verlegt ist. Dazu wird der methode **ausgabe** die adresse des feldes **$alle** übergeben, in dem alle referenzen gespeichert sind. Die methode entnimmt der reihe nach die referenzen dem feld und ruft damit die methode **anzeige** auf. Natürlich hätte man die anweisungen der methode **anzeige** auch in die methode **ausgabe** stellen können, aber es soll hier ja ein wenig mit den möglichkeiten gespielt werden.

```
<?php
   $ende  = chr(13) . chr(10);   //  zeilenende

//   klassendefinition
class Person
{   private $id;
    private $name;
    private $vorname;
    private $abteil;
    private $beitrag;

    public static $nr = 0;        //  statische variable
    const verein = "FC Mau";      //  klassen konstante
```

```
    function __construct($nam, $vor)
    {   self::$nr   = self::$nr + 1;
        $this->id   = 1000 + self::$nr;
        $this->name = $nam;
        $this->vorname = $vor;
        $this->org     = self::verein;
    }
    static function kopieren($orig, $nam, $vorn)
    {   $neu = new Person($nam, $vorn);
        $neu->name    = $orig->name . "-" . $nam;
        $neu->abteil  = $orig->abteil;
        $neu->beitrag = $orig->beitrag;
        return $neu;
    }
    function __clone()
    {   self::$nr = self::$nr + 1;
        $this->id = 1000 + self::$nr;
        $this->name = $this->name . "-geklont";
    }
    function aendern($abt, $bei = 10.50)
    {   $this->abteil  = $abt;
        $this->beitrag = $bei;
    }

    function anzeige()
    {   global $ende;
        $wert = sprintf("% 5.2f", $this->beitrag);
        echo "<tr><td>$this->id</td><td>$this->name</td>"
             . "<td>$this->vorname</td>"
             . "<td>$this->org - $this->abteil</td>"
             . "<td>$wert</td></tr>" . $ende;
    }

    static function ausgabe($liste)
    {  global $ende;
       echo "<p>im " . self::verein . " gibt es "
            . self::$nr . " mitglieder</p>" . $ende;
       echo "<table style='width: 600px' class='tbohne'>" . $ende;
       echo "<thead>" . $ende;
       echo "<tr><th style='width: 10%;'>nr</th>"
            . "<th style='width: 30%;'>name</th>"
            . "<th style='width: 30%;'>vorname</th>"
            . "<th>abteilung</th>"
            . "<th style='width: 15%;'>beitrag</th></tr>" . $ende;
       echo "</thead>" . $ende;
       echo "<tbody>" . $ende;
       $anz = self::$nr;
       for ($lauf=1; $lauf<=$anz; $lauf++)
       {   $fall = $liste[$lauf];
           $fall->anzeige();
       }
       echo "</tbody>" . $ende;
       echo "</table>" . $ende;
    }
}
```

Bei der verarbeitung werden zunächst drei objekte erzeugt und ihre referenzen im feld **$alle** gespeichert. Bemerkenswert ist die indizierung mit hilfe der statischen variablen **$alle[Person::$nr]** die variable enthält nach dem erzeugen eines objekts die laufende nummer des objekts. Mit der methode **ändern** werden an den eigenschaften der objekte änderungen vorgenommen. Schließlich wird mit der methode **kopieren** das objekt **$fall2** in das neue objekt **$fall4** kopiert und dort etwas abgeändert. Zuletzt wird das objekt **$fall3** als **$fall5** geklont.

```
   //  verarbeitung
   $alle  = array(0, 0, 0);       //  feld für referenzen
   $fall1 = new Person("Valentin", "Karl");
   $alle[Person::$nr] = $fall1;
   $fall2 = new Person("Karlstadt", "Liesel");
   $alle[Person::$nr] = $fall2;
   $fall3 = new Person("Rembremerdeng", "Wrdlbrmft");
   $alle[Person::$nr] = $fall3;
   echo "<p><b>objekte original</b></p>" . $ende;
   Person::ausgabe($alle);

   //   eigenschaften ändern
   $fall1->aendern(3, 15.80);
   $fall2->aendern(2, 20.00);
   $fall3->aendern(1);
   echo "<p><b>objekte geändert</b></p>" . $ende;
   Person::ausgabe($alle);

   $fall4 = Person::kopieren($fall2, "Quatsch", "Elvira");
   $alle[Person::$nr] = $fall4;

   $fall5 = clone $fall3;
   $alle[Person::$nr] = $fall5;
    echo "<p><b>objekte nach kopieren und "
         . "klonen</b></p>" . $ende;
   Person::ausgabe($alle);
?>
```

objekte original

im FC Mau gibt es 3 mitglieder

nr	name	vorname	abteilung	beitrag
1001	Valentin	Karl	FC Mau	0.00
1002	Karlstadt	Liesel	FC Mau	0.00
1003	Rembremerdeng	Wrdlbrmft	FC Mau	0.00

objekte geändert

im FC Mau gibt es 3 mitglieder

nr	name	vorname	abteilung	beitrag
1001	Valentin	Karl	FC Mau - 3	15.80
1002	Karlstadt	Liesel	FC Mau - 2	20.00
1003	Rembremerdeng	Wrdlbrmft	FC Mau - 1	10.50

objekte nach kopieren und klonen

im FC Mau gibt es 5 mitglieder

nr	name	vorname	abteilung	beitrag
1001	Valentin	Karl	FC Mau - 3	15.80
1002	Karlstadt	Liesel	FC Mau - 2	20.00
1003	Rembremerdeng	Wrdlbrmft	FC Mau - 1	10.50
1004	Karlstadt-Quatsch	Elvira	FC Mau - 2	20.00
1005	Rembremerdeng-geklont	Wrdlbrmft	FC Mau - 1	10.50

11.5 objekt-felder

Unter dem schlagwort kann man verschiedene dinge verstehen, beispielsweise ein feld von referenzen auf objekte. Das ist einfach und wurde im vorhergehenden beispiel (11.4.4) schon vorgeführt. Es geht auch etwas anders, etwa eine eigenschaft, die nicht einen wert, sondern ein feld von werten darstellt. Das sieht dann so aus:

In der klasse **Artikel** gibt es die statische eigenschaft **$namtab**, die aus einem feld von zeichenketten besteht, außerdem die eigenschaft **$artnam** mit dem typ **public**. Beim erzeugen eines objekts mit dem konstruktor wird der parameter **$nr** übergeben. Der konstruktor entnimmt damit aus **$namtab** eine zeichenkette und versorgt damit die eigenschaft **$artnam**. Diese eigenschaft des objekts wird dann angezeigt. Das ist alles.

```
<?php
class Artikel
{   public $artnam;
    static $namtab = array("unbekannt", "spaten", "rechen",
                           "harke", "eimer", "becher" );

    function __construct($nr)
    {
        $this->artnam = self::$namtab[$nr];
    }
}

$fall = new Artikel(3);
echo "<p class='font10'>gewählt wurde: $fall->artnam</p>";
?>
```

Komplizierter wird die sache, wenn eine eigenschaft ein feld von objekten ist und zwar von objekten einer anderen klasse, wie in dem nächsten beispiel.

beispiel

Die klasse **Person** hat die eigenschaft **$bestell**, der man überhaupt nicht ansieht, dass sich dahinter ein feld von objekten der klasse **Bestellen** verbirgt. Beim erzeugen eines objekts wird dem konstruktor eine referenz auf ein solches feld übergeben (s.u), der konstruktor versorgt damit die eigenschaft **$bestell**, d.h. diese eigenschaft enthält dann eine referenz auf ein feld von referenzen, die auf objekte der klasse **Bestellen** verweisen. Die klasse **Person** enthält ferner die methode **zusatz**, mit der ein objekt der klasse **Bestellen** erzeugt wird. Mit der referenz auf dieses objekt wird das feld **$bestell** erweitert. Es gibt ferner noch die methode **anzeigen**, mit der ein objekt angezeigt wird. Dazu wird dann noch die methode **zeigen** der klasse **Bestellen** benötigt. Das ist natürlich unnötig kompliziert, aber es soll ja gezeigt werden, was man so alles machen kann.

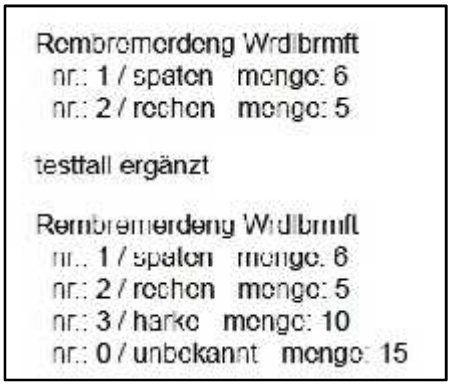
Die klasse **Bestellen** wiederholt das spiel des vorigen beispiels (eigenschaft als feld von zeichenketten), außerdem gibt es noch die anzahl der artikel.

```php
<?php
   $ende  = chr(13) . chr(10);
//   klasse Person
class Person
{   private $name;
    private $vorname;
    private $bestell;            //  feld von objekten

    function __construct($nam, $vor, $best)
    {   $this->name    = $nam;
        $this->vorname = $vor;
        $this->bestell = $best;  //  feld von objekten
    }

    function zusatz($nr, $anz)
    {   $erg = new Bestellen($nr, $anz);
        $x = count($this->bestell);   //  weiteres objekt der
        $this->bestell[$x] = $erg;    //  klasse Bestellen einf.
    }

    function anzeigen()
    {   global $ende;
        echo "<p >$this->name $this->vorname<br />" . $ende;
        $zahl = count($this->bestell);     //  anzahl objekte
        for ($x=0; $x<$zahl; $x++)         //  der klasse Bestellen
        {   $fall = $this->bestell[$x];
            $fall->zeigen();
        }
    }
}

//   klasse Bestellen
class Bestellen
{   private $artnr;
    private $artnam;
    private $menge;
    static $namtab = array("unbekannt", "spaten", "rechen",
                           "harke", "eimer", "becher" );

    function __construct($nr, $anz)
    {   if (($nr < 0) || ($nr > 5))
            $nr = 0;
        $this->artnr = $nr;
        $this->menge = $anz;
        $this->artnam = self::$namtab[$nr];
    }

    function zeigen()
    {   global $ende;
        echo "- nr.: $this->artnr  / $this->artnam - "
             . "menge: $this->menge<br />" . $ende;
    }
}
```

Ausführen des beispiels

Es wird das objekt **Testfall** der klasse **Person** erzeugt, dem konstruktor wird übergeben: name, vorname und ein feld mit zwei objekten der klasse **Bestellen**. Das objekt wird angezeigt. Dann werden mit der methode **zusatz** für das objekt Testfall zwei weitere objekte der klasse Bestellen erzeugt. Die unsinnige angabe -3 als artikelnummer soll nur verdeutlichen, dass man bei klassendefinitionen auch mit fehlerhaften angaben rechnen sollte.

```
   //   objekt erzeugen
   $testfall = new Person("Rembremerdeng", "Wrdlbrmft",
               array(new Bestellen(1,6),
                     new Bestellen(2, 5)));
   //   objekt  anzeigen
   $testfall->anzeigen();
   //   bestellung erweitern
   $testfall->zusatz(3, 10);
   $testfall->zusatz(-3, 15);
   echo "<p>testfall ergänzt</p>" . $ende;
   $testfall->anzeigen();
?>
```



11.6 vererbung

Es ist möglich, methoden und eigenschaften von einer klasse an eine andere klasse zu vererben, d.h. weiter zu geben. Die vererbende klasse bezeichnet man dann als **basisklasse**, die klasse, die erbt ist eine **abgeleitete** klasse.

class *Basis*
{ eigenschaften
methoden
}

class *Erbe* **extends** *Basis*
{ eigenschaften
methoden
}

zugriff zu eigenschaften
In der klasse ***Erbe*** können eigenschaften der klasse ***Basis*** verwendet werden, allerdings müssen sie dort den typ **protected** haben. Umgekehrt gilt das gleiche. Natürlich könnte man in beiden fällen die eigenschaften auch als **public** deklarieren, aber ohne zwingende notwendigkeit macht man eigenschaften nicht überall verfügbar.

aufruf von methoden
In der klasse ***Erbe*** können methoden der klasse ***Basis*** wie folgt aufgerufen werden:

Basis::*methode*([*parameter*])

parent::*methode*([*parameter*])

Die zweite art des aufrufs wird empfohlen. In der klasse **Basis** können keine methoden der klasse **Erbe** aufgerufen werden. Von außerhalb der klassen können die methoden beider klassen wie gewohnt aufgerufen werden.

Erzeugen von objekten
Soll ein objekt eigenschaften beider klassen enthalten, muß man es in der klasse **Erbe** mit dem konstruktor erzeugen und dann den konstruktor der klasse **Basis** aufrufen (vgl. beispiel). Soll ein objekt nur eigenschaften einer der beiden klassen enthalten, erzeugt man es wie gewohnt.

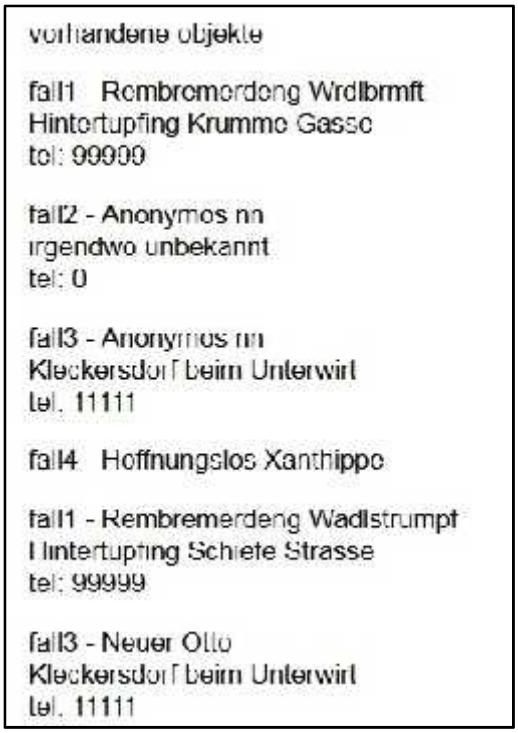
beispiel

Es sind zwei klassen definiert, die basisklasse **Person** mit den eigenschaften einer person und die abgeleitete klasse **Ansch** mit der anschrift. Es werden drei objekte der klasse **Ansch** erzeugt. Dabei sollten eigentlich werte für alle eigenschaften übergeben werden (fall1). Werden gar keine werte (fall2) oder nur die werte für die anschrift (fall3) übergeben, werden default-werte verwendet. Der konstruktor von **Ansch** ruft den konstruktor von **Person** auf, um die eigenschaften name und vorname zu versorgen. Dann werden die drei objekte mit der methode **zeigen** (klasse **Ansch**) angezeigt. Die methode ruft die methode **anzeigen** (klasse **Person**) auf. Mit fall4 wird ein objekt der klasse **Person** erzeugt, das nur mit der methode **anzeigen** der klasse **Person** angezeigt werden kann. Bei dem objekt fall1 werden mit der methode **aendern** (klasse **Ansch**) eigenschaften geändert, dabei wird auch die methode **change** (klasse **Person**) aufgerufen, um ggf. die eigenschaft name oder vorname zu ändern. Bei fall3 werden die eigenschaften name und vorname geändert, dazu wird nur die methode **change** der klasse **Person** aufgerufen. Die geänderten objekte werden angezeigt, jetzt aber mit der methode **extra** der klasse **Ansch**. Diese methode benötigt nicht die methode **anzeigen** (klasse **Person**). Die eigenschaft vorname, die den typ **protected** hat, wird direkt angesprochen und die eigenschaft name wird mit der methode **holnam** (klasse **Person**) geholt. Der methode **extra** wird nämlich als parameter eine referenz übergeben, die an die methode **holnam** weitergereicht wird, die dann das gewünschte ergebnis liefert, ein wenig umständlich, aber es soll gezeigt werden, was man so alles machen kann.

hinweis
Versuchsweise wurde folgende lösung konstruiert: wenn beim erzeugen eines objekts der klasse **Ansch** kein name übergeben wird, werden die eigenschaften name und vorname nicht erzeugt und können dann später mit einer eigenen methode erzeugt werden. Es hat funktioniert, aber die lösung war maßlos verwickelt und wird deshalb hier nicht gezeigt.

```
<?php
   $ende  = chr(13) . chr(10);

//   basisklasse Person
class Person
{   private $name;
    protected $vorname;

    function __construct($nam, $vor)
    {   $this->name    = $nam;
        $this->vorname = $vor;
    }

    function anzeigen($par)
    {   global $ende;
        echo "$par - $this->name $this->vorname" . $ende;
    }

    function change($nam, $vorn)
    {   if ($nam != "no")
            $this->name = $nam;
        if ($vorn != "no")
            $this->vorname = $vorn;
    }

    function holnam($param)
    {
        return $param->name;
    }
}
```

```
//   klasse Ansch von Person abgeleitet
class Ansch extends Person
{   private $ort;
    private $strasse;
    private $telefon;

    function __construct($o="irgendwo", $s="unbekannt",
                         $t=0, $nam="Anonymos", $vor="nn")
    {   $this->ort = $o;
        $this->strasse = $s;
        $this->telefon = $t;
        parent::__construct($nam, $vor);
    }

    function zeigen($par)
    {   global $ende;
        echo parent::anzeigen($par) . "<br />"
             . "$this->ort $this->strasse<br />"
             . "tel: $this->telefon</p>" . $ende;
    }

    function extra($par, $fall)
    {   global $ende;
        $nam = parent::holnam ($fall);
        echo "<p>$par - $nam $this->vorname <br />"
             . "$this->ort $this->strasse<br />"
             . "tel: $this->telefon</p>" . $ende;

    }

    function aendern($ort="no", $str="no", $tel="no",
                     $nam="no", $vorn="no")
    {   if ($ort != "no")
            $this->ort = $ort;
        if ($str != "no")
            $this->strasse = $str;
        if ($tel != "no")
           $this->telefon = $tel;
        parent::change($nam, $vorn);
    }
}

//   objekte erzeugen
$fall1 = new Ansch("Hintertupfing", "Krumme Gasse",
                   "99999", "Rembremerdeng", "Wrdlbrmft");
$fall2 = new Ansch();
$fall3 = new Ansch("Kleckersdorf", "beim Unterwirt",
                   "11111");

echo "<p>vorhandene objekte</p>" . $ende;
$fall1->zeigen("fall1");
$fall2->zeigen("fall2");
$fall3->zeigen("fall3");
$fall4 = new Person("Hoffnungslos", "Xanthippe");
$fall4->anzeigen("fall4");

$fall1->aendern("no", "Schiefe Strasse", "no", "no",
                "Wadlstrumpf");
$fall1->extra("fall1", $fall1);
$fall3->change("Neuer", "Otto");
$fall3->extra("fall3", $fall3);
?>
```



11.7 serialisierung / deserialisierung

Ein objekt serialisieren bedeutet, es mit seinen eigenschaften in eine zeichenkette zu schreiben. Das gegenstück dazu ist die deserialisierung, bei der mit hilfe einer zeichenkette, die ein serialisiertes objekt enthält, wieder ein objekt erzeugt und eine referenz auf das objekt in einer variablen gespeichert wird. Diese referenz muß nicht die gleiche sein, wie beim serialisieren.

In der regel schreibt man die zeichenketten mit **fputs** in eine textdatei, um objekte zu sichern und liest sie mit **fgets** von dort, um objekte wieder zu erzeugen. Vor dem schreiben muß man an die zeichenketten das zeilenende-zeichen (chr(13) . chr(10)) anfügen (vgl. 8.5, 8.6). Da die zeichenketten sehr unterschiedliche länge haben können, sollte man zudem beim schreiben die maximale länge ermitteln.

$zeile = **serialize**(*$referenz*);

$referenz = **unserialize**(*$zeile*);

beispiel

Es werden zwei objekte der klasse **Person** erzeugt, angezeigt und serialisiert. Die zeichenketten werden zu zwei objekten deserialisiert und die objekte angezeigt.

```
<?php
   $ende  = chr(13) . chr(10);

//   klassendefinition
class Person
{   private $name;
    private $vorname;
    private $abteil;
    private $beitrag;
```

```
    function __construct($nam, $vor)
    {   global $ende;
        $this->name    = $nam;
        $this->vorname = $vor;
        $this->abteil  = 0;
        $this->beitrag = 0;
    }

    function aendern($abt, $bei = 10.50)
    {   $this->abteil  = $abt;
        $this->beitrag = $bei;
    }

    function __toString()
    {   return "$this->name $this->vorname abteilung: "
                . "$this->abteil beitrag: $this->beitrag";
    }
}

//   PHP-routine
echo "<p><b>serialisieren</b></p>" . $ende;
$fall1 = new Person("Valentin", "Karl");
$fall2 = new Person("Karlstadt", "Liesel");
$fall1->aendern(3, 15.80);
$fall2->aendern(2, 20.00);
echo "$fall1 <br />" . $ende;
echo "$fall2 <br />" . $ende;
$zeile1 = serialize($fall1);
$zeile2 = serialize($fall2);
echo "<p><b>deserialisieren</b></p>" . $ende;
$testa = unserialize($zeile1);
$testb = unserialize($zeile2);
echo "$testa <br />" . $ende;
echo "$testb <br />" . $ende;
?>
```

12. anhang

12.1 wordwrap - zeichenkette zerlegen

Die funktion zerlegt eine zeichenkette nach einer wählbaren anzahl von zeichen mit einem zeichen oder einer zeichenkette. Durch eine option kann bestimmt werden, dass worte, die länger sind als die angegebene zeichenlänge zerlegt werden.

$textneu = wordwrap($text, $width, $break, $cut);

$textneu	zeichenkette mit dem ergebnis der funktion
$text	zeichenkette, die bearbeitet wird
$width	anzahl der zeichen, nach denen die zeichenkette zerlegt wird.
$break	zeichen oder zeichenkette; damit wird die zeichenkette zerlegt.
$cut	steuert die zerlegung
	true worte, die länger sind als **$width**, werden durch **$break** zerlegt. Folgen worte aufeinander, die zusammen länger als **$width** sind, wird vor dem folgewort zerlegt.
	false es werden keine worte zerlegt..

achtung
Die funktion hat eine schwäche, wenn die zeichenkette im **Unicode** codiert ist und 2-byte-zeichen enthält (geschlossene umlaut, €, §, °, ², ³, µ siehe 8.9) wertet die funktion diese zeichen als zwei zeichen und zerreißt ggf. ein solches zeichen zu zwei nicht darstellbaren zeichen. Im Internet wurde die funktion **wordwrap_utf8** gefunden, die mit diesen speziellen zeichen kein problem hat, ziemlich stur mit der angegeben zeichenzahl zerlegt, aber dabei nicht immer sauber zählt.

beispiel

Es wird nach maximal 10 zeichen mit der zeichenkette **
** zerlegt.

```
<?php
$text = "12345678901234567890 xx yy zz aaaaaaa 12äöüxx890aaaaÄÖÜaaa";
$texta = wordwrap($text, 10, "<br >", true);
$textb = wordwrap_utf8($text, 10, "<br />", true);
?>
```

Man beachte, wie von **wordwrap** das letzte wort zerlegt wird: der erste teil enthält scheinbar nur 7 zeichen, in wirklichkeit sind es aber 10, den die zeichen **äöü** sind jeweils 2-byte-zeichen. Weil die zeichen **ÄÖÜ** ebenfalls 2-byte-zeichen sind , liegt die trennstelle mitten in dem zeichen **Ö**, das in zwei schmierzeichen zerrissen wird.

ergebnis worderap

```
1234567890
1234567890
xx yy zz
aaaaaaa
12äöüxx
890aaaaÄ
Üaaa
```

ergebnis wordwrap_utf8

```
1234567890
1234567890
xx yy zz
aaaaaaa 12
äöüxx890aa
aaÄÖÜaaa
```

funktion wordwrap_utf8

Die folgende funktion wurde im Internet auf der seite **miliaw.de** von Milian Wolff gefunden.

```
<?php
function wordwrap_utf8($str, $width, $break, $cut = false)
{    if (!$cut)
     {    $regexp = '#^(?:[\x00-\x7F]|[\xC0-\xFF][\x80-\xBF]+){'.$width.',}\b#U';
     }
     else
     {    $regexp = '#^(?:[\x00-\x7F]|[\xC0-\xFF][\x80-\xBF]+){'.$width.'}#';
     }
     $str_len = preg_match_all('/[\x00-\x7F\xC0-\xFD]/', $str, $var_empty);
     $while_what = ceil($str_len / $width);
     $i = 1;
     $return = '';
     while ($i < $while_what)
     {    $i++;
          preg_match($regexp, $str, $matches);
          if(isset($matches[0]))
          {    $string = $matches[0];
               $return .= $string.$break;
               $str = substr($str, strlen($string));
          }
     }
     return $return.$str;
}
```