- Allgemein -

PHP ist eine Scriptsprache zur Erstellung dynamischer Websites. Die erste Version entwickelte 1994 Rasmus Lerdorf, der eigentlich nur eine Möglichkeit zur Programmierung seines eigenen Webservers suchte. Er nannte seine kleine Skriptmaschine "Personal Homepage Tools". Die Applikation stellte er ins Internet und ließ die freie Verbreitung zu. So entstand PHP, als Abkürzung zu "**P**ersonal **H**ome **P**age". Später entwickelten die Open Source-Jünger, bekannt für kryptische Abkürzungen, die rekursive Version "**P**HP **H**yperText **P**reprocessor".

Den Weg zum professionellen Programm ging PHP erst 1995. Unter dem Namen PHP / FI (Form Interface) wurde die erste wirklich nutzbare Version veröffentlicht. PHP / FI wird als Version 2 angesehen. Seit dieser Zeit entwickelte sich PHP rasant weiter. Zum einen trug dazu die freie Verfügbarkeit bei. Die ist aber bei ASP (Active Server Pages) und Perl auch gegeben. PHP ist im Gegensatz dazu aber eine Sprache, darren einziger Zweck das Web ist. ASP kommt mit VBScript daher, einem BASIC Derivat, das nur schwer an die Bedürfnisse eines Webservers angepaßt werden kann. Der größte Nachteil von ASP ist aber die Einschränkung auf Microsoft-Plattformen. Perl hingegen ist zwar plattformübergreifend, wurde aber schon 1986 entwickelt. Zu dieser Zeit war das Internet noch nicht bekannt. Aus der Betrachtung der Nachteile entstand daher eine Skriptsprache, die alle Vorteile der Konkurrenz in sich vereint - PHP.

PHP besticht außerdem durch einen fast schon grandiosen Funktionsumfang. Hier kann keine andere Sprache mithalten. Besonders deutlich wird dies bei den Datenbankfunktionen. PHP unterstützt mehr Datenbanken direkt, als die meisten Programmierer überhaupt vom Namen her kennen. Den Erfolg macht nicht zuletzt die gute Unterstützung für die Datenbank MySQL aus, die wie PHP auch als Open Source verfügbar ist.

- Grundlagen -

Sehr praktisch an PHP ist, dass es in HTML-Dateien eingebettet werden kann. Die Datei läuft dann durch den PHP-Interpreter und wird dann zum Betrachter geschickt.

Hier die Möglichkeiten PHP in eine Datei zu schreiben:

```
<script language=php>
.. code ..
</script>
```

oder

```
<?php
.. code ..
?>
```

meist jedoch nutzt man folgendes:

```
<?
.. code ..
?>
```

Damit eine PHP-Datei auch ausgeführt wird, muss sie eine bestimmte Dateiendung haben. Folgende sind möglich: **.php .php3 .php4 .phtml**

Aber auch einfache .html Dateien können, falls der Server richtig konfiguriert ist, vom Interpreter verarbeitet werden.

Hier eine Beispieldatei (*Beispiel 1*):

```
<html>
<body>
<?
echo "Hallo Welt!";
?>
</body>
</html>
```

Im Browser betrachtet erscheint dann der Satz: *Hallo Welt!* Und wenn man sich jetzt den Quelltext im Browser anschaut sieht man nur die HTML- und Bodytags, sowie den Satz Hallo Welt! Der Eigentliche PHP Code wird im Quelltext nicht angezeigt. Befehle werden in PHP generell mit einem Semikolon "**;**" abgeschlossen, nur wenn der Befehl alleine zwischen <? und ?> steht, ist dies optional.

Bestimmte Befehle sollen nur ausgeführt werden, wenn bestimmte Bedingungen zutreffen. Ein Beispiel wäre, dass beim Eintragen eines neuen Eintrags in einen Newsletter erst überprüft werden soll, ob überhaupt alle Felder ausgefüllt wurden.

```
<?
if ( strlen($Name) < 1) echo "Sie haben keinen Namen
angegeben";
if ( strlen($Email) < 1) echo "Sie haben keine email-Adresse
angegeben";
else echo "Alles okay!";
?>
```

IF bedeutet übersetzt "wenn" - und den Sinn hat diese Funktion auch, denn wenn die Bedingung wahr ist, dann sollen die folgenden Aktionen ausgeführt werden.

Dabei ist es möglich eine auszuführende Aktion wie oben direkt dahinter zu schreiben, oder aber wenn es mehrere Aktionen sind, sie in **Klammern** zu schreiben, wie bei Schleifen:

```
<?
if ( strlen($Name) < 1)
{
echo "Sie haben keinen Namen angegeben<br>";
echo "und das finde ich nicht nett!";
...
}
?>
```

Und natürlich kann man wie man Schleifen ineinander verschachteln kann auch IF-Abfragen ineinander verschachteln.

Die Funktion **strlen()** liefert die Anzahl der Zeichen innerhalb einer Variable.

So liefert strlen("hallo"); die Zahl 5. Die IF Schleife ist also erfüllt, wenn der Inhalt der Variable $Name aus weniger als einem Zeichen besteht.

Mit **echo** wird Text an den Browser übergeben. Der Browser führt diesen Text direkt aus.

```
<?
echo "Hallo<br>";
?>
```

Hier wird also der Text Hallo vom Browser ausgegeben. HTML Befehle wie hier das **
** werden vom Browser interpretiert und umgesetzt.

Der PHP Befehl **\n** hingegen erzeugt lediglich im HTML-Quelltext einen Zeilenumbruch, der vom Browser nicht umgesetzt wird. Er dient lediglich dazu, den Quelltext besser lesbar zu machen.

```
<?
echo "Hallo\n";
?>
```

Natürlich können damit auch komplette HTML-Blöcke an den Browser übergeben werden:

```
<?
echo <<<Tabelle
<table>
  <tr>
    <th>Name</th>
    <th>Adresse</th>
  </tr>
</table>
Tabelle;
```

Der Block wird stets durch **<<<** eingeleitet.

Hier einige Vergleichsmöglichkeiten zweier Variablen

$var1	**==**	**$var2**	wenn beide den selben Inhalt haben
$var1	**!=**	**$var2**	wenn sie unterschiedlichen Inhalt haben
$var1	**<**	**$var2**	wenn var1 kleiner als var2 ist
$var1	**<=**	**$var2**	hier wenn sie kleine oder gleich var2 ist
$var1	**>**	**$var2**	wenn var1 größer als var 2 ist
$var1	**>=**	**$var2**	hier wenn var1 größer oder gleich var2 ist

Ein möglicher Vergleich wäre dann:

```
$string = "hallo";

echo "Der String ist " . strlen($string) . " Zeichen lang<br>";

if (strlen($string > 4)
{
echo "auf jeden Fall is er länger als 4 Zeichen";
}
else echo "Er ist 4 oder weniger Zeichen lang!";
```

Das **else** führt alle Aktionen aus, die ausgeführt werden sollen, wenn obige Bedingung nicht zutrifft, und diese Aktionen können auch wieder in Klammern geschrieben werden.

Eine weitere interessante Funktion von PHP ist das einschließen von Dateien. Nehmen wir einmal an, wir hätten ein größeres Script, in dem ein bestimmter Teil immer wieder vorkommt. So könnt man diesen Teil in eine separate Datei schreiben und dann einfach an der gewünschten Stelle nur noch darauf verweisen:

```
<html>
<body>
<?
include("../hallo.php4");
?>
</body>
</html>
```

In diesem Script wird z.B. die Datei hallo.php4 aufgerufen, die ein Verzeichnis höher liegt. Diese könnte dann wie folgt aussehen:

```
<?
Echo "Hallo";
?>
```

Dies hätte dann das gleiche Ergebnis wie die Beispieldatei *Beispiel 1* oben.

Es macht übrigens keinen Unterschied, wie man die einzelnen Befehle anordnet.

```
<HTML>
<HEAD>
<TITLE><? echo $title ?></TITLE>
...
```

Dies entspricht dem nachfolgenden Script. Wobei das folgende Beispiel natürlich übersichtlicher ist.

```
<HTML>
<HEAD>
<TITLE>
<?
   echo $title /* hier wird eine Variable verwendet */
?>
</TITLE>
...
```

Leerzeichen, Zeilenumbrüche und Tabulatoren spielen in PHP keine Rolle. Mann kann die Befehle also beliebig einrücken um die Übersichtlichkeit zu steigern.

Kommentare werden mit **/*** eingeleitet und mit ***/** beendet. Sie können sich auch über mehrere Zeilen erstrecken. Um lediglich einen Kommentar am Ende einer Zeile einzugeben genügt es, den Kommentar mit **//** zu beginnen

```
<HTML>
<HEAD>
<TITLE>
<?
   echo $title // hier wird eine Variable verwendet
?>
</TITLE>
...
```

— Einfache Rechenoperationen —

Folgende Rechenoperationen stehen in PHP zur Verfügung:

"+"	Addition	$N+$M
"-"	Subtraktion	$N-$M
"*"	Multiplikation	$N*$M
"/"	Division	$N/$M
"%"	Reste-Bildung	$N%$M 23%17 ergibt 6 weil 23/17 ergibt 1 Rest 6
"."	Verknüpfung	$N = "langer"; $M = "kurzer"; echo $N.$M ergibt "langerkurzer"

Dazu kommen noch ein paar Abkürzungen, um dem Programmierer das Leben zu erleichtern:

$N++	erhöht $N um 1
++$N	erhöht $N ebenfalls um 1
$N--	erniedrigt $N um 1
--$N	erniedrigt $N ebenfalls um 1

Der Unterschied zwischen $N++ und ++$N ist:

```
<?
$N=0;
echo $N++;
?>
```

Gibt "0" aus, anschließend wird $N auf den Wert "1" erhöht.

```
<?
$N=0;
echo ++$N;
?>
```

Erhöht zuerst $N auf "1" und gibt den Wert "1" aus.

Um komplexe logische Ausdrücke zu erstellen benötigt man auch Vergleichsoperatoren. Die folgende Tabelle gibt einen Überblick:

==	Gleichheit
===	Identität
!=	Ungleichheit
>	Größer als
<	Kleiner als
>=	Größer als oder gleich
<=	Kleiner als oder gleich

Neu in PHP 4 ist der Identitätsoperator ===. Damit wird nicht nur der Inhalt einer Variablen auf Gleichheit geprüft, sondern auch der Datentyp. Außerdem sollte man nie den Unterschied zwischen = und == verwechseln.

```
if ($var =17)
{
echo $var;
}
```

Hier war sicher gemeint, dass die Variable ausgegeben werden soll, wenn der Inhalt gleich 17 ist. Praktisch passiert aber folgendes. Die Variable wird auf den Wert 17 gesetzt - unabhängig von ihrem bisherigen Inhalt. Der Wert 17 wird dann ausgegeben.

Gleichheit wird immer mit == geprüft.

Man kann die einzelnen Vergleich auch kombinieren:

```
($i==10) && ($j>=) --> $i gleich 10 und $j
größer als 0
($i==10) || ($j==0) --> $i gleich 10 oder
$j gleich 0
```

Zufallszahlen werden häufig benötigt, um Vorgänge zu steuern oder Kennwörter zu erzeugen. Mit dem Befehl mt_rand([$min],[$max]) werden Zufallszahlen erzeugt.

```
$var = mt_rand(5,10);
```

Gibt eine Zufallszahl zwischen 5 und 10 aus.

Natürlich gibt es in PHP auch logische Operatoren.

$x and $y	UND Verknüpfung
$x or $y	ODER Verknüpfung
$x xor $y	NOR Verknüpfung (Wahr, wenn beide gleich)
$x && $y	ebenfalls UND Verknüpfung
$x || $y	ebenfalls ODER Verknüpfung
!$x	Negierung (aus FALSE wird TRUE)

– Variablen –

In jedem Programm kommt man früher oder später in die Verlegenheit, Werte (Daten) speichern zu müssen. Für die Speicherung während der Ausführung eines Scripts stehen Variablen zur Verfügung. Variablen beginnen immer mit $

Zum testen des Scripts **hier** klicken

```
<?
$text = "Ich bin ein String !";
echo $text;
echo "<br>";
echo $text,$text,$text;
echo "<br>";
$L = "langer";
$K = "kurzer";
echo "Ich bin ein $L$L$L$L$L Text !";
echo "<br>";
echo "Ich bin ein $K Text !";
echo "<br>";

$I = 10;
$J = 5;
echo $I,"+",$J,"=",$I+$J;
?>
```

Der Typ der Variablen (ganze Zahl, Gleitpunktzahl, String) wird ja nach Verwendung von PHP automatisch bestimmt. Der Benutzer braucht sich darum nur in Spezialfällen kümmern.

Variablen sind in PHP allerdings nur in dem Kontext gültig, in dem sie definiert wurden. Wenn man eine Variable Außerhalb eines Blocks definiert, ist sie innerhalb eines Blocks nicht verfügbar.

```
<?
$zahl = 22;
function ausgabe ()
  {
    echo $zahl;
  }
ausgabe()
?>
```

Das obige Beispiel wird nicht wie erwartet funktionieren, da die Variable Zahl innerhalb des Blocks nicht verfügbar ist. Der Block wird durch die beiden geschweiften Klammern {} begrenzt.

Der Zugriff auf globale Variablen ist dennoch mit dem Schlüsselwort **global** möglich.

```
<?
$zahl = 22;
function ausgabe ()
  {
    global $zahl;
    echo $zahl;
  }
ausgabe()
?>
```

Mit dem Schlüsselwort global teilt man PHP mit, dass die Variable in einem globalen Kontext bereits existiert. Der Inhalt der Variablen wird dann übernommen.

Da PHP speziell zur Erzeugung von dynamischen Webseiten geschaffen wurde, ist es ein Kinderspiel, Eingaben, die aus HTML-Formularen stammen, zu bearbeiten.

Zum testen des Scripts **hier** klicken

```
<HTML>
<HEAD>
<Title>Ein einfaches Formular</Title>
</HEAD>
<BODY>

<FORM ACTION="Bsp_variablen3.php4" METHOD=POST>
<INPUT NAME="beliebigerName">
<INPUT TYPE="submit">
</FORM>

</BODY>
</HTML>
```

In obiger HTML Datei wird ein Formular erzeugt, in dem beim klicken auf den Button die Datei *Bsp_variablen3.php4* aufgerufen wird. Diese Datei könnte dann wie folgt aussehen:

```
<?
echo "Sie haben <b>$beliebigerName</b> eingegeben";
?>
```

Der Text, der im Formularfeld **beliebigerName** eingegeben wurde, ist in PHP automatisch in der Variablen **$beliebigerName** verfügbar.

– Schleifen –

Schleifen werden benötigt, um Programmteile mehrfach zu durchlaufen. Neben der Einsparung an Tipparbeit ist vor allem die variable Festlegung der Schleifendurchläufe interessant. Schleifen ohne feste Laufvariable werden durch Bedingungen gesteuert. Der Zustand des logischen Ausdrucks bestimmt, ob die Schleife weiter durchlaufen wird oder nicht.

Die häufigste Schleifenart ist die WHILE-Schleife, die in fast jeder Programmiersprache zu finden ist. Die Bedingung wird mit jedem Eintritt in die Schleife getestet. Solange der Ausdruck TRUE zurückgibt, wird die Schleife durchlaufen. Wenn der ausdruck also schon beim Eintritt in die Schleife FALS ergibt, wird die Schleife überhaupt nicht durchlaufen.

Zum testen des Scripts **hier** klicken

```
<?
$counter = 0;
$test = 6;
while ($test > $counter)
{
echo "Aktueller Zähler: $counter<br>";
$counter++;
}
?>
```

Die Schleife wird solange durchlaufen, bis $counter den Wert "6" aufweist.

Der Test der Bedingung am Anfang der WHILE-Schleife hat einen wesentlichen Nachteil. Es ist Möglich, dass die Bedingung so wirkt, dass der Inhalt nie durchlaufen wird. Die ist vor allem schlecht, wenn der Inhalt zur weiteren Verarbeitung benötigt wird. Die Lösung dieses Problems ist die DO...WHILE-Schleife.

```
<?
$counter = 0;
$test = 6;
do {
    echo "Aktueller Zähler: $counter<br>";
    $counter++;
} while ($Counter <= $test)
?>
```

Der Einzige Unterschied ist die Reihenfolge der Abarbeitung. Hier wird zuerst die Schleife einmal durchlaufen und am Ende die Abbruchbedingung gesestet. Auch dann, wenn die Abbruchbedingung bereits beim Schleifeneintritt FALS ist, wird der Block mindestens einmal durchlaufen.

Schleifen lassen sich auch vor dem Ende abbrechen.

Zum testen des Scripts **hier** klicken

```
<?
$counter = 0;
$test = 100;
while ($test > $counter)
{
echo "Aktueller Zähler: $counter<br>";
$counter++;
if ($counter == 50) break;
}
echo "<br>Schleife abgebrochen";
?>
```

Hier wird die Schleife mit dem **break**-Befehl verlassen, sobald $counter den Wert "50" erreicht hat.

Die vorangegangenen Beispiele dienten vor allem der Erläuterung der Befehle, die feste Vorgabe von unteren und oberen Grenzen ist eigentlich keine typische Anwendung von WHILE-Schleifen. In solchen Fällen werden besser FOR-Schleifen eingesetzt. Die Abbruchbedingung ist allerdings auch hier ein normaler logischer Ausdruck. Zusätzlich wird eine numerische Variable mitgeführt - die Zählvariable.

Die FOR-Schleife ist komplexer als die bisher vorgestellten Schleifen:
FOR(START, BEDINGUNG, ITERATION);
Mit START wird die Schleife initialisiert, d.h. normalerweise wird die Variable, die die Schleifendurchläufe zählt, auf den Anfangswert gesetzt.
BEDINGUNG gibt die Abbruchbedingung an.
In ITERATION wird die Variable, die die Schleifendurchläufe zählt, erhöht bzw. erniedrigt.

Zum testen des Scripts **hier** klicken

```
<?
for($i=0; $i <= 10; $i++) {
echo "i ist jetzt: $i<br>";
}
?>
```

Dies ist die einfachste Form der Anwendung. Die Schleife zählt einfach von 0 bis 10. Die Variable in den drei Elementen der FOR-Schleife muss nicht durchgehend verwendet werden. Dies ist zwar im Hinblick auf die Lesbarkeit der Skripte zu empfehlen, notwendig ist es jedoch nicht, wie das folgende Beispiel zeigt:

Zum testen des Scripts **hier** klicken

```
<?
$j = 0;
for($i=0; $i <= 10; $j++) {
echo "$i. j ist jetzt: $j<br>";
$i++;
}
?>
```

Raffinierte FOR-Schleife: Abbruchbedingung und Iteration nutzen unterschiedliche Variablen.

Alle drei Parameter der FOR-Schleife sind optional. Ohne Iterationsvariable wird die Schleife endlos durchlaufen. In diesem Fall kann die Schleife wieder wie oben schon gesehen mit BREAK verlassen werden.

In PHP 4 steht eine neue Form der FOR-Schleife zur Verfügung: **FOREACH**. Damit wird ein Array durchlaufen.

```
FOREACH(array as ememente) {
...
}
```

Dabei wird das Array *array* von Anfang bis Ende durchlaufen und bei jedem Schleifendurchlauf wird das aktuelle Element der Variablen **array element** zugewiesen.

Auf Arrays wird in einem späteren Kapitel näher eingegangen.

– Fallunterscheidung –

Die Syntax des Befehls IF lehnt sich an die Programmiersprache C an. Entsprechend knapp fällt die Schreibweise aus:

```
<?
if ($tag == "Montag") echo "Heute ist Montag";
?>
```

Der IF-Befehl besteht aus dem Schlüsselwort und dem in runden Klammern stehenden logischen Ausdruck. Wenn der Ausdruck Wahr ist, wird der nachfolgende Befehl oder Block (in geschweiften Klammern) ausgeführt. Ist der Ausdruck Falsch, wird die Programmausführung mit dem nächsten Befehl fortgesetzt.

Zum testen des Scripts **hier** klicken

```
<?
$tag = date("l");
if ($tag =="Monday") echo "Heute ist Montag";
if ($tag =="Tuesday") echo "Heute ist Dienstag";
if ($tag =="Wednesday") echo "Heute ist Mittwoch";
if ($tag =="Thursday") echo "Heute ist Donnerstag";
if ($tag =="Friday") echo "Heute ist Freitag";
if ($tag =="Saturday") echo "Heute ist Samstag";
if ($tag =="Sunday") echo "Heute ist Sonntag";
?>
```

Oft ist es notwendig, nicht nur auf das Eintreten eines Ereignisses zu reagieren, sondern auch die negative Entscheidung zu behandeln. In solchen Fällen wird der IF-Befehl um ELSE ergänzt. Der Befehl oder Block hinter ELSE wird ausgeführt, wenn die Bedingung nicht zutrifft.

Zum testen des Scripts **hier** klicken

```
<?
$tag = date("w");
if ($tag == 0 or $tag == 7) {
echo "Wochenende !!";
} else {
echo "Arbeitszeit !!";
}
?>
```

Wenn man ein Script wie das obige mit den Wochentagen aufbaut, welches mehrere aufeinanderfolgende Bedingungen gegen ein und dieselbe Variable testet, ist der IF-Befehl sehr augwendig. Mit SWITCH steht ein Befehl zur Verfügung, der solche Listen eleganter aufbaut.

Zum testen des Scripts **hier** klicken

```
<?
$tag = date("w");
switch($tag)
{
case Saturday:
echo "Samstag";
break;

case Sunday:
echo "Sonntag";
break;

default:
echo "unter der Woche";
}
?>
```

Wichtig in diesem Zusammenhang ist der BREAK-Befehl, der die Arbeitsweise von SWITCH wesentlich beeinflusst. Natürlich kann der BREAK-Befehl auch weggelassen werden. Jedoch werden dann immer alle Befehle durchlaufen, was bei sehr großen Scripten nicht vorteilhaft ist.

– Datum- und Zeitfunktionen –

Für die Arbeit mit Daten ist oft das aktuelle Datum von großer Bedeutung. Man sollte aber beachten, dass der Server sich immer auf sein eigenes Systemdatum bezieht. Wenn man seine Seiten z.B. in den USA hosten lässt, wird natürlich grundsätzlich zuerst einmal die USA Zeit ausgegeben.

Der Befehl **DATE** formatiert ein Datum für die Ausgabe.

Zum testen des Scripts **hier** klicken

```
<?
echo date ("l dS of F Y");
?>
```

Innerhalb der Formatierungsanweisung sind folgende Symbole von Bedeutung. Alle anderen Zeichen (wie oben z.B. of) werden ignoriert und unverändert ausgegeben.

a	"am" oder "pm"
A	"AM" oder "PM"
d	Tag des Monats mit 2 Stellen und führender Null: "01" bis "31"
D	Tag der Woche als Abkürzung mit drei Buchstaben: "Fri"
F	Monat, ausgeschrieben: "January"
h	Stunde im 12-Stunden-Format: "01" bis "12"
H	Stunde im 24-Stunden-Format: "00" bis "23"
g	Stunde im 12-Stunden-Format ohne führende Null: "1" bis "12"
G	Stunde im 24-Stunden-Format ohne führende Null: "0" bis "23"
i	Minuten: "00" bis "59"
j	Tag des Monats ohne führende Null: "1" bis "31"
l	(kleine "L") Wochentag, voll ausgeschrieben: "Friday"
L	Boolescher Wert, der bestimmt, ob das Datum in einem Schaltjahr liegt: "0" oder "1"
m	Monat mit führender Null: "01" bis "12"
n	Monat ohne führende Null: "1" bis "12"
M	Monat als Abkürzung: "Jan"
s	Sekunden mit führender Null: "00" bis "59"
S	Das Suffix der englischen Ordnungszahlen: "th", "nd" usw.
t	Anzahl der Tage in einem Monat: "28" bis "31"
U	Sekunden seit Beginn der UNIX-Expoche (01.01.1970)
w	Numerische Darstellung des Wochentags: "0" (Sonntag) bis "6" (Samstag)
Y	Vierstellige Ausgabe des Jahres: "1999" oder "2000"
y	Zweistellige Ausgabe des Jahres: "99" oder "00"
z	Tag im Jahr: "00" bis "365"
Z	Differenz zur Zeitzone in Sekunden "-43200" bis "43200" Entspricht -12 bis +12 Std.

Interessant ist auch der Befehl MKTIME. Damit lässt sich der Zeitstempel für ein bestimmtes Datum ermitteln:

Zum testen des Scripts **hier** klicken

```
<?
echo "Der 1. Juli 2000 ist ein ";
echo date("l", mktime(0,0,0,7,1,2000));
?>
```

Hier wird z.B. ermittelt, was für ein Wochentag am 1.7.2000 war.

Um einen Zeitstempel für ein bestimmtes Datum zu erhalten verwendet man folgenden Befehl:
$variable = mktime(stunde, minute, sekunde, monat, tag, jahr, sz)
Der Parameter *sz* ist optional. Wird er auf "1" gesetzt, nimmt die Funktion an, dass sich das Datum in der Sommerzeit befindet. "0" entspricht der Winterzeit. Default ist "-1".

Noch interessanter dürfte der Befehl **STRFTIME** sein. Der große Vorteil hier ist die Tatsache, dass man die Ausgabe mit **SETLOCALE** an die sprachlichen Besonderheiten eines Landes anpassen kann.

Zum testen des Scripts **hier** klicken

```
<?
echo strftime("Englisches Format: %A %x");
echo "<br>";
setlocale ("LC_TIME", "de_DE");
echo strftime("Deutsches Format: %A %x");
?>
```

Die Handhabung des STRFTIME-Befehls ist zunächst identisch mit dem DATE-Befehl. Lediglich die Parameter unterscheiden sich:

%a	Abkürzung des Wochentags: "Mon"
%A	Voller Name des Wochentags: "Monday"
%d	Abkürzung des Monats: "Jan"
%B	Monat, ausgeschrieben: "January"
%c	Vollständige Datums- und Zeitangabe
%d	Tag des Monats mit 2 Stellen und führender Null: "01" bis "31"
%H	Stunde im 24-Stunden-Format: "00" bis "23"
%I	Stunde im 12-Stunden-Format: "01" bis "12"
%j	Tag im Jahr: "001" bis "366"
%m	Monat ohne führende Null: "1" bis "12"
%M	Minuten als Ziffer: "0" bis "59"
%p	"am" oder "pm"
%S	Sekunden als Ziffer: "0" bis "59"
%U	Wochennummer im Jahr, startet die Zählung am ersten Sonntag
%W	Wochennummer im Jahr, startet die Zählung am ersten Montag
%w	Numerische Darstellung des Wochentags: "0" (Sonntag) bis "6" (Samstag)
%x	Vollständige Datumsangabe
%X	Vollständige Zeitangabe
%y	Zeistellige Ausgabe des Jahres: "99" oder "00"
%Y	Vierstellige Ausgabe des Jahres: "1999" oder "2000"
%Z	Differenz zur Zeitzone in Sekunden "-43200" bis "43200" Entspricht -12 bis +12 Std.

Der Befehl SETLOCALE ist wie folgt aufgebaut:
SETLOCALE(category, localID)

Der Parameter *category* wird durch folgende Werte bestimmt:

"LC_ALL"	Wirkt sich auf alle weiteren Angaben aus.
"LC_COLLATE"	Wirkt auf Zeichenkettenvergleiche (ab PHP 4)
"LC_CTYPE"	Wirkt auf die Zeichensetzung
"LCMONETARY"	Wirkt auf Währungsfunktionen
"LC_NUMERIC"	Bestimmt das Dezimaltrennzeichen (ab PHP 4)
"_LC_TIME"	Wirkt auf Datums- und Zeitfunktionen

Der Parameter *localeID* setzt das Land und die Sprache. Für Deutschland wäre es dann "de_DE". Für die Schweiz gibt es zwei Varianten: "de_CH" und "fr_CH".

– Dateien lesen und schreiben –

Dateien lesen und schreiben gehört zu den elementaren Vorgängen bei der PHP Programmierung. Zum lesen und schreiben werden folgende Befehle verwendet:

READFILE
Die Funktion READFILE liest eine Datei und sendet den gesamten Inhalt ohne weitere Bearbeitung an die Standardausgabe - dies ist im Normalfall der Browser. Ohne weitere Bearbeitung heiß auch, dass die einzelnen Zeilen einfach hintereinander angehängt werden.
readfile("Dateiname");

Zum testen des Scripts **hier** klicken

```
<?
readfile("Test.txt");
?>
```

FILE
Analog funktioniert auch FILE, die gelesene Datei wird aber in einem Array abgelegt.
$filearray = file("Dateiname");

FOPEN
Hiermit wird eine Datei geöffnet.
$datei = fopen("Dateiname","attribut");
Das Attribus gibt an, wie die Datei geöffnet wird. Folgende Attribute stehen zur Verfügung:

r	nur lesen, begonnen wird am Dateianfang
r+	lesen und schreiben, begonnen wird am Dateianfang
w	nur schreiben, Existiert die Datei, wird ihr Inhalt gelöscht. Existiert sie nicht, wird versucht sie zu erzeugen

w+	lesen und schreiben, ansonsten wie "w"
a	nur schreiben, begonnen wird am Ende der Datei. Existiert sie nicht, wird sie erstellt
a+	lesen und schreiben, ansonsten wie "a"

Alle Attribute können mit "b" kombiniert werden (z.B. "br","br+") um den Zugriff auf Binärdateien zu ermöglichen.

FCLOSE
Geöffnete Dateien müssen wieder geschlossen werden, um Systemressourcen zu schonen und anderen Prozessen den Zugriff zu ermöglichen.
fclose($datei);

FGETS
Die Funktion FGETS liest aus einer Datei; die Funktion FPUTS schreibt in die Datei. FGETS liest von der Position des Dateizeigers, bis eines der drei folgenden Ereignisse eintritt:

- Die Anzahl der Bytes (**Länge**), die angegeben wurde, ist erreicht
- Ein Zeilenende wurde erreicht
- Das Dateiende wurde erreicht

```
$var = fgets($datei, 4096);
```

Ein konkretes Beispiel könnte so aussehen:

Zum testen des Scripts **hier** klicken

```
<?
$datei = fopen("Test.txt","r");
while (!feof($datei)) {
  $zeile = fgets($datei,1000);
  echo $zeile;
  echo "<br>";
}
fclose($datei);
?>
```

- In der 2. Zeile wird festgelegt, welche Datei wie geöffnet werden soll. Soll später auf die Datei zugegriffen werden, so benutzt man die Variable **$datei**.
- **feof($datei)** ist wahr, sobald das Ende der Datei erreicht wird. *eof* (End Of File) wird negiert (mit dem "!") Schleife läuft, solange EOF nicht erreicht ist.
  $zeile = fgets($datei,1000); liest maximal die nächsten 1000 Zeichen, hört aber auf, sobald eine neue Zeile beginnt, oder das Ende der Zeile erreicht ist.
- Schließlich muss die Datei mit **fclose** geschlossen werden

Mit FWRITE wird in eine Datei geschrieben

```
fwrite($datei, "ein Text");
```

Hier ein konkretes Beispiel, welches das aktuelle Datum, die Uhrzeit sowie die IP-Adresse des Clients in eine Datei schreibt und den Inhalt der Datei anschließend ausgibt.

Zum testen des Scripts **hier** klicken

```
<?
$fp = fopen("test/logfile.log","a");
fputs($fp, date("d.M.Y h:m:s",time()).",
IP:".$REMOTE_ADDR."<br>\n");
fclose($fp);
?>

<P>
Das ist der Inhalt der Log-Datei:<br><br><br>
</P>

<?
echo "Es sind mitlerweile ";
echo count(file("test/logfile.log"));
echo " Einträge. So oft wurde die Datei schon
aufgerufen.<br><br>";
readfile("test/logfile.log");
?
```

Der Befehl **count** zählt aus wie vielen Zeilen die Datei besteht.

Etwas problematisch ist das gezielte Löschen einer einzelnen Zeile einer Datei. Nur mit der Kombination mehrerer Befehle ist dies möglich. Das folgende Beispiel zeigt die prinzipielle Vorgehensweise:

```
<?
$datei = "Test.txt";                  # Name der Datei
$line = 17;                           # zu löschende Zeile
$myfile = file($datei);               # einlesen in ein Array
unset $myfile[$line];                 # löschen des Elements
$fh = fopen($datei, "w");             # überschreibend öffnen
fputs($f, implode("\n", $myfile));    # Array in String
fclose($fh);                          # Datei schließen
?>
```

– Zugriff auf Datenbanken –

Eines der Haupteinsatzgebiete von PHP ist der Online-Zugriff auf Datenbanken. Eine Datenbank enthält Datensätze, die mit einer speziellen Sprache bearbeitet werden können. Zum Beispiel können bestimmte Datensätze ausgelesen werden, neue Datensätze hinzugefügt werden, Datensätze können aktualisiert oder gelöscht werden. All diese Vorgänge nennt man *Abfrage.*

Weit verbreitet sind sogenannte SQL-Datenbanken, d.h. Datenbanken, die mit der Sprache SQL (Structured Query Language) bearbeitet werden können. PHP kann mit Datenbanken verschiedener Hersteller umgehen, besonders beliebt ist die Datenbank MySQL, da sie relativ schnell und zudem kostenlos ist.

Um eine SQL-Abfrage mit PHP auszuführen, muss zuerst die Datenbank geöffnet werden (vergleichbar mit dem Zugriff auf eine Datei aus dem vorherigen Kapitel), dann wird die SQL-Befehlszeile an die Datenbank geschickt, die Antwort der Datenbank wird aufgenommen und schließlich wird die Datenbank wieder geschlossen.

SQL-Grundlagen

SQL-Datenbanken bestehen aus einer oder mehreren sogenannten Tabellen. Jeder Datensatz der Datenbank ist in genau einer Zeile der Tabelle gespeichert.

Tabelle: **Kneipen**

ID	Name	Art	Note	Kommentar
1	Herzogkeller	Biergarten	1	Schöner Baumbestand
2	Glenk	Biergarten	3	Gute Bratwürste
3	...	...	...	...

Man kann nun auf diese Tabelle zugreifen. Hierfür gibt es im wesentlichen vier Abfragemöglichkeiten:

- Auslesen: SELECT
- Einfügen: INSERT
- Überschreiben: UPDATE
- Löschen: DELETE

Will man z.B. alle Namen auslesen, gibt man den SQL-Befehl **SELECT Name FROM Kneipen;**
Mit **SELECT Name,Art FROM Kneipen;** wird der Name und die Art ausgelesen
Mit **SELCET * FROM Kneipen;** wird schließlich die ganze Zeile ausgelesen

Die auszulesenden Datensätze können noch weiter spezifiziert werden. **SELECT * FROM Kneipen WHERE Note=1;** gibt nur die Datensätze aus, die in der Spalte "*Note*" eine "*1*" enthalten. Sortieren kann man die Datensätze natürlich auch. **SELECT * FROM Kneipen WHERE Note=1 ORDER BY Name;** sortiert die Ausgabe nach der Spalte "*Name*".

Neue Datensätze werden folgendermaßen hinzugefügt:
INSERT Kneipen (Name,Art,Note,Kommentar) VALUES ('Glenk',*Biergarten','1','Schöner Baumbestand');

Vorhandene Datensätze können mit Update überschrieben werden:
UPDATE kneipen SET Note='2',Kommentar='Die Bratwürste lassen nach' WHERE ID=2;

SQL-Befehle mit PHP

Die obigen SQL-Befehle sollen nun mit PHP verwendet werden. Als Beispiel sollen alle Datensätze der Tabelle "*Kneipen*" aus der Datenbank "*Datenbank1*" ausgegeben werden.

Zuerst muss eine Verbindung von PHP zum Datenbank-Server hergestellt werden:

```
$verbindung = @mysql_connect("127.0.0.1","nobody","");
```

"**127.0.0.1**" gibt die Adresse des SQL-Servers an. "**nobody**" ist ein gültiger Benutzername für die Datenbank. **""** bedeutet, dieser Benutzer benötigt kein Passwort.

Man kann überprüfen, ob die Verbindung funktioniert:

```
if (!$verbindung) {
    echo "Keine Verbindung möglich !\n";
    exit;
}
```

Steht die Verbindung, kann man die Abfrage starten. Zur besseren Übersicht sollte man die SQL-Abfrage in eine Variable schreiben:

```
$abfrage = "SELECT Name,Art FROM Kneipen";
```

Mit dem folgenden Befehl wird nun die Abfrage auf den Datenbank-Server geschickt. Das Ergebnis wird in der Variablen **$erg** gespeichert.

```
$erg =
mysql_db_query("Datenbank1",$abfrage,$verbindung);
```

Die erste Variable in der Klammer gibt die Datenbank an, die man abfragen will. Die zweite Variable enthält den SQL-Befehl, die dritte Variable enthält die Verbindung, die mit *mysql_connect* geschaffen wurde.

Damit wäre die Datenbank-Abfrage eigentlich beendet. Nun müssen aber die von der Datenbank gelieferten Datensätze noch ausgelesen werden. Der obige Befehl hat die beiden Spalten *Name* und *Art* angefordert. Dies werden nun Zeilenweise aus der Variablen *$erg* ausgelesen:

```
list($Name,$Art) = mysql_fetch_row($erg);
```

Die Variable *$erg* enthält nach der Abfrage Zeilen mit jeweils 2 Spalten (*Name* und *Art*). **mysql_fetch_row($erg)** gibt genau eine Zeile aus. Zu Beginn steht eine Art Zeiger auf der ersten Zeile, führt man den Befehl **mysql_fetch_row($erg)** aus, rutscht der Zeiger zur nächsten Zeile usw. Will man alle Zeilen ausgeben, empfiehlt sich eine Schleife:

```
while (list($Name,$Art) = mysql_fetch_row($erg)) {
    echo "$Name ist ein(e) $Art<br>\n";
}
```

Die Ergebnisse werden also zeilenweise zu HTML-Text verarbeitet.

Am Ende empfiehlt es sich, die Verbindung zur Datenbank zu schließen:

```
mysql_close($verbindung);
```

Natürlich kann man auch einzelnen Einträge (Zeilen) aus der Tabelle einer SQL-Datenbank löschen. Im obigen Beispiel sollte man den zu löschenden Datensatz über das Feld **ID** angeben, denn dieses Feld enthält für jeden Datensatz eine eindeutige Nummer. Will man nun z.B. den Datensatz *ID=120* löschen, sieht der Befehl wie folgt aus:

```
$abfrage = "DELETE FROM Kneipen WHERE ID=120";
```

Achtung: Vergisst man den WHERE-Teil, werden **alle** Datensätze der Tabelle gelöscht !!

Dies war natürlich nur eine sehr kleiner Teil der Möglichkeiten mit SQL-Datenbanken, der allerdings die grundlegenden Funktionen erklären dürfte.

– Funktionen –

Ein wichtiger Bestandteil eines jeden Programmiersprache ist die Fähigkeit, mehrere Befehle zu einem einzigen Befehl zusammenzufassen, d.h. neue Funktionen zu definieren. PHP hat diese Fähigkeit selbstverständlich auch.

Folgendes Beispiel berechnet aus wie vielen Sekunden eine Anzahl von Tagen besteht:

Zum testen des Scripts **hier** klicken

```
<?
function tag ($anzahl) {
    $sekunden = 3600 * 24 * $anzahl;
    return $sekunden;
}

$x = tag(7);
Echo "7 Tage bestehen aus $x Sekunden !";
?>
```

$anzahl nennt man auch Argument der Funktion. Eine Funktion kann mehrere Argumente, oder auch kein Argument besitzen.

Oben im Beispiel wird mit **function** die Funktion definiert. Sie kann nun beliebig oft im Script mit einem beliebigen Wert (z.B. **tag(7)**) aufgerufen werden.

– .htaccess –

Oftmals tut sich für einen Webmaster die Frage auf: Wie kann ich einzelne Dateien oder Ordner vor unerwünschtem Zugriff schützen? Nun, es gibt mehrere Möglichkeiten, wie z.B. CGI-Scripte. Aber eine einfachere und sicherere Methode ist das Verwenden von **.htaccess-Dateien**. Um mit .htaccess Web-Seiten zu schützen muss auf dem Webserver "*Apache*" laufen (viel verbreiteter Webserver).

Um ein Verzeichnis zu schützen benötigt man 2 Dateien. Zunächst benötigt man eine Datei namens **.htaccess**, die man in das Verzeichnis kopiert, dass vor fremdem Zugriff geschützt werden soll. Außerdem benötigt man eine Datei, in der sich die Benutzernamen und Passwörter (in verschlüsseltem Zustand) der User befinden. In unserem Beispiel **.htpasswd**. Beide Dateien müssen einen speziellen Inhalt haben.

Die .htaccess-Datei hat folgenden:

```
AuthUserFile /kunden/homepages/22/d23295760/.htpasswd
AuthName "Passwortgeschuetzter Bereich: ihredomain.de"
AuthType Basic
require valid-user
```

Die .htpasswd-Datei hat einen ziemlich simplen Aufbau:

```
user:aXmiNQVPt4PhE
user2:aXmiNQVPt4PhE
```

1. .htaccess:

AuthUserFile ../.htpasswd bezeichnet die Position der Passwortdatei (in unserem Fall .htpasswd). Wohl gemerkt ist, dass nicht die URL der Datei angegeben wird, sondern der Absolute Pfad. Die Datei muss nicht .htpasswd heißen. Sie kann z.B. auch pass oder pass.txt heißen

AuthName "..." gibt an wie der zu schützende Bereich heißen soll. Der angegebene Wert wird in dem Abfragedialog angegeben.

AuthType gibt die Autorinfizierungsmethode an. Die meisten Web-Server unterstützen nur Basic.

require gibt an wer sich anmelden darf und wer nicht. D.h. wenn **require user**, dann kann sich nur der Benutzer mit dem Benutzername user anmelden.
In unserem Fall steht **valid-user**. Mit valid-user kann sich jeder Benutzer, der in der Passwortdatei steht anmelden.

2. .htpasswd

Der Aufbau der .htpasswd-Datei ist wahrscheinlich schon klar:

Benutzername:verschlüsseltes Passwort

Zu beachten ist, dass Benutzername und Passwort durch einen Doppelpunkt voneinander getrennt sind. Wichtig ist, dass keine Leerzeilen vor oder hinter dem Doppelpunkt sind!

Die Frage ist nun, wie werden die Passwörter verschlüsselt. An dieser Stelle kommt nun PHP ins Spiel. Hier gibt es den Befehl CRYPT, der unter Verwendung der Standard-DES-Verschlüsselungsmethode von UNIX verschlüsselt.

Zum testen des Scripts **hier** klicken

```
<HTML>

<HEAD>
<title>Der CRYPT Befehl</title>
</HEAD>

<BODY>
<p align="center"><u>Beispiel zum verschlüsseln eines
Passworts</u></p>
<br>

<!-- HIER BEGINNT DER PHP TEIL -->


<?
//Generrieren des Formulars
echo "<br><br>Bitte das gewünschte Kennwort
eingeben:<br><br>";
echo "<form action=\"$PHP_SELF\" method=post>";
echo "<INPUT type=text name=pass> ";
echo "<INPUT type=submit name=submit value=submit>";
echo "</form>";

if($pass)
{                              //Kontrollieren, ob ein Wert
angegeben wurde
do_crypt($pass);               //wenn ja, führe Funktion aus
(weiter unten)
}

function do_crypt($pass)
{
$passC=crypt($pass,yl);                                  //Erzeugen
des Crypts
echo "<br><b>Das Kennwort lautet:</b> $passC";
      //Ausgabe des Crypts
}

?>


<!-- HIER ENDET DER PHP TEIL -->


<br><br>
Der CRYPT Befehl verschlüsselt unter Verwendung der Standard-
DES-Verschlüsselungsmethode von UNIX.
</BODY>
</HTML>
```

Dieses Script erstellt ein simples HTML-Formular. Die einzige Ausnahme ist, das in "action" nicht ein Script steht, sondern "$PHP_SELF". Warum aber? Nun, $PHP_SELF ist eine Variable, die immer den aktuellen Scriptnamen zurückgibt. Man kann das Script nennen wie man Lust und Laune hat. - es ruft sich immer selber auf.

Dateiupload per Browser–

Der Dateiupload ist eine interessante Funktion. Das Skript zeigt, wie dies besonders komfortabel erfolgen kann.

Voraussetzung für dieses Skript ist das Vorhandensein eines Unterverzeichnisses mit dem Namen **_upload**. Ansonsten sind keine Änderungen am Skript notwendig.

```
<html>
<head>
<title>Datei Upload</title>
<meta name="title" content="Datei-Upload">
</head>
<body>


<?
if($action):
?>

<h1>Datei-Upload</h1>
<h3>Ergebnis des Uploads</h3>
<div class=text>

<?
// Zeitbegrenzung: beachten Sie hier die Dateigroesse!
set_time_limit(120);
$path1 = dirname($PATH_TRANSLATED)."/_upload/"; // Pfadangabe
fuer das Ziel
// maximal 4 Dateien gleichzeitig
for($i = 0; $i < 4; $i++){
// Bildung der Variablennamen
switch($i){
case 0:
$source = $file1;
$source_name = $file1_name;
break;
case 1:
$source = $file2;
$source_name = $file2_name;
break;
case 2:
$source = $file3;
$source_name = $file3_name;
break;
case 3:
$source = $file4;
$source_name = $file4_name;
break;
}
if ($source <> "none") {
if ($error1 <> 1) {
$dest = $path1.$source_name;
if (copy($source, $dest)) {
echo "<b>$source_name</b> wurde hochgeladen<br>\n";
} else {
echo "Schreibrechte im Zielverzeichnis fehlen<br>\n";
```

```
$error1 = 1;
}
}
@unlink($source);
}
}
?>


<br>
<a href="<?php echo $PHP_SELF ?>">Zurück</a>
</div>

<?
else:
?>

<h1>Datei-Upload</h1>
<h3>Upload starten</h3>
<div class=text>
Wählen Sie eine oder mehrere Dateien aus.<br>
Limit: 1000 KByte.<br>
<form method="post"
    enctype="multipart/form-data"
    action="<?php echo $PHP_SELF ?>">
<input type="hidden" name="MAX_FILE_SIZE" value="1000000">
Datei 1: <input type="file" name="file1" size="30"><br>
Datei 2: <input type="file" name="file2" size="30"><br>
Datei 3: <input type="file" name="file3" size="30"><br>
Darei 4: <input type="file" name="file4" size="30"><br><br>
<input type="submit" name="action" value="Upload starten">
</form>

<?
endif;
?>

</div>
</body>
</html>
```

Das Script ist eigentlich ziemlich unspektakulär. Kern des Skripts ist der Formular-Befehl:

```
<form method="post"
    enctype="multipart/form-data"
    action="<?php echo $PHP_SELF ?>">
```

Das Skript ruft sich dabei selbst auf. Durch Auswerten des Namens der Send-Schaltfläche wird beim zweiten Durchlauf der obere Teil des Skripts gestartet und die Übertragenen Dateien werden kopiert.

Interessant ist die Ermittlung des Zielpfades. Dies setzt voraus, das ein Unterverzeichnis UPLOAD existiert:

```
$path1 = dirname($PATH_TRANSLATED)."/_upload/";
```

Der Kopiervorgang wird nur gestartet, wenn tatsächlich eine Datei übertragen wurde. Falls das nicht der Fall ist, enthält der Rückgabewert die Zeichenkette "none", entsprechend erfolgt die Auswertung:

```
if ($source <> "none")
```

Aus dem Pfad und dem Namen wird dann der komplette Name für den Kopiervorgang zusammengesetzt:

```
$dest = $path1.$source_name;
```

Der Kopiervorgang selbst steht in einer IF-Anweisung. So lassen sich Fehler sofort auswerten und entsprechende Meldungen anzeigen:

```
if (copy($source, $dest))
```

ENDE