PHP Befehle in HTML einbauen

```
<? ... ?>
<?php ... ?>
<script language="php"> ... </script>
<% ... %>
```

Dazu gibt es vier Möglichkeiten, die links dargestellt sind. Standard ist *zweite* dargestellte Möglichkeit, also **<?php ?>**

PHP testen

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

Aufruf einer Funktion `phpinfo()`, die alle relevanten Informationen zur verwendeten PHP-Version bereitstellt

Bildschirm-Ausgabe

```
echo "Hallo";
echo "Hallo <br>";
echo $a;
echo"<h1>Kneipentest-Auswertung</h1>";
```

Ausgabe eines Textes
Ausgabe eines Textes (inkl. Zeilenumbruch)
Ausgabe eines Variablen-Werts (Var. $a)
Ausgabe eines Textes in Überschrift1

„Escapen" von bestimmten Zeichen

```
echo "Eine neue Zeile: \n, ein
Tabulator: \t, ein Slash \\ und
einmal den String \\n und noch
mal das \" im String
";
```

Eine neue Zeile:
, ein Tabulator: , ein Slash \ und einmal den String \n und noch mal das " im String

Kommentare

```
// Ein Beispielkommentar

/*
ein Kommentar-Bereich, der ueber
mehrere Zeilen gehen kann.
*/
```

Einzeilige Kommentare - eingeleitet durch **//**

Mehrzeilige Kommentare - eingeleitet durch **/*** und beendet mit ***/**

Variablen: Strings

Variablen sind *case-sensitive*. Das bedeutet, die Variablen $ID, $Id, $iD oder $id sind alle verschieden

```
$text = "Ich bin ein String !";
echo $text;
$i = 10;
$j = 5;
echo ($i."+".$j."=".($i+$j));

$vorname = "Max";
$nachname = "Mustermann";
$name = $vorname." ".$nachname;
echo $name;
```

Wertzuweisung
Ausgabe mit „echo"
Verknüpfung von Text und Variablen-Ausgabe mit dem *Punkt-Operator*.

Ergibt auf dem Bildschirm:
Max Mustermann

Variablen: Zahlen

In PHP gibt es 2 Zahlentypen. Einmal sind das Integer-Zahlen (Ganzzahlen) und einmal die Float- bzw. Double-Zahlen (Fließkommazahlen).

```
$g = 9.81;




echo 4 . 5; // Ausgabe 45
echo 4.5; // Ausgabe 4.5
```

Bei Float-Zahlen muss man beachten, dass das Kommazeichen ein Punkt ist. Das Kommazeichen ist nämlich schon für die Trennung der Parameter einer Funktion belegt

Problem: Der Punkt als Verkettungsoperator.

Rechnen mit Zahlen

Die Grundrechenarten sind +, -, * und /.
Der Doppelpunkt : ist nicht das Divisionszeichen, dies ist schon *belegt*.

```
$rest = 20%7;
echo "Rest von 20/7 ist: ".$rest;
// Ergebnis: 6 da 20=(2*7)+6 = 14+6
```

Der **Modulo-Operator** ist das Prozentzeichen **%**. Mit ihm lässt sich der ganzzahlige Rest einer Division bestimmen.

Kontrollstrukturen: Boolean-Datentyp

Einführung: In PHP gibt es einen Variablentyp, der genau 1 Bit Speicher belegt („**Boolean**"). In diesem Bit kann dann entweder 0 oder 1 stehen. Statt 0 und 1 sagt man auch true bzw. wahr (für die 1) oder auch false bzw. falsch (für die 0).

```
$name = true;
$name2 = false;
```

`$name` ist vom Typ Boolean, hat den Wert "true"
`$name2` ist vom Typ Boolean, hat den Wert "false"

Kontrollstrukturen: Schleifen

Geschlossene, kopfgesteuerte Schleife mit „for"

```
$t = "Ausgabetext!<BR>";
for ($i=0;$i<10;$i++)
{echo $t;}
```

Zählvariable ist `$i`. Abbruchbedingung hier: $i=10. Erhöhung der Zählervariablen pro Durchlauf um 1. ($i++). Anzahl der Schleifendurchläufe immer bekannt: geschlossene Schleife.

Offene, kopfgesteuerte Schleife mit „while"

```
while (bedingung) {
  // Programmcode
}
```

Wird die Bedingung am Anfang der Schleife auf „true" geprüft, läuft die Schleife.

Offene fußgesteuerte Schleife mit „do while"

```
do {
  // PHP-Code
}
while (bedingung);
```

ACHTUNG: Semikolon hinter While!!!

Wird die Bedingung am Ende der Schleife auf „true" geprüft, läuft die Schleife.

Kontrollstrukturen: Fallunterscheidung 1

```
if ($i<0)
{  echo "$i ist kleiner als Null"; } //if
else { echo "$i ist nicht kleiner als Null"; }//else
```

Kontrollstrukturen: Fallunterscheidung 2

```
switch ($name)
{  case "Heinrich":
   echo "Ich bin der kluge Heinrich";
   break;
   case "Hans":
   echo "Ich bin der dumme Hans";
   break;
   default:
   echo "Wir sind der Rest";  }//switch
```

Die variable `$name` wird nach den Fällen Variableninhalt=Heinrich und Variableninhalt=Hans geprüft. Der entsprechende Zweig (`case`) wird bis zur break-Anweisung ausgeführt.
Der `default`-Wert fängt alle anderen Fälle auf.

Kontrollstrukturen: Vergleichsmöglichkeiten

• `$i==10`	• Ist $i gleich 10?
• `$i!=10`	• Ist $i ungleich 10?
• `$i>=10`	• Ist $i größer oder gleich 10?
• `$i<=10`	• Ist $i kleiner oder gleich 10?
• `($i==10) && ($j>0)`	• Ist $i gleich 10 und $j größer als 0?
• `($i==10) \|\| ($j==0)`	• Ist $i gleich 10 oder $j gleich 0?

```
$x = "10"; //String mit Wert 10
$y = 10;   //Zahl mit dem Wert 10
var_dump($x == $y);
// gibt 'bool(true)' aus
```

Die **drei Gleichheitszeichen** hintereinander sind keine Schreibfehler. Dieser Vergleichsoperator funktioniert so ähnlich wie der 'normale'

`var_dump($x === $y);` `// gibt 'bool(false)' aus`	Vergleichsoperator ==. Ein Vergleich mit === liefert erst dann true wenn Variableninhalt UND Variablentyp übereinstimmen. Links ein Beispiel:

Dateien hinzuladen / einbinden

`include("dateiname.inc.php");`
 	
`require_once ("dateiname.inc.php");`
	Modulares Programmieren: Hinzuladen weiterer php-Dateien. (`include`)
	`require_once`: (oder `include_once`) Datei wird explizit nur einmal hinzugeladen, um Fehler zu vermeiden.
	dateiname.inc.php: „inc": Eine „include-Datei"

GET / POST Variablen

`http://www.xy.com/check.php?clan=345&action=add`	GET und POST Variablen werden genutzt, um einem Script Werte übergeben. Dies kann man über *Cookies*, *Formulare* oder über die *URL* machen. Wie man Werte über eine URL übergibt hat man bestimmt schon mal gesehen: Das „?" startet den Übergabebereich, mehrere Variablen/Werte werden mit „&" verknüpft. Da man in der URL die Daten sieht, ist die Methode hier die **GET**-Methode
`$_GET['clan']mit dem Wert "345"` `$ GET['action']mit dem Wert "add"`	In der Datei `check.php` können so die Werte ausgelesen werden.
`GET`	Übertragung der Daten im URL Versendbare Datenmenge: meistens max. 1024 Byte GET-Anfragen als Lesezeichen abspeicherbar GET-Anfragen werden vom Browser gecacht
`POST`	Übertragung der Daten im Dokument-Body Versendbare Datenmenge: nicht beschränkt POST-Anfragen werden vom Browser *nicht* gecacht.

isset() und empty(): Überprüfen von Eingaben

`$var = "";`
`$x = isset($var);`
`var_dump($x);//gibt "bool(true)" aus`
`$y = isset($andere_var);`
`var_dump($y);//gibt "bool(false)" aus`
 	
`if(empty($_POST['username']))`
`{echo "Bitte geben sie einen Benutzernamen ein";  }`
	Die Funktion `isset()` liefert **true**, wenn die übergeben Variable oder das übergebene Arrayelement definiert ist und **false**, wenn dies nicht so ist.
	`!isset($variable)` prüft, ob es eine Variable *nicht* gibt
	Leere Variablen sind definierte Variablen!
	
	Ob eine Variable leer ist (z.B. bei Formularen) kann man mit der Funktion `empty()` prüfen.

Formulare mit PHP

Das Handling von Formularen ist extrem wichtig: über Formulare erfahren wir, was der Benutzer tun möchte!

HTML-Formulare sind der einzige praktikable Weg, wie Daten vom Browser zum Server und damit zu Ihrem Skript gelangen können. Ein Formular besteht aus dem <form>-Tag mit verschiedenen Parametern und den Feld-Elementen, HTML-Tags zur Darstellung von Eingabefeldern.

Element	Beschreibung	Attribute
text	Einzeiliges Eingabefeld	size, value, name
checkbox	Kontrollkästchen	value, checked, name
radio	Optionsfelder	value, checked, name
submit	Sendeschalter	value, name
reset	Schalter zum Rücksetzen	value, name
password	Verdecktes Eingabefeld	size, value, name
hidden	Unsichtbares Feld	value, name
button	Schaltfläche	value, name
image	Bild, ersetzt submit	src, name, Bildattribute (img)
file	Eingabefeld und Schaltfläche	name, accept

Abbildung 1: Formular-Elemente

Um es vorweg zu nehmen: Es gibt viele Frameworks wie jQuery, die eine optimale Formular-Gestaltung erlauben und viele Erleichterungen („Validitätsprüfung": Sind alle Eingaben ok?) bieten.

Grundlagen:

Formularauswertung

formular.htm

```
. . .
<FORM ACTION="tuwas.php" METHOD=POST>
<INPUT NAME="xyz">
<INPUT TYPE="submit">
</FORM>
. . .
```

Formular wird von der Datei `tuwas.php` ausgewertet.
„`xyz`" ist die Variable für den Eingabetext
Achtung: Globale Variablen=Sicherheitslücke!!!
php.ini: **`RegisterGlobals=Off`** setzten und in php mit: `$_POST['xyz']` den Wert holen

tuwas.php

```
. . .
echo "Sie haben ".$_POST['xyz']."
eingegeben";
```

Die Variable „`xyz`" wird auf dem Bildschirm ausgegeben.

auswahlformular.htm

```
<FORM METHOD="POST" ACTION="auswertung.php">
 <SELECT SIZE="1" name="D1">
  <OPTION VALUE="1">Filme komplett</OPTION>
  <OPTION VALUE="2">Film-Detais</OPTION>
 </SELECT>
<INPUT TYPE="submit" VALUE="Abschick" name="B5">
<INPUT TYPE="reset" VALUE="Zurücksetz" name="B6">
</FORM>
```

Formular mit Drop-Down-Menü: Auswertung der Variable (hier D1 mit Werten von 1 oder 2 findet in der Datei auswertung.php statt)

Tipp: Lesen Sie im moodle-System den Bereich „erweiterte Formularbehandlung" durch!

Arrays mit PHP

Um verschiedene Werte in nur einer Variable zu speichern, verwendet man in PHP ein *Array*. Bisher konnte jede Variable nur einen Wert speichern, in PHP können in einem Array beliebig viele Werte gespeichert werden, auch von unterschiedlichen Typen. So kann ein PHP Array z.B. eine Zeichenkette (String/Text), eine Ganzzahl und eine Zahl mit Kommastellen enthalten und auf jeden dieser Werte kann man separat zugreifen.

Eindimensionales Array	
`$wochentage = array("Sonntag","Montag","Dienstag", "Mittwoch","Donnerstag","Freitag","Samstag");`	Einem PHP-Array wird ganz normal ein Variablenamen zugeordnet, hier `wochentage`, allerdings erfolgt die Zuweisung der Daten nicht einfach durch das Gleichheitszeichen, sondern danach kommt noch der Befehl: `array();`
`echo $wochentage[1];`	Einfacher Zugriff auf Array-Elemente, hier auf den ZWEITEN Eintrag: `Montag`. Arrays beginnen immer beim Element-Index „0"

`$wochentage [] = "Feiertach";`
 	
`echo $wochentage [7];`
	Hinzufügen von Array-Elementen: Ganz einfach!

Assoziatives Array	

`$wochentage = array(`
`"so" => "Sonntag",`
`"mo" => "Montag",`
`"di" => "Dienstag",`
`"mi" => "Mittwoch",`
`"do" => "Donnerstag",`
`"fr" => "Freitag",`
`"sa" => "Samstag");`
	Bei großen Arrays wird es natürlich irgendwann umständlich, zu wissen, welche Nummer/Index zu welchem Wert gehört, darum gibt es *assoziative PHP Arrays*.
	Das heißt, man kann für einen Wert einen Schlüssel/Key zuweisen, dies kann z.B. eine andere Zeichenkette sein.
	Die Zuweisung erfolgt per: „=>"

`echo $wochentage["mo"];`	*Zugriff* auf das Array-Element mit Benennung des Keys.
`$wochentage["mo"] = "Monday";`	*Ändern* eines Array-Wertes
`$wochentage["fei"] = "feiertach";`	*Erzeugen* eines Array-Wertes

Mehrdimensionales Array

In einem Array kann man ein weiteres Array, und in diesem Array wieder ein Array speichern, und so weiter. Solche Arrays nennt man dann *mehrdimensionale Arrays*. Die Dimension gibt dabei an, wie Tief diese Verschachtelung geht. Ein normales Array wäre ein 1-dimensionales Array, wenn jetzt in dem Array ein weiteres Array gespeichert ist, ist dies ein 2-dimensionales Array.

```
$mitarbeiter = array(
  array("Klaus", "Zabel"),
  array("Arnie", "Meier"),
  array("Willi", "Brand")
);
```

Erzeugen eines zweidimensionalen Arrays: Das Aussehen des Arrays:

	Spalte: 0	**Spalte: 1**
Reihe: 0	Klaus	Zabel
Reihe: 1	Arnie	Meier
Reihe: 2	Willi	Brand

```
echo "Vorname: ".$mitarbeiter[0][0];
echo " Nachname: ".$mitarbeiter[0][1];
```

Zugriff auf das zweidimensionale Array:
`$mitarbeiter[`**`REIHE`**`][`**`SPALTE`**`];`
Ausgabe: Vorname: `Klaus` Nachname: `Zabel`

Mehrdimensionales assoziatives Array

```
$mitarbeiter = array();
$mitarbeiter[] =
array("Vorname"=>"Klaus","Nachname"=>"
Zabel");
$mitarbeiter[] =
array("Vorname"=>"Arnie","Nachname"=>"
Meier");
$mitarbeiter[] =
array("Vorname"=>"Willi",
"Nachname"=>"Brand");
```

Erzeugen eines zweidimensionalen, assoziativen Arrays:
Zuerst ein leeres Array erzeugt, dann Werte eingetragen.
Das Aussehen des Arrays:

	Spalte: Vorname	**Spalte: Nachname**
Reihe: 0	Klaus	Zabel
Reihe: 1	Arnie	Meier
Reihe: 2	Willi	Brand

```
echo "Vorname:
".$mitarbeiter[0]["Vorname"];
echo " Nachname:
".$mitarbeiter[0]["Nachname"];
```

Zugriff auf das Array:
`$mitarbeiter[`**`REIHE`**`][`**`SPALTE`**`];`
Ausgabe: Vorname: `Klaus` Nachname: `Zabel`

Arbeiten mit Arrays: Anzahl der Elemente

```
$mitarbeiter = array("Bob","Peter");

echo count($mitarbeiter);
```

Wenn wir in einem Array dynamisch neue Elemente hinzufügen können, dann ist es sehr wichtig zu wissen, *wie viele Elemente* das Array überhaupt enthält.
Ausgabe: 2

Arbeiten mit Arrays: Elemente ausgeben

```
$mitarbeiter = array("Bo","Pa","Li");
for($i=0; $i < count($mitarbeiter);
$i++)
   {
   echo $mitarbeiter[$i]."<br>";
   }
```

Zum Ausgeben aller Elemente in einem Array benutzt man gewöhnlich eine Schleife, dies kann eine `for`, eine `while` oder eine `foreach` Schleife sein:

for-Schleife

```
$i = 0;
while($i < count($mitarbeiter))
   {
   echo $mitarbeiter[$i]."<br>";
   $i++;
   }
```

while-Schleife

```
foreach($mitarbeiter AS $name)
   {
   echo $name."<br>";
   }
```

foreach-Schleife (Besonderheit, gibt es nicht in jeder Programmiersprache) Im Schleifenkopf definieren wird, dass die Werte in `$name` gespeichert werden soll. Unter dieser Variable können wir dann innerhalb der Schleife auf den jeweiligen Namen des Mitarbeiters zugreifen.

Assoziative Arrays: Noch mehr Dimensionen

```
$mitarbeiter = array();
$mitarbeiter["Klaus"]["Vorname"] =
"Klaus";
$mitarbeiter["Klaus"]["Nachname"]
= "Zabel";
$mitarbeiter["Klaus"]["Kinder"][]
= "Klaus-Junior";
$mitarbeiter["Klaus"]["Kinder"][]
= "Kind2";
```

In der neuen Spalte `Kinder` wird ein weiteres Array erzeugt. Das Aussehen des Arrays:

	Spalte: Vorname	**Spalte: Nachname**	**Spalte: Kinder**	
Reihe: Klaus	Klaus	Zabel	Klaus-Junior	Kind2

```
echo "Vorname: ".$mitarbeiter["Klaus"]["Vorname"];
echo " Nachname: ".$mitarbeiter["Klaus"]["Nachname"];
echo "<br> Er hat ";
echo count($mitarbeiter["Klaus"]["Kinder"])." Kinder";
```

Ausgabe eines Kinds / bzw. aller Kinder mit `foreach`

```
//Ausgabe von Kind1:
//$mitarbeiter["Klaus"]["Kinder"][0];

echo "<br> Kinder: <br>";
foreach($mitarbeiter["Klaus"]["Kinder"] AS $name)
   {
   echo $name."<br>";
   }
```

Zugriff auf Datenbanken

1: Verbindung zur Datenbank

`$db=mysql_connect("localhost","root","");`
 	
`if ($db==false) {`
`  echo "Keine Verbindung möglich!";`
`  exit;`
`}`
	Die Variable `$db` erhält das Ergebnis des Datenbank-Aufbaus. Dies kann anschließend per Abfrage mit „if" getestet werden. localhost=lokaler Rechner, root=user, "leer"=Passwort

`mysql_select_db("kneipentest",$db) or exit ("Datenbank kann nicht geöffnet werden");`	Auswahl d. Datenbank (hier „kneipentest") oder Fehlermeldung

2: SQL-Befehle verwenden

`$sql = "SELECT * FROM kneipen";`	SQL-Abfrage wird in eine Variable `$sql` geschrieben

`$erg = mysql_query ($sql,$db)`
 	
`or die ("Fehlermeldung=". mysql_error());`
	Das Ergebnis der Datenbank-Anfrage mit dem SQL-Befehl "landet" in einer Ergebnis-Variablen (`$erg`). Diese Variable ist ein „Ergebnis-Handle", also eine Resource-ID, mit der man auf das Ergebnis (also eine Art innere Tabelle) zugreifen kann. Sie liefert **kein Array**, **keine Zahl** und **kein String** zurück
	Hinten: Fehlerroutine

3: Ausgabe der Ergebnisse

`$anz = mysql_num_rows($erg);`
`for ($i=0; $i<$anz;$i=$i+1)`
`{`
`  $a=mysql_result($erg, $i, "id");`
`  $b=mysql_result($erg, $i, "Name");`
`  $c=mysql_result($erg, $i, "Art");`
`  echo $b; echo $c;`
`} //for`
	Die Variable `$anz` bekommt die Anzahl der Datensätze ("Reihen").
	Per Schleife werden dann aus jeder „Reihe" die einzelnen Zellen in Variablen ausgelesen (in `$a`, `$b`, `$c`) und auf dem Bildschirm ausgegeben (mit `echo`). Dies wird für jede Reihe wiederholt.

Oder eine andere Variante für Schritt 3:	
`while (list($b,$c) = mysql_fetch_row($erg))` `  {echo $b; echo $c;}//while`	**Andere Möglichkeit** mit `mysql_fetch_row($erg)` Weitere Möglichkeiten mit `mysql fetch assoc` oder `mysql_fetch_array`
Noch eine andere (weit verbreitete) Variante für Schritt 3:	

`while ($row = mysql_fetch_assoc($result))`
`{`
`  echo $row["id"];`
`  echo $row["Name"];`
`  echo $row["Art"];`
`}`
	Solange eine Zeile mit Daten existiert, wird dies in dem *assoziativen Array* `$row` abgelegt.
	Wenn Sie `extract($row)` innerhalb der folgenden Schleife verwenden, können Sie damit die Variablen `$id`, `$Name` und `$Art` erzeugen.

4: Datenbank-Verbindung beenden

`mysql_close($db);`	Datenbank-Verbindung wird geschlossen.

PHP-Funktionen

Wenn gleicher Programmcode häufig ausgeführt wird, sollte dieser Programmcode in einer Funktion ausgelagert werden. Statt also jedes Mal den gleichen Code zu haben, ruft man an den entsprechenden Stellen nur noch die Funktion auf, die dann die eigentliche Arbeit verrichtet.
Das Verhalten einer Funktion wird durch ihre *Parameter* beeinflusst. Eine Funktion kann beliebig viele Parameter besitzen, dies schließt keine Parameter genauso ein wie unendlich viele Parameter. Des Weiteren *kann* eine Funktion einen Wert zurückliefern.

Eigene Funktionen

```
function NAME-DER-FUNKTION ( ÜBERGABEWERTE )
{
   AUSZUFÜHRENDER CODE;
   return ( ZURÜCKGEBENER WERT );
}
```

Hier die allgemeine Struktur / Syntax einer PHP-Funktion

```
$ZURUECKGEGEBENER_WERT =  NAME-DER-FUNKTION ( ÜBERGABEWERTE );
```

Aufgerufen wird eine Funktion dann so. Manchmal (siehe nächstes Beispiel) ist kein Rückgabewert vorhanden.

Konkretes Beispiel: die Funktion `ausgabe_uhrzeit()`: Funktion OHNE Rückgabewert

```
function ausgabe_uhrzeit()
{
    echo "<p>Es ist gerade: ". date("H:i:s"). "</p>";
}
```

`date("H:i:s")` ist übrigens eine weitere, vordefinierte Funktion (siehe unten)

```
ausgabe_uhrzeit();
```

Konkretes Beispiel: die Funktion `dividieren()`: Funktion MIT Übergabewerten UND Rückgabewert

```
function dividieren($zahl,$quotient)
{
 $erg = bcdiv($zahl,$quotient, 2);
 // geht auch so:
 // $erg = $zahl/$quotient;
 return ( $erg );
}
```

Mit `bcdiv()` wird der erste Operand (`$zahl`) durch den zweiten Operanden (`$quotient`) dividiert. Der Parameter Genauigkeit (`2`) bestimmt die Nachkommastellen im Ergebnis.

```
$dividend = 33;
$divisor  = 5;

$wert = dividieren($dividend, $divisor);
echo "Berechnung von $dividend / $divisor = $wert ";
```

Benutzen der Funktion

Vorgabewerte für Funktionen bestimmen

```
function dividieren( $zahl=1, $quotient=1 )
...
```

So können Variablen mit Werten belegt werden, falls keine Parameter übergeben werden.

Es ist sicher sehr sinnvoll, eigene Funktionen zu dokumentieren, dazu eignen sich Werkzeuge, wie **PhpDocumentor**.

Vordefinierte Funktionen

Es gibt sehr viele vorgefertigte Funktionen für PHP, die ebenso gut dokumentiert sind. So gibt es Funktionen für Datenbanken, Zeichenketten, Dateien, Datumsfunktionen, Mathematische Funktionen, usw. **Hier** finden Sie eine Übersicht.

Eine kleine Auswahl von Funktionen:

Funktion	**Beschreibung**
string **date**(string format [, int timestamp])	Datumsfunktion, siehe Doku für genauere Benutzung
string **decbin**(int number)	Umwandlung von dec→bin
int **hexdec**(string hex_string)	Umwandlung von hex→dec
float **sin**(float arg)	Sinusfunktion
float **sqrt**(float arg)	Wurzelfunktion
int **chmod**(string filename, int mode)	Ändern der Dateirechte (UNIX only), ebenso chown, etc.
int **fopen**(string filename, string mode [, int use_include_path])	Öffnen einer Datei
int **fclose**(int fp)	Schließen einer Datei
int **file_exists**(string filename)	Funktion, um zu überprüfen, ob eine Datei auf einem Server existiert oder nicht.
string2 **md5** (string1)	Verschlüsselung nach dem md5-Verfahren des Strings1, Ergebnis in String2.
bool **mail**(string to, string subject, string message [, string additional_headers])	Mail-Funktion, siehe Doku für genauere Benutzung
int **strlen**(string str)	Funktion gibt die Länge einer Zeichenkette (str) zurück
int **strncmp**(string str1, string str2, int len)	Mit strncmp() kann man zwei Zeichenketten (str1 und str2) auf Binärbasis miteinander vergleichen.

$GATEWAY_INTERFACE
$SERVER_NAME
$SERVER_SOFTWARE
$DOCUMENT_ROOT
$HTTP_CONNECTION
$SERVER_ADMIN
$SERVER_PORT
$SCRIPT_NAME
$REMOTE_ADDR
	Vordefinierte Variablen, die unterschiedliche Informationen enthalten.
	Welche Adresse hat der Besucher?
	`$ip = $_SERVER["REMOTE_ADDR"];`
	`echo $ip;`
	Auf welcher Seite ist der Nutzer gerade?
	`$seite = $_SERVER["PHP_SELF"];`
	`echo $seite;`
	Welchen Browser nutzt der Besucher?
	`$browser = $_SERVER["HTTP_USER_AGENT"];`
	`echo $browser;`

Datums- und Zeitfunktionen

Mit `date()` kann man eine Zeitangabe formatieren oder auswerten. Die Zeitangabe übergeben Sie im Parameter `timestamp`. Lassen Sie diesen Parameter leer, nimmt die Funktion die aktuelle Zeit. Der Parameter `format` ist ein String, der festlegt, welche Informationen über die Zeitangabe Sie benötigen. In diesem String sind folgende Platzhalter möglich (*: Ausgabe mit führenden Nullen):
a - "am" oder "pm"
A - "AM" oder "PM"
d - Tag des Monats *(01 - 31)
D - Tag der Woche (Wed - 3stellig)
F - Monatsangabe (December - ganzes Wort)

Beispiel	**Ausgabe:**
`<?PHP` `echo date("d M Y") . " ";` `echo date("Y m d") . " ";` `echo date("d n y") . " ";` `echo date("D, d m Y") . " ";` `echo date("l, d m Y") . " ";` `echo date("l dS of F Y h:i:s A") . " ";` `echo "Dieser Monat hat " . date(t) . " Tage";` `?>`	29 May 2001 2001 05 29 29 5 01 Tue, 29 05 2001 Tuesday, 29 05 2001 Tuesday 29th of May 2001 06:20:57 AM Dieser Monat hat 31 Tage