Einführung

von Heike Gierisch und Katja Mayer

Überblick

1 Einführung PHP

Einsatzgebiete von PHP

- Generierung dynamischer Webseiten
  - HTML (Haupteinsatzgebiet)
  - PDF-Dateien
  - Bilder
  - XML-Dateien
- Auswertung von Formularen
- Datenbank-Anbindung (z.B. MySQL)
- Senden und Empfangen von Cookies

Vorteile von PHP

- schnell erlernbar (wichtig für den Anfänger)
- großer Funktionsumfang (wichtig für den Profi)
- Plattform-übergreifend (Linux, Windows, MAC OS, …)
- kostenlos
- Skripte laufen auf dem Webserver.
  - Nur der HTML-Code wird übermittelt, der User kennt den Quellcode nicht.
  - Der Browser braucht keine speziellen Fähigkeiten, die hat der Server.
  - Es kann auf Textdateien und Datenbanken auf dem Server zugegriffen werden.

Voraussetzungen für PHP

Programmierer:
- Editor (Syntax-Highlighting ist praktisch)
- Upload-Möglichkeit am Webserver (FTP)

Webserver mit laufendem PHP-Parser:
- Linux-Rechner (Apache-Server und PHP bereits integriert) oder
- Windows-Rechner mit Apache-Server und PHP

Anwender:
- Browser ohne weitere Plug-Ins
- Zugang zum Internet

2 HTML-Kenntnisse: Grundstruktur

```
<html>
    <head>      Kopfdaten des Dokuments: Titel, Meta-Tags, Einbindung von
    </head>     Stylesheets …
```

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

Inhalt der Seite:
Hier steht alles, was im Browser angezeigt wird.

2.1 HTML-Kenntnisse: Tabellen

```
<table border="1">
    <caption align="top">Tabellenueberschrift</caption>
    <tr>
     <td>Zeile 1 Zelle 1</td>
     <td>Zeile 1 Zelle 2</td>
    </tr>
    <tr>
     <td>Zeile 2 Zelle 1</td>
     <td>Zeile 2 Zelle 2</td>
    </tr>
</table>
```

Tabellenueberschrift

Zeile 1 Zelle 1	Zeile 1 Zelle 2
Zeile 2 Zelle 1	Zeile 2 Zelle 2

2.2 HTML-Kenntnisse: Hyperlinks

```
<a href= "http://www.php.net/docs.php">
    PHP-Handbuch
</a>
```

PHP-Handbuch

SelfHTML

2.3 HTML-Kenntnisse: Formulare

```
<form action="Was_soll_passieren" method= "get oder post">
    <input name="" type="text" value=""><br>
    <input type="submit">
    <input type="reset">
</form>
```

2.3.1 HTML-Formulare: action

`action="mailto:name@provid.er"`
Formulardaten werden an die angegebene Adresse geschickt. Dieses Verfahren ist unsicher, da das Versenden der Email von Einstellungen beim Anwender abhängt.

`action="http://www.website.de/datei.htm"`
Es wird eine neue Seite aufgerufen (HTML, PHP oder CGI).
Es gelten die normalen Referenzierungsregeln von HTML für absolute und relative Adressierung.

2.3.2 HTML-Formulare: method

`method="get"`
Die Eingabedaten werden als Parameter an die Adresse (`action`) angehängt.
Beschränkung: Es werden max. 255 Zeichen inklusive der URI übermittelt

`method="post"`
Die Eingabedaten werden direkt (für den Anwender nicht sichtbar) an den Webserver geschickt.
Die Länge der übermittelten Zeichen ist unbeschränkt

2.3.3 HTML-Formulare: Eingabefelder

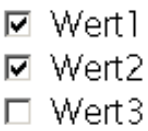
einzeiliges Eingabefeld:
```
<input name="a" type="text" size="10">
```
mehrzeiliges Eingabefeld (hier mit 5 Zeilen à 20 Zeichen):
```
<textarea name="b" rows="5" cols="20"></textarea>
```
Der eingegebene Text wird als Wert der Variable `name` übermittelt.

2.3.4 HTML-Formulare: Radio-Buttons

```
<input type="radio" name="a" value="1">Wert1<br>
<input type="radio" name="a" value="2">Wert2

Nur einer der beiden Werte kann markiert werden.
Es wird dann 1 oder 2 übermittelt.
```

2.3.5 HTML-Formulare: Checkbox

```
<input type="checkbox" name="a"   value="1"> Wert1<br>
<input type="checkbox" name="b"   value="2"> Wert2<br>
<input type="checkbox" name="c"   value="3"> Wert3
```



Es können mehrere Werte angekreuzt werden.

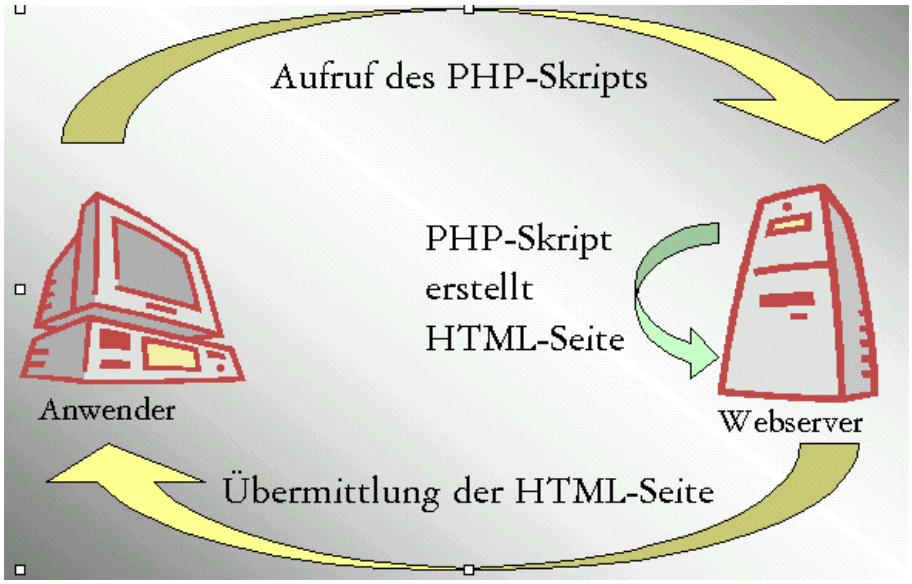
3 PHP-Tags

`<?php` PHP-Befehle; `?>` — für kurze Passagen sinnvoll

```
<script language="php">
      PHP-Befehle;
</script>
```
für längere PHP-Programme

`<?..?>` und `<% ... %>` — Diese Kurzversionen sind nicht immer verfügbar. Also: nicht verwenden!!

3.1 Client-Server-Prinzip



3.2 Speichern der PHP-Dateien

Der Programmierer speichert Skripte in speziellem Verzeichnis des Webservers. Beispiel Linux-Server mit Samba:
IP-Adresse des Webservers: 192.168.104.7, Verzeichnis: htdocs
Datei speichern unter: \\192.168.104.7\htdocs\test.php

Achtung: \ („Backslashes“) verwenden!

3.3 Ausführen der PHP-Dateien

Der Anwender ruft PHP-Seite vom Webserver mit dem Browser auf:
//192.168.104.7/test.php

Achtung: / („Slashes“) verwenden!

(Bei Linux-Webserver „htdocs“ nicht mehr angeben!)

Aufruf, falls Webserver = eigener Rechner:
//localhost/test.php (evtl. mit „htdocs“)

3.4 Grundregeln der Syntax

Befehle enden immer mit ;

Kommentare (äußerst wichtig!!):

einzeilig

```
// Hier steht der Kommentar
```

mehrzeilig

```
/* In dieser Zeile
   und hier
   und hier auch
   steht Kommentar
*/
```

Bildschirmausgabe:
echo "Dieser Text erscheint genau so am Bildschirm.";

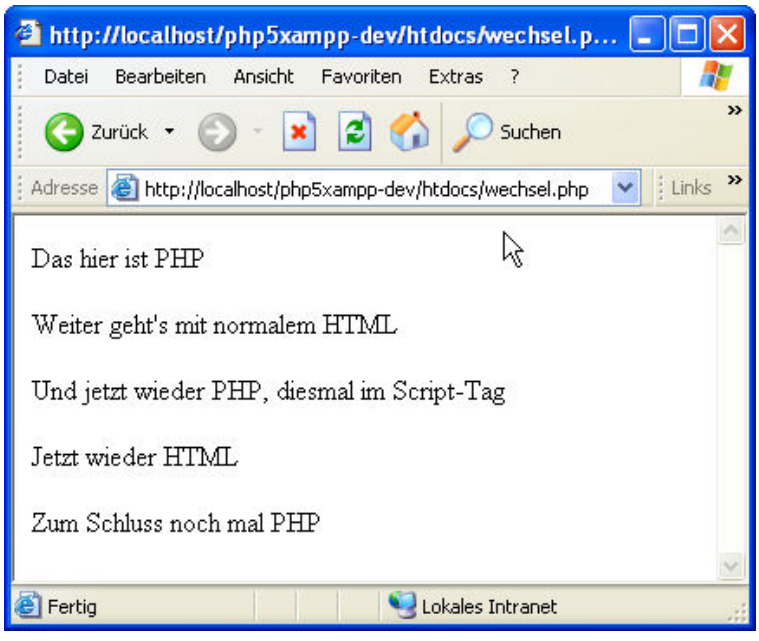
3.5 Datumsfunktion

Mit der Funktion `date` lassen sich die Datumsfunktionen aufrufen. Syntax:
echo date ("[Parameter]"); z.B. echo date ("Y-m-d");

Y	Jahreszahl , vierstellig (2001)
m	Monat (01-12)
n	Monat (1-12)
d	Tag des Monats (01 - 31)
j	Tag des Monats (1-31)
H	Stunde im 24-Stunden-Format (00-23)
i	Minuten (00-59)

3.6 Wechsel PHP-HTML

```
<html>
  <head>
  <!-- Hier wird der Wechsel zwischen PHP und HTML gezeigt -->
  </head>
  <body>
    <?php echo "Das hier ist PHP"; ?>

    <p>Weiter geht's mit normalem HTML</p>

    <SCRIPT language="php">
       echo "Und jetzt wieder PHP, diesmal im Script-Tag";
    </SCRIPT>

    <p>Jetzt wieder HTML</p>

    <?php echo "Zum Schluss noch mal PHP"; ?>

  </body>
</html>
```



4 Datentypen, Variablen, Konstanten

4.1 Datentypen

Zahl
- Ganzzahl (integer) Zahl ohne Punkt
- Fließkommazahl Zahl mit Punkt 1.6

Zeichenketten (string) "..."
Arrays
Objekte

4.2 Variable

Variable sind Speicherzellen, die über einen Namen (den Bezeichner) angesprochen werden. In der Speicherzelle wird der Wert der Variablen abgelegt.
Beispiel: `$a = 7;`
Der Bezeichner ist a, der Wert von a ist 7.

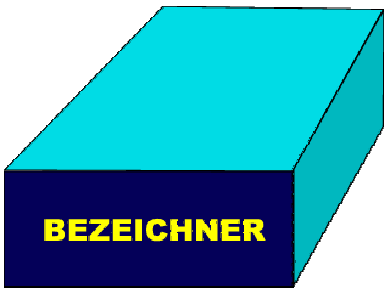
4.3 Benennung von Variablen und Funktionen

Dollar-Zeichen+Buchstabe oder Unterstrich am Anfang, kein Leerzeichen, nur bestimmte ASCII-Zeichen (vgl. Tabelle):
~~`$3max`~~ `$max3`
~~`$ Grüße!`~~ `$grüße ß=223 ü=252` ~~`!=33`~~
Groß- und Kleinschreibung wird unterschieden
`$Max ≠ $max`
keine für PHP reservierten Begriffe
~~`$echo`~~ `$ausgabe`
Selbsterklärende Namen verwenden
~~`$a, $b`~~ `$summand_1, $summand_2`

4.4 Konstanten

Konstante haben kein `$`-Zeichen. Sie werden aber in GROSSBUCHSTABEN geschrieben.

vordefinierte Konstanten: z.B. π : M_PI
Selbstdefinierte Konstanten:
Syntax: `define ("KONSTANTE","€");`



4.4 ASCII-Tabelle mit erlaubten Zeichen

Angegeben ist die dezimale Kodierung der ASCII-Zeichen. Aus der linken Spalte nimmt man die Zehnerzahl, aus der ersten Zeile die Einer.
Beispiel: Das Zeichen A ist das ASCII-Zeichen Nummer 65, das à-Zeichen ist die Nummer 224.

	0	1	2	3	4	5	6	7	8	9
0										
1										
2										
3										
4									0	1
5	2	3	4	5	6	7	8	9		
6						A	B	C	D	E
7	F	G	H	I	J	K	L	M	N	O
8	P	Q	R	S	T	U	V	W	X	Y
9	Z									
10										
11										
12								□	€	□
13	‚	ƒ	„	…	†	‡	ˆ	‰	Š	‹
14	Œ	□	Ž	□	□	‘	’	“	”	•
15	–	—	˜	™	š	›	œ	□	ž	Ÿ
16		¡	¢	£	¤	¥	¦	§	¨	©
17	ª	«	¬		®	¯	°	±	²	³
18	´	µ	¶	·	¸	¹	º	»	¼	½
19	¾	¿	À	Á	Â	Ã	Ä	Å	Æ	Ç
20	È	É	Ê	Ë	Ì	Í	Î	Ï	Ð	Ñ
21	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û
22	Ü	Ý	Þ	ß	à	á	â	ã	ä	å
23	æ	ç	è	é	ê	ë	ì	í	î	ï
24	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù
25	ú	û	ü	ý	þ	ÿ				

5 Wichtige Operatoren

.	Verkettung von Strings
`$a + $b`	Addition
`$a - $b`	Subtraktion
`$a * $b`	Multiplikation
`$a / $b`	Division
`$a = 3`	Zuweisung eines Wertes
`$a++`	$a = $a+1 (Inkrement)
`!A`	nicht
`A && B`	Bedingung A und Bedingung B wahr
`A || B`	Bedingung A oder Bedingung B oder beide wahr

5.1 Zeichen-Verkettung

Variablen können direkt in den echo-Befehl mit doppelten Hochkommas eingegeben werden:

```
echo "Das Quadrat von $zahl ist $quadrat."
```

Mit dem Punkt-Operator werden zwei Variablen oder Variable und String verknüpft:

```
echo "Sehr geehrte Frau ".$nachname.",";
```

5.2 Datentypen bei Verkettungen

Die Verkettung mit . (Punkt-Operator) ➔ Datentyp String

```
$a = "5Euro"; $b = "70ct";
$c = $a . $b;
echo $c; //Ausgabe: 5Euro70ct
```

Die Verkettung mit + (Plus-Operator) ➔ Datentyp float oder integer (je nach Inhalt des Strings)

```
$a = "5Euro"; $b = "70ct";
$c = $a + $b;
echo $c; //Ausgabe: 75
```

5.3 Runden

kaufmännisches Runden: `$a = round (1.95583,2)`
ergibt den Wert 1.96
Der zweite Parameter (hier 2) gibt die Stellen hinter dem Komma an.

Abrunden: `floor (1.95583)`
Diese Funktion liefert diejenige ganze Zahl, die kleiner oder gleich dem Parameter in der Klammer ist.

Aufrunden: `ceil (1.95583)`
Diese Funktion liefert diejenige ganze Zahl, die größer oder gleich dem Parameter in der Klammer ist.

Abbrechen: `bcdiv(7,3,2)` (auch `bcmul, bcadd, bcsub`)
Diese Funktion berechnet die Division 7:3 auf 2 Stellen hinter dem Komma genau.

5.4 Formatierte Ausgabe

Die Funktion
`number_format($zahl,$stellenzahl)`
gibt die Zahl `$zahl` auf `$stellenzahl` gerundet zurück. Der Wert der Variablen `$zahl` wird allerdings nicht verändert.
Beispiel:

```
$a = 1.4937;
echo number_format($a,2);
```

liefert den String 1.50 als Ausgabe.

6 Kontrollstrukturen

6.1 Verzweigungen: if

Bedingte Ausführung von Anweisungen

```
if (Ausdruck)
{
  Anweisung(en) // Ausführung nur, wenn Ausdruck "wahr"
}
```

Beispiel:

```
if ($a < $b)
{
    echo "$a ist kleiner als $b";
    $a = $a + $b;
}
```

6.1.1 Vergleichsoperatoren

Gleichheit (Zahlen und Zeichenketten)

```
$a = =  25;
```

Dieser Ausdruck ist wahr, wenn der Wert der Variablen `$a 25` ist.

Ungleichheit (Zahlen und Zeichenketten)

```
$a != 25;
```

Kleiner < kleiner oder gleich (nur für Zahlen): <=

Größer > größer oder gleich (nur für Zahlen): >=

6.1.2 Verzweigungen: if ... else

```
if (Ausdruck)
{
    Anweisung(en)1 // Ausführung nur, wenn Ausdruck "wahr"
}
else
{
Anweisung(en)2 // Ausführung, wenn Ausdruck „falsch"
}
```

Beispiel:

```
if ($a < $b)
    { echo "$a ist kleiner als $b"; }
else
    { echo "$a ist größer als $b"; }
```

6.1.3 Verzweigungen: switch...case

```
switch (Variable)
{
     case Bedingung1:
     Anweisungen
     break;

     case Bedingung2:
     Anweisungen
     break;
     ...
}
```

Achtung:
`break` am Ende von `case` ist wichtig, da sonst die folgenden `case`-Bedingungen auch ausgeführt werden.

6.2 Schleifen

Schleifen sind Anweisungen, die wiederholt ausgeführt werden.
Schleifen mit Abbruchbedingung: **while**
Die Anweisung wird so oft ausgeführt, bis die Abbruchbedingung eintritt.
Schleifen mit fester Anzahl: **for**
Es gibt eine feste Anzahl von Durchläufen der Anweisung.

6.2.1 while

Struktur: `while (Bedingung) {Anweisung }` Prüfung am Anfang!!

```
<?php
$zahl = 2;
while($zahl<100)
{
$quadrat = $zahl*$zahl;
echo "Das Quadrat von $zahl ist $quadrat<br>";
$zahl++;
}
?>
```

6.2.2 do-while

Struktur: `do {Anweisung } while (Bedingung)`
Prüfung am Ende, d.h. die Anweisung wird mindestens ein Mal durchlaufen.

```
<?php
$zahl = 2;
do {
$quadrat = $zahl*$zahl;
echo "Das Quadrat von $zahl ist $quadrat<br>";
$zahl++;
}
while($zahl<100)
?>
```

6.2.3 for

Struktur: `for (Anfangsbedingung; Endbedingung; Iteration) {Anweisung }`

```
<?php
for ($zahl=2;$zahl<=100; $zahl++)
{echo ("Das Quadrat von $zahl ist ".$zahl*$zahl.".<br>");
}
?>
```

7 Funktionen

nicht Funktion im mathematischen Sinn als eindeutige Abbildung

f: A $\mapsto$ B

besser: „Unterprogramm" oder „Prozedur" bzw. „Funktionsprozedur"

Ziel: Zerlegung und Strukturierung umfangreicher Programme
zentrales Speichern häufig benötigter Befehlsfolgen für verschiedene Programme

7.1 Funktionstypen

Funktionen ohne Parameter:
Hier wird immer eine bestimmte Befehlsfolge unabhängig von äußeren Einflüssen ausgeführt.
Funktionen mit Parameter(n):
Die „äußeren" Parameter beeinflussen die Ausführung der Befehlsfolge.
Funktionen mit Rückgabewert:
Die aufrufende Stelle erhält ein Ergebnis der Funktion, mit dem weitergearbeitet werden kann.

7.2 vordefinierte Funktionen

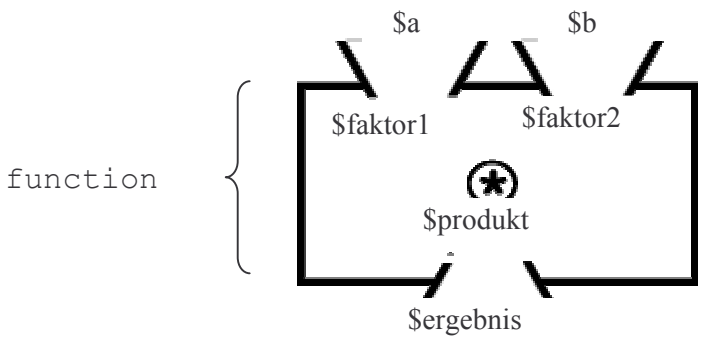
String-Funktionen zur Veränderung von Zeichenketten
mathematische Funktionen
Array-Funktionen
Datums- und Zeit-Funktionen
Netzwerk-Funktionen
Datenbank-Funktionen
…

7.3 selbstdefinierte Funktionen

Definition von Funktionen:

```
function fname($param1,…,$paramn)   Name
{
// Anweisungen der Funktion         optional
return $Rueckgabewert;
}
```

Gültigkeit von Variablen `$faktor1, $faktor2` und `$produkt` sind nur innerhalb der Funktion gültig



Am besten erfolgt die Definition der Funktion vor deren Aufruf (z.B. im `<head>`-Bereich).

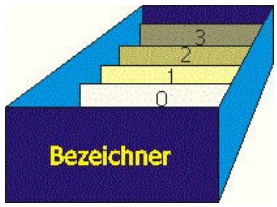
8 Arrays

Ein Array ist ein Feld von Variablen.
Zweck: Gruppierung zusammengehörender Werte unter einem Namen anstelle vieler einzelner Variablen (z.B. Daten eines Messreihe).
`array()` ist ein Sprachkonstrukt zur Erzeugung eines Feldes von Werten (keine echte PHP-Funktion).
Es gibt zwei Arten von Arrays in PHP:
indizierte (oder numerische) Arrays
assoziative Arrays

8.1 Indizierte Arrays

Einem numerischen Index (= Schlüssel) werden Werte zugewiesen.

```
$fächer   [0] =     "Mathematik";
$fächer   [1] =     "Englisch";
$fächer   [2] =     "Latein";  …
    ↑     ↑
    Schlüssel   Wert
```



Ausgabe mit `for`-Schleife:

```
for ($i=0;$i<count($fächer);$i++)
{echo "$fächer[$i]<br>";}
```

`count()` zählt die Anzahl der Elemente im Array.

Schlüssel automatisch erstellen:

```
$b[] = 5;
$b[] = 10;
$b[] = 15;
```

ergibt: `$b[0]=5  $b[1]=10  $b[2]=15`

Kurzform:

```
$b = array(0,1,4,9);
```

ergibt: `$b[0]=0  $b[1]=1  $b[2]=4  $b[3]=9`

Schlüssel explizit zuweisen:

```
$c=array(1=>20, 5=>30, 7=>35);
```

8.2 Assoziative Arrays

Der Index ist ein String.

```
$formularfeld["anrede"]    =   "Frau";
$formularfeld["vorname"]   =   "Emilie";
```

```
$formularfeld["nachname"] =  "Tischbein"; …
```

Schlüssel Wert

Ausgabe mit `foreach`-Schleife:

```
foreach($formularfeld as $kopie)
{ echo "$kopie "; }
```

ergibt: Frau Emilie Tischbein

`foreach` ist eine Schleife zum Abarbeiten von Anweisungen für jedes Element des Arrays. Der Reihe nach wird der Wert des aktuellen Array-Elements in `$kopie` kopiert und damit die Anweisungen ausgeführt. Die Werte im Array ändern sich dabei nicht.

foreach-Schleife mit Ausgabe des Schlüssels

```
foreach($formularfeld as $schlüssel => $kopie)
```

Wertepaar im Array

```
{ echo "\$formularfeld[$schlüssel] ist: $kopie<br>"; }
```

ergibt: $formularfeld[anrede] ist: Frau
$formularfeld[vorname] ist: Emilie
$formularfeld[nachname] ist: Tischbein

9 Auswertung von Formularen

Formulare werden als HTML geschrieben (vgl. Einführung HTML)

```
<input type="text" name="zahl-1" >
<input type="text" name="zahl-2" >
```

Mehrere Formularfelder bilden ein Array.
PHP-Skripte können über den name-Tag auf den Inhalt des Formularfelds zugreifen.

```
$zahl=$_GET[ "zahl-1" ]
$zahl=$_GET[ "zahl-2" ]
```

Es gibt also zwei Dateien: eine HTML-Seite mit dem Formular. Über `action="auswertung.php"` ruft sie die gewünschte PHP-Datei auf, die die Auswertung übernimmt und das Ergebnis ausgibt.

9.1 Daten-Übergabe an PHP-Skript

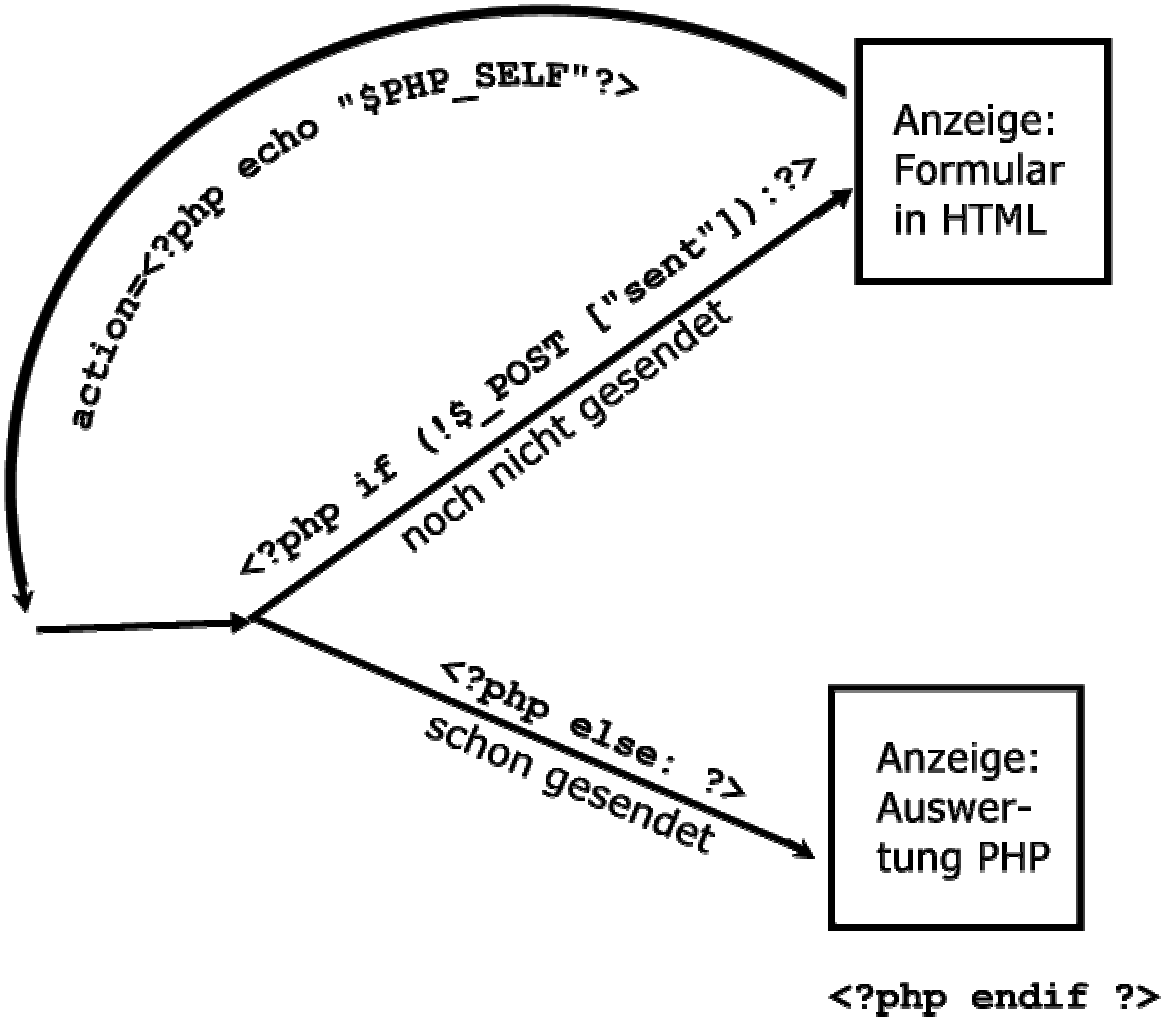
Beim Abschicken des Formulars werden die Daten **als Strings** (auch Zahlen !!!) in einem Array `$_GET` oder `$_POST` abgelegt.

```
$zahl = $_GET  [ "zahl-1 ]
$zahl = $_POST [ "zahl-2 ]
```

php-Variable `name` im HTML-Formular

9.2 Formular in PHP

Bisher: HTML-Datei "`formular.htm`" ruft PHP-Datei "`skript.php`" auf.
Nachteil: Man muss zwei Dateien pflegen.
Andere Möglichkeit: PHP-Skript schreibt zuerst das Formular und wertet erst aus, wenn es abgeschickt wurde.




```
<?php if (!$_POST ["sent"]):?>
<form name="form1" method="post"
    action=<?php echo "$PHP_SELF"?> >
   Bitte geben Sie Ihren Namen ein:
<input type="text" name="name"><br>
  <input type="hidden" name="sent" value="1">
  <input type="submit" name="OK" value="OK">
</form>
<?php else: ?>
<?php $name=$_POST["name"];
    echo "Name: ".$name;
?>
<?php endif ?>
```

10 Dateizugriff

10.1 Dateizugriff: fopen(), fgets()

```
$meld=fopen($datei,modus);
```

Datei öffnen: - Lesen (modus="r")
- Schreiben (modus="w")
- Anhängen (modus="a")

In `$meld` werden die Zugriffsdaten (Dateiname, Modus ...) gespeichert.
`$text=fgets($meld,[$anzahl_z])`
Auslesen einer Zeile (oder von `$anzahl_z` Zeichen) aus zuvor geöffneter Datei (`$meld` existiert)
Setzen des Dateizeigers eine Zeile tiefer

10.2 Dateizugriff: fwrite(),fclose()

`fwrite($meld,$text)`
Schreiben des Strings `$text` in die geöffnete Datei (Modus "a" oder "w")
Bei Erfolg gibt `fwrite()` `TRUE` zurück, andernfalls `FALSE`.
`fclose($meld)`
Schließen der zuvor geöffneten Datei (`$meld` existiert)
Bei Erfolg gibt `fclose()` `TRUE` zurück, andernfalls `FALSE`.

10.3 Beispiel Dateizugriff

```
<?php
$meld= fopen("text.txt","r");
while (!feof($meld))
 { $zeile= fgets($meld);
   echo "Zeile: $zeile<br>";
 }
fclose($meld);
?>
```

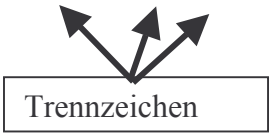
Prüfung, ob Dateiende erreicht wurde

Bei jedem Durchlauf der `while`-Schleife wird eine Zeile in `$zeile` gespeichert und Zeile für Zeile ausgegeben.

10.4 Zerlegung von Strings

Ein String `$s` besteht aus mehreren Substrings, die durch ein bestimmtes Zeichen (z.B. einen Strichpunkt) von einander getrennt sind.
`$teil = explode(";",$s)` zerlegt `$s` in Teilstrings und legt diese im Array `$teil` ab.
Beispiel für String-Zerlegung
Zeile aus einer Liste für Lehrersprechzeiten:

```
$s="Mayer;Katja;Mo;3. Std.";
```




```
$teil = explode(";",$s);
$teil[0] = "Mayer";
$teil[1] = "Katja";
$teil[2] = "Mo";
$teil[3] = "3. Std.";
```

10.5 Zugriff auf Dateien: file()

```
$zeilen = file($datei)
```

liest die ganze Datei `$datei` zeilenweise in das Array `$zeilen`.

```
<?php
$a = file("text.txt");
foreach($a as $t)
{ echo "Zeile: $t<br>";}
?>
```

`$a[0]` ➔ Zeile: Das ist ein Text in der ersten Zeile.
`$a[1]` ➔ Zeile: Und hier steht der Text der zweiten Zeile.

Literaturempfehlungen

PHP Galileo <Openbook> http://www.galileocomputing.de/
Für Einsteiger. Mit HTML-Wiederholung, Übungen und Lösungen.

PHP Handbuch http://www.php.net/docs.php
Für erfahrene Programmierer, die es genau wissen wollen. Zum Teil etwas unbeholfen aus dem Englischen übersetzt. Keine Suchfunktion.

SELF-PHP http://www.selfphp4.de/
Zum Nachschlagen. Nach dem berühmten Vorbild SELF-HTML. Wenig Erklärung , aber ausführliche Übersicht.