

PHP

In diesem Teil der Veranstaltung¹ wird in die PHP-Programmierung eingeführt. Voraussetzung ist, dass bereits eine Programmiersprache bekannt ist. Es ist also **keine** Einführung in das Programmieren.

Inhaltsverzeichnis

1Historie.....	5
2Grundlagen.....	9
3Variablen, Konstanten und Datentypen.....	17
4Funktionen und Gültigkeitsbereich von Variablen	26
5Schleifen.....	43
6Bedingungen.....	44
7Rekursion.....	49
8Arrays.....	51

¹Der erste Teil der Veranstaltung ist an den php-Kurs <http://www.php-experts.de/> angelehnt.

9	Formulare.....	59
10	Datenbankzugriff.....	70
10.1	Connect.....	70
10.2	Select.....	72
10.3	Insert.....	76
10.4	Delete.....	77
11	Sessionverwaltung.....	79
11.1	Cookies.....	79
11.2	Session	81
12	Zugriff auf Server-Ressourcen.....	90
12.1	Datei Upload (PC->Server).....	90
12.2	Dateien und Verzeichnisse.....	93
12.2.1	Schreiben einer Dateien.....	93
12.2.2	Lesen aus einer Datei.....	94
12.2.3	Löschen einer Datei	98

12.3Mail.....	98
12.4Serverprogramme ausführen.....	100
13Programmieren im Grossen.....	102
13.1Strukturierung durch mehrere Dateien.....	102
13.2Objektorientierung.....	105
13.2.1.OOP als Klassifizieren von Problemen.....	105
13.2.2.Probleme der Prozeduralen Programmierung.....	106
13.2.3.Klassen und Objekte in PHP.....	106
13.2.4.Objektserialisierung.....	119
14PHP und Sicherheit.....	122
14.1register_globals.....	122
14.2Validieren von Eingaben.....	124
14.3SQL-Injection.....	125
14.4Command Injection.....	128
14.5 Cross Site Scripting (XSS).....	129

15PHP und JavaScript.....131

1 Historie²

PHP ist als Erweiterung von Web-Servern entstanden, um relativ einfach dynamische Web-Seiten erstellen zu können. Dynamisch bedeutet, dass der Inhalt der Web-Seite von Daten, die in Backend-Systemen (wie Datenbanken) gespeichert sind, abhängen.

PHP/FI

PHP ist der Nachfolger eines älteren Produktes, PHP/FI. PHP/FI wurde **1995** von **Rasmus Lerdorf** geschaffen. Ursprünglich war PHP/FI ein **Set von Perl Skripten** zur Erfassung der Zugriffe auf seinen Webauftritt. Er nannte dieses Set von Skripten 'Personal Home Page Tools'. Als dann mehr Funktionalität benötigt wurde, schrieb Rasmus eine viel größere **Umsetzung in C**, welche auch mit Datenbanken kommunizieren konnte, und den Benutzern die Entwicklung einfacher dynamischer Webapplikationen ermöglichte. Rasmus entschloss sich, den Sourcecode von PHP/FI zu veröffentlichen, sodass ihn jeder benutzen, von Fehlern bereinigen, und weiter entwickeln konnte.

PHP/FI stand für **P**ersonal **H**ome **P**age / Forms Interpreter, und beinhaltete manches an Funktionalität des PHP wie wir es heute kennen. Es besaß Variablen wie in Perl, eine automatische Interpretation von Formularvariablen und eine in HTML eingebettete Syntax. Die Syntax selbst war der von Perl ähnlich, wenn auch viel eingeschränkter, einfach, und ziemlich inkonsistent.

1997 war PHP/FI 2.0, die zweite Überarbeitung der C Implementierung, Kult für einige tausend Benutzer weltweit (geschätzt). Etwa 50.000 Domains berichteten PHP/FI 2.0 installiert zu ha-

² aus <http://de.php.net/history>

ben, was mit ca. 1% der Domains im Internet zu Buche schlug. Obwohl manche Leute diesem Projekt ein Stück Code beisteuerten, war es insgesamt immer noch ein Ein-Mann-Projekt.

PHP/FI 2.0 wurde im November **1997** offiziell **freigegeben**, nachdem es die meiste Zeit seines Lebenszyklus als verschiedene Betaversionen verbracht hatte. Es wurde kurz danach von den ersten Alphaversionen von PHP 3 abgelöst.

PHP 3

PHP 3.0 war die erste Version, die dem heutigen PHP sehr gleicht. Es wurde 1997 von **Andi Gutmans** und **Zeev Suraski** neu geschrieben, nachdem PHP/FI 2.0 ihrer Meinung nach für die Entwicklung ihrer eCommerce Applikation für ein Universitätsprojekt arbeiteten, viel zu schwach war. Auf die Basis der bestehenden Benutzer von PHP/FI aufbauend, entschieden sich Andi, Rasmus und Zeev zur Kooperation, und kündigten PHP 3.0 als den offiziellen Nachfolger von PHP/FI 2.0 an, und die Entwicklung von PHP/FI 2.0 wurde größtenteils eingestellt.

Eine der größten **Stärken** von PHP 3.0 waren die starken Erweiterungsmöglichkeiten. Zusätzlich zu der soliden Infrastruktur für eine Menge an **Datenbanken**, Protokollen und APIs, lockten vor allem die Erweiterungsmöglichkeiten von PHP 3 dutzende von Entwicklern an, welche sich beteiligten, und neue Erweiterungsmodule einbrachten. Möglicherweise war das der Schlüssel zu dem gewaltigen Erfolg von PHP 3.0. Weitere besondere Merkmale von PHP 3.0 waren die Unterstützung für **objektorientierte** Syntax und die viel bessere sowie konsistentere Sprachsyntax.

Die gesamte neue Sprache wurde unter einem neuen Namen veröffentlicht, welche die im Namen PHP/FI 2.0 vorhandene Implizierung einer eingeschränkten persönlichen Nutzung beseitigte. Es wurde einfach 'PHP' genannt, ein rekursives Akronym für PHP: Hypertext Preprocessor.

Gegen Ende 1998 wuchs PHP auf eine installierte Basis von (geschätzten) zehntausenden Benutzern und hunderttausenden Websites, auf denen PHP installiert war, heran. An seinem Höhepunkt war PHP 3.0 auf etwa 10% der Webserver im Internet installiert.

PHP 3.0 wurde im Juni **1998** nach einer neunmonatigen öffentlichen Testphase offiziell freigegeben.

PHP 4

Im Winter 1998, kurz nach der offiziellen Freigabe von PHP 3.0, begannen **Andi Gutmans** und **Zeev Suraski** den **Kern** von PHP **umzuschreiben**. Die Ziele waren eine verbesserte Leistung von komplexen Applikationen, und eine verbesserte Modularität des Basiscodes. Solche Applikationen wurden durch die neuen Leistungsmerkmale von PHP 3.0, der Unterstützung einer großen Auswahl von Datenbanken und APIs von Drittanbietern möglich gemacht, aber PHP 3.0 war nicht dafür entworfen, solche komplexen Applikationen auch effizient zu handhaben.

Die neue **Engine**, titulierte als '**Zend Engine**' (aus den Vornamen Zeev und Andi gebildet), entsprach diesen Zielen im Design erfolgreich, und wurde zum ersten Mal Mitte 1999 eingeführt. PHP 4.0, das auf dieser Engine, verbunden mit einer großen Auswahl an zusätzlichen Leistungsmerkmalen basiert, wurde im Mai 2000 offiziell freigegeben, fast zwei Jahre nach seinem Vorgänger PHP 3.0. Zusätzlich zu der stark verbesserten Leistung, inkludierte PHP 4.0 andere wichtige **Leistungsmerkmale**, wie Unterstützung für viele **weitere Webserver, HTTP-Sessions, Ausgabepufferung**, sicherere Wege im Umgang mit Benutzereingaben, und verschiedene neue Sprachkonstrukte.

PHP 4 ist die derzeit aktuellste freigegebene Version von PHP. Die Arbeit an der Modifikation und Verbesserung der Zend Engine zur Integration der neuen für PHP 5.0 entworfenen Leistungsmerkmale hat bereits begonnen.

Heute wird PHP von (schätzungsweise) hunderttausenden Entwicklern verwendet, und es wird von mehreren Millionen Sites berichtet, auf welchen PHP installiert ist, was mit über 20% der Domains im Internet zu Buche schlägt.

Das Entwicklerteam von PHP umfasst dutzende Entwickler, sowie dutzende andere, welche an PHP verwandten Projekten wie PEAR oder dem Dokumentationsprojekt arbeiten.

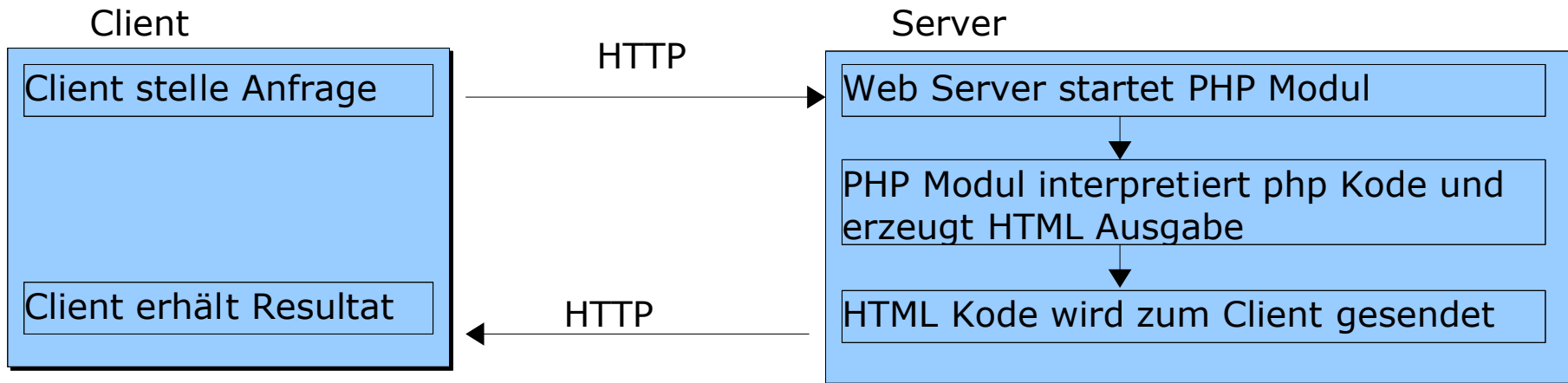
PHP 5

Die Zukunft von PHP ist hauptsächlich von seinem Kern, der Zend Engine, geprägt. PHP 5 wird die neue Zend Engine 2.0 enthalten. Weitere Informationen zur Zend Engine finden Sie auf der php Web-Site.

2 Grundlagen

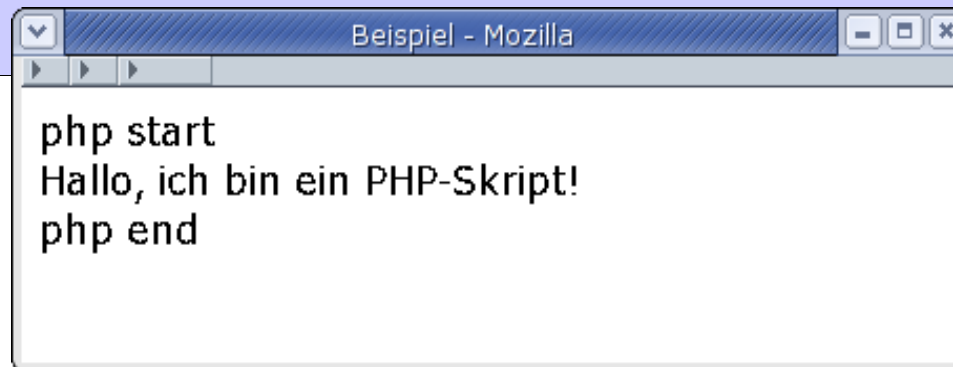
PHP ist eine **Skriptsprache**, die hauptsächlich im Web-Umfeld verwendet wird. **Voraussetzung** ist also ein **Interpreter**, der die PHP-Anweisungen liest, interpretiert und ausführt.

PHP Programme lösen mehr und mehr CGI Skripten ab, da die Sprache **in HTML integriert** ist und ein PHP Programm nicht als eigenständiger Prozess (so wie CGI) abläuft.



Das erste Programm:

```
$ cat erstesProgramm.php
<html>
  <head>
    <title>Beispiel</title>
  </head>
  <body>
    php start
    <br>
    <?php
      echo "Hallo, ich bin ein PHP-Skript!";
    ?>
    <br>
    php end
  </body>
</html>
$
```



Wir verwenden **Apache** mit der **PHP-Erweiterung**. Damit ist es möglich, **PHP-Kode in HTML-Seiten einzubetten**, der dann interpretiert werden kann.

```
$ cat echo.php
<html>
<head><title>echo</title></head>
<body>
<?php      // Ab hier beginnt PHP, nur noch PHP-Syntax ist erlaubt
           // dieser Teil wird vom PHP-Interpreter interpretiert
           // auf diese Weise bettet man PHP in HTML ein
           echo "Hallo "
?>
,
<?php
           echo " Welt!<br>"
?>
</body>
</html>
$
```



Öffnen und Schließen des PHP-Teils ist **jederzeit möglich**. Der PHP-Interpreter wird serverseitig ausgeführt. Die PHP-Teile werden vor dem Senden der Seite zum Client durch das Ergebnis des Ausführens des Skriptes ersetzt.

Ein PHP-Programm kann auch ohne Browser ausgeführt werden, dann muss es dem PHP-Interpreter als Argument mitgegeben werden.

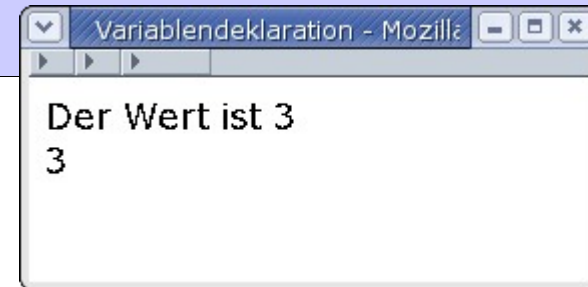
```
$ cat halloWelt.php
<?php
    echo "Hallo, Welt!\n"
?>
$ php halloWelt.php
Content-type: text/html
X-Powered-By: PHP/4.3.4RC1

Hallo, Welt!
$
```

In den folgenden Beispielen wird der umgebene HTML-Kode meistens ausgeblendet!

Variablen werden **nicht** deklariert. Alle Variablennamen beginnen mit einem '\$' Zeichen. Innerhalb von Anführungszeichen (") werden Variablen durch ihre Inhalte ersetzt.

```
$ cat Variablendeklaration.php
<?php
$variable = 3;    // Deklaration durch Wertzuweisung
echo "Der Wert ist $variable <br>";
echo $variable;
?>
$
```



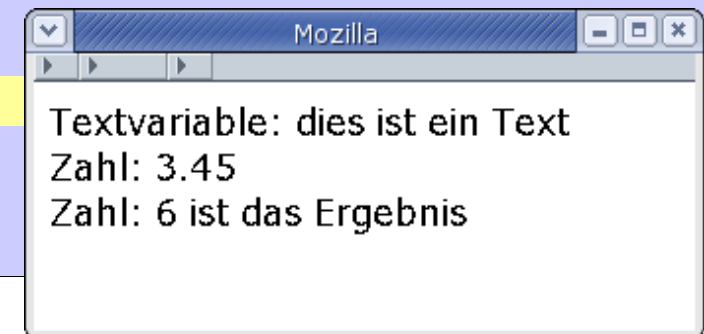
Kommentare werden C-like angegeben, also durch /* ... */ oder durch //.

3 Variablen, Konstanten und Datentypen

Der Typ einer Variablen ergibt sich implizit durch den Typ des Wertes, der der Variablen zuletzt zugewiesen wird. Es gibt drei Grundtypen:

- double,
- string und
- integer

```
$ cat Grundtypen.php
<?php
$zahl = 3.45; // double
$variable = 'dies ist ein Text'; // string
echo "Textvariable: $variable<br>";
echo "Zahl: $zahl<br>";
$variable = 3; // integer
echo "Zahl: ", $variable + 3, " ist das Ergebnis";
?>
$
```



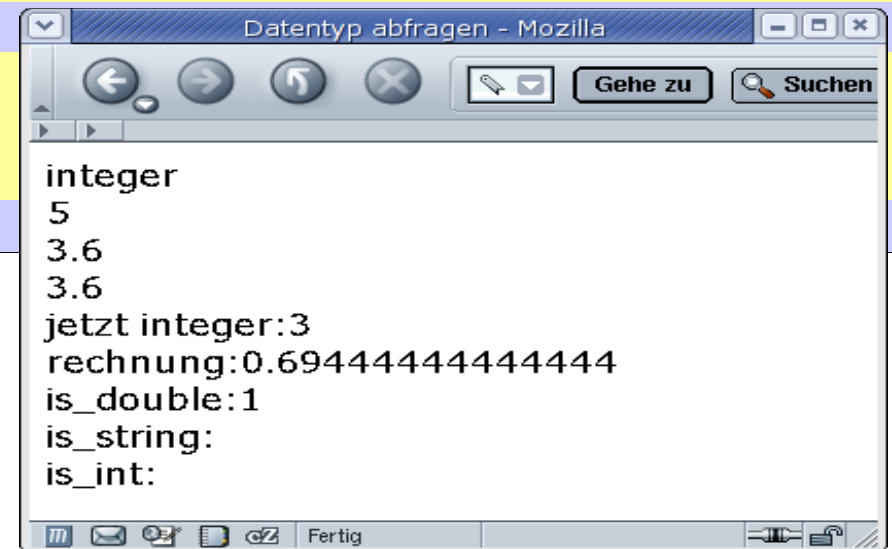
Der Typ einer Variable kann sich ändern.

```
$ cat Typaenderung.php
<?php
$zahl1=3;          // int
echo $zahl1;
echo "<br>";
$zahl2=3.5;
echo $zahl2;
echo "<br>";
$zahl1=$zahl2;    // int->double
echo $zahl1;
echo "<br>";
$zahl1="sdfsfas"; // double->string
echo $zahl1;
echo "<br>";
?>
$
```



Den implizit gesetzten Datentyp kann man mit `gettype` abfragen. Mit `settype` kann man den Datentyp explizit setzen, das gilt allerdings nur bis zur nächsten Zuweisung. Der Datentyp kann auch über verschiedenen boolsche Funktionen ausgewertet werden.

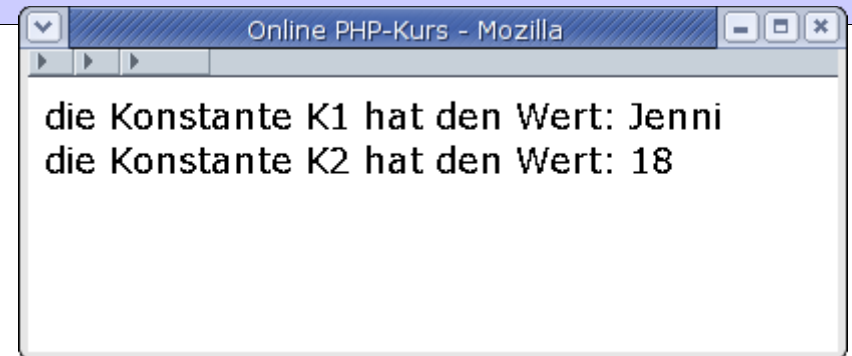
```
$ cat AbfrageSetzenDatentyp.php
$zahl1=5;
echo gettype($zahl1), "<br>"; // Datentyp abfragen
echo $zahl1, "<br>";
$zahl2=3.6;
echo $zahl2, "<br>";
$zahl1=$zahl2; // -> int
echo $zahl1, "<br>";
settype($zahl1, "integer"); // Datentyp setzen
echo "jetzt integer:", $zahl1, "<br>";
$zahl1 = 2.5; // jetzt wieder double
echo "rechnung:", $zahl1/$zahl2, "<br>";
echo "is_double:", is_double($zahl1), "<br>";
echo "is_string:", is_string($zahl1), "<br>";
echo "is_int: ", is_int($zahl1), "<br>";
$
```



```
Datentyp abfragen - Mozilla
integer
5
3.6
3.6
jetzt integer:3
rechnung:0.6944444444444444
is_double:1
is_string:
is_int:
```

Konstanten können mit "define" definiert werden. Ihr Wert ändert sich nicht und ihrem Namen ist kein "\$" vorangestellt.

```
$ cat Konstanten.php
<?php
define ("K1", "Jenni" );
define ("K2", 18);
echo "die Konstante K1 hat den Wert: ",K1, "<br>";
echo "die Konstante K2 hat den Wert: ",K2, "<br>";
?>
$
```



Hörsaalübung:

Schreiben Sie ein php-Programm, in dem je eine Variable und eine Konstante aller Grundtypen von php definiert werden, die dann ausgegeben werden.

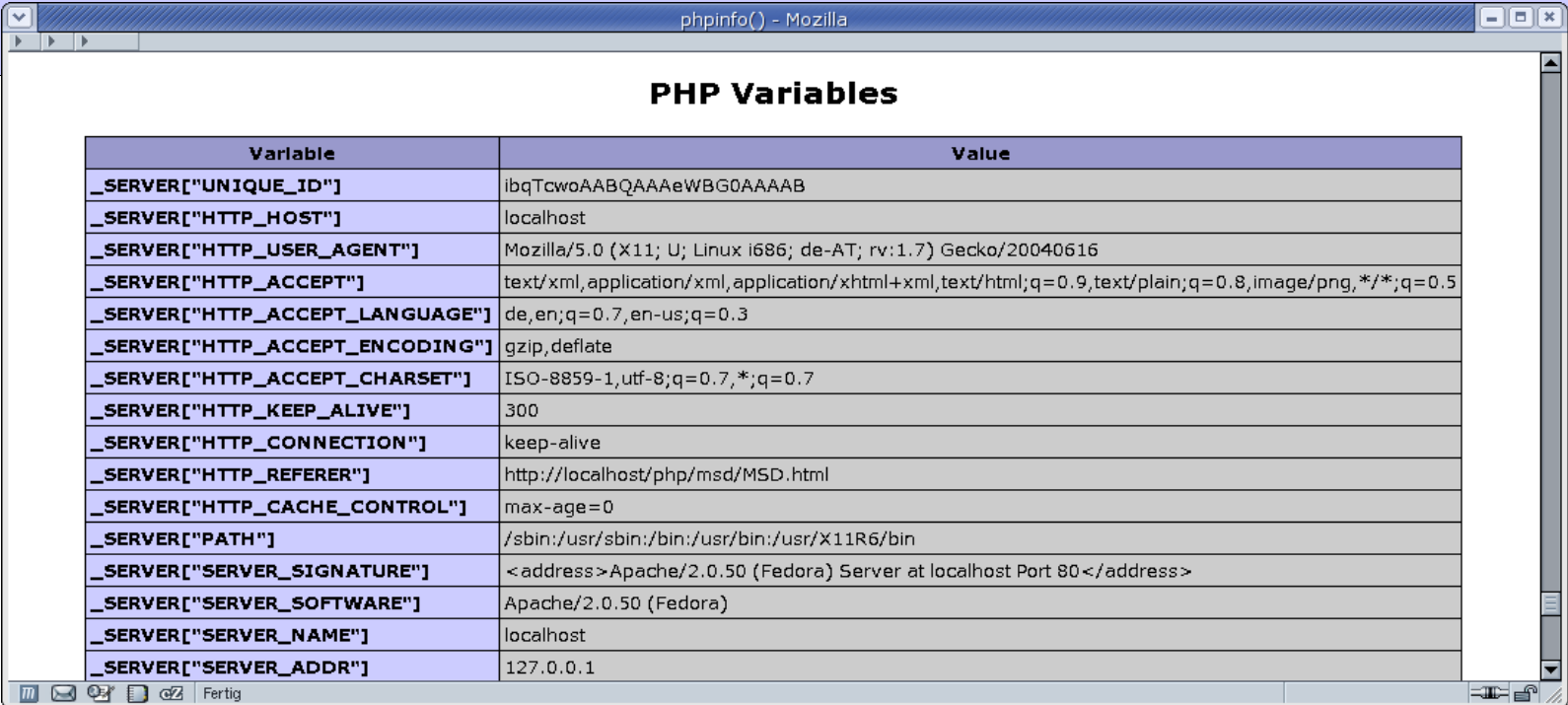
Rufen Sie das Programm von der Kommandozeile und aus ihrem Browser heraus auf.

4 Funktionen und Gültigkeitsbereich von Variablen

PHP erlaubt als Strukturierungsmittel Funktionen. Eine Funktion wird wie in Java oder C aufgerufen durch Angabe des Funktionsnamen mit (ev. leerer) Parameterliste.

Funktionsaufruf:

```
$ cat phpinfo.php
<?php
phpinfo(); // Funktionsaufruf
?>
$
```



Variable	Value
_SERVER["UNIQUE_ID"]	ibqTcwoAABQAAeWBG0AAAAB
_SERVER["HTTP_HOST"]	localhost
_SERVER["HTTP_USER_AGENT"]	Mozilla/5.0 (X11; U; Linux i686; de-AT; rv:1.7) Gecko/20040616
_SERVER["HTTP_ACCEPT"]	text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
_SERVER["HTTP_ACCEPT_LANGUAGE"]	de,en;q=0.7,en-us;q=0.3
_SERVER["HTTP_ACCEPT_ENCODING"]	gzip,deflate
_SERVER["HTTP_ACCEPT_CHARSET"]	ISO-8859-1,utf-8;q=0.7,*;q=0.7
_SERVER["HTTP_KEEP_ALIVE"]	300
_SERVER["HTTP_CONNECTION"]	keep-alive
_SERVER["HTTP_REFERER"]	http://localhost/php/msd/MSD.html
_SERVER["HTTP_CACHE_CONTROL"]	max-age=0
_SERVER["PATH"]	/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin
_SERVER["SERVER_SIGNATURE"]	<address>Apache/2.0.50 (Fedora) Server at localhost Port 80</address>
_SERVER["SERVER_SOFTWARE"]	Apache/2.0.50 (Fedora)
_SERVER["SERVER_NAME"]	localhost
_SERVER["SERVER_ADDR"]	127.0.0.1

Im folgenden Beispiel ist eine **Deklaration** einer Funktion ohne Parameter und Rückgabewerte zu sehen.

```
$ cat Funktionsdefinition.php
<?php
function ausgabe() {           // Funktionsdefinition
    echo "TEST<br>test2<br>";
}

ausgabe();                     // Funktionsaufruf
?>
$
```

Funktionen können Rückgabewerte haben:

```
$ cat Return.php
<?php
function myPi() {              // Funktionsdefinition
    return 3.14;
}

echo myPi();                   // Funktionsaufruf
?>
$
```

Funktionen können **Parameter** besitzen. Der Typ der Parameter wird nicht explizit deklariert, die **Typen der Parameter** ergeben sich **implizit beim Aufruf** der Funktion.

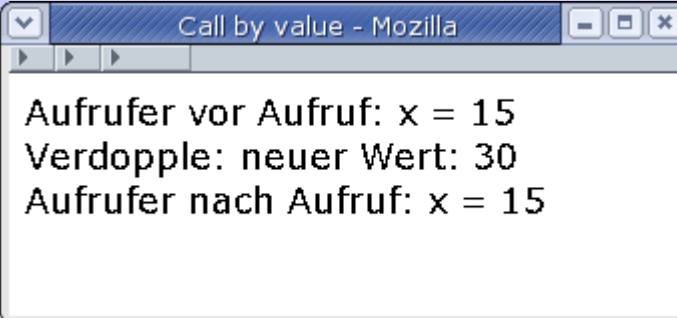
```
$ cat Parameter.php
<?php
function ausgabe($wert1, $wert2) {
    echo "ausgabe: wert1=$wert1 \n <br> Wert2=$wert2 \n <br>";
}

ausgabe("hier", 3.4);           // string, double
ausgabe("dort", "x");          // string, string
$zahl=2;
ausgabe("am schluss", $zahl); // string, int
?>
$
```

Voreingestellt erfolgt die **Parameterübergabe** durch einen "**Call by value**"-Mechanismus, d.h. es wird eine lokale Kopie des aktuellen Parameter angelegt und an die Funktion übergeben. Daher kann die Funktion **keine** Werte des Aufrufers verändern.

```
$ cat CallByValue.php
<?php
function Verdopple($wert) {
    $wert = $wert * 2;
    echo "Verdopple: neuer Wert: $wert<br>";
}

$x = 15;
echo "Aufrufer vor Aufruf: x = $x <br>";
Verdopple($x);
echo "Aufrufer nach Aufruf: x = $x <br>";
?>
$
```

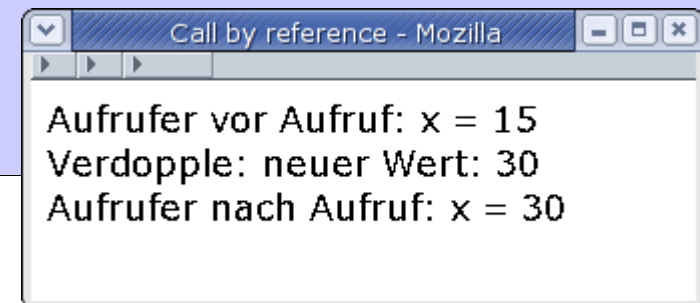


```
Call by value - Mozilla
Aufrufer vor Aufruf: x = 15
Verdopple: neuer Wert: 30
Aufrufer nach Aufruf: x = 15
```

Mit dem "&"-Zeichen vor dem Variablennamen kann man einen **"Call-by-reference"** erzwingen. Wird die formale Parameter in der Funktion verändert, so verändert sich auch der Inhalt der übergebenen Variablen im aufrufenden Programm.

```
$ cat CallByReference.php
<?php
function Verdopple(&$wert) {
    $wert = $wert * 2;
    echo "Verdopple: neuer Wert: $wert<br>";
}

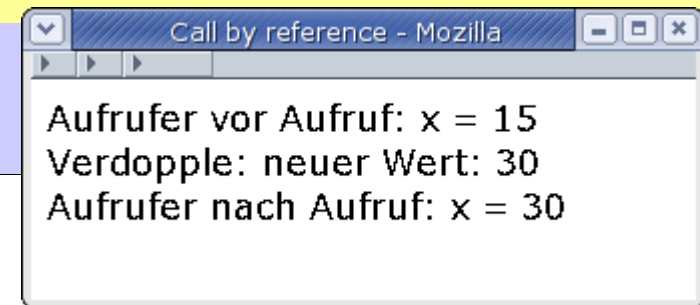
$x = 15;
echo "Aufrufer vor Aufruf: x = $x <br>";
Verdopple($x);
echo "Aufrufer nach Aufruf: x = $x <br>";
?>
$
```



Man kann den "**Call-by-reference**" auch im aufrufenden Programm **erzwingen**.

```
$ cat CallByReference.php
<?php
function Verdopple($wert) {
    $wert = $wert * 2;
    echo "Verdopple: neuer Wert: $wert<br>";
}

$x = 15;
echo "Aufrufer vor Aufruf: x = $x <br>";
Verdopple(&$x);
echo "Aufrufer nach Aufruf: x = $x <br>";
?>
$
```



Tafelübung:

Welche Ausgabe erzeugt folgendes Programm:

Funktionen und Gültigkeitsbereich von Variablen

```
<?php
function incr1($wert) {
    $wert++;
    return $wert;
}
function incr2(&$wert) {
    $wert++;
    return $wert;
}

$x = 1;
echo " vor Aufruf incr1: x = $x \n";
$y = incr1($x);
echo "nach Aufruf incr1: x = $x y = $y \n";

$x = 1;
echo " vor Aufruf incr2: x = $x \n";
$y = incr2($x);
echo "nach Aufruf incr2: x = $x y = $y\n";

$x = 1;
echo " vor Aufruf incr1: x = $x \n";
$y = incr1(&$x);
echo "nach Aufruf incr1: x = $x y = $y\n";
?>
```

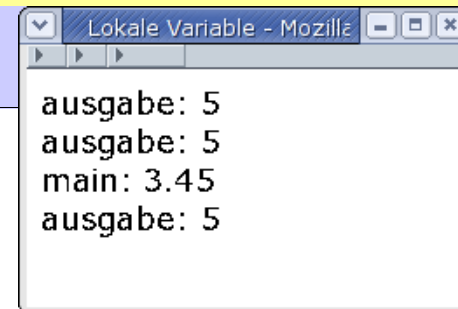
Der Gültigkeitsbereich von Variablen ist in PHP wie folgt geregelt:

Variablen in Funktionen sind lokal, das heißt Variablen gleichen Namens in einer anderen Funktion beeinflussen nicht den Inhalt der Variablen in der ursprünglichen Funktion.

Im folgenden Beispiel behält die Variable `$zahl` im Hauptprogramm stets den Wert 5.0.

```
$ cat LokaleVariable.php
<?php
function ausgabe(){
    $zahl = 5.0;        // lokal in ausgabe
    echo "ausgabe: $zahl<br>";
}

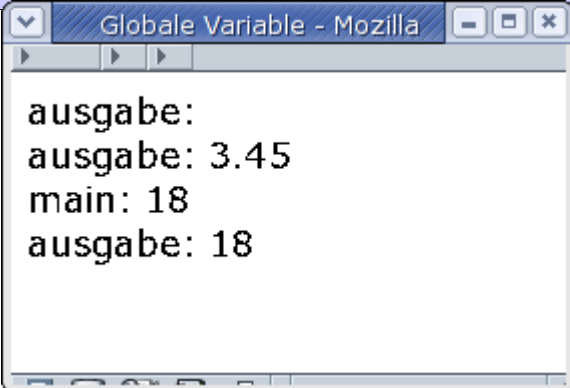
ausgabe();
$zahl = 3.45;
ausgabe();
echo "main: $zahl<br>";
ausgabe();
?>
$
```



Durch `global` wird definiert, dass in einer Funktion die Variable im Hauptprogramm gemeint ist.

```
$ cat Global.php
<?php
function ausgabe() {
    global $zahl;
    echo "ausgabe: $zahl<br>";
    $zahl = 18;
}

ausgabe();
$zahl = 3.45;
ausgabe();
echo "main: $zahl<br>";
ausgabe();
?>
$
```



```
ausgabe:
ausgabe: 3.45
main: 18
ausgabe: 18
```

Auch ohne die Variable als global zu deklarieren kann auf die globale Variable `$zahl` über `$GLOBALS["zahl"]` zugegriffen werden.

```
$ cat Globals.php
<?php
function ausgabe() {
    echo "ausgabe: ", $GLOBALS["zahl"], "<br>";
}

$zahl = 3.45;
ausgabe();
?>
$
```



5 Schleifen

Schleifen sind in PHP so wie in Java oder C++ in unterschiedlicher Ausprägung möglich, als Schleifen mit und ohne abweisendem Charakter.

```
$ cat while.php
<?php
function fac($i) {                                // Berechnung Fakultät von i
    $res = 1;
    while ($i > 0) {
        $res = $res * $i;
        $i = $i - 1;
    }
    return $res;
}

echo fac(3);
?>
$
```

Hörsaalübung

Formulieren Sie die Funktion fac mit einer do-while-Schleife, danach mit einer for-Schleife!

6 Bedingungen

Variablen können initialisiert worden sein, oder man hat es vergessen. Es gibt in PHP **keine** Fehlermeldung wenn man auf nicht initialisierte Variablen zugreift. Mit `isset` kann abgefragt werden, ob eine Variable initialisiert ist.

Boolsche Variablen werden als integer gespeichert und mit `is_bool` kann abgefragt werden, ob eine Variable einen als bool-Wert interpretierbaren Inhalt hat.

Bedingungen werden mittels if-else sowie switch-case wie in Java bzw. **C++** kodiert.

Bedingungen

```
$ cat abfragen.php
<?php
// Mit isset kann abgefragt werden, ob eine Variable gesetzt ist
echo "var: " , $var, "<br>";
echo "isset: ",isset($var),"<br>";
$var="";
// mit empty() kann abgefragt werden, ob eine Variable leer ist.
// 0 oder "" werden z. B. als leer interpretiert
echo "isset: ",isset($var),"<br>";
echo "empty: ",empty($var),"<br>";
$var="0";
echo "isset: ",isset($var),"<br>";
echo "empty: ",empty($var),"<br>";
$var=(3>4);
echo "\nvar: $var\n<br>";
echo "isset: ",isset($var),"<br>";
echo "empty: ",empty($var),"<br>";
echo "is_bool: ", is_bool($var),"<br>";
$var=(4>3);
echo "\nvar: $var\n<br>";
echo "isset: ",isset($var),"<br>";
echo "empty: ",empty($var),"<br>";
// mit is_bool kann abgefragt werden, ob eine Variable einen bool-Wert hat.
echo "is_bool: ", is_bool($var),"<br>";
?>
```

Die **Behandlung** von **true** und **false** verdeutlicht folgendes Beispiel (BehandlungTrueFalse.php)

```
<?php
// eine nicht gesetzte Variable wird als false interpretiert
if($var)
    echo("ja<br>");
else
    echo("nein<br>");

// eine leere Variable wird als false interpretiert
$var="";
if($var)
    echo("ja<br>");
else
    echo("nein<br>");

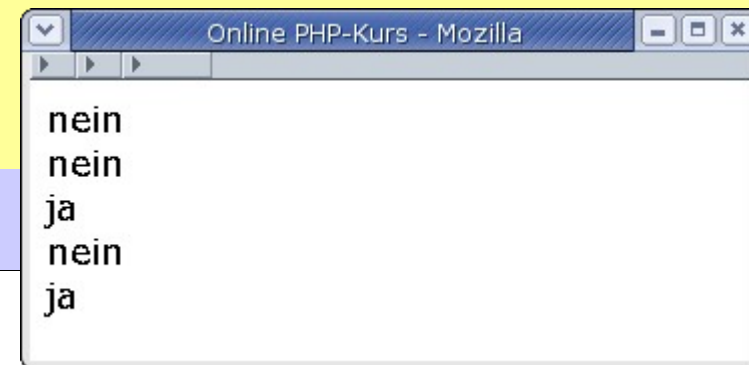
// eine gesetzte Variable wird als true interpretiert, wenn
// ihr Inhalt nicht auf false hindeutet (0, "" etc.)
$var=1;
if($var)
    echo("ja<br>");
else
    echo("nein<br>");
```

Bedingungen

```
$var=0;  
if($var)  
    echo("ja<br>");  
else  
    echo("nein<br>");
```

```
$var="sdf2";  
if($var)  
    echo("ja<br>");  
else  
    echo("nein<br>");
```

```
?>  
$
```



Im folgenden Beispiel erfolgt in Abhängigkeit der Variable \$zaehler eine Ausgabe in den HTML-Quelltext. Der "default"-Teil wird durchgeführt, wenn kein anderer Abschnitt ausgeführt wurde.

Bedingungen

```
$ cat switch.php
<?php
for ($zaehler=1; $zaehler<10; $zaehler++) {
    switch($zaehler) {
        case 1:
        case 2:
            echo "$zaehler <br>";
            break;

        case 3:
            echo "drei <br>";
            break;
        case 4:
            echo "vier <br>";
            break;
        default:
            echo "groesser 4 <br>";
            break;
    }
}
?>
$
```

7 Rekursion

Eine Funktion kann sich rekursiv selbst aufrufen.

```
$ cat fac.php
<?php
function fac($i) {           // Berechnung Fakultät von i
    if ($i == 0 || $i == 1)
        return 1;
    else
        return $i * fac($i-1);
}

echo fac(3);
?>
$
```

Hörsaalübung

Der größte gemeinsame Teiler zweier natürlicher Zahlen ist definiert als:

rekursiv:

$$ggt(q,s) = q \text{ wenn } s = 0$$

$$ggt(s,q\%s) \text{ sonst}$$

iterativ:

$$ggt(q,s) =$$

1.setze $t = s$

2.bestimme $s = q \% s$

3.setze $q = t$

4.wiederhole die Schritte 1 - 3 solange bis $r = 0$; dann ist q das Ergebnis

Formulieren Sie ein PHP-Programm mit

a) rekursiver Funktion für den ggt

b) nicht rekursiver Funktion für den ggt

8 Arrays

In einem Array können mehrere Werte unter einem Variablennamen zusammengefasst werden.

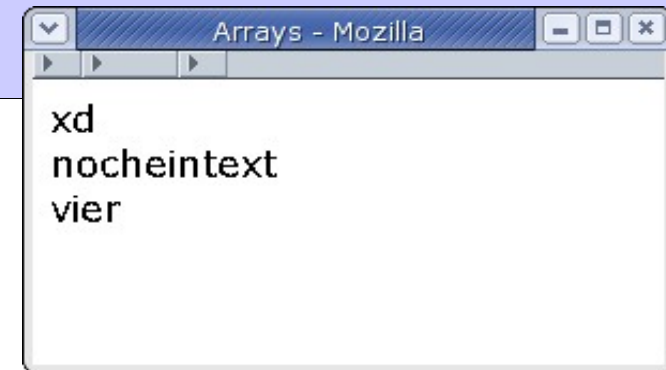
Auf einzelne Werte in diesem Array kann über den Index zugegriffen werden. In PHP sind Arrays null-indiziert, das heißt auf den ersten Wert ("x") kann mit `$v[0]` zugegriffen werden, auf den zweiten Wert mit dem Index 1 etc.

Arrays

```
$ cat array.php
<?php
$v[]="xd";
$v[]="nocheintext";
$v[]="dritter text";
$v[]="vier";

echo $v[0],"<br>";
echo $v[1],"<br>";
echo $v[3],"<br>";

?>
$
```



Arrays

Beim Füllen eines Array kann der zu verwendende Index für einen neuen Wert auch explizit angegeben werden:

Mit der Funktion `count()` kann die Anzahl der Elemente eines Array abgefragt werden.

```
$ cat array02.php
<?php
$array[0]="Text 1";
$array[3]="nocheintext";

echo count($array), "!", $array[2], "!";
$array[2]="dritter text";
$array[1]="vier";
echo $array[1], "<br>";
?>
$
```



Neben den numerisch indizierten Arrays gibt es in PHP auch assoziative Arrays, bei denen einem Textschlüssel ein Wert zugeordnet wird.

Der Schlüssel kann mit oder ohne Anführungsstriche angegeben werden. Verwendet man die Syntax ohne Anführungsstriche, so dürfen bestimmte Zeichen allerdings nicht verwendet werden, z.B. Leerzeichen.

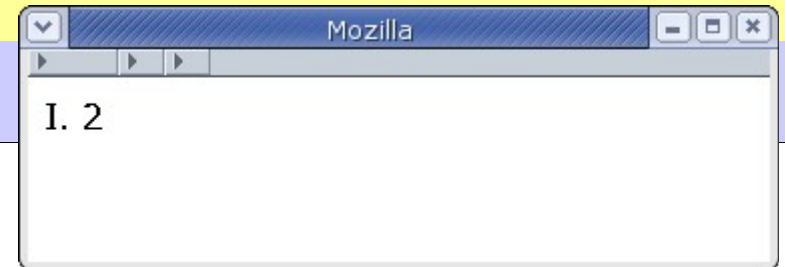
```
$ cat assozArray.php
<?php
$stel[alouis]="123";
$stel["jenni"]="18";
$stel[Donald]="987 12";
echo $stel["jenni"],"<br>";
?>
$
```

Mit der array-Funktion kann auch eine assoziatives Array aufgebaut werden:

```
<?php
$stel=array ("alouis" => "123",
            "jenni" => "18",
            "donald" => "987 12" );
echo $stel["jenni"],"<br>";
?>
```

In PHP sind mehrdimensionale Array möglich:

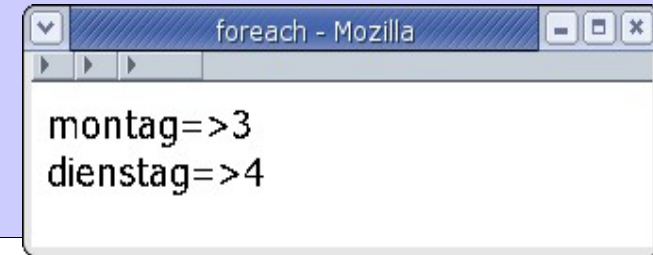
```
$ cat mehrdimArray.php
<?php
$matrix = array ("W1" => array("I. 1", "I. 2"),
                "W2" => array("II. 1", "II. 2")
                );
echo $matrix["W1"][1], "<br>";
?>
$
```



Für Arrays gibt es eine eigene Schleifenkonstruktion, die sich besonders bei assoziativen Arrays anbietet. Die folgende **foreach**-Schleife wird für jedes Element des Arrays `$x` einmal durchlaufen, wobei der jeweilige Wert aus dem Array in die Variable `$x`, der Index bzw. Schlüssel in die Variable `$key` kopiert wird.

Arrays

```
$ cat foreach.php
<?php
$x=array("montag"=>3,"dienstag"=>4);
foreach ($x as $key=>$value)
    echo $key,"=>", $value,"<br>";
?>
$
```



Hörsaalübung:

Schreiben Sie eine php-Funktion, die ein Array als Parameter besitzt und den größten im Array gespeicherten Wert als Ergebnis zurück liefern.

Realisieren Sie ein php-Testprogramm für ihre Funktion.

9 Formulare

In **HTML** können **Formulare** mit den HTTP-Methoden GET oder POST realisiert werden. In beiden Fällen werden die Inhalte von HTML-**Formularfeldern** automatisch als **lokale Variablen** in PHP **importiert**. Zusätzlich sind diese Variablen noch in zwei assoziativen Arrays `$HTTP_GET_VARS` und `$HTTP_POST_VARS` verfügbar.

Bei der Behandlung von Formularen erfolgen **mindestens zwei HTTP-Anfragen** beim Webserver:

- Die **erste** Anfrage zeigt das **leere Formular** an.
- Mit der **zweiten** Anfrage werden die Daten des **ausgefüllten Formulars** an den Server geschickt. Dieser schickt als Reaktion eine HTML-Seite an den Client, das die Daten erfolgreich an den Server übermittelt wurden.

Diese **beiden Seitenaufrufe** werden meistens vom *selben PHP-Skript* behandelt. Anhand der Existenz der durch GET übermittelten Variablen wird entweder das Formular erzeugt oder die übermittelten Daten werden verarbeitet.

Im folgenden Beispiel wird dies verdeutlicht, wir berechnen Schaltjahre und verwenden die **POST**-Methode:

Formulare

```
$ cat schaltjahre.php
<html>
  <head>
    <title>Schaltjahrberechnung</title>
  </head>
  <body>
    Bitte Jahreszahl eingeben: <br>
    <form method="POST">
      <input type="text" name="T" size="20">
      <input type="submit" value="Submit" name="B1">
    </form><br>
```

```
<?php
function schaltjahr($jahr) {
    if ($jahr%4==0 && $jahr%100!=0 || $jahr%400==0)
        return true;
    else
        return false;
}

$x = $_POST[T];
if ($x!="") {
    $x = (int)$x;
    if ($x != 0) {
        print "Das Jahr " . $x . " ist " ;
        if (schaltjahr($x))
            print "<b>ein</b> ";
        else
            print "<b>kein</b> ";
        print "Schaltjahr";
    } else
        print "Eingabe keine gueltige Jahreszahl !";
}

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

Da in der Regel zwei umfangreiche HTML-Quelltexte erzeugt werden müssen ist häufig übersichtlicher diese im HTML-Teil zu erstellen, d.h. den PHP im if und else-Teil zunächst zu schließen und anschließend wieder zu öffnen.

Formulare

```
$ cat formular.php
```

```
<body>
```

```
<h1>Formulare auswerten</h1>
```

```
<?php
```

```
$name = $_HTTP_GET_VARS["name"];
```

```
if (isset($name)) { // Wurde das Formular bereits ausgefüllt?
```

```
// Dann erfolgt hier Verarbeitung der gesendeten Daten
```

```
?>
```

```
Sie haben folgende Nachricht gesendet:<P>
```

```
<B>Ihr Name:</B> <?php echo $name;?><BR>
```

```
<FORM method=get action=formular.php>
```

```
<INPUT type=submit value="Neue Nachricht senden?">
```

```
</FORM>
```



Formulare

```
<?php
}
else          // Wenn keine Daten übermittelt wurden, muss zunächst
              // das Formular angezeigt werden.
{
?>
<H3>Namensabfrage</h3>
<FORM method=get action=formular.php>
<TABLE>
<TR>
<TD>Mein Name:</TD>
</TR>
<TR>
<TD><INPUT type=text name=name size=50</TD>
</TR>
```



Formulare - Mozilla

Formulare auswerten

Namensabfrage

Mein Name:

Neue Nachricht senden?

Fertig

Formulare

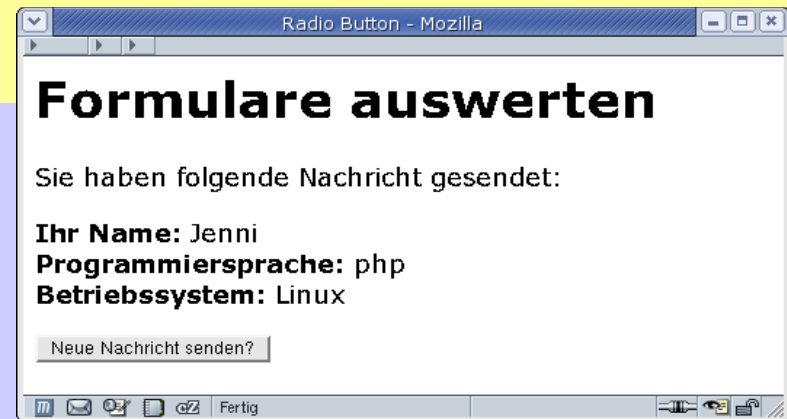
```
<TR>
<TD>
<INPUT type=submit value="Neue Nachricht senden?">
</TD>
</TR>
</TABLE>
</FORM>

<?php          // Ende der IF-Abfrage, else-Zweig
}
?>
</body>
$
```

Radio Button sind wie folgt abfragbar:

Formulare

```
$ cat Radio.php
<html>
<head><title>Radio Button</title></head>
<body>
<h1>Formulare auswerten</h1>
<?php
$name = $_HTTP_GET_VARS["name"];
$ps = $_HTTP_GET_VARS["ps"];
$bs = $_HTTP_GET_VARS["bs"];
if (!empty($name)) { // Formularauswertung
    echo "Sie haben folgende Nachricht gesendet:<P>";
    echo "<B>Ihr Name:</B> $name<BR>";
    echo "<B>Programmiersprache:</B> $ps<BR>";
    echo "<B>Betriebssystem:</B> $bs<BR>";
    echo "<FORM method=post action=Radio.php>";
    echo "<INPUT type=submit value=\"Neue Nachricht senden?\">";
    echo "</FORM>";
} else { // Abfragen
?>
<H3>Namensabfrage</h3>
<FORM method=get action=Radio.php>
Mein Name:
<INPUT type=text name=name size=50><br>
<H3>Programmiersprache</h3>
PHP <INPUT type=radio name=ps value=php>
```



Formulare

```
ASP <INPUT type=radio name=ps value=asp>
PERL <INPUT type=radio name=ps value=perl>
<br>
<H3>Betriebssystem</h3>
Windows <INPUT type=radio name=bs value=Windows>
Linux <INPUT type=radio name=bs
      value=Linux>

<br>
<INPUT type=submit value="Neue Nachricht
senden?">
</FORM>
<?php
}
?>
</body>
</html>
$
```



Hörsaalübung:

Schreiben Sie ein php-Programm, das ein Formular mit 2 Textfeldern anzeigt.

Durch einen Submit-Button sollen die in den Textfeldern eingegebenen Werte addiert werden und das Ergebnis angezeigt werden.

Sollen **Daten** zwischen **Seiten ausgetauscht** werden, wird häufig mit **versteckten** Variablen gearbeitet:

Die Ursprungsseite definiert eine **hidden**-Feld in einem Formular, die aufgerufene Seite kann den Wert des Feldes über `$HTTP_GET/POST_VARS` abfragen.

Formulare

```
$ cat WertuebergabeUrsprung.php
<html><head><title>Hidden Felder</title></head>
<body>
Mein Name:
<FORM method=post action=WertuebergabeAufgerufeneSeite.php>
  <INPUT type=text name=name size=50><br>
  <INPUT type=hidden name=versteckt value=18><br>
  <INPUT type=submit value="Neue Nachricht senden?">
</FORM>
</body>
</html>
$
$ cat WertuebergabeAufgerufeneSeite.php
<html><head><title>&Uuml;bergabe</title></head>
<body>
<?php
  $name =  $HTTP_POST_VARS["name"];
  $v =  $HTTP_POST_VARS["versteckt"];
  echo "Folgedne Werte sind &uuml;bergeben worden:<BR>";
  echo "<B>Name: </B> $name<BR>";
  echo "<B>versteckt: </B> $v<BR>";
?>
</body>
</html>
$
```

10 Datenbankzugriff

Hier wird gezeigt, wie die Verbindung von PHP-Programmen auf MySQL-Datenbanken hergestellt wird und Daten abgefragt und hinzugefügt werden können.

10.1 Connect

Zum Zugriff auf Daten einer MySQL Datenbank wird **zunächst** eine **Verbindung zum MySQL Server** hergestellt, **dann** wird die **Datenbank ausgewählt**.

Dazu wird die PHP-Funktion `mysql_connect()` verwendet. Verwendet wird auch die PHP-Funktion `die`.

Mit „`void die(string message)`“ kann man ein laufendes Skript abbrechen und eine eigene Fehlermeldung (`message`) an den Browser senden.

Datenbankzugriff

```
$ cat connect.sh
#!/usr/local/bin/php -q
<?php
    $host = "localhost";
    $user = "jenni";
    $passwd = "lopez";
    $database = "msd";

    echo "Verbindung zum MySQL-Server herstellen ... \n";
    $db = mysql_connect($host, $user, $passwd) or
        die("Verbindung zum MySQL Server fehlgeschlagen! \n");
    echo "Verbindung zum MySQL-Server erfolgt! \n";

    echo "mit Datenbank $database verbinden ... \n";
    @mysql_select_db($database, $db) or
        die("Verbindung mit $database fehlgeschlagen! \n");
    echo "mit Datenbank $database verbunden! \n";
?>
$ connect.sh
Verbindung zum MySQL-Server herstellen ...
Verbindung zum MySQL-Server erfolgt!
mit Datenbank msd verbinden ...
mit Datenbank msd verbunden!
$
```

Zum Zugriff auf Daten einer MySQL Datenbank wird zunächst eine Verbindung so wie oben beschrieben zum MySQL Server hergestellt, dann wird die Datenbank ausgewählt und der Zugriff wird mittels SQL-Anweisungen definiert.

10.2 Select

Daten selektieren kann man nach erfolgreicher Verbindung zur Datenbank, indem zunächst eine Anfrage definiert wird und das Resultat der Abfrage verwendet wird, um eine Ausgabe zu erzeugen.

```
$ cat select.sh
#!/usr/local/bin/php -q
<?php
    $host = "localhost";
    $user = "jenni";
    $passwd = "lopez";
    $database = "msd";

    // Verbindung zur MySQL Datenbank
    $db = mysql_connect($host, $user, $passwd) or
        die("Verbindung zum MySQL Server fehlgeschlagen!\n");
    @mysql_select_db($database, $db) or
        die("Verbindung mit $database fehlgeschlagen!\n");
```

```
// DB Query definieren
$sql_query = "SELECT kunr, name, adresse "
            ."FROM kunde "
            ."ORDER BY kunr";

// Query an Datenbank
$result = mysql_query($sql_query, $db);

// Ausgabe des Ergebnisses (eine Zeile pro Satz)
if ($result) {
    while ($row = mysql_fetch_array($result)) {
        echo $row["kunr"] . " " . $row["name"] . "\n";
    }
} else
    echo "Fehler bei SELECT!\n";
```

```
?>
$ select.sh
1 as
2 jl
3 bp
$
```

Ein **select**, das eine **HTML-Tabelle** erzeugt:

```
$ cat tabelle.php
<html>
...
  <body>
    <?php
      $host = "localhost";
      $user = "jenni";
      $passwd = "lopez";
      $database = "msd";

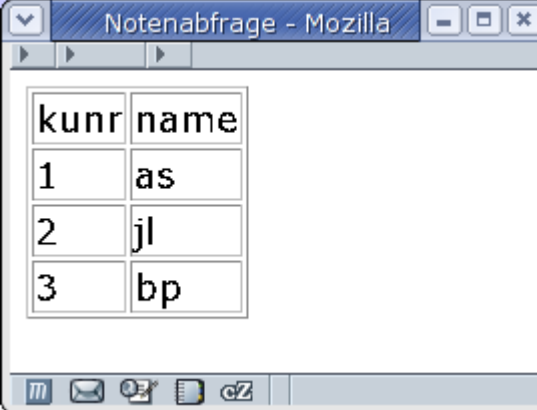
      // Verbindung zur MySQL Datenbank
      $db = mysql_connect($host, $user, $passwd) or
          die("Verbindung zum MySQL Server fehlgeschlagen!");
      @mysql_select_db($database, $db) or
          die("Verbindung mit $databas e fehlgeschlagen!<br>");

      // DB Query definieren
      $sql_query = "SELECT kunr, name, adresse "
          ."FROM kunde "
          ."ORDER BY kunr";

      // Query an Datenbank
      $result = mysql_query($sql_query, $db);
```

```
// Ausgabe des Ergebnisses (eine Zeile pro Satz)
if ($result) {
    echo "<table border=1>";
    echo "<tr><td>kunr</td><td>name</td></tr>";
    while ($row = mysql_fetch_array($result)) {
        echo "<tr>"
            . "<td>". $row["kunr"] . "</td>"
            . "<td>". $row["name"] . "</td>"
            . "</tr>";
    }
    echo "</table>";
} else
    echo "Fehler bei SELECT!<br>";

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



kunr	name
1	as
2	jl
3	bp

10.3 Insert

Einfügen ist nun einfach, die Technik ist die selbe (Querrystring, mysql_query()):

```
$ cat insert.sh
#!/usr/local/bin/php -q
<?php
    ...
    // DB Query definieren
    $sql_query = "INSERT INTO kunde VALUES (4, \"Dagobert\", null)";

    // INSERT an Datenbank
    $result = mysql_query($sql_query, $db);

    // Ausgabe ob erfolgreich
    if ($result)
        echo "INSERT erfolgreich!\n";
    else
        echo "Fehler bei INSERT!\n";
?>
$ insert.sh
INSERT erfolgreich!
$
```

10.4 Delete

Löschen ist genauso über SQL-delete zu realisieren:

```
$ cat delete.sh
#!/usr/local/bin/php -q
<?php
    ...

    // DB Query definieren
    $sql_query = "DELETE FROM kunde where kunr = 4;";

    // DELETE an Datenbank
    $result = mysql_query($sql_query, $db);

    // Ausgabe ob erfolgreich
    if ($result)
        echo "DELETE erfolgreich!\n";
    else
        echo "Fehler bei DELETE!\n";
?>
$
```

Hörsaalübung

Realisieren Sie eine php-Funktion, die sich mit einer Datenbank verbindet. Datenbank, Benutzer und Passwort sind in der Funktion zu deklarieren.

1. Entwickeln Sie ein Testprogramm, das sich mit der Datenbank verbindet und das Ergebnis (hat funktioniert, nicht funktioniert) zurück liefert.
2. Benutzen Sie diese Funktion, um ein Programm zu schreiben, das alle Konten mit zugehörigem Name des Kontoinhabers als HTML-Tabelle ausgibt.

11 Sessionverwaltung

Eine Session (Sitzung) ist eine **abstrakte Vorstellung** von einer zeitlich begrenzten **Umgebung**, deren Verhalten und Aussehen von **Sitzungsdaten bestimmt** sind. Die Sitzungsdaten werden von der Anwendung verwaltet und über die Dauer der Anwendung hinweg gespeichert.

Dadurch kann z.B. ein Benutzer (Sitzungsdaten sind Id und zuletzt benutzte Funktion der Anwendung) bei jedem Start der Anwendung jeweils „weiterarbeiten“ dort wo er zuvor die Anwendung beendet hat.

Im Web-Umfeld ist die Sessionverwaltung kompliziert, da **HTTP keine Sessions** kennt:

eine Web-Seite wird angefordert, die Seite wird geladen und der Browser schliesst die Verbindung zum Server.

Will man also über den Aufruf einer Seite hinweg Sitzungsdaten speichern, muss die Web-Anwendung dies selbst implementieren. Das kann prinzipiell auf zwei Arten erfolgen:

- clientseitig auf dem Rechner, auf dem der Browser gestartet wurde (Cookies)
- serverseitig auf dem Web-Server, der die Web-Seite bereit stellt (module session).

11.1 Cookies

Cookies bieten dem Server die Möglichkeit, Sitzungsdaten auf dem Rechner des Benutzers abulegen.

Das folgende Beispiel verwendet einen Cookie, um die Anzahl der Besuche einer Seite innerhalb von 1 Minute zu zählen.

```
$ cat cookie.php
<?php
    if (!isset($_COOKIE['anzahl']))
        $zaehler = 0;
    else
        $zaehler = $_COOKIE['anzahl'] + 1;
    // Cookies werden im HTTP-Header übertragen. Dieser wird vor der HTML-
    // Seite übertragen, daher muß der Cookie gesetzt werden, bevor durch echo,
    // einen HTML-Teil oder anderes bereits HTML-Quelltext ausgegeben wurde.
    setcookie ("anzahl", $zaehler, time()+60);
?>
<html>
<head><title>Cookies</title></head>
<body>
<h1>Cookies</h1>
<?php
    echo "Sie waren in der letzten Minute schon $zaehler mal hier!";
?>
</body>
</html>
$
```

-> live

Cookies haben folgende Einschränkungen:

- Ein Cookie kann nicht grösser als **2.048** Byte sein
- Es können (pro Domäne) maximal **20 Cookies** abgelegt werden

Da wegen Sicherheitsproblemen Cookies benutzerindividuell **abschaltbar** sind, sind sie **keine** gute Realisierungsidee für eine Sitzungsverwaltung.

Man sollte stets ein Test-Cookie setzen und das vor Verwendung anderer Cookies abfragen, um sicher zu sein, dass der Benutzer Cookies erlaubt.

11.2 Session

In PHP existieren viele Funktionen zur Sessionverwaltung. Hier sollen nur die wichtigsten erläutert werden.

PHP Sessions speichern Sitzungsdaten auf dem Webserver in speziellen Dateien. Die Verbindung zum Webbrowser des Benutzers erfolgt über eine **SessionID**.

Die **Session** wird **gestartet** bzw. eine bestehende Session wiederhergestellt durch `session_start()` und **beendet**, wenn der **Browser beendet** wird.

Zunächst ein einfaches Beispiel eines **Seitenzählers**:

Sessionverwaltung

```
$ cat counter.php
<?php
// start the session
session_start();           // muss vor erster Ausgabe stehen!
if(!$_SESSION['count']){
    $_SESSION['count'] = 1; // registrieren der Sitzungsvariablen count
} else {
    $_SESSION['count']++;
}
?>
<html>
<head><title>Session</title></head>
<body>
<h1>Session</h1>
    Sie waren schon <?php echo $_SESSION['count']; ?> mal hier!<br>
</body>
</html>
$
```

-> live

Auf die SessionId und alle **registrierten Sitzungsdaten** kann man zugreifen:

```
$ cat view.php
<?php
// start the session
session_start();           // muss vor erster Ausgabe stehen!
if(!$_SESSION['count']){
    $_SESSION['count'] = 1; // registrieren der Sitzungsvariablen count
} else {
    $_SESSION['count']++;
}
$_SESSION['count']++       // registrieren der Sitzungsvariablen test

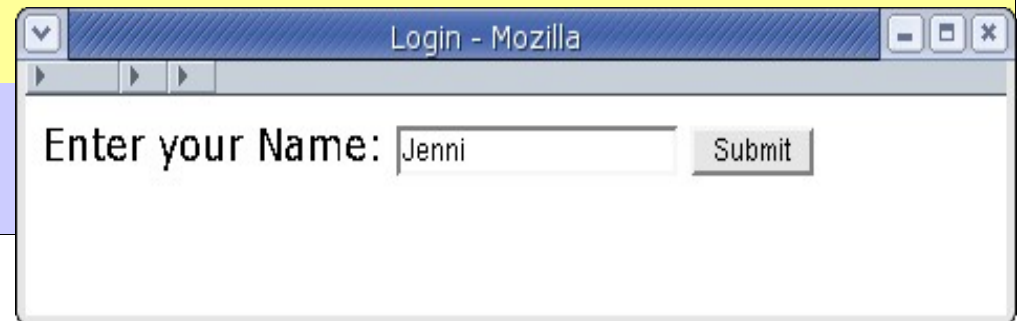
echo "SessionID: ", session_id(), "<br>";
echo "Sitzungsvariable: ", print_r($_SESSION), "<br>";
?>
<html>
<head><title>Session</title></head>
<body>
    Test
</body>
</html>
$
```

-> live

Eine häufige Verwendung von Sessions ist der **Login-Prozess** und die anschließende Begrüßung als bekannter Benutzer. Über verschiedene Seiten hinweg wird der Benutzer über Sitzungsdaten „gespeichert“.

Login-Maske

```
$ cat page1.php
<?php
// start the session
session_start();
?>
<html>
<head><title>Login</title></head>
<body>
<FORM METHOD="POST" ACTION="page2.php">
  Enter your Name: <input type="text" name="name">
  <input type="SUBMIT" value="Submit">
</FORM>
</body>
</html>
$
```

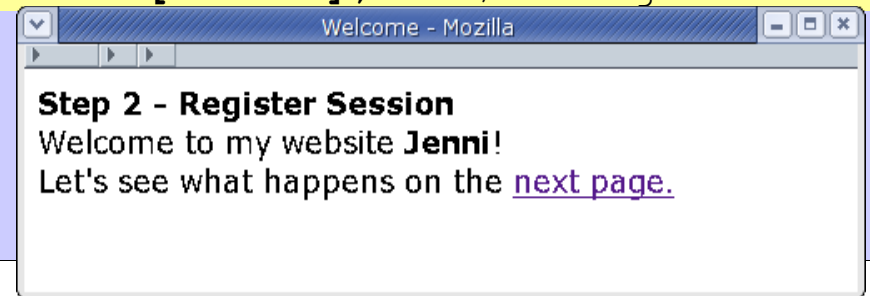


Die POST-Aktion ruft page2.php auf. In page2.php wird der **Name von der Formeingabe übernommen** und als **Sitzungsdatum registriert**:

```
$ cat page2.php
<?php
// start the session
session_start();
echo "<strong>Step 2 - Register Session </strong><br>";

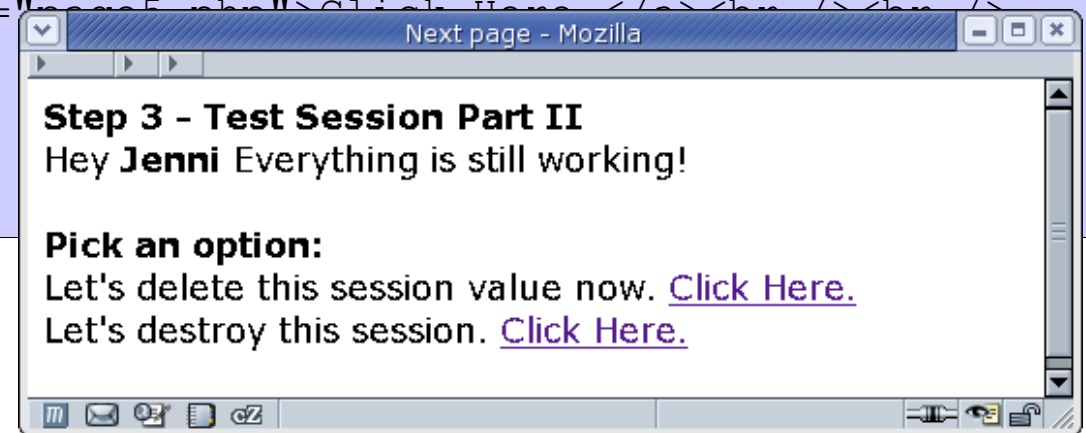
// Get the user's input from the form
$name = $_POST['name'];

// Register session key with the value
$_SESSION['name'] = $name;
// Display the Session information:
?>
<html>
<head><title>Welcome</title></head>
<body>
  Welcome to my website <strong><?php echo $_SESSION['name']; ?></strong>!<br>
  Let's see what happens on the
  <a href="page3.php">next page.</a><br>
</body>
</html>
$
```



Ein Klick auf „next page“ bewirkt, dass page3 geladen wird. Dabei wird der **Name über die Sitzungsdaten übernommen.**

```
$ cat page3.php
<?php
// start the session
session_start();
?>
<html>
<head><title>Next page</title></head>
<body>
  <strong>Step 3 - Test Session Part II </strong><br />
  Hey <strong><?php echo $_SESSION['name']; ?></strong> Everything is still
  working!<br /><br />
  <strong>Pick an option:</strong><br />
  Let's delete this session value now. <a href="page4.php">Click Here.</a><br
  />
  Let's destroy this session. <a href="page5.php">Click Here.</a><br /><br />
</body>
</html>
$
```



Einzelne Sitzungsdaten können „**unregistered**“ werden, ohne dass die Sitzung selbst zerstört wird. Auch kann eine ganze Sitzung beendet werden, ohne den Browser zu schliessen.

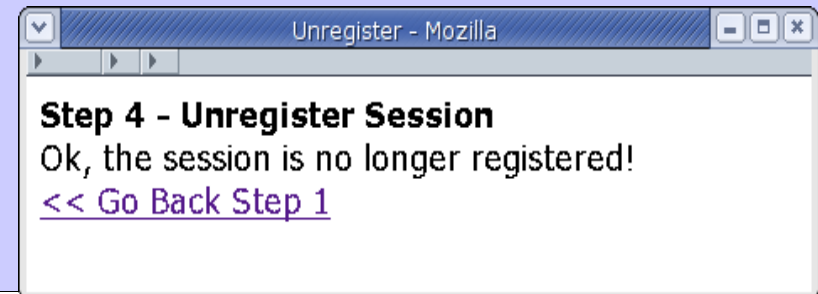
Als Anwendungsfall dient der Logout-Prozess, bei dem gespeicherte Namen und Passworte „zerstört“ werden müssen, damit man sich erneut einloggen kann.

Sitzungsdaten **entregistrieren**:

```
$ cat page4.php
<?php
// start the session
session_start();
header("Cache-control: private"); //IE 6 Fix

$_SESSION['name'] = FALSE;
// or $_SESSION['name'] = '';

echo "<strong>Step 4 - Unregister Session </strong><br />";
if($_SESSION['name']){
    echo "The session is still registered.<br /><br />";
} else {
    echo "Ok, the session is no longer registered! <br />";
    echo "<a href=\"page1.php\"><< Go Back Step 1</a><br /><br />";
}
?>
<html>
<head><title>Unregister</title></head>
<body>
</body>
</html>
$
```



Eine ganze Sitzung beenden:

```
$ cat page5.php
<?php
// start the session
session_start();
$_SESSION = array(); // delete all session data
session_destroy(); // destroy session
echo "<strong>Step 5 - Destroy This Session </strong><br />";
if($_SESSION['name']){
    echo "The session is still active";
} else {
    echo "Ok, the session is no longer active! <br />";
    echo "<a href=\"page1.php\"><< Go Back Step 1</a>";
}
?>
<html>
<head><title>Session destroy</title></head>
<body>
</body>

</html>
$
```



12 Zugriff auf Server-Ressourcen

In PHP-Programmen ist es oftmals erforderlich, auf Ressourcen des Web-Servers zuzugreifen. Wir diskutieren den Zugriff auf Email, Dateien und ausführbare Programme.

12.1 Datei Upload (PC->Server)

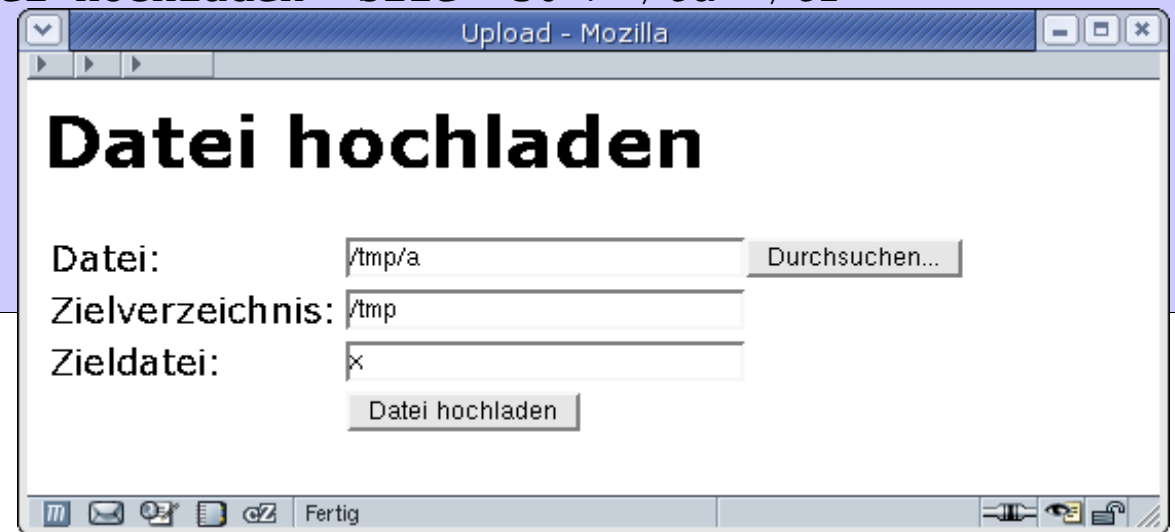
Wenn Dateien vom PC des Benutzers auf dem Server abgelegt werden sollen, wird häufig eine ftp-Verbindung aufgebaut, um die Daten zu übertragen. Ftp ist ein unsicheres Protokoll, deshalb wird es von vielen Web-Anbietern nicht mehr unterstützt.

Mit PHP kann man das Hochladen vom Client zum Web-Server leicht realisieren, ohne ftp zu verwenden.

Zunächst entwickeln wir ein **Formular**, um die Daten der **hochzuladenden Datei** eingeben zu können:

```
$ cat uploadForm.html
<html>
<head><title>Upload</title></head>
<body>
<h1>Datei hochladen</h1>
<table>
<form method=post action=upload.php>
<tr><td>Datei:</td>
<td><input type="File" name="quelldatei" size="30"></td></tr>
<tr><td>Zielverzeichnis:</td>
<td><input type="Text" name="zielverzeichnis" size="30"></td></tr>
<tr><td>Zieldatei:</td>
<td><input type="Text" name="zieldatei" size="30"></td></tr>
<tr><td></td>
<td><input type="submit" value="Datei hochladen" size="30"></td></tr>
</form>
</table>

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



Damit wird die Datei durch HTTP-POST **übertragen** und in einer **temporären Datei auf dem Server gespeichert**. Name, Grösse und Typ dieser Datei sind abfragbar über das globale Array `$_FILES`. Mittels der PHP-Funktionen `rename` ist dann der Zieldateiname festzulegen.

```
$ cat upload.php
<?php
    $zv = $_HTTP_POST_VARS['zielverzeichnis'];
    $zd = $_HTTP_POST_VARS['zieldatei'];
    echo "source name: ",    $_FILES['quelldatei']['name'], "<br>";
    echo "source type: ",    $_FILES['quelldatei']['type'], "<br>";
    echo "source size: ",    $_FILES['quelldatei']['size'], "<br>";
    echo "source tmp_name: ", $_FILES['quelldatei']['tmp_name'], "<br>";
    echo "destination: ", $zv, "/", $zd, "<br>";
    if (file_exists($zv."/". $zd)==1)
        echo "Datei existiert", "<br>";
    else
        if (rename($_FILES['quelldatei']['tmp_name'], $zv."/". $zd) == 1)
            echo "Datei uebertragen", "<br>";
        else
            echo "Fehler beim Upload!", "<br>";
?>
$
```



12.2 Dateien und Verzeichnisse

Alle elementaren Datei- und Verzeichnis-Systemfunktionen, die auf Betriebssystemebene bekannt sind, stehen als PHP-Funktionen zur Verfügung.

Hier soll lediglich das Lesen, Schreiben und Löschen einer Datei behandelt werden.

12.2.1.Schreiben einer Dateien

Bevor eine Datei gelesen oder beschrieben werden kann, muss sie geöffnet werden. Zum Öffnen und Schliessen einer Datei dienen die Funktionen `fopen()` und `fclose()`. Wir verwenden `fwrite()`, um Text in eine Datei zu schreiben.

```
$ cat fwrite.php
<?PHP
$text = "Dieser Text wird gespeichert";
$fp = fopen ("datei.txt", "w");           // Oeffnen zum Schreiben
fwrite($fp, $text,strlen($text));        // schreiben
fwrite($fp, "\n",1);                     // noch mal schreiben
fclose($fp);                             // schliessen
?>
$
```

```
$ php -q fwrite.php
$ cat datei.txt
Dieser Text wird gespeichert
$
```

12.2.2. Lesen aus einer Datei

Mit „string fgets(int fp, int length)“ kann man aus einer Datei (fp) eine Zeile mit der Länge length (inByte) auslesen.

Sollte die Zeile länger sein als der in length vorgegebene Wert, so wird die Zeile bis zur angegebenen Länge gelesen und der Rest abgeschnitten.

Kommt es beim Lesen der Datei zu einem Fehler, so wird false zurückgeliefert.

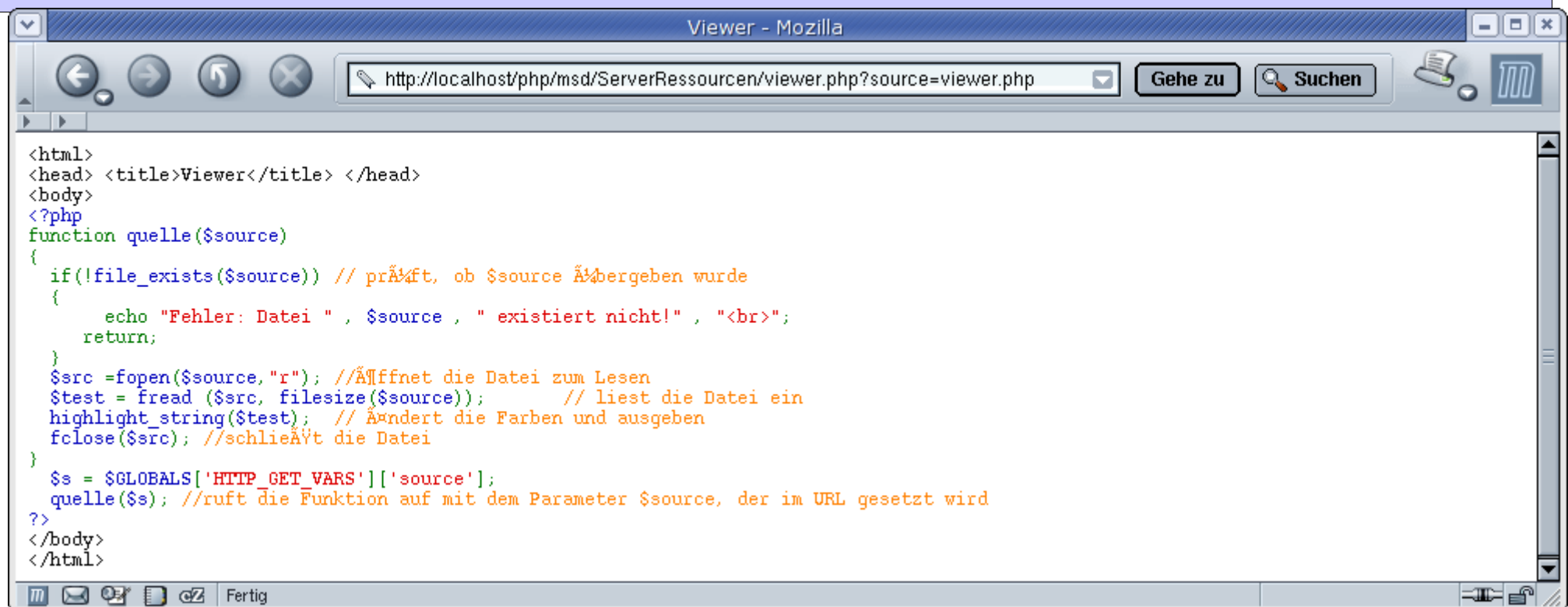
```
$ cat fgets.php
<?php
$datei="datei.txt";
$fp = fopen ($datei,"r");
$str = fgets($fp, 80);
fclose($fp);
echo $str;
echo "\n";
?>
$ php -q fgets.php
Dieser Text wird gespeichert
$
```

Zugriff auf Server-Ressourcen

Mit „string fread(int fp, int length)“ kann man Binärdaten aus einer Datei (fp) lesen. Der zweite Parameter für die Länge (length) bestimmt, wie viel der Datei gelesen werden soll (max. bis zum Dateiende).

Diese Funktion wird nachfolgend verwendet, um php-Quelldateien im Browser anzuzeigen.

```
<html>
<head> <title>Viewer</title> </head>
<body>
<?php
```



The screenshot shows a Mozilla browser window titled "Viewer - Mozilla". The address bar contains the URL "http://localhost/php/msd/ServerRessourcen/viewer.php?source=viewer.php". The browser's content area displays the source code of the PHP script, which is color-coded. The code defines a function named "quelle" that checks if a file exists, reads its contents using "fread", and highlights the string. The script then calls this function with the "source" parameter from the URL. The browser's status bar at the bottom shows "Fertig".

```
<html>
<head> <title>Viewer</title> </head>
<body>
<?php
function quelle($source)
{
    if(!file_exists($source)) // prüft, ob $source übergeben wurde
    {
        echo "Fehler: Datei " , $source , " existiert nicht!" , "<br>";
        return;
    }
    $src =fopen($source,"r"); //Öffnet die Datei zum Lesen
    $test = fread ($src, filesize($source)); // liest die Datei ein
    highlight_string($test); // ändert die Farben und ausgeben
    fclose($src); //schließt die Datei
}
$s = $GLOBALS['HTTP_GET_VARS']['source'];
quelle($s); //ruft die Funktion auf mit dem Parameter $source, der im URL gesetzt wird
?>
</body>
</html>
```

```
function quelle($source)
{
    if(!file_exists($source)) // prüft, ob $source übergeben wurde
    {
        echo "Fehler: Datei " , $source , " existiert nicht!" , "<br>";
        return;
    }
    $src =fopen($source,"r"); //öffnet die Datei zum Lesen
    $test = fread ($src, filesize($source)); // liest die Datei ein
    highlight_string($test); // ändert die Farben und ausgeben
    fclose($src); //schließt die Datei
}
$s = $GLOBALS['HTTP_GET_VARS']['source'];
quelle($s); //ruft die Funktion auf mit dem Parameter $source, der im URL ge-
setzt wird
?>
</body>
</html>
$
```

Mit „array **file**(string filename [, int use_include_path])“ kann man eine ganze Datei (filename) zeilenweise in ein Array einlesen. Das Zeilenumbruchzeichen am Ende jeder Zeile wird als letztes Zeichen n das entsprechende Array-Element übernommen.

Wird der optionale Parameter (use_include_path) auf 1 gesetzt, so wird auch innerhalb des Include-Pfads (wird in der php.ini bestimmt) nach der Datei gesucht.

```
$ cat file.php
<?php
$datei = "tel.txt";
$array = file($datei);
for($x=0;$x<count($array);$x++){
    echo $array[$x];
}
?>
$ cat tel.txt
Jenni          18
Dagobert       1000
Daisy          123
$ php -q file.php
Jenni          18
Dagobert       1000
Daisy          123
$
```

12.2.3.Löschen einer Datei

Mit „`int unlink(string filename)`“ kann man eine Datei (filename) vom Server löschen. Sollte ein Fehler auftreten, so gibt diese Funktion false zurück. Sie können die interne Fehlermeldung von PHP unterdrücken, indem Sie der Funktion ein "@" voranstellen. Somit wird dann nur die eventuell von Ihnen erzeugte Fehlermeldung ausgegeben.

```
$ cat unlink.php
<?php
$datei = "nicht_da.txt";
if (@unlink($datei))
    echo "Die Datei $datei wurde gelöscht!\n";
else
    echo "Konnte die Datei $datei nicht löschen!\n";
?>
$ php -q unlink.php
Konnte die Datei nicht_da.txt nicht löschen!
$
```

12.3 Mail

Mit „`bool mail(string to, string subject, string message [, string add_headers])`“ kann man eine E-Mail im Text- oder HTML-Format an eine oder mehrere Personen versenden. Man kann in dieser Mail einen Empfänger (to), einen Absender, ein CC , ein BCC und sogar ein Attachment festlegen, die alle beim Versand berücksichtigt werden.

Der Betreff (subject) und die Nachricht (message) werden dann mit den obigen Daten versendet. Im Mailheader (add_headers) kann man verschiedene Angaben (From, Cc, Bcc etc) ma-

chen, welche jeweils durch einen Zeilenvorschub (\n) getrennt sein müssen.

Will man eine E-Mail an mehrere Personen senden, so schreibt man sie im to-Bereich und trennt die einzelnen Adressen durch ein Komma(,). Im optionalen Parameter `additional_parameters` kann man Befehlszeilenargumente an das Mail-Programm übergeben.

Das Kontaktformular meiner Web-Seite verwendet die PHP mail-Funktion, um Anfragen an mich zu senden, hier ein php-Skript, das mir eine Nachricht von Jennifer sendet:

```
$ cat mailFromJenni.php
<?php
    $res = mail("a.schuette@fbi.fh-darmstadt.de",      // receipend
               "Message from Jenni",                // subject
               "Hi Alois, nice course!\n",          // message
               "From: Jenni Jennifer.Lopez@aol.com"); // sender

    if ($res)
        echo "Mail erfolgreich versendet";
    else
        echo "Mail nicht versendet";
    echo "\n";
?>
$ php -q mailFromJenni.php
Mail erfolgreich versendet
$
```

12.4 Serverprogramme ausführen

Die Funktion „`string system (string befehl [, int return_var])`“ ähnelt der C Version der Funktion sehr, indem es einen übergebenen *Befehl* ausführt und dessen Ausgabe anzeigt. Wir als zweiter Parameter der Funktion eine Variable übergeben, so wird der Rückgabestatus des Befehls in diese geschrieben.

```
$ cat system.php
<?php
    $res = 0;
    system('date', $res);
    echo $res, "\n";
?>
$ php -q system.php
Do Aug 19 12:01:27 CEST 2004
0
$
```

Die Funktion „`void passthru (string befehl [, int return_var])`“ ähnelt der Funktion `system()`, da sie ebenfalls einen Befehl ausführt. Ist der Parameter `return_var` angegeben, wird der Rückgabestatus des UNIX-Befehls hier abgelegt. Sie sollten diese Funktion an Stelle `system()` benutzen, wenn es sich bei der Ausgabe des Unix-Befehls um **binäre Daten** handelt, welche direkt zum Browser zurückgeschickt werden müssen.

Hörsaalübung:

Schreiben Sie ein PHP-Programm zum kopieren von Dateien. In einem Formular sollen die Namen der Quell- und Zieldateien eingebbar sein.

-> copy.php

13 Programmieren im Grossen

Php-Anwendungen bestehen aus mehreren „Modulen“, die jeweils in einer Datei gespeichert sind. Grosse Anwendungen werden mehr und mehr objektorientiert entworfen. Hier werden diese Möglichkeiten von PHP gezeigt.

Dies ist **keine** Einführung in die objektorientierte Programmierung (OOP)!

13.1 Strukturierung durch mehrere Dateien

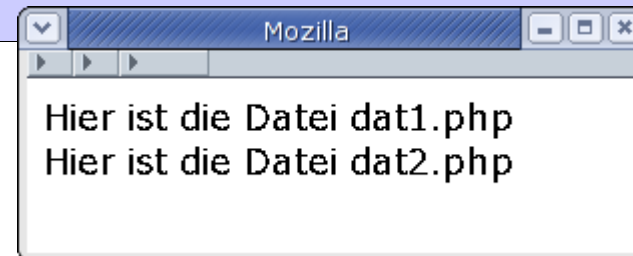
„`require(file)`“ hat zur Folge, dass vom PHP-Parsing-Modus in den HTML-Modus geschaltet wird und die angegebene **Datei während der Parser-Phase eingelesen** und ausgewertet wird. Ist innerhalb der zu inkludierenden Datei PHP-Code, so muss dieser in gültigen PHP-Start-(<?PHP) und End-Tags (?>) eingebunden werden.

„`require_once(file)`“ hat zur Folge, dass an seiner Stelle der Inhalt einer anderen Datei ausgegeben wird und somit ersetzt sich der Befehl durch die Datei. Der Hauptunterschied liegt darin, dass bei dem Befehl `require_once` der einzubindende Code einer Datei **nur einmal** eingebunden wird.

```
$ cat dat.php
<?php
require_once("dat1.php");
mache_etwas();
require_once("dat2.php");
?>

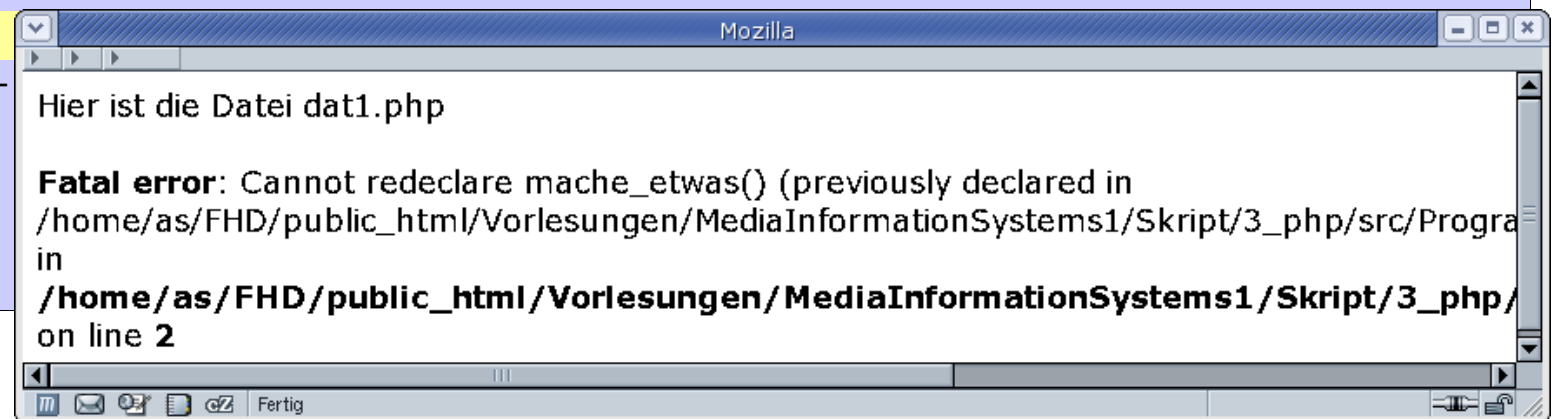
$ cat dat1.php
<?php
function mache_etwas(){
    echo "Hier ist die Datei dat1.php<br>";
}
?>

$ cat dat2.php
<?php
require_once("dat1.php");
echo "Hier ist die Datei dat2.php";
?>
$
```



Ein **Fehler** passiert, wenn man hier in dat2 require verwendet, da dann die Funktion „make_etwas“ **zwei mal definiert** würde.

```
$ cat dat_err.php
<?php
require_once("dat1.php");
make_etwas();
require_once("dat2_err.php");
?>
$ cat dat1.php
<?php
function make_etwas(){
    echo "Hier ist die Datei dat1.php<br>";
}
?>
$ cat dat2_err.php
<?php
require("dat1.php");
echo "Hier ist die Datei dat2.php";
?>
$
```



„`include(file)`, `include_once(file)`“ hat zur Folge, dass vom PHP-Parsing-Modus in den HTML-Modus geschaltet wird und die angegebene **Datei während der Laufzeit eingelesen** und **ausgewertet** wird. Ist innerhalb der zu inkludierenden Datei PHP-Code, so muss dieser in gültigen PHP-Start- (`<?PHP`) und End-Tags (`?>`) eingebunden werden.

Damit ist man in der Lage, die **Datei dynamisch** über **Laufzeitvariablen** zu bestimmen.

13.2 Objektorientierung

13.2.1.OOP als Klassifizieren von Problemen

Die **Grundidee** der Objektorientierten Programmierung ist es, einen Ausschnitt der **realen Welt** in einem Programm **abzubilden**.

In allen wissenschaftlichen Disziplinen hat sich das ***hierarchische Ordnungsprinzip*** bewährt:

Einteilung der **Objekte** der realen Welt in **Klassen**.

Beispiel Biologie: Einteilung der Lebewesen die Klassen Tiere, Pflanzen, Wirbeltiere, ...

Dabei sind Gruppen von **Objekten** mit **gleichen Eigenschaften** zu einer **Klasse** zusammengefasst: gehört ein Objekt zu einer Klasse, dann hat es auch deren Eigenschaften.

OOP ist ein **Klassifizieren von Problemen**, weniger eine Beschreibung der Aktionen (dies ist Hauptgegenstand der Prozeduralen Programmierung).

13.2.2. Probleme der Prozeduralen Programmierung

Eines der **Probleme** der **Prozeduralen** Programmierung ist die Wiederverwendung und Anpassung von Code. Software wird häufig um neue Funktionen erweitert, indem ein „**ähnliches**“ **Modul kopiert** und **angepasst** wird. Dann wird es in das System integriert.

Diese Vorgehensweise hat zwei große Nachteile:

- Das kopierte und modifizierte Modul muss **vollständig getestet** werden, also auch bereits getestete Teilfunktionalität (des Originals).
- Durch die Redundanzen müssen **Fehler**, die im redundanten Teil auftreten in **allen Kopien bereinigt** werden. Dies ist eine weitere Fehlerquelle und sehr aufwendig.

Die **Redundanzen** im Quellcode **erschweren** die **Wartbarkeit** des so entstandenen Systems – schlimmstenfalls ist das System nicht mehr wartbar.

Durch OOP soll dies verhindert werden, da Gemeinsamkeiten in Oberklassen zusammengefasst werden und die Erweiterungen in abgeleiteten Klassen behandelt werden.

13.2.3. Klassen und Objekte in PHP

Eine Klasse ist eine Sammlung von **Attributen** (Variablen) und **Methoden** (Funktionen), die mit diesen Variablen arbeiten.

Klassendefinition

Eine **Klasse** wird folgendermaßen **definiert**:

```

$ cat Klassendefinition
<?php
class Cart {
    var $items;                // Artikel in unserem Einkaufswagen

    function add_item ($artnr, $num) { // Füge dem Einkaufswagen $num Arti-
kel                                // der Sorte $artnr zu
        $this->items[$artnr] += $num;
    }

    function remove_item ($artnr, $num) { // Nimm $num Artikel von $artnr aus
// dem Einkaufswagen
        if ($this->items[$artnr] > $num) {
            $this->items[$artnr] -= $num;
            return true;
        } else {
            return false;
        }
    }
}
?>

```

In diesem Beispiel wird eine **Klasse** "Cart" definiert. Sie besteht aus einem **assoziativen Array von Produkten** im Einkaufswagen und **zwei Funktionen** zum Hinzufügen und Entfernen von Artikeln.

Klassen sind Typen, das heißt sie sind die „**Blaupausen**“ für reale Variablen. Um sie zu nutzen, muss zunächst eine **Variable** mit dem Operator `new` **angelegt** werden.

```
<?php
$cart1 = new Cart;
$cart1->add_item("10", 1);

$cart2 = new Cart;
$cart2->add_item("0815", 3);
```

Dies erstellt die Objekte `$cart1` und `$cart2` aus der Klasse `Cart`. Dann wird die Funktion `add_item()` des `$cart1` Objektes aufgerufen, um `$cart1` einen Artikel mit der Artikelnummer 10 hinzuzufügen. 3 Artikel mit der Artikelnummer 0815 werden `$cart2` hinzugefügt.

Zu beachten ist, dass die Variable `$cart->items`, und nicht `$cart->$items` genannt wird, da ein **Variablenname** in PHP nur **ein einziges Dollarzeichen** hat.

```
$cart->items = array("10" => 1); // korrekt, einfaches $  
  
// falsch, denn $cart->$items wird zu $cart->""  
$cart->$items = array("10" => 1);  
  
// richtig aber fraglich, ob dies erwünscht war:  
// $cart->$myvar wird zu $cart->items  
$myvar = 'items';  
$cart->$myvar = array("10" => 1);
```

Innerhalb einer Klassendefinition ist nicht bekannt, unter welchem Namen das Objekt im Programm erreichbar sein wird: Als die Klasse `Cart` geschrieben wurde war nicht bekannt, dass das Objekt später `$cart1` oder `$cart2` genannt wird. Deshalb kann man innerhalb der Klasse `Cart` selbst auch **nicht** `$cart1->items` schreiben. Um nun die eigenen Funktionen und Variablen innerhalb einer Klasse anzusprechen, können Sie die **Pseudo-Variable** `$this` verwenden, die das aktuelle Objekt referenziert.

Vererbung

Oft braucht man Klassen mit Eigenschaften, die in einer anderen Klasse definiert sind. So ist es eine gute Vorgehensweise, eine in allen Ihren Projekten verwendbare Oberklasse zu definieren, und diese dann den Bedürfnissen Ihrer einzelnen Projekte anzupassen.

Um dies zu erleichtern, können Klassen andere Klassen **erweitern**. Die erweiterte bzw. abgeleitete Klasse verfügt über alle Variablen und Funktionen der Basisklasse, und was Sie in der er-

weiteren Definition hinzufügen. Es ist **nicht** möglich, etwas von einer Klasse wegzunehmen, d.h. man kann keine existierenden Variablen oder Funktionen 'weg definieren'. Eine **Unterklasse** ist immer von **einer einzigen Oberklasse** abgeleitet, d.h. **Mehrfachvererbung** wird **nicht** unterstützt. Klassen werden mittels dem Schlüsselwort '**extends**' erweitert.

```
$ cat Vererbung.php
class Cart {
    ...
}
```

```

class Named_Cart extends Cart { // Named_Cart ist von Cart abgeleitet
                                // Cart ist Basisklasse von Named_Car
                                // Named_Cart ist Unterklasse von Cart
                                // Cart ist Oberklasse von Named_Cart
                                // Named_Cart erweitert Cart

    var $owner;
    function set_owner ($name) {
        $this->owner = $name;
    }
}
?>
$

```

Hier wird die Klasse `Named_Cart` definiert, die über **alle** Variablen und Funktionen von `Cart`, **plus** der Variable `$owner` und der Funktion `set_owner()` verfügt.

Man kann einen `Named_Cart` auf dem üblichen Weg erstellen und nun auch den Besitzer (`owner`) bestimmen und erfragen. Man kann noch immer die normalen `Cart` Funktionen an `Named_Cart` anwenden:

```

$ncart = new Named_Cart; // Erstellt einen Named_Cart
$ncart->set_owner("Jenni"); // den Besitzer festlegen
print $ncart->owner; // den Besitzer ausgeben
$ncart->add_item("10", 1); // vererbte Funktionalität von Cart
echo $ncart->items["10"]; // Zugriff auf Attribut der Basisklasse

```

Konstruktoren

Konstruktoren sind Funktionen innerhalb einer Klasse, die automatisch aufgerufen werden, sobald mittels `new` eine neue Instanz erstellt wird.

Ein **Konstruktor**³ ist eine **Funktion** der Klasse oder ihrer Basisklasse, die den **selben Namen** hat wie die **Klasse**, in der sie definiert wurde.

```
$ cat Konstruktor.php
<?php
class Cart {
    var $items;           // Artikel in unserem Einkaufswagen
    var $numberItems;    // Anzahl Artikel

    function Cart () { // Konstruktor
        $this->numberItems = 0;
    }

    function add_item ($artnr, $num) {
        $this->items[$artnr] += $num;
        $this->numberItems++;
    }
}
```

³ In PHP 3 und PHP 4 verhalten sich die Konstruktoren unterschiedlich. Die PHP 4 Semantik wird dringend empfohlen.
PHP3-Semantik: Ein Konstruktor ist eine Funktion mit dem selben Namen wie die Klasse (Basisklassen werden nicht beachtet).

```
function remove_item ($artnr, $num) {
    if ($this->items[$artnr] > $num) {
        $this->items[$artnr] -= $num;
        $this->numberItems--;
        return true;
    } else {
        return false;
    }
}

$cart = new Cart;
echo $cart->numberItems, "\n";
$cart->add_item("10", 1);
echo $cart->numberItems, "\n";
?>
$
```

Im nächsten Beispiel ist der Konstruktor in der Basisklasse definiert:

```
$ cat Konstruktor02.php
<?php
class Cart {
    var $items; // Artikel in unserem Einkaufswagen
    var $numberItems;

    function Cart () { // Konstruktor of Cart
        $this->numberItems = 0;
    }

    function Named_Cart () { // Konstruktor of Named_Cart
        $this->owner = "nobody";
    }

    function add_item ($artnr, $num) {
        $this->items[$artnr] += $num;
        $this->numberItems++;
    }

    function remove_item ($artnr, $num) {
        if ($this->items[$artnr] > $num) {
            $this->items[$artnr] -= $num;
            $this->numberItems--;
            return true;
        } else {
```

```
        return false;
    }
}
}
class Named_Cart extends Cart {
    var $owner;

    // Konstruktor ist in Basisklasse

    function set_owner ($name) {
        $this->owner = $name;
    }
}

$ncart = new Named_Cart;
echo $ncart->owner, "\n";
?>
$
```

Der :: Operator

Allgemein kann man mit „Klassenname :: Klassenfunktion“ Methoden aufrufen. Zum Zugriff auf Attribute oder Methoden der Basisklasse kann dieser Operator „::“ verwendet werden, z.B. um auf **Funktionen in Basisklassen** bzw. auf Funktionen in Klassen zuzugreifen, **die noch keine Instanzen haben**.

```
$ cat Operator::.php
<?php
class A {
    function example() {
        echo "I am the original function A::example().\n";
    }
}

// Es gibt kein Objekt der Klasse A.
// Dies wird folgendes ausgegeben
A::example();
?>
$ php -q Operator::.php
I am the original function A::example().

$
```

Das obige Beispiel ruft die Funktion `example()` der Klasse A auf. Nachdem noch kein Objekt der Klasse A existiert, können wir nicht `$a->example()` oder ähnliches schreiben. Stattdessen rufen wir `example()` als '**Klassenfunktion**' auf, d.h. als Funktion der Klasse selbst und nicht irgendein Objekt dieser Klasse.

Es gibt Klassenfunktionen, aber **keine** Klassenvariablen. Tatsächlich gibt es zur Zeit des Aufrufs kein Objekt. Deshalb darf eine **Klassenfunktion keine Objektvariablen** benutzen und sie darf `$this` **ebenfalls nicht** benutzen.

```

$ cat Operator::02.php
<?php
class A {
    function example() {
        echo "I am the original function A::example().<br>\n";
    }
}
class B extends A {
    function example() { // Redefinition
        echo "I am the redefined function B::example().<br>\n";
        A::example();
    }
}
// Erstellt ein Objekt der Klasse B.
$b = new B;
$b->example();
?>
$ php Operator\:\:02.php
I am the redefined function B::example().<br>
I am the original function A::example().<br>
$

```

In dem obigen Beispiel definiert Klasse B die Funktion `example()` neu. Die **ursprüngliche Definition** in Klasse A ist **überschattet** und nicht länger verfügbar, außer man verweist mittels des `::`-Operators speziell auf `example()` in Klasse A. In diesem Kontext **besteht ein Objekt**, das

Objektvariablen haben kann. Deshalb kann man auch `$this` und Objektvariablen verwenden, wenn sie von innerhalb einer Objektfunktion verwendet werden.

Anstatt im Code den wörtlichen Namen der Basisklasse zu verwenden, kann man den speziellen Namen `parent` verwenden, der sich auf die Basisklasse bezieht.

```
$ cat parent.php
class A {
    function example() {
        echo "I am A::example() and provide basic functionality.\n";
    }
}
class B extends A {
    function example() {
        echo "I am B::example() and provide additional functionality.\n";
        parent::example();
    }
}
$b = new B;
$b->example();
?>
$ php -q parent.php
I am B::example() and provide additional functionality.
I am A::example() and provide basic functionality.
$
```

13.2.4. Objektserialisierung

Ein **Objekt lebt** nur, solange das erzeugende **PHP-Programm in Ausführung** ist. Will man Objekte über einzelnen PHP-Aufrufen (Seiten) hinweg verwenden, muss man sie **Zwischen-**

speichern. Da Objekte komplexe Gebilde sind, stellt PHP zwei Funktionen zur Verfügung, um die komplexe Struktur in eine „verwendbare“ Form zu bringen: `serialize()` und `unserialize()`.

`serialize()` gibt eine **Zeichenkette** zurück, die eine **Byte-Strom-Repräsentation** irgendeines in PHP **speicherbaren Wertes** enthält.

`unserialize()` kann diese Zeichenkette verwenden, um die **ursprünglichen Variablenwerte** wieder **herzustellen**. Die Verwendung von `serialize` zum Speichern eines Objektes wird alle Variablen (Attribute) innerhalb eines Objektes speichern. Die Funktionen in einem Objekt werden nicht gespeichert, sondern nur der Name der Klasse.

Um ein Objekt wieder **deserialisieren** zu können, muss die **Klasse** dieses Objektes **definiert sein**. Das heißt, wenn man ein Objekt `$a` der Klasse A in `page1.php` hat und dieses serialisiert, erhält man eine Zeichenkette, die sich auf die Klasse A bezieht, und alle Werte der in `$a` enthaltenen Variablen enthält. Wenn man `$a` der Klasse A in `page2.php` mittels `unserialize` wiederherstellen will, muss die Definition von Klasse A in `page2.php` vorhanden sein. Dies kann zum Beispiel durch das **Speichern** der **Klassendefinition** von Klasse A in einer **Include-Datei**, und das Einbinden dieser Datei sowohl in `page1.php` und `page2.php` realisiert werden.

```
$ cat serialize.incl
```

```
<?php  
class A {          // Klassendefinition  
    var $one = 1;  
    function show_one() {  
        echo $this->one;  
    }  
}
```

```
$ cat serialize.php
```

```
<?php  
include("serialize.incl");// Definition der Klasse A  
$a = new A;           // Erzeugen einen Objektes der Klasse A  
$s = serialize($a);   // Konvertieren in speicherbare Form  
  
$fp = fopen("serialize.store", "w"); // speichern in Datei  
fputs($fp, $s);  
fclose($fp);  
?>
```

```
$ cat unserialize.php
<?php
include("serialize.incl");
$fp = fopen ("serialize.store","r");
$s = fgets($fp, 1000);    // Lesen des serial. Objektes
$a = unserialize($s);    // Umwandeln in int. Repräsentation

$a->show_one();          // Verwenden einer Methode des Objektes a
echo "\n";
?>
$ php unserialize.php
1
$
```

Wenn man mit **Sessions arbeiten** und `session_register()` verwendet, um **Objekte zu registriert**, so werden **diese Objekte am Ende der PHP Seite serialisiert**, und in jeder folgenden Seite **automatisch via unserialize wiederhergestellt**. Das heißt, dass diese Objekte auf jeder Seite auftauchen können, sobald sie Teil einer Session sind.

14PHP und Sicherheit

PHP-Anwendungen sind häufig Ziel von Angriffen. Hier sollen die wesentlichen Aspekte zum Realisieren von sicheren Anwendungen erläutert werden.

14.1 register_globals

Die Datei **php.ini** definiert die *Einstellungen* des PHP-**Interpreters**. Die Einstellungen

```
register_globals = Off/On
```

definiert, dass die über URIs übergebenen Variablen als **Globalvariablen** verwendet werden oder nicht.

Dadurch lassen sich über jeden Aufruf eines PHP-Programms in der URL beliebige Variablen ins Programm einschleusen.

Schwachstelle:

Ein Formular ruft die Datei global.php auf und übergibt ihr per GET den Inhalt des Eingabefeldes als Variable hallo.

```
<form action="global.php" method="GET">
<input type="text" name="hallo"></input>
<input type="submit">
</form>
```

Das Programm global.php enthält eine Variable:

```
<?php
echo $hallo;
?>
```

Ein Klick auf den Submit-Button übergibt hallo aus dem Formular ins Programm. global.php zeigt den Wert der Variablen. Doch hier sitzt eine Sicherheitslücke sein: Denn genauso kann jemand die Datei mit

`http://.../global.php?hallo=neuer+Wert`

aufrufen und Daten (=Variablenwerte) manipulieren.

Der **Programmfluss** kann dadurch **modifiziert** werden (geheim.php):

```
<?php
$geheim=0;
if ($geheim == 1) {
    echo „geheime Info“;
} else
    echo „nicht geheime Info“;
?>
```

Der Aufruf

`http://.../global.php?geheim=1`

bewirkt das Ausgeben geheimer Information.

Abhilfe: register_globals ausschalten und \$_POST/GET verwenden!

14.2 Validieren von Eingaben

Alle Benutzereingaben (URI oder Eingabefelder) sind zu validieren.

Im folgenden Beispiel soll das Programm zeigeBild.php ein Bild anzeigen.

```
http://.../zeigeBild.php?bild=auto.gif
```

Dadurch wird das Bild auto.gif im selben Verzeichnis, indem auch das Programm steht angezeigt.

Schwachstelle:

```
http://.../zeigeBild.php?bild=/home/as/privateBilder/jenni.gif
```

zeigt meine privaten Bilder.

Abhilfe: Eingabe validieren, hier mit „basename“

```
<?php
echo basename($_GET['bild']);
);
?>
```

14.3 SQL-Injection

Eine SQL-Injection ist eine SQL-Anweisung die von Hackern in Benutzereingaben eingeschleust wird, um die Datenbank zu manipulieren.

```

<?PHP
$sql = "SELECT * FROM users
      WHERE username='" . $_POST['username'] . "'
      AND password='" . $_POST['password'] . "'";

echo 'Query: ' . $sql . '<br />';

$result = mysql_query($sql);

$rows = mysql_num_rows($result);

if ($rows > 0) {
echo 'You are logged in!<br />';
} else {
echo 'You are not allowed here!<br />';
}
?>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
<input type="text" name="username" /><br />
<input type="text" name="password" /><br />
<input type="submit" />
</form>

```

Ein Angreifer könnte nun im Passwort-Feld folgendes eingeben:

' OR username LIKE '%'

```
SELECT * FROM users WHERE username='' AND password='' OR username LIKE '%'
```

Dieser Modifizierte Query würde **alle Einträge** der User Tabelle auswählen und somit wäre der Passwortschutz ausgehebelt.

Abhilfe: Variablen escapen oder **magic quotes** auf dem Server **aktivieren**

```
<?php
function safeEscapeString($string) {
    if (get_magic_quotes_gpc()) {
        return $string;
    } else {
        return mysql_real_escape_string($string);
    }
}

$sql = "SELECT * FROM users
WHERE username='" . safeEscapeString($_POST['username']) . "'
AND password='" . safeEscapeString($_POST['password']) . "'";

?>
```

14.4 Command Injection

Die php-Funktionen `exec()` und `system()` dienen dazu, serverseitig Kommandos auszuführen. Auch hier könnte über Parameter ein nicht gewolltes Programm eingeschleust werden.

```
<?php
$eingabe='hallo';
$eingabe='`date`'; // wird in Formularfeld eingegeben
system("echo $eingabe");
?>
```

Wird im Formularfeld anstelle von „hallo“ „`date`“ eingegeben, so wird das Kommando `date` ausgeführt.

Abhilfe: `escapeshellcmd()` und `escapeshellargs()`

```
<?php
$eingabe=escapeshellcmd('hallo');
$eingabe=escapeshellcmd('`date`'); // wird in Formularfeld eingegeben
system("echo $eingabe");
?>
```

Die Funktion `escapeshellargs` stellt den Parameter in Anführungsstriche, damit Leerzeichen nicht dazu führen, dass mehrere Parameter des Kommandos existieren.

14.5 Cross Site Scripting (XSS)

Unter "Cross Site Scripting" versteht man das "Einschleusen" von fremden HTML-Kode auf einer Web-Seite.

Damit kann ein Angreifer folgende Ziele verfolgen:

- Falsche Anzeige der WWW-Seite
 - Verunstaltung (z.B. unleserlich),
 - Anzeige fremder Inhalte (Texte, Bilder)
- Ausspähen von Daten, z.B. Zugangsdaten (insb. Cookies), durch Einbringen von JavaScript-Kode
- Ausführen von fremden Programm-Kode (insb. JavaScript)

Betroffen von solchen XSS-Problemen sind prinzipiell alle PHP- und CGI-Programme, die externe Parameter (wie Formulareingaben) verarbeiten, z.B. Gästebücher, Such- oder Anmeldeformulare.

```
<?PHP
$eingabe=$_POST['name'];
echo $eingabe;
?>
<form method="post" action="<?php echo $_SERVER['PHP_SELF']; ?>">
<input type="text" name="name" /><br />
<input type="submit" />
```

Wird in das Formularfeld eingegeben

```
<script>alert('Hallo');</script>
```

so wird der JavaScript Code ausgeführt, hier ein alert. Aber es ist denkbar, hier JavaScript Code einzugeben, durch den eine Verbindung zu einem Server aufgebaut wird, der dann Zugriff auf den PC des Benutzers hat.

Abhilfe: `strip_tags()` löscht alle in einer Zeichenkette eingefügten Tags, also auch `<script>`.

15PHP und JavaScript

PHP und JavaScript wird häufig zusammen verwendet:

- JavaScript zur Validierung von HTML-Forms Eingaben
- PHP, um das Formular zu erzeugen und weiterzuverarbeiten