

PHP

PHP - Grundlagen

Übersicht

PHP steht für **PHP**: **H**ypertext **P**reprocessor.

Die Sprache PHP findet gemeinhin Gebrauch als serverseitige Skriptsprache für die Programmierung von Webseiten mit dynamischem Inhalt.

PHP läuft auf den meisten Betriebssystemen und mit nahezu jedem Webserver. Die häufigste Webserverkombination ist Linux/Apache.

PHP - Grundlagen

Übersicht

Was leistet PHP?

- mit geringem Aufwand Zugriff auf Datenbanken (nativ und per ODBC)
- Schnittstellen für viele Protokolle (HTTP, FTP, IMAP, NNTP, ...)
- Bilder, PDFs, Shockwave-Filme generieren
- automatische Verwaltung von Ressourcen (Speicher, Dateihandles, ...)
- Verwaltung von HTTP-Authentifizierung, Sessions und Cookies
- PHP-Skripte auch auf der Kommandozeile ausführbar
- u.v.m.

Viele Erweiterungen sind über die Modulbibliotheken PEAR und PECL verfügbar.

PHP - Grundlagen

Übersicht

grundlagen/hallo_welt.php

```
1 <html>
2   <head>
3     <title>Hallo</title>
4   </head>
5   <body>
6     <?php echo "Hallo Welt!\n"; ?>
7   </body>
8 </html>
```

Das obige Dokument nachdem es von PHP interpretiert wurde sieht wie folgt aus

grundlagen/hallo_welt.html

```
1 <html>
2   <head>
3     <title>Hallo</title>
4   </head>
5   <body>
6     Hallo Welt!
7   </body>
8 </html>
```

PHP - Grundlagen

Übersicht

Skripte können durch übergebene Parameter beeinflusst werden.

Das folgende Skript gibt ebenfalls **Hallo Welt!** aus, wenn beim Aufrufen der Parameter **name** mit dem Wert **Welt** angegeben wurde.

grundlagen/hallo_x.php

```
1 <html>
2   <head>
3     <title>Hallo</title>
4   </head>
5   <body>
6     <?php echo "Hallo " . $_GET['name'] . "!\n"; ?>
7   </body>
8 </html>
```

PHP - Grundlagen

Übersicht

Zur Funktionsweise von PHP:



Quelle: <http://de.wikipedia.org/wiki/PHP>

- Ein Client (Browser) sendet eine Anfrage an den Webserver.
- Der Webserver nimmt die Anfrage entgegen und liest die PHP-Datei ein.
- Anschließend übergibt der Webserver die Datei an den PHP-Interpreter.
- Dieser arbeitet die Eingabe ab und gibt das Ergebnis an den Webserver zurück.
- Der Webserver liefert anschließend die Daten an den Client (Browser) zurück.

Syntax

PHP - Grundlagen

Syntax

PHP ähnelt den Sprachen C/C++, Java, JavaScript und Perl.

Man kann PHP-Code in andere Daten (normalerweise HTML) einbetten wie im vorherigen Beispiel. Der PHP-Interpreter interpretiert alles, was sich zwischen den Tags `<?php` und `?>` befindet als PHP-Code und reicht die Daten sowie die Ausgabe des Codes an den Webserver weiter.

Eine bessere Alternative hinsichtlich der Lesbarkeit bei größerem Code sind einschließende Tags um die ganze Datei:

`grundlagen/hallo_x2.php`

```
1 <?php
2     echo "<html><head><title>Hallo</title></head><body>Hallo ";
3     echo $_GET['name'];
4     echo "!</body></html>";
5 ?>
```

PHP - Grundlagen

Syntax

Der HTML-Quelltext des letzten Beispiels ist nicht besonders gut lesbar; Abhilfe schaffen Templates. Dabei handelt es sich um mit Platzhaltern geschriebene HTML-Dateien, die vom PHP-Skript eingelesen und mit Variablen belegt ausgegeben werden.

Eine weitere Möglichkeit besteht im Verwenden der Heredoc-Syntax

grundlagen/hallo_x3.php

```
01 <?php
02     echo <<<EOD
03 <html>
04     <head>
05         <title>Hallo</title>
06     </head>
07     <body>
08         Hallo {$_GET['name']}!
09     </body>
10 </html>
11 EOD;
12 ?>
```

PHP - Grundlagen

Syntax - Variablen

Variablen fangen in PHP mit einem **\$** an, dem der Variablenname folgt. Für diesen gilt:

- er beginnt mit einem Buchstaben oder Unterstrich¹
- darauf können recht viele Buchstaben, Zahlen oder Unterstriche folgen
- es wird zwischen Groß- und Kleinschreibung unterschieden

Eine Variable sieht meistens so aus: **\$text**

¹ Umlaute sind erlaubt, ihre Verwendung ist aber nicht besonders ratsam

PHP - Grundlagen

Syntax - Variablen

In PHP sind folgende Typen definiert:

- Boolean: entweder **true** oder **false**
- Integer: ganze Zahlen in dezimaler, hexidezimaler oder oktaler Schreibweise¹
- Float: Reelle Zahlen (die Nachkommastellen werden durch einen Punkt abgetrennt)
- String: Zeichenketten in einfachen oder doppelten Anführungszeichen
- Array: lineare Felder, bestehend aus beliebigen Elementen beliebigen Datentyps
- Object: Instanzen von Klasse im Bereich der objektorientierten Programmierung
- NULL: der Typ einer Variablen, die keinen Wert hat (d.h. ihr Wert ist auch **NULL**)

Einer Variablen ist i.d.R. kein expliziter Typ zugeordnet. Bei Bedarf wird eine Typkonvertierung durchgeführt.

¹ gekennzeichnet durch eine führende **0x** (hexadezimal) bzw. **0** (oktal)

PHP - Grundlagen

Syntax - Variablen - Boolean

Eine Variable kann jederzeit als Boolean-Wert interpretiert werden, gleich welchen Typ sie hat. In folgenden Fällen wird sie als **false** ausgewertet, andernfalls als **true**.

- Boolean: **false**
- Integer: **0**
- Float: **0.0**
- String: **''** (leerer String) und **'0'**
- Array: **[]** (leeres Array)
- Object: Objekte ohne eigene Daten
- **NULL**

PHP - Grundlagen

Syntax - Variablen - Boolean

Vergleichsoperationen liefert als Ergebnis einen Wert vom Typ Boolean:

- `==` wahr, wenn beide Werte gleich sind
- `===` wahr, wenn beide Werte und Typen gleich sind
- `!=` oder `<>` Gegenteil von `==`
- `!==` Gegenteil von `===`
- `<` und `>` wahr, wenn der erste Wert kleiner bzw. größer als der zweite ist
- `<=` und `>=` wahr, wenn `==` oder `<` bzw. `>` gilt

PHP - Grundlagen

Syntax - Variablen - Boolean

Logikoperatoren haben ebenfalls Boolean als Ergebnistyp

- **and** und **&&** wahr, wenn beide Werte **true** sind¹
- **or** und **||** wahr, wenn mindestens einer der beiden Werte **true** ist¹
- **xor** wahr, wenn genau einer der beiden Werte **true** und der andere **false** ist
- **!** wahr, wenn der nachfolgende Wert **false** ist

¹ Es besteht ein Unterschied zwischen beiden Operatoren bezüglich der Reihenfolge ihrer Auswertung

PHP - Grundlagen

Syntax - Variablen - Integer

Mit ganzen Zahlen kann man rechnen; definiert sind die Operationen **+**, **-**, ***** und **%**. Letzteres bezeichnet den Rest einer Division. Die Division selbst ist für Integer-Werte nicht definiert - sie werden aber bei Verwendung des **/**-Operators zu Float-Werten konvertiert.

Weiterhin definiert sind die Inkrement- und Dekrementbildung à la **\$a++** oder **--\$a**.

PHP - Grundlagen

Syntax - Variablen - Integer

Eine Variable wird als Integer-Wert interpretiert, wenn eine Funktion einen solchen verlangt. In dem Fall gilt für die Konvertierung:

- **false** ergibt 0, **true** ergibt 1
- Fließkommawerte werden gerundet
- Strings werden als 0 interpretiert, wenn sie nicht mit einer Zahl beginnen

PHP - Grundlagen

Syntax - Variablen - Float

Für Fließkommazahlen gibt es eine Vielzahl mathematischer Funktionen. So geben beispielsweise alle trigonometrischen Funktionen Float-Werte zurück.

Als Besonderheit können Fließkommazahlen als Ergebnis von Rechenoperationen die "Werte" **unendlich** und **NaN** (not a number) aufweisen. Letzteres ist der Fall, wenn eine Funktion für einen bestimmten Wert nicht definiert ist.

Diese Eigenschaften können mit den Funktionen **is_nan()** und **is_infinite()** bzw. **is_finite()** abgefragt werden.

PHP - Grundlagen

Syntax - Variablen - Float

Für Fließkommazahlen gilt bezüglich der Konvertierung in etwa das selbe wie für die ganzen Zahlen.

grundlagen/float.php

```
1 <?php
2     $s = '1.5e3 kleine Schweinchen' + 1;
3     echo $s;
4 ?>
```

Ausgegeben wird **1501**.

PHP - Grundlagen

Syntax - Variablen - Strings

Strings werden in einfache oder doppelte Anführungszeichen eingeschlossen. Sollen die Begrenzungszeichen selbst in der Zeichenketten vorkommen, müssen sie mit einem Backslash \ **escaped** werden.

Der Unterschied zwischen beiden Varianten besteht in der Behandlung des Inhaltes zwischen dem Anfang und dem Ende der Zeichenkette: Einfache Anführungszeichen führen dazu, daß der Inhalt unverändert ausgegeben wird (mit Ausnahme der escapeden Anführungszeichen). Im Fall von doppelten Anführungszeichen werden im String geschriebene Variablennamen durch ihre Werte ersetzt. Darüberhinaus werden weitere Escape-Sequenzen erkannt und ihrer Bedeutung entsprechend behandelt.

`grundlagen/strings.php`

```
1 <?php
2     $a = 'You' ;
3     echo "$a'll be a superstar!\n" ;
4 ?>
```

PHP - Grundlagen

Syntax - Variablen - Strings

Zum Aneinanderfügen von Strings gibt es den `.`-Operator. Mit ihm hätte das vorherige Beispiel wie folgt aussehen können.

`grundlagen/strings2.php`

```
1 <?php
2     $a = 'You';
3     echo $a.'\'ll be a superstar!'. "\n";
4 ?>
```

PHP - Grundlagen

Syntax - Variablen - Strings

Für größere Mengen Text, deren Formatierung nicht vernachlässigt werden sollte, eignet sich die bereits erwähnte Heredoc-Syntax.

`grundlagen/hallo_x3.php`

```
01 <?php
02     echo <<<EOD
03 <html>
04     <head>
05         <title>Hallo</title>
06     </head>
07     <body>
08         Hallo {$_GET['name']}!
09     </body>
10 </html>
11 EOD;
12 ?>
```

PHP - Grundlagen

Syntax - Variablen - Strings

Da PHP primär bei der Verarbeitung und Aufbereitung von Text eingesetzt wird, sind für Variablen dieses Typs sehr viele Funktionen bereits definiert.

Einige Beispiele und die Ausgabe des Skriptes:

grundlagen/strings3.php

```
1 <?php
2     echo str_replace('Welt','Erde',"Hallo Welt!\n");
3     echo htmlentities("<b>einen</b> Eßlöffel Öl\n");
4     similar_text('Mausefalle','Raubüberfall',$percent1);
5     similar_text('Mausefalle','Schneetreiben',$percent2);
6     echo "$percent1, $percent2\n";
7 ?>
```

Hallo Erde!

einen Eßlöffel Öl

63.636363636364, 17.391304347826

PHP - Grundlagen

Syntax - Variablen - Arrays

Arrays können für beliebig verschachtelte Datenstrukturen genutzt werden. Ein Array besteht aus einer geordneten Sammlung von Werten, die mit einem Schlüssel indiziert sein können (auch als Hash bezeichnet).

grundlagen/array.php

```
1 <?php
2   $arr1 = array(1,2,3,4);
3   $arr2 = array(1,'blah',25,array('a','b'));
4   $arr3 = array(1969 => 'Heinemann', 1974 => 'Scheel', 1979 => 'Carstens');
5   $arr4 = array('kurs'=>'php', 'raum'=>'3.11','anzahl'=>14);
6 ?>
```

Der Zugriff auf die Array-Variable geschieht, indem der Index in eckigen Klammern [] an die Variable angehängt wird (bei verschachtelten Strukturen mehrere []).

Eine Zugriff auf ein Array könnte z.B. so aussehen: `$filme[2006]['mai'][9] = array('laenge' => 114, 'darsteller' => array('Philip Seymour Hoffman', 'Catherine Keener'));`

PHP - Grundlagen

Syntax - Variablen - Arrays

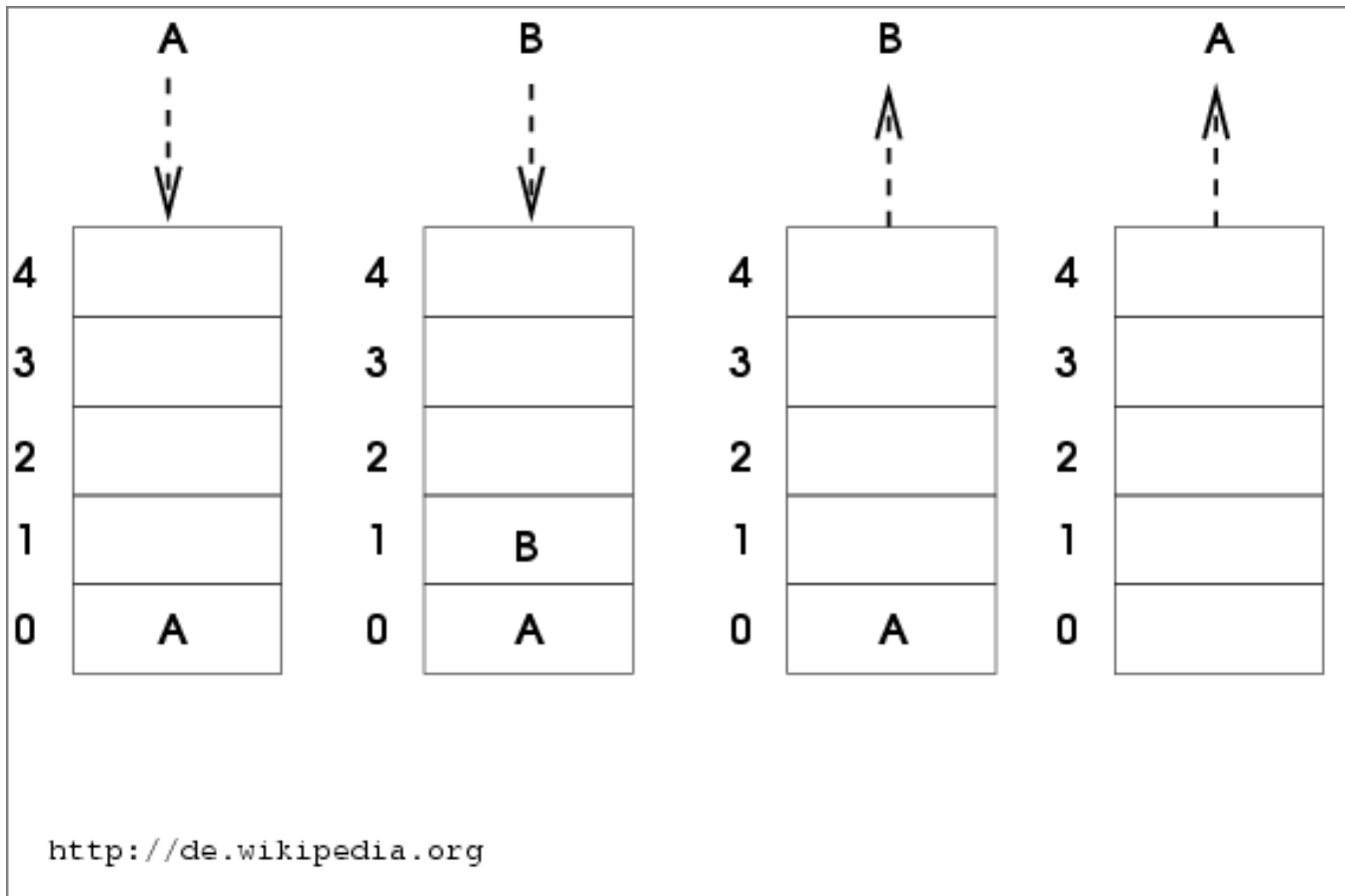
Für Arrays existieren ebenfalls sehr viele Funktionen. Neben dem direkten Zugriff auf benannte Array-Elemente kann ein Array ebenfalls als Keller- oder Schlangenspeicher benutzt werden (**LIFO** bzw. **FIFO**).

- **array_push()** fügt einem Array am "oberen" Ende ein Element hinzu
- **array_pop()** entfernt ein Element vom "oberen" Ende
- **array_shift()** fügt einem Array am "unteren" Ende ein Element hinzu
- **array_unshift()** entfernt ein Element vom "unteren" Ende

PHP - Grundlagen

Syntax - Variablen - Arrays

Die Kombination von `array_push()` und `array_pop()` bzw. `array_shift()` und `array_unshift()` verwendet ein Array als Kellerspeicher.

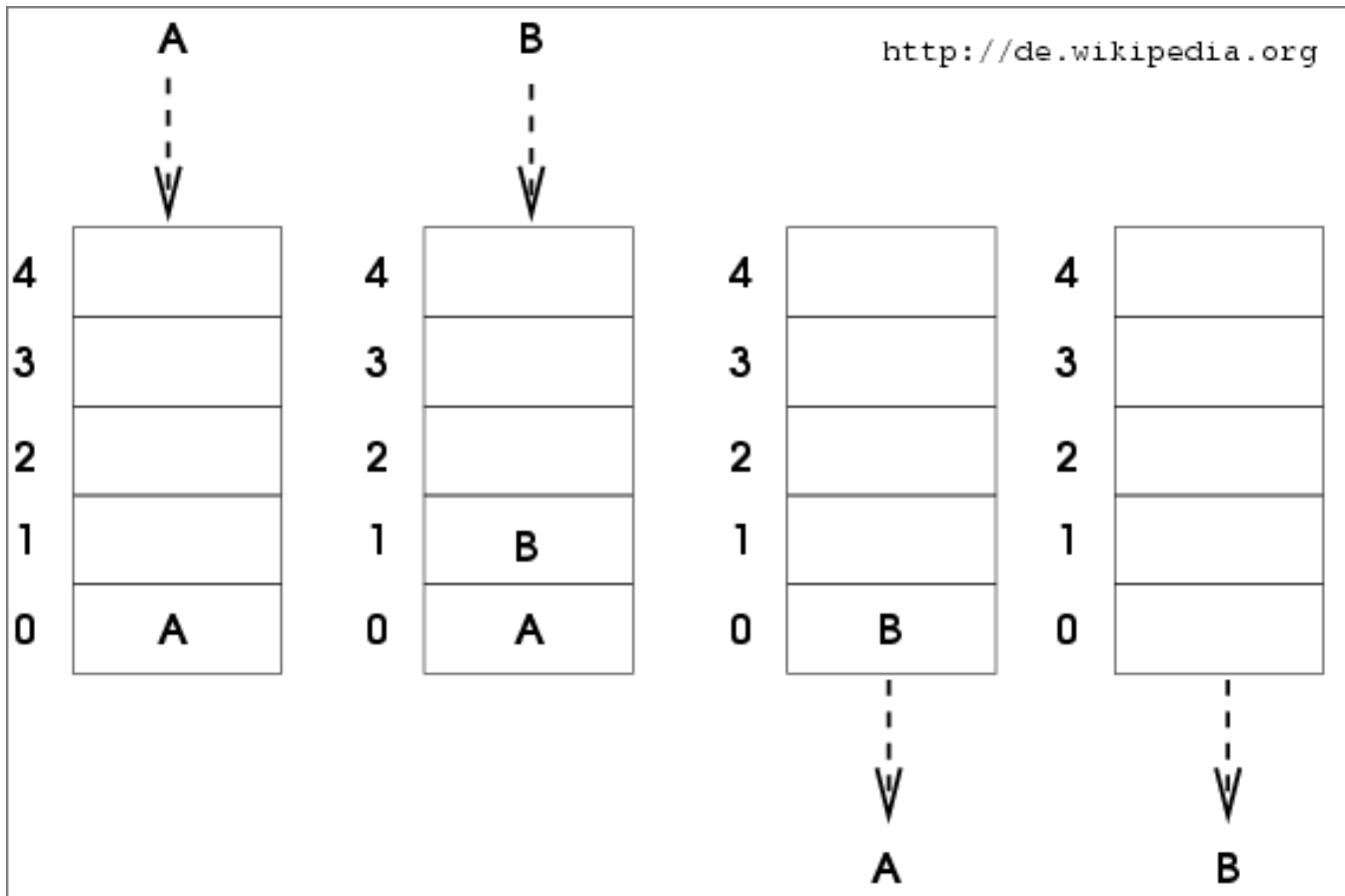


Quelle: <http://de.wikipedia.org/wiki/LIFO>

PHP - Grundlagen

Syntax - Variablen - Arrays

`array_push()` und `array_unshift()` bzw. `array_shift()` und `array_pop()` zusammen simulieren einen Schlangenspeicher.



Quelle: <http://de.wikipedia.org/wiki/FIFO>

PHP - Grundlagen

Syntax - Variablen - Arrays

Weitere häufig verwendete Funktionen:

- **count()** gibt die Anzahl der Elemente in einem Array zurück
- **explode()** splittet einen String anhand eines gegebenen Separators in ein Array
- **implode()** fügt ein Array mit einem Trennstring zu einer Zeichenkette zusammen
- **array_keys()** liefert die Schlüssel eines Arrays in einem neuen Array

PHP - Grundlagen

Syntax - Variablen - Klassen/Objekte

Auf Klassen kommen wir später zurück. Hier nur ein kleines Beispiel:

grundlagen/klassen.php

```
01 <?php // geliehen von http://tut.php-q.net/klassen.html
02
03 class Meerschweinchen
04 {
05     var $hunger = 0;
06
07     // Default-Werte sind nicht zwingend noetig, hier wird die Laenge im
08     // Konstruktor definiert
09     var $laenge;
10
11     // Dies ist der Konstruktor
12     function Meerschweinchen($laenge = 10)
13     {
14         $this->setLaenge($laenge);
15     }
16
17     function wasBistDu()
18     {
19         $message = "Ich bin ein Meerschweinchen mit ".$this->getLaenge()
20                 ." Millimeter langen Zaehnen.\r\n";
21         return $message;
22     }
23
```

PHP - Grundlagen

Syntax - Variablen - Klassen/Objekte

```
24     function getLaenge()  
25     {  
26         return $this->laenge;  
27     }  
28  
29     function setLaenge($neueLaenge)  
30     {  
31         if (!is_numeric($neueLaenge)) {  
32             return false;  
33         }  
34         $this->laenge = (int)$neueLaenge;  
35         return $this->laenge;  
36     }  
37 }  
38  
39 // Wir erzeugen ein paar Meerschweinchen  
40 // - genauer gesagt Instanzen der Klasse ``Meerschweinchen''  
41 $Washu  = new Meerschweinchen(11);  
42 // Der Parameter des Konstruktors ist in unserem Fall optional  
43 $Ebichu = new Meerschweinchen;  
44  
45 // Inspizieren wir die Meerschweinchen...  
46 print $Washu->wasBistDu();  
47 print $Ebichu->wasBistDu();  
48  
49 ?>
```

PHP - Grundlagen

Syntax - globale Variablen

Um das Verhalten eines PHP-Skriptes zu beeinflussen, gibt es mehrere Möglichkeiten, an dieses Parameter zu übergeben. Diese werden in allen Fällen als globales Array zur Verfügung gestellt.

Wird PHP als Kommandozeilen-Anwendung ([CLI](#)) ausgeführt, ist ein Array mit Namen **\$argv** definiert, in dem alle Parameter über ihren Index angesprochen werden können.

grundlagen/argv.php

```
1 <?php
2     foreach($argv as $var => $value)
3     {
4         echo "$var: $value\n";
5     }
6 ?>
```

PHP - Grundlagen

Syntax - globale Variablen

Ein Aufruf des obigen Skriptes der Form `php argv.php eins zwei drei` liefert folgende Ausgabe:

`0: argv.php`

`1: eins`

`2: zwei`

`3: drei`

Der erste Eintrag mit dem Index 0 ist - wie bei den meisten Skriptsprachen üblich - der Name des aufgerufenen Skriptes selbst.

Die Anzahl der übergebenen Parameter läßt sich sowohl über die Variable `$argc` als auch über `count($argv)` abfragen.

PHP - Grundlagen

Syntax - globale Variablen

In der Verwendung in Kombination mit einem Webserver, gibt es im Wesentlichen zwei Möglichkeiten der Parameterübergabe.

Die einfachere geschieht über die Adresszeile (**URL**) und nennt sich **GET**.

Das **HTTP** Protokoll sieht vor, daß Parameter in der Form **var=value**, jeweils durch ein **&** verkettet, an die Adresse mittels eines **?** angehängt werden können.

Das sieht dann beispielsweise so aus:

`http://localhost/php/grundlagen/get.php?eins=zwei&drei=vier`

Die Parameter finden sich in dem als Hash fungierenden Array **`$_GET`** wieder.

PHP - Grundlagen

Syntax - globale Variablen

Der **URL** (Uniform Resource Locator) ist zwar offiziell nicht in der Länge beschränkt, in der Praxis sollte man aber keine URLs mit mehr als 255 Zeichen verwenden. Mit dieser Einschränkung können per **GET** nur begrenzt Daten übertragen werden. Das Anwendungsgebiet der **GET**-Methode ist vielmehr die Auswahl von Informationen als die Übermittlung von Daten zur Weiterverarbeitung.

Die Alternative für letzteren Fall heißt **POST** und überträgt die Daten im **Header** einer HTTP-Anfrage. Der Zugriff auf die Daten erfolgt über ein Array mit Namen **\$_POST**

Als Überprüfung, ob überhaupt Daten mit der einen oder anderen Methode übergeben wurden, eignet sich die Funktion **empty()** angewendet auf das jeweilige Array.

PHP - Grundlagen

Syntax - globale Variablen

Weiterhin definiert sind folgende Variablen:

- **\$_SERVER** ein Array mit Informationen über den Webserver, auf dem PHP läuft¹
- **\$_FILES** ein Liste der beim aktuellen Aufruf hochgeladenen Dateien
- **\$_COOKIE** ein Array mit Cookie-Variablen, falls ein Cookie gesetzt wurde
- **\$_SESSION** ein Array mit Session-Variablen, wenn eine Session gestartet wurde

¹ Hierbei sind **\$_SERVER['REMOTE_ADDR']**, **\$_SERVER['REQUEST_URI']** und **\$_SERVER['HTTP_USER_AGENT']** vorrangig von Interesse.

PHP - Grundlagen

Syntax - globale Variablen

Mittels **global \$name** wird PHP angewiesen, die Variable **\$name** aus einer übergeordneten Struktur lokal verfügbar zu machen. Ein Zugriff auf diese ist auch über das globale Array **\$GLOBALS** möglich, und zwar mit **\$GLOBALS['name']**.

PHP - Grundlagen

Syntax - variable Variablen

Auf Variablen kann ebenfalls variabel zugegriffen werden. Dabei können Variablen mit ungültigen Namen entstehen, auf die aber mithilfe von `{ }` trotzdem zugegriffen werden kann.

grundlagen/varvar.php

```
1 <?php
2     $name = 'pi';
3     $$name = 3.14159;
4     echo "$pi\n";
5
6     $wert = 3.14159;
7     $$wert = 'pi';
8     echo "${3.14159}\n";
9 ?>
```

PHP - Grundlagen

Syntax - Debugging

Beim Debuggen von Skripten sind die Funktionen `var_dump()` und `print_r()` von großer Hilfe.

`grundlagen/var_dump.php`

```
1 <?php
2     $arr2 = array(1, 'Sag' => 'blah', 'Zebra' => 25, array('a', 'b'));
3     var_dump($arr2);
4 ?>
```

```
array(4) {
    [0]=>
    int(1)
    ["Sag"]=>
    string(4) "blah"
    ["Zebra"]=>
    int(25)
    [1]=>
    array(2) {
        [0]=>
        string(1) "a"
        [1]=>
        string(1) "b"
    }
}
```

PHP - Grundlagen

Syntax - Debugging

grundlagen/print_r.php

```
1 <?php
2     $arr2 = array(1, 'Sag' => 'blah', 'Zebra' => 25, array('a', 'b'));
3     print_r($arr2);
4 ?>
```

Array

```
(
    [0] => 1
    [Sag] => blah
    [Zebra] => 25
    [1] => Array
        (
            [0] => a
            [1] => b
        )
)
```

PHP - Grundlagen

Syntax - Konstanten

Konstanten werden mit der Funktion `define()` definiert und können anschließend ausgelesen aber nicht geändert werden.

Üblicherweise werden Konstantenbezeichner groß geschrieben:

```
define(VERSION, '1.2.3.4');
```

Um variabel auf Werte von Konstanten zuzugreifen zu können, liefert die Funktion `constant()` den Wert der Konstanten, deren Name als String übergeben wird.

Die Funktion `defined()` gibt zurück, ob eine Konstante mit dem angegebenen Namen definiert ist.

PHP - Grundlagen

Syntax - Konstanten

PHP stellt sogenannte **Magische Konstanten** bereit, bei denen es sich im Grunde um Funktionen auf Ebene des Interpreters handelt.

- **__LINE__** steht für die Zeilennummer, die der Interpreter gerade bearbeitet
- **__FILE__** steht für den Dateinamen des gerade in Abarbeitung begriffenen Skriptes
- **__FUNCTION__** steht für den Namen der Funktion¹
- **__CLASS__** steht für den Namen der Klasse¹
- **__METHOD__** steht für den Namen der Klassenmethode¹

¹ innerhalb derer diese Konstante verwendet wird

PHP - Grundlagen

Syntax - Kontrollstrukturen

Für bedingte Anweisungen bietet PHP das Standard-Repertoire: **if**, **if-else** und **if-elseif-else**.

grundlagen/elseif.php

```
01 <?php
02     if ($a > $b)
03     {
04         echo "a ist größer als b";
05     }
06     elseif ($a == $b)
07     {
08         echo "a ist gleich b";
09     }
10     else
11     {
12         echo "a ist kleiner als b";
13     }
14 ?>
```

PHP - Grundlagen

Syntax - Kontrollstrukturen

Zur Behandlung mehrerer Wertvergleiche mit der gleichen Variable ist es möglich, mehrere **elseif**-Blöcke hintereinander zu verwenden. Eine bessere Alternative dafür ist die **switch**-Anweisung.

grundlagen/switch.php

```
01 <?php
02     // Skript gibt <Parameter1> hoch <Parameter2> aus
03
04     switch ($argc)
05     {
06         case 2:
07             // nur ein Parameter uebergeben, also quadrieren wir
08             $argv[2] = 2;
09
10         case 3:
11             echo $argv[1].'^'.$argv[2].' = '.pow($argv[1],$argv[2])."\n";
12             break;
13
14         default:
15             echo "Ich brauche einen oder zwei Parameter!\n";
16     }
17 ?>
```

PHP - Grundlagen

Syntax - Kontrollstrukturen

Folgende Schleifentypen sind in PHP definiert:

- **while** die Bedingung wird vor jedem Durchlauf ausgewertet
- **do ... while** die Bedingung wird nach jedem Durchlauf ausgewertet
- **for** aus C entlehnte Zählschleife mit drei Parametern
- **foreach** zum Durchlaufen eines Arrays

PHP - Grundlagen

Syntax - Kontrollstrukturen

Einer **for**-Schleife sieht drei Parameter vor, die aber nicht definiert sein müssen:

- eine Startanweisung, die zu Beginn der Schleife einmal ausgeführt wird
- eine Bedingung, die vor jedem Durchlauf geprüft wird (wie bei einer **while**-Schleife)
- eine Anweisung, die nach jedem Durchlauf ausgeführt wird

grundlagen/for.php

```
1 <?php
2   for ($i=1, $n=1; !is_infinite($n); $i++)
3   {
4       echo "$i! = " . ($n *= $i) . "\n";
5   }
6 ?>
```

PHP - Grundlagen

Syntax - Kontrollstrukturen

Eine **foreach**-Schleife benötigt als Parameter das zu durchlaufende Array und ein Muster, das angibt, wie die Variablenbelegung innerhalb der Schleife aussehen soll. Möglich sind die Belegungen **`$value`** und **`$var => $value`**.

grundlagen/foreach.php

```
1 <?php
2     $arr3 = array(1969 => 'Heinemann', 1974 => 'Scheel', 1979 => 'Carstens');
3
4     foreach ($arr3 as $year => $name)
5     {
6         echo "$year: $name\n";
7     }
8
9 ?>
```

PHP - Grundlagen

Syntax - Kontrollstrukturen

Eine andere Möglichkeit, ein Array zu durchlaufen, ist mit den Funktionen **each()** und **list()** gegeben.

grundlagen/each.php

```
01 <?php
02     $arr3 = array(1969 => 'Heinemann', 1974 => 'Scheel', 1979 => 'Carstens');
03
04     // setzt den internen Zeiger des Array zurueck
05     reset($arr3);
06
07     while (list($year, $name) = each($arr3))
08     {
09         echo "$year: $name\n";
10     }
11 ?>
```

Beide Varianten unterscheiden sich hinsichtlich ihrer Funktionalität nicht. Je nach Kontext kann die Verwendung der einen oder anderen Variante sinnvoller sein.

PHP - Grundlagen

Syntax - Kontrollstrukturen

In jeder Schleife können die Anweisungen **break** und **continue** verwendet werden, um die Schleifenbearbeitung abubrechen bzw. zum nächsten Schritt überzugehen. Innerhalb eines **switch**-Blockes bewirken **break** und **continue** einen Sprung zum Ende des Blockes.

PHP - Grundlagen

Syntax - Kontrollstrukturen

Beide Anweisungen können mit einem numerischen Parameter versehen werden, der angibt, auf die wievielte Ebene vom aktuellen Block ausgehend sie sich bezieht.

grundlagen/break.php

```
01 <?php
02     for ($i=1; $i<=10; $i++)
03     {
04         echo str_pad($i,2)." - ";
05         for ($j=1; $j<=10; $j++)
06         {
07             if ($i == $j)
08             {
09                 break;
10             }
11             elseif ($i*$j > 42)
12             {
13                 echo "\n";
14                 continue 2;
15             }
16             echo "$j ";
17         }
18         echo "\n";
19     }
20 ?>
```

PHP - Grundlagen

Syntax - Funktionen

Funktionen werden in PHP mit dem Schlüsselwort **function** gefolgt von einem Funktionsnamen und einer Parameterliste eingeleitet. In **{ }** folgt der Inhalt der Funktion.

Die Parameterliste wird in **()** notiert und kann leer sein; die einzelnen Parameter erhalten einen Variablennamen als Bezeichner innerhalb des Funktionsblocks.

grundlagen/function.php

```
01 <?php
02
03     function lf()
04     {
05         echo "\n";
06     }
07
08     echo 'Erste Zeile';
09     lf();
10     echo 'Zweite Zeile';
11     lf();
12 ?>
```

PHP - Grundlagen

Syntax - Funktionen

Parameter können per **call-by-value** (Standard) oder **call-by-reference** übergeben werden. In letzterem Fall wird vor dem Variablenbezeichner ein **&** notiert.

Wertrückgabe von Funktionen erfolgt durch die **return**-Anweisung. Dabei kehrt der Interpreter zur Stelle des Funktionsaufrufes zurück und setzt den Parameter der Anweisung (sofern einer gegeben ist) als Ergebnis des Funktionsaufrufes.

Funktionen, die eine Referenz zurückgeben wollen, müssen vor dem Funktionsnamen ebenfalls den Referenz-Operator **&** aufweisen. (Darüberhinaus muß bei der Verwendung der Referenz-Zuweisungsoperator **=&** statt **=** verwendet werden.

PHP - Grundlagen

Syntax - Funktionen

Ein Überladen von Funktionen ist in PHP nicht möglich, aber einzelne Funktionsparameter können mit einem konstanten Vorgabewert versehen werden und sind somit beim Funktionsaufruf optional. Die mit einem Vorgabewert versehenen Parameter sollten in der Parameterliste am Ende stehen, da der letzte nicht-optionale Parameter über die Mindestanzahl der Parameter entscheidet.

grundlagen/function2.php

```
01 <?php
02
03     function hallo($anzahl,$wer='Welt')
04     {
05         for ($i=1; $i <= $anzahl; $i++)
06         {
07             echo "Hallo $wer!\n";
08         }
09     }
10
11     hallo(2,'Teilnehmer');
12     hallo(3);
13 ?>
```

PHP - Grundlagen

Syntax - Funktionen

Mit den Funktionen `func_num_args()`, `func_get_arg()` und `func_get_args()` kann man auch direkt auf die übergebenen Parameter zugreifen und so eine dynamische Parameteranzahl realisieren.

grundlagen/add.php

```
01 <?php
02
03 function add()
04 {
05     $res = 0;
06     foreach(func_get_args() as $int)
07     {
08         $res += $int;
09     }
10     return $res;
11 }
12
13 echo add().', '.add(1,7).', '.add(1,2,3,4,5)."\n";
14 ?>
```

PHP - Grundlagen

Syntax - Funktionen

Funktionen können wie Variablen variabel angesprochen werden. Die Klammern signalisieren dem Interpreter, die Variable als Funktionsnamen aufzufassen.

grundlagen/varfunc.php

```
01 <?php
02
03     function eins()
04     {
05         echo "Funktion 1\n";
06     }
07
08     function zwei()
09     {
10         echo "Funktion 2\n";
11     }
12
13     $func = 'eins';
14     $func();
15     $func = 'zwei';
16     $func();
17 ?>
```

Dateifunktionen

PHP - Grundlagen

Dateifunktionen

Die einfachste Art, in PHP auf den Inhalt einer Datei zuzugreifen, ist die Funktion **file_get_contents()**. Diese liefert das Ergebnis in einem String zurück. Falls bei der jeweiligen PHP-Installation die Konfigurationsoption **allow_url_fopen** auf **"1"** gesetzt wurde (Standard), ist es möglich, anstelle des Dateinamens auch eine **URL** zu verwenden.

`grundlagen/file_get_contents.php`

```
1 <?php
2     $url = 'http://www.ims.uni-stuttgart.de/lehre/teaching/2006-SS/php/';
3     $content = file_get_contents($url);
4     echo $content;
5 ?>
```

Neben **HTTP/HTTPS** sind auch Zugriffe auf andere Quellen wie z.B. **SSH** möglich.

file_put_contents() heißt analog die einfachste Funktion, um eine Datei zu schreiben, sofern ein Schreibzugriff möglich ist.

PHP - Grundlagen

Dateifunktionen

Die Funktion `readfile()` liest eine Datei und gibt sie gleichzeitig aus.

Ebenso liest `file()` den Inhalt einer Datei komplett ein. Allerdings gibt diese Funktion den Inhalt zeilenweise in einem Array zurück.

PHP - Grundlagen

Dateifunktionen

Bei größeren Datenmengen ist es meist ratsam, eine Datei stückweise einzulesen. Das funktioniert am besten mit der Funktion **fgets()**, **fopen()** und **fclose()** sind vonnöten, um die Datei zu öffnen bzw. zu schließen (was die anderen Funktionen von sich aus erledigen).

grundlagen/fgets.php

```
01 <?php
02     $filename = $argv[1];
03     $handle = fopen($filename, 'r');
04     while (!feof($handle))
05     {
06         // nur 4KB Speicher in Benutzung
07         $buffer = fgets($handle, 4096);
08         echo $buffer;
09     }
10     fclose ($handle);
11 ?>
```

Die analoge Funktion zum Schreiben heißt **fputs()**.

PHP - Grundlagen

Dateifunktionen

Viele Dateifunktionen sind Operationen auf UNIX-Dateisystemen entlehnt. So sind die Funktionen `chmod()`, `chown()` und `chgrp()` den gleichnamigen Befehlen entsprechend definiert. `fileperms()`, `fileowner()` und `filegroup()` lesen die entsprechenden Informationen zu der angegebenen Datei aus.

`copy()`, `unlink()`, `mkdir()` und `rmdir()` dienen zum Kopieren und Löschen von Dateien sowie dem Anlegen und Entfernen von Verzeichnissen.

Mit der Funktion `file_exists()` kann die Existenz einer Datei geprüft werden, `glob()` gibt ein Array von Dateien, die einem angegebenen Suchmuster entsprechen zurück.

Nachtrag

PHP - Grundlagen

Nachtrag

Zum Einbinden anderer Skripte wird normalerweise der `include()`-Befehl verwendet. Will man vermeiden, die gleiche Quelle mehrfach einzubinden, kann man `include_once()` verwenden. Letzteres ist ratsam, wenn es bei der einzubindenden Quelle um Klassendefinitionen oder Funktionsdeklarationen handelt.

PHP behandelt die angegebene Quelle als (HTML-)Daten, so daß die umschließende Verwendung der Tags `<?php` und `?>` notwendig ist, wenn es sich um ein Skript handelt.

PHP - Grundlagen

Nachtrag

Für die Verwendung **regulärer Ausdrücke** gibt es in PHP zwei Möglichkeiten: Zum einen die erweiterten **POSIX**-Ausdrücke, zum anderen die Perl-kompatiblen (**PCRE**).

Die POSIX-Ausdrücke werden in den Funktionen **ereg()**, **ereg_replace()** und **split()** verwendet. Für die Perl-kompatiblen Ausdrücke gibt es analog die Funktionen **preg_match()**, **preg_replace()** und **preg_split()**. Darüberhinaus sind weitere nützliche Funktionen definiert:

- **preg_match_all()** gibt alle Treffer des Suchmusters zurück
- **preg_replace_callback()** verwendet eine Funktion zum Ersetzen der Treffer
- **preg_grep()** nimmt ein Array und liefert nur die passenden Einträge zurück