PHP

Grundlagen, Dynamische Webseiten

Themenübersicht

Basics

- Grundlagen
- Sprachkonstrukte
- Funktionen
- Closures
- Namespaces

Dynamische Webseiten

- Formulare
- Datenverarbeitung, Validierung, filter-Funktionen
- Datenbanken, Passwort-API
- Cookies & Sessions

- PHP 7
- Composer & PHP-Frameworks

Grundlagen

Entwickelt 1995 von Rasmus Lerdorf als „Personal Home Page“

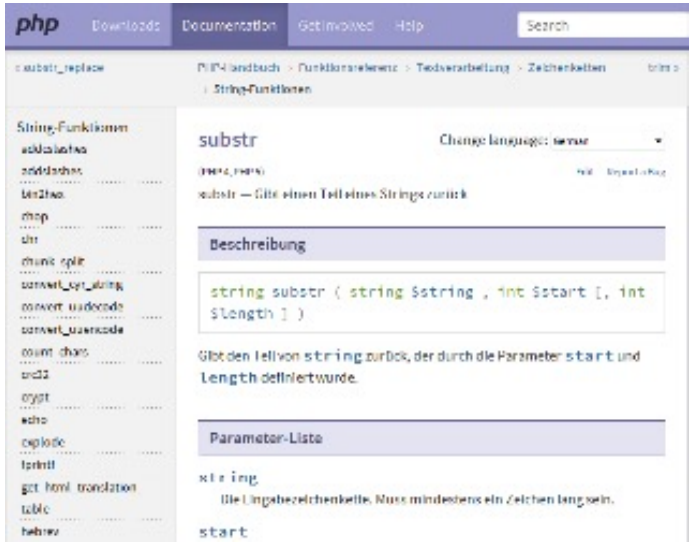
- Später umbenannt in „PHP: Hypertext Processor“

Historie

- Lange Zeit eine sehr simple Skriptsprache
- „Richtige“ Objektorientierung erst seit PHP 5.0
- PHP 6 als ursprünglicher Nachfolger von PHP 5 geplant
  - > Gescheitert auf Grund zahlreicher Probleme
  - > Versionsnummer 6 wurde nie vergeben
- Aktuelle Version: PHP 7 (Juni 2016)

PHP Documentation Group

- php.net



Grundlagen

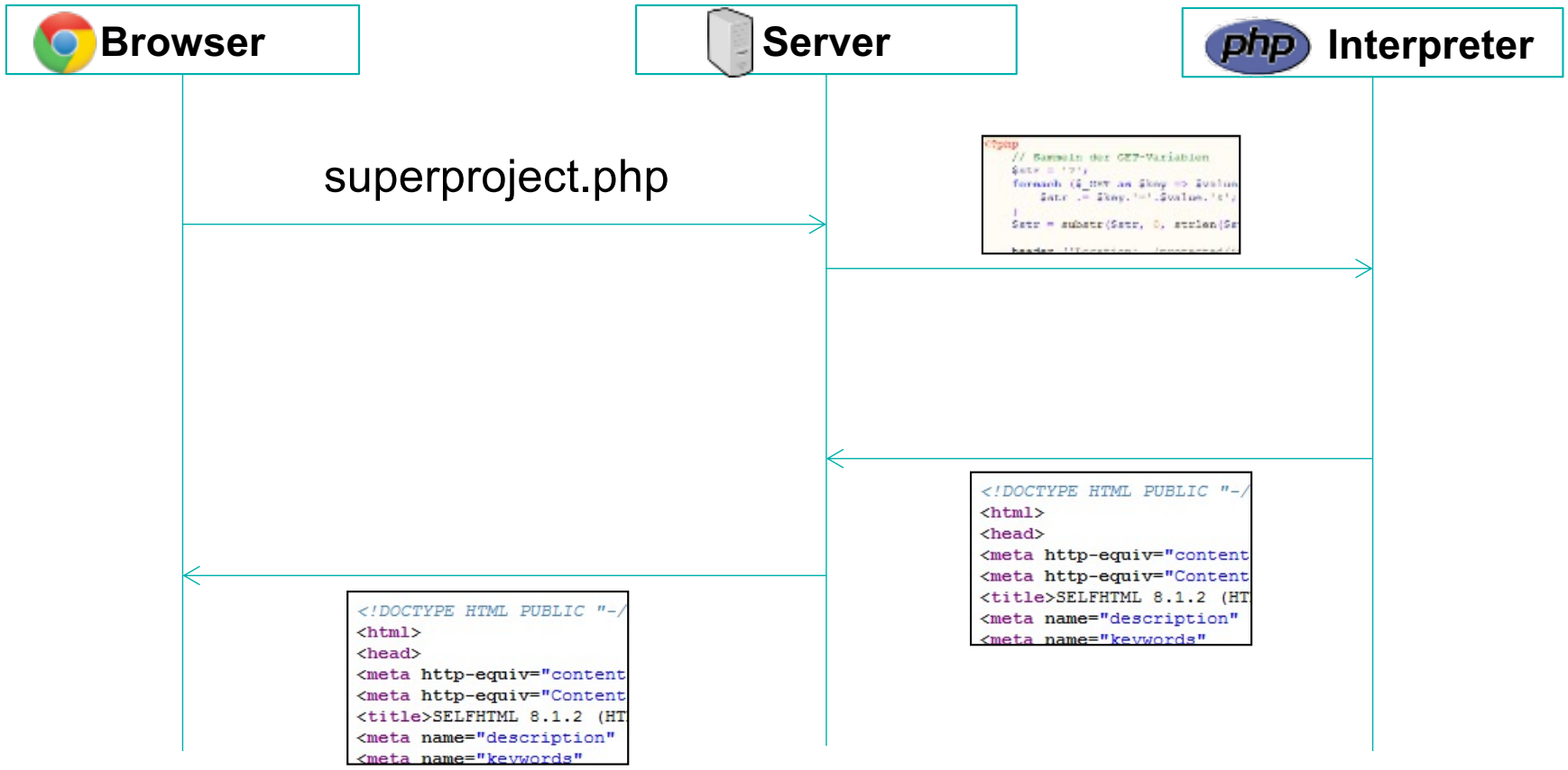
Voraussetzungen an den Server

- Installierter PHP-Interpreter notwendig
- Typisch für Webentwicklung ist ein „LAMP"-Stack
  > Linux, Apache, MySQL, PHP

Ablauf

- Browser stellt die Anfrage an den Server
- Dateiendung „.php" signalisiert dem Server, dass die Datei PHP-Code beinhaltet
- PHP-Interpreter durchläuft die Datei und interpretiert den Quellcode
- Die Ausgabe des PHP-Scripts wird in das an den Browser gesendete HTML integriert, bzw. ist das HTML-Dokument

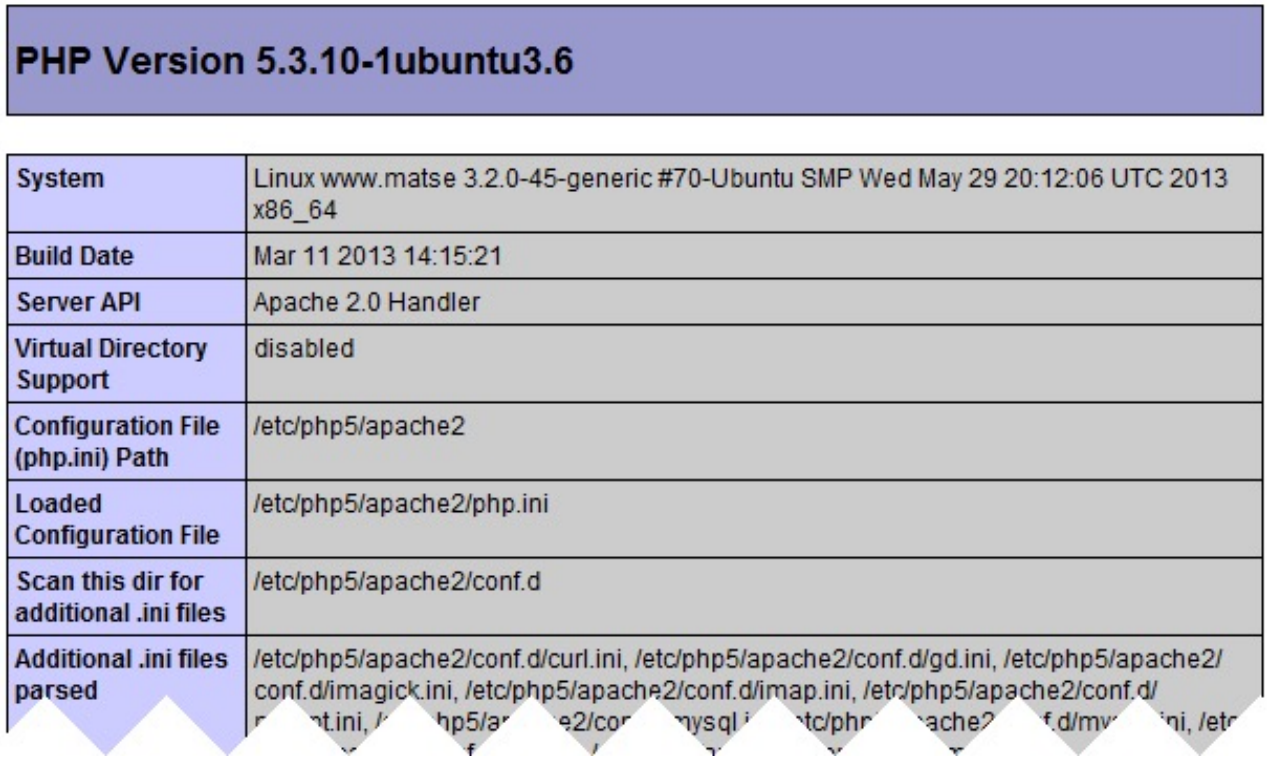
Aufruf einer PHP-Datei (vereinfacht)



Wichtig: Der Code ist für den Nutzer nicht sichtbar.
Es wird die Ausgabe der Ausführung übertragen

Beispiel: test.php

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



PHP Version 5.3.10-1ubuntu3.6

System	Linux www.matse 3.2.0-45-generic #70-Ubuntu SMP Wed May 29 20:12:06 UTC 2013 x86_64
Build Date	Mar 11 2013 14:15:21
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	/etc/php5/apache2/conf.d
Additional .ini files parsed	/etc/php5/apache2/conf.d/curl.ini, /etc/php5/apache2/conf.d/gd.ini, /etc/php5/apache2/conf.d/imagick.ini, /etc/php5/apache2/conf.d/imap.ini, /etc/php5/apache2/conf.d/

Syntax
Beispiele

beispiel2.php

```
<?php
    $titel = 'Web-Engineering';
    $gruss = 'Willkommen im Kurs!';
?>
<!DOCTYPE html>
<html>
<head>
    <title><?php echo $titel; ?></title>
</head>
<body>
    <?php echo $gruss; ?>
</body>
</html>
```

Syntax
Beispiele

beispiel2.php

- Produziert die folgende HTML-Ausgabe:

```
<!DOCTYPE html>
<html>
<head>
    <title>Web-Engineering</title>
</head>
<body>
    Willkommen im Kurs!
</body>
</html>
```

Syntax

Vieles bekannt aus anderen Sprachen

- if, for, while…
- Funktionen
- Klassen und Objekte

Einige Konzepte unbekannt

- Fehlende Variablendeklaration
- Closures
- Cookies & Sessions

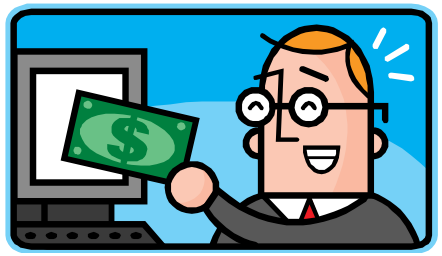
Syntax
Variablen

Variablen

- Dargestellt durch Dollar-Zeichen ($) gefolgt vom Namen
- **Keine explizite Deklaration**!
- **Kein fester Typ**!
- Nicht initialisierte Variablen haben einen vom Kontext abhängigen Vorgabewert (false, null, leerer String oder leeres Array)

```php
<?php
    $name = 'Max';
    $id = 42;
    $pi = 3.14;

    $foo += 5;
?>
```



Datentypen

Typ	Beschreibung	Beispiel/Literal
boolean	Wahrheitswert	true, false
integer	Ganzzahl	0, 123, -12, 0xFF
float / double	Fließkommazahl (IEEE 754)	3.14, 1.2e5
string	Zeichenkette	'foo', "bar"
array	(assoziatives) Arrray	array(), array(1, 2, 3), array('key' => 'val')
object	Objekt	new Car (wobei Car eine Klasse ist)
NULL	Variable ohne Wert	NULL
resource	Referenz zu einer Ressource	mysql_connect('localhost', …)
callable (nicht primär)	Funktionszeiger	function() { }

- Weitere Informationen: http://php.net/manual/language.types.php

Typumwandlung

- Implizit:

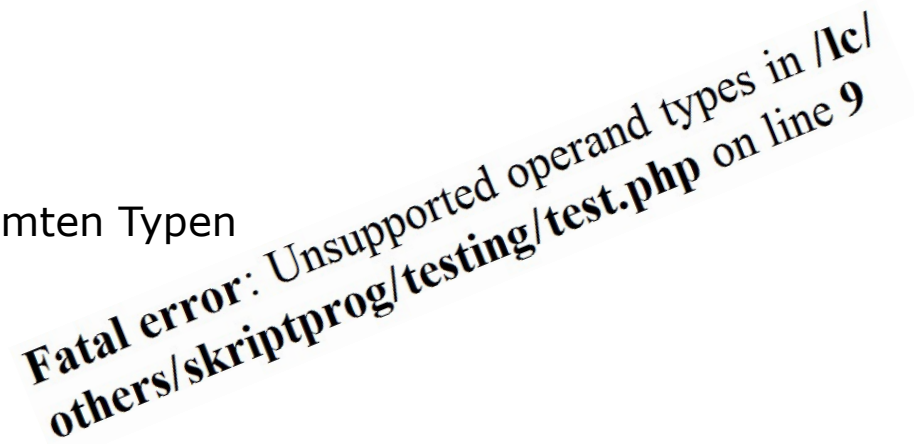
```php
$foo  = '5';                    // '5'
$foo += 3;                      // 8
$foo  = '10 Mann' + $foo;       // 18
$foo += true;                   // 19
$foo += 1.5;                    // 20.5
$foo += array();                // FATAL ERROR
```

> Führt schnell zu Problemen

> Besser explizit casten

  – Datenbanken erwarten oft einen bestimmten Typen

> Implizit casten besser vermeiden!



- Weitere Informationen: https://php.net/manual/language.types.type-juggling.php

Typumwandlung

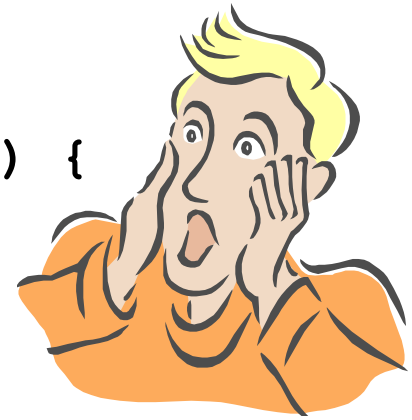
- Explizit:

```php
$foo = '1.5';            // '1.5'
$foo = (double) $foo;    // 1.5
$foo = (int) $foo;       // 1
$foo = (boolean) $foo;   // true
$foo = (string) $foo;    // '1'
```

cast to	false	1	0	1.7	-1.2	''	'1'	'php'	array()
boolean	false	true	false	true	true	**false**	true	**true**	**false**
int	0	1	0	1	-1	0	1	**0**	0
double	0	1	0	1.7	-1.2	0	1	**0**	0
string	''	'1'	'0'	'1.7'	'-1.2'	''	'1'	'php'	ERROR

Vergleich von Variablen

```
if (true == 'php' && 'php' == 0 && 0 != true) {
    echo 'true == "php" == 0 != true';
}
```



> Mit == auf Gleichheit zu prüfen kann zu Problemen führen
  - Beispiel: strpos

$a === $b

- **Typstarker Vergleich!**
- Nur bei gleichem Typen und gleichem Inhalt von $a und $b wird true zurückgegeben

Typschwache Vergleiche mittels ==

	TRUE	FALSE	*1*	*0*	*-1*	*"1"*	*"0"*	*"-1"*	NULL	*array()*	*"php"*	*""*
TRUE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE
1	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	TRUE
-1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
"1"	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
array()	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
"php"	TRUE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	TRUE

Quelle: http://php.net/manual/de/types.comparisons.php

Typstarke Vergleiche mittels ===

	TRUE	FALSE	*1*	*0*	*-1*	*"1"*	*"0"*	*"-1"*	NULL	*array()*	*"php"*	*""*
TRUE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
1	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
0	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
-1	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"1"	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
"0"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
"-1"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
NULL	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
array()	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
"php"	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE
""	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

Quelle: http://php.net/manual/de/types.comparisons.php

Erzeugung eines Strings

- Einfache Anführungszeichen
- Steuerzeichen ($, \n) werden ignoriert (bzw. nicht interpretiert)

- Beispiele:

```
echo 'Einfacher String';
echo 'Auch ein Zeilenumbruch
ist kein Problem, der wird aber beim Rendering im
Browser zumeist aufgehoben';

echo 'I\'ll be back';  // I'll be back
echo 'Dateien: C:\\*'; // Dateien: C:\*

$fuenf = 5;
echo '2+3 = $fuenf';   // 2+3 = $fuenf
```

Erzeugung eines Strings

- Doppelte Anführungszeichen
- Steuerzeichen ($, \n) werden interpretiert
- Variablen werden erkannt
- Beispiele:

```php
echo "Zeilenumbruch folgt\n";

$fuenf = 5;
echo "2+3 = $fuenf";     // 2+3 = 5
echo "2+3 = {$fuenf}";   // 2+3 = 5
```

- Für Strings, die keine Variablen enthalten wird meist ein einfaches Anführungszeichen verwendet
- Für SQL-Queries wird meist ein doppeltes Anführungszeichen verwendet

Verkettung

- Punktoperator
- Beispiele:

```php
$fh = 'FH' . ' ' . 'Aachen'; // 'FH Aachen'

$sql = "SELECT 'id' FROM 'users' "
     . "WHERE 'name' = 'Max' ";

$id = 305;
$ort = 'Raum ' . $id; // 'Raum 305'

$ort .= ' (3. OG)';   // 'Raum 305 (3. OG)'
```

Syntax
Arrays

Eigenschaften

- **Keine feste Größe**
- Assoziative Arrays möglich (eigentlich sogar eher der Standard)
- Mischformen von String und Integern als Key erlaubt
  - > Bedeutet nicht, dass dies auch sinnvoll ist…
- Als Wert ist alles erlaubt!
- Numerische Arrays können Lücken haben
  - > Ebenfalls nicht zu empfehlen…

Erstellung:

```php
$normal = array();
$mix1   = array(2, 4, 8, 'Tomate');
$mix2   = array('foo' => 'bar', 12 => true);
$colors = array('rgb1' => array(0,0,255));
```

Zugriff und Bearbeitung:

```php
$arr = array();   // Alternativ: []
$arr[] = 'EINS'; // array('EINS')
$arr[] = 'ZWEI'; // array('EINS', 'ZWEI')

$arr[0] = 'NEU'; // array('NEU', 'ZWEI')
$arr['a'] = 5;   // array(0 => 'NEU', 1 => 'ZWEI',
                 //       'a' => 5)

echo $arr[1];    // 'ZWEI'
```

Syntax
Sprachkonstrukte

Grundsätzliches bekannt aus Java

- Bedingungen
  - > if
  - > else if
  - > else

- Schleifen
  - > for
  - > while
  - > do-while
  - > break
  - > continue

foreach-Schleife

```php
foreach ($arr as $value) {
    // ...
}

foreach ($arr as $key => $value) {
    // ...
}
```

Weitere Informationen: http://php.net/manual/control-structures.foreach.php

Syntax
Sprachkonstrukte

Dateien mittels *require* und *include* einbinden

- Ermöglicht Auslagerung und Mehrfachnutzung des Codes
- Bindet den Code aus der Datei ein
- Geltungsbereich von der Position des Befehlsaufruf abhängig

```
require './lib/Database.class.php'
```

> „Fatal Error", falls ein Fehler beim Einbinden auftritt
  - Die Ausführung wird sofort abgebrochen
  - Was aber ist mit der eigentlich anzuzeigenden Web-Seite?

```
include './lib/Encoding.class.php'
```

> Warning, falls ein Fehler beim Einbinden auftritt
  - Skript wird weiterausgeführt
  - Web-Seite kann angezeigt werden

Dateien mittels *require* und *include* einbinden

- *include* nur bei Dateien, die **für den Programmablauf optional** sind
  - > I.d.R. ist die Benutzung von *require* sinnvoller, aber bitte ohne Fehler!
- „Geklammerter Syntax“ möglich

```php
require('./lib/Database.class.php');
include('./lib/Encoding.class.php');
```

- Eine Pfadangabe führt dazu, dass nicht im *include_path* gesucht wird
  - > include_path gibt an in welchem Verzeichnis PHP einzubindende Dateien suchen soll (für Bibliotheken zu gebrauchen)

- *require_once* und *include_once* binden die Datei nur einmal ein, auch wenn der Funktionsaufruf öfter erfolgt

```php
require_once('./lib/Database.class.php');
```

Sonstige Befehle

```
bool isset($variable)
```

> Prüft ob eine Variable exisitert und nicht *NULL* ist

```
bool empty($var)
```

> Prüft, ob eine Variable einen Wert enthält
> Für die folgenden Werte wird *TRUE* zurückgegeben:
  - Leere Zeichenkette (''), leeres Array (*array()*)
  - `0, 0.0, '0.0'`
  - *NULL, FALSE*
  - In einer Klasse deklarierte, aber nicht belegt Variable

Sonstige Befehle

```
bool define($name, $value)
```

> Setzt Konstante die in allen Geltungsbereichen verfügbar ist
> Nur für Werte benutzen, die für die ganze Applikation relevant sind

> Beispiele:

```
define('ROOT_DIR', '/my_project');
define('VERSION', '3.1.24');
```

> Auslesen des Wertes **ohne $-Zeichen**:

```
echo VERSION;
```

Syntax
Errors & Exceptions

Exceptions

```
try {
    funcWhichMightThrowException();
} catch (Exception $e) {
    echo 'Problem calling ...: ' . $e->getMessage();
}
```

- Eigene Exception-Klassen können von Exception abgeleitet werden

```
class MyException extends Exception {...}
throw new MyException();
```

- Beim Abfangen kann der Exception-Name angegeben werden

- Die meisten PHP-Funktionen werfen im Fehlerfall keine Exception

Weitere Informationen: http://php.net/manual/language.exceptions.php

Syntax
Funktionen

Wie Variablen sind auch Funktionen nicht typsicher!

- Funktionen können manchmal Zahlen, manchmal Strings zurückgeben
- In der Regel ist es aber sinnvoll, dass eine Funktionen nur einen Typen zurückgibt.

Angabe von Vorgabewerten möglich

- Beispiel:

```php
function machKaffee($typ = 'Kaffee') {
    return 'Eine Tasse ' . $typ;
}

echo machKaffee();              // 'Eine Tasse Kaffee'
echo machKaffee('Cappucino');   // 'Eine Tasse Cappucino'
echo machKaffee('Espresso');    // 'Eine Tasse Espresso'
```

Syntax
Funktionen

Call-By-Reference

```php
function toggle(&$value) {
    $value = ($value + 1) % 2;
}
$a = 1;
toggle($a);
// $a === 0
```

Variable Anzahl von Parametern

```php
function tolerant() {
    $numargs = func_num_args(); // Anzahl
    $arg1 = func_get_arg(0);    // 1. Parameter
    $args = func_get_args();    // Array mit Parametern
}
tolerant(1, 2, 4);
```

Syntax
Funktionen

Variablen im Geltungsbereich außerhalb der Funktion sind nur durch „global"-Schlüsselwort zu erreichen

- Beispiel:

```
$a = 1;
$b = 2;
function incA() {
    global $a;     // $b ist unbekannt
    $a++;
}

incA();
echo $a; // 2
```

Lambdas

- Anonyme Funktionen ohne Namen
- Funktionszeiger wird in Variable gespeichert
  - > Variable kann wie Funktion benutzt werden
  - > Kann als Argument übergeben werden
- Einfacher Austausch oder die Übergabe von Funktionen als Parameter möglich

Lambdas

```
$eineFunktion = function () {
   echo 'Ich bin eine anonyme Funktion!';
};

// Aufruf:
$eineFunktion();
```

Lambdas

- Es geht auch mit Parametern

```php
$array = range(1, 10);              // Array mit [1,..,10]

$mySort = function($a, $b) {  // Anonyme Funktion
    return $b - $a;             // gespeichert in $mySort
};

usort($array, $mySort);      // Eigene Funktion
                             // zur Sortierung nutzen
```

Weitere Informationen: https://php.net/manual/en/function.usort.php

Lambdas

```
function echoText(callable $callback) {
    echo $callback();
}

echoText(function () {
    return 'Ich bin cool';
});
echoText(function () {
    return 'Ich bin cooler';
});

$eineFunktion = function () {
    return 'Ich der coolste!';
};
echoText($eineFunktion);
```

Ein callable ist eine Funktion, die aufgerufen werden kann

Wir sehen hier das Prinzip einer Wegwerffunktion

Closures

```
$dayPrepare = function() {
    $dayNames = array('Mo', 'Di', 'Mi', 'Do',
                      'Fr', 'Sa', 'So');

    return function($day) use ($dayNames) {
        return $dayNames[$day];
    };
};

$dayNameGetter = $dayPrepare();

echo $dayNameGetter(1); // Di
```

Anonyme Funktion in einer anonymen Funktion

Variable *$dayNames* aus äußerem Geltungsbereich wird benutzt

Aufruf der inneren anonymen Funktion**, die immer noch Zugriff auf die Variable *$dayNames* hat**. Wir gehen hier von Lambdas über zu Closure

Closures

- Ein Lambda wird zum **Closure,** wenn es **Zugang zu Variablen** bekommt**, die normalerweise nicht zur Verfügung stehen** würden.
- In PHP gibt es Closures für anonyme und benannte Funktionen. Leider können viele Programmiersprachen nicht beides, so dass unglücklicherweise Closures oft mit anonymen Funktionen/Lambdas gleichgesetzt werden

```php
function func() {
    $i = 0;
}
isset($i); // <- false
```

```php
function func() {
    $i = 1;
    return function() use ($i) {
        echo $i;
    };
}

$x = func();
$x(); // 1
```