EINE EINFÜHRUNG IN

PHP - HYPERTEXT PREPROCESSING

PHP

HISTORIE UND UMGEBUNG

HISTOR

PHP - HYPERTEXT PREPROCESSING

- PHP wurde 1995 von Rasmus Lerdorf entwickelt.
  Der Begriff stand damals für Personal Home Page Tools.

- PHP ist in C geschrieben.

- PHP 3 wurde von Andi Gutmans und Zeev Suraski (Zend Technologies Ltd.) neu geschrieben. Lerdorf kooperierte dabei mit Gutmans und Suraski.

- Diese Version brachte die Verbreitung von PHP bedeutend voran und löste PERL ab.

PHP 4

- Mit PHP 4 wurden Ausführungsgeschwindigkeit und die Sicherheit bei Verwendung globaler Variablen verbessert:
- Unterstützung für weitere Webserver
- Sessionmanagement
- Ausgabepufferung
- Neue Befehle und Sprachkonstrukte.

PHP5

- Seit 2004 gibt es Version 5.0
  Sie basiert auf der Zend Engine II:

- Verbessertes Objektmodell, objektorientierte Anwendungen, Exceptions, Reflections;
  SQLite Integration;
  Erweiterungen bei XML- und DOM-Handhabung.

PHP7

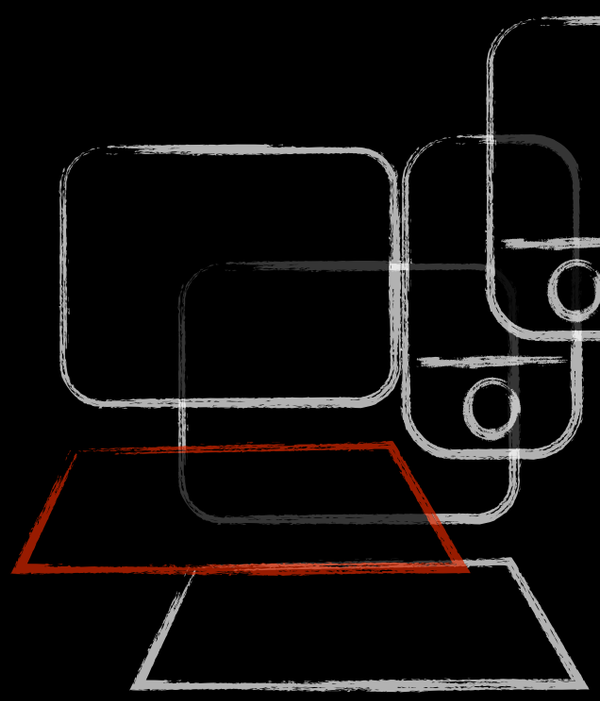
- Am 01.02.2018 wurden das Release 7.2.2 veröffentlicht.
- Alle Informationen zu PHP sind unter php.net zu finden.

EIGENSCHAFTEN VON PHP

- PHP läuft serverseitig und kommt fast ausschließlich im Kontext von HTML und Webanwendungen vor.
- Ist eine Scriptsprache.
- Kann prozedural oder objektorientiert verwendet werden.
- Ist dynamisch typisiert.

Clients

Server



REQUEST
index.php?email=…

erzeugtes HTML

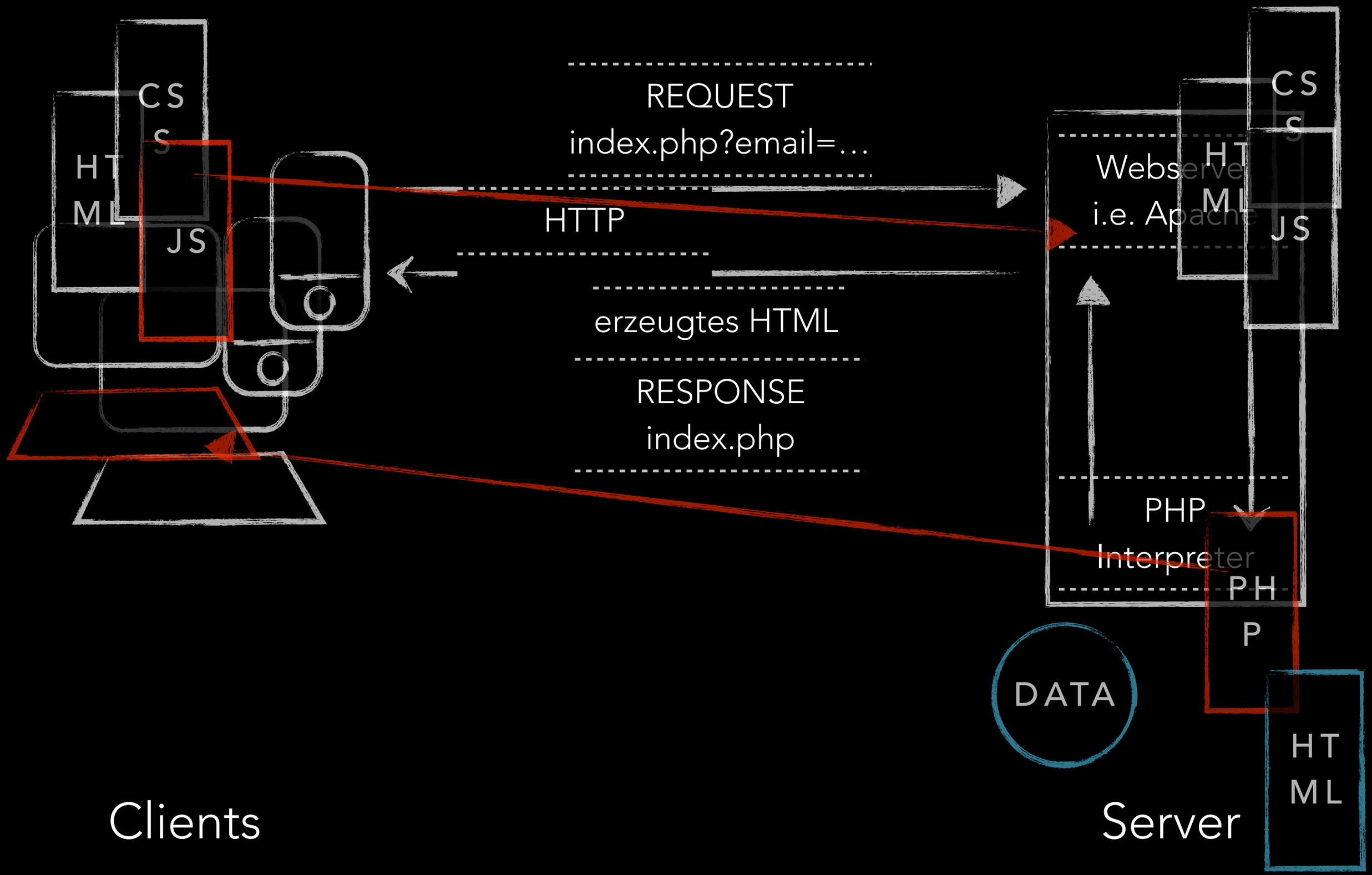
RESPONSE
index.php

Webserver
i.e. Apache

PHP
Interpreter

Clients

Server


CS
S
HT
ML
JS
REQUEST
index.php?email=…
HTTP
erzeugtes HTML
RESPONSE
index.php
Webserver
i.e. Apache
CS
S
HT
ML
JS
PHP
Interpreter
PH
P
DATA
HT
ML
Clients
Server

INTEGRATED DEVELOPMENT ENVIRONMENT SOFTWARE, EDITOREN

- Eclipse, Aptana, PHPStorm, Netbeans, Visual Studio
- Integriertes Debugging, Dateiverwaltung, Projektstrukturen

EDITOREN

- Notepad++, Sublime Text, Coda, …

DIREKTE EINBINDUNG IN HTML MÖGLICH

Der Programmcode kann in die HTML-Quelldatei geschrieben werden.

```
<title><?php echo $page->title; ?></title>


<html> ...
   <?php echo 'hallo $a Welt<br>' . "\n"; ?>
   ...
</html>
```

EINBINDUNG IN HTML

```
In komplexen Applikationen werden PHP und HTML getrennt.

ROOT
| index.php
| app
|   | templates/template.html
|   | includes/functions.php
|   | includes/classes.php
|   | main.php
| assets
|   | css/style.css
|   | js/script.js
| libraries
   | jquery
   | bootstrap
. . .
```

BASIS-FUNKTIONSUMFANG VON PHP

SPRACHE

VARIABLEN

VARIABLEN

```
Jede Variable beginnt mit einen $

$a, $_a, $a1
$lowerCamelCase, $UpperCamelCase
$object->member


Konstanten werden per Konvention groß geschrieben:

define('PI', 3.14152);
echo PI;
```

TYPEN WERDEN DURCH WERTEZUWEISUNG ODER TYPECASTING ERZEUGT

```
// $a ist ein Integer!
   $a = 1234;

// $b ist ein Double mit !
// dem Wert 1234!
   $b = (double) $a;
```

SCHWACH TYPISIERTE VARIABLEN

```
Typen können überschrieben werden.

String        Alphanumerische Zeichenketten
Integer       Ganzzahlen
Float         Fließzahlen
Boolean       true oder false (PHP 4: 1 oder 0)
Array            Wertelisten
```

DIE FUNKTION GETTYPE()

```
// Den Typ abfragen

$a = 2;
(float) $a;
echo 'GETTYPE: ' . gettype($a) . BR;
```

TYPE CASTING MIT PHP

```
$a = 4;
(double) $a;

(int), (integer)                cast to integer
(bool), (boolean)               cast to boolean
(float), (double), (real)       cast to float
(string)                        cast to string
(binary)                        cast to binary string (PHP 6)
(array)                         cast to array
(object)                        cast to object
(unset)                         cast to NULL (PHP 5)
```

VARIABLEN PRÜFEN

```
// Prüfen, ob eine Variable existiert
echo 'IS_NULL: '       .  is_null($f) . BR;
echo 'NOT IS_NULL: '   . !is_null($f) . BR;

// Prüfen, ob in einer Variablen ein Wert steht
echo 'ISSET: '         .  isset($f) . BR;
echo 'NOT ISSET: '     . !isset($f) . BR;
```

DIE FUNKTION EMPTY()

```
// Abfragen, ob die Variable existiert
// und ein Wert enthalten ist

echo 'EMPTY: '     .  empty($a) . BR;
echo 'NOT EMPTY: ' . !empty($a) . BR;
```

FLIEßKOMMA-PROBLEM!

```
echo floor((0.1+0.7)*10);
```

KONSTANTEN

```
Konstanten ändern ihren Wert zur Laufzeit nicht mehr.

define('CONST', 'konstanter Wert');
define('LINE', "\r\n");
```

ARITHMETIK UND ZUWEISUNGEN

ARITHMETRISCHE OPERATOREN I

```
$a + $b;        -> Addition
$a - $b;        -> Subtraktion
$a * $b;        -> Multiplikation
$a / $b;        -> Division
$a % $b;        -> Modulo (Ganzahliger Rest der
                   Division von $b/$a)

$a = 2;
$b = 3;
echo $a + $b . "\n";
```

ARITHMETRISCHE OPERATOREN II

```
$a += 5;          // entspricht $a = $a + 5;
$a *= 2;          // entspricht $a = $a * 2;
$i++;             // entspricht $i = $i + 1;
$i--:             // entspricht $i = $i - 1;
```

```
$a = 2;
echo $a += 5 . '\n';

$i = 1;
echo $i++ . '\n';
echo ++$i . '\n';
```

NOTATIONEN FÜR ANDERE ZAHLENSYSTEME

```
// Oktalzahl (entspricht 83 dezimal)
   $a = 0123;
!
// Hexadezimalzahl
// (entspricht 2989 dezimal)
$a = 0xbad;
```

WERTELISTEN SIND DATENCONTAINER.

ARRAYS

ARRAYS - WERTELISTEN ALS DATENCONTAINER

```
$monat[0] = 'Januar';
$monat[1] = 'Februar';
$monat[2] = 'Maerz';
...
$monat[8] = 'September';
$monat[9] = 'Oktober';
$monat[10] = 'November';
$monat[11] = 'Dezember';


Iteration über numerisch indizierte Arrays:

for ($i=0;$i<sizeof($monat);$i++)
{
  echo $monat[$i] . '\n';
}
```

EIN EINDIMENSIONALES ARRAY ALS TABELLE ABGEBILDET

JANUAR
FEBRUAR
MÄRZ
...
SEPTEMBER
OKTOBER
NOVEMBER
DEZEMBER

EIN ZWEIDIMENSIONALES ARRAY ALS TABELLE II

JANUAR	31
FEBRUAR	28
MÄRZ	31
...	
SEPTEMBER	30
OKTOBER	31
NOVEMBER	30
DEZEMBER	31

EIN NUMERISCH UND ASSOZIATIV INDIZIERTES ZWEIDIMENSIONALES ARRAY

```
$monat[0]['Name']='Januar';  $monat[0]['Tage'] = 31;
$monat[1]['Name']='Februar'; $monat[1]['Tage'] = 28;
$monat[2]['Name']='Maerz';   $monat[2]['Tage'] = 31;
...!
$monat[8]['Name']='September'; $monat[8]['Tage'] = 30;
$monat[9]['Name']='Oktober'; $monat[9]['Tage'] = 31;
$monat[10]['Name']='November'; $monat[10]['Tage'] = 30;
$monat[11]['Name']='Dezember'; $monat[11]['Tage'] = 31;



$monat[0] = array('Name' => 'Januar', 'Tage' => 31);
```

ARRAY SCHREIBWEISE II

```
$monat = array(
   0 => 'Januar',
   1 => 'Februar'
);

$monat = array(
   0 => array('Januar' ,  31),
   1 => array('Februar' , 28)
}

$monat[0][0] = 'Januar';
$monat[0][1] = 31;
$monat[1][0] = 'Februar';
$monat[1][1] = 28;
```

METHODEN FÜR ARRAYS

ARRAY-OPERATOREN

```
$a + $b      Vereinigung von $a und $b.

$a==$b       Gleichwerigkeit TRUE wenn $a und $b die
             gleichen Schlüssel- und Wert-Paare enthalten.

$a === $b    Identität!TRUE wenn $a und $b die gleichen
             Schlüssel- und Wert-Paare in der gleichen
             Reihenfolge enthalten.!

$a!=$b       Ungleichheit    TRUE wenn $a nicht gleich $b ist.

$a<>$b       Ungleichheit    TRUE wenn $a nicht gleich $b ist.!

$a !== $b    nicht identisch! TRUE wenn $a nicht identisch
             zu $b ist.
```

BEFEHLE IM ARRAY KONTEXT

```
array_pad()  Pad array to the specified length with a value

list()       Assign variables as if they were an array

count()      Count all elements in an array,
             or something in an object!

range()      Create an array containing range of elements
```

ARRAY_PAD()

```
$input = array(12, 10, 9);

$result = array_pad($input, 5, 0);
// result is array(12, 10, 9, 0, 0)

$result = array_pad($input, -7, -1);
// result is array(-1, -1, -1, -1, 12, 10, 9)
```

LIST()

```
$info = array('coffee', 'brown', 'caffeine');

// Listing all the variables:

list($drink, $color, $power) = $info;
echo "$drink is $color and $power makes it special.\n";
```

COUNT()

```
$b[0] = 7;! $b[5] = 9;! $b[10] = 11;

$result = count($b);   // $result == 3
```

```
array_change_key_case
array_chunk
array_column
array_combine
array_count_values
array_diff_assoc
array_diff_key
array_diff_uassoc
array_diff_ukey
array_diff
array_fill_keys
array_fill
array_filter
array_flip
array_intersect_assoc
array_intersect_key
array_intersect_uassoc
array_intersect_ukey
array_intersect
array_key_exists
array_keys
array_map
array_merge_recursive
array_merge
array_multisort
array_pad
array_pop
array_product
array_push
array_rand
array_reduce
array_replace_recursive
array_replace
array_reverse
array_search
array_shift
array_slice
array_splice
array_sum
array_udiff_assoc
array_udiff_uassoc
array_udiff
array_uintersect_assoc
array_uintersect_uassoc
array_uintersect
array_unique
array_unshift
array_values
array_walk_recursive
array_walk
```

ARRAY_KEY_EXISTS()

Prüft, ob ein Schlüssel existiert:

```
$search_array = array('first' => 1, 'second' => 4);

if (array_key_exists('first', $search_array)) {
    echo "The 'first' element is in the array";
}
```

ARRAY_KEYS()

Gibt ein Array mit den Schlüsseln eines Arrays zurück:

```
$array = array(0 => 100, "color" => "red");
print_r(array_keys($array));

$array = array("blue", "red", "green", "blue", "blue");
print_r(array_keys($array, "blue"));
```

ARRAY_MERGE()

```
Vereinigt zwei Arrays, doppelte Keys werden überschrieben.

$array1 = array("color" => "red", 2, 4);
$array2 = array("a", "b", "color" => "green",
                "shape" => "trapezoid", 4);

$result = array_merge($array1, $array2);
```

```
-> Array
(
    [color] => green
    [0] => 2
    [1] => 4
    [2] => a
    [3] => b
    [shape] => trapezoid
    [4] => 4
)
```

ARRAY_POP()

```
Löscht den letzen Eintrag eines Array und reindiziert.

$stack = array("orange", "banana", "apple", "raspberry");

$fruit = array_pop($stack);

-> Array (
    [0] => orange
    [1] => banana
    [2] => apple
)
```

ARRAY_PUSH()

```
$stack = array("orange", "banana");

array_push($stack, "apple", „raspberry");
```

```
-> Array
(
    [0] => orange
    [1] => banana
    [2] => apple
    [3] => raspberry
)
```

ARRAY_SHIFT()

```
Gibt den ersten Eintrag zurück und entfernt diesen
aus dem Array.

$stack = array("orange", "banana", "apple", raspberry");

$fruit = array_shift($stack);

-> Array
(
    [0] => banana
    [1] => apple
    [2] => raspberry
)
```

ARRAY_SPLICE()

```php
$input = array("red", "green", "blue", "yellow");

array_splice($input, 2);
// $input is now array("red", "green")

$input = array("red", "green", "blue", "yellow");

array_splice($input, 2, -2);
// $input is now array("red", "yellow")
```

VERGLEICHSOPERATOREN

```
==     // Gleich (Wertevergleich)
!=     // Ungleich (Wertevergleich)
===    // Gleich (Typen- und Wertevergleich)
!==    // Ungleich (Typen- und Wertevergleich)
>      // Größer
<      // Kleiner
>=     // Größer gleich
<=     // Kleiner gleich
```

TYPENSICHER VERGLEICHEN IST STANDARD

```
=     // Wertezuweisung
==    // Gleich (vergleichend, ist genau gleich)
===   // Typensicheres Gleich
!==   // Typensicheres Ungleich
```

```
$a = mysql_select_db(...);

$a == 1;     // richtig
$a === 1;    // falsch

true == (boolean)$a
```

DEN ABLAUF EINES PROGRAMMS STEUERN:
WIEDERHOLUNGEN, ABFRAGEN UND VERZWEIGUNGEN.

KONTROLLSTRUKTUREN

IF

```
$a = 4;
$b = 3;

if ($a > $b)
  print 'a ist groesser als b';
```

IF

Falls man mehr als einen Befehl hat, der ausgeführt werden soll, so muß die Befehle in geschweifte Klammern einschließen:

```
if ($a>$b) {
  print 'a ist groesser als b';
  $b = $a;
}
```

ELSE

```
if ($a>$b)
  print 'a ist groesser als b';
else
  print 'a ist nicht groesser als b';
```

Zwei Anweisungen, die abhängig von einer Bedingung alternativ ausgeführt werden sollen.

ELSEIF

```
if ($a > $b) {
    print 'a ist groesser als b';
} elseif ($a == $b) {
    print 'a ist gleich b';
} else {
    print 'a ist kleiner als b';
}
```

Bedingungen mit mehreren Varianten.

TERNÄRE SCHREIBWEISE

```
$a < $b
      ? 'a ist kleiner als b'
      : '';

$a < $b ? 'a ist kleiner als b' : '';
```

Entspricht -> wenn $a < $b, dann ..., sonst ...

WHILE

```
Solange die Bedingung `expr` erfüllt ist, wird die Anweisung
`statement` ausgeführt.

WHILE (expr) statement { . . . }

$i=1;
while ($i<=10) {
  print $i++;
}


// Alternativ:
WHILE (expr) : statement ... ENDWHILE;
```

DO WHILE

```
$i=0;

do {
    print $i++;
} while ($i>0);
```

Do führt einen ersten Durchlauf durch den Codeblock und prüft erst dann, ob die Bedingung für einen weiteren Durchlauf noch gegeben ist.

FOR

```
for ($i=1; $i<=10; $i++) {
    print $i;
}


for ($i=1;;$i++) {
    if ($i > 10) {
        break;
    }
    print $i;
}
```

Codezeilen wiederholt ausführen!
Über eine Indexvariable wird die Anzahl der Iterationen festgelegt.

FOR OHNE EXPRESSIONS?

```
$i=1;

for (;;) {
    if ($i>10) {
      break;
    }
    print $i;
    $i++;
}

Seltsam: keine Iterationsparameter,
aber es funktioniert.
```

SWITCH

```
switch ($i) {
    case 0:
        print 'i ist gleich 0';
        break;
    case 1:
        print 'i ist gleich 1';
        break;
    default:!
        print 'i ist ungleich (a oder 1)';
        break;
}
```

Ein Merfachschalter, ähnlich der If – Else Anweisung,
aber werteabhängig und schneller.

SWITCH VARIATION

```
switch($i) {
    case 0:
        print 'i ist gleich 0';
        break;
    case 1:
    case 2:
        print 'i ist gleich 1 oder 2';
}
```

„Durchfallen lassen“ einzelner Cases:

FOREACH MIT WERTEN

```
$tage = array('Montag', 'Dienstag', 'Mittwoch',
              'Donnerstag', 'Freitag', 'Samstag',
              'Sonntag');

foreach ($tage as $tag)
   echo $tag;

// dasselbe mit for
for ($i=0;$i<count($tage);$i++)
   echo $tage[$i];
```

Über Array iterieren: Wert für Wert aus einer Werteste erfragen oder abarbeiten.

FOREACH MIT SCHLÜSSEL/WERTEN

```
foreach ($myArray as $key=>$val)
    echo '$key: $val\n';

Abfragen von Schlüssel und Wert.
```

CONTINUE

```
$i = 10;
while ($i > 0) {
    $i--;
    if (!($i%2))
        continue;
    echo $i;
}
```

Schleifen iteration gezielt fortsetzen, ohne die weiteren Programmzeilen auszuführen.

TRY - CATCH

- Mit try etwas versuchen, mit Catch einen Funktionsfehler abfangen.
- Die Fehlermeldung selbst erhalten Sie mit der Methode getMessage().
- Dateinamen und Zeilennummer zur Fehlermeldung gibt es mit getLine() und getFile().

FEHLER WERFEN UND FEHLER FANGEN

```php
// Eine Funktion wirft einen Fehler
    function inverse ( $x = 0 ) {
        if ( !$x ) {
            throw new Exception('Division by Zero!!' . BR);
        } else {
            return 1/$x;
        }
    }
// Der Programmablauf fängt den Fehler auf und gibt ihn aus
    try {
        echo inverse(5) . BR;
        echo inverse(0) . BR;
    } catch ( Exception $e ) {
        echo 'Exception catched: ' . $e->getMessage() . BR;
    }
    echo 'continuing ...';
```

EINBINDEN VON EXTERNEN PROGRAMMTEILEN

INCLUDE UND REQUIRE

```
include('file.inc.php');

if ($bedingung){
  include('file.inc.php');
}

require 'other-file.php'


Fehlerhafte include-Dateien erzeugen Warnings und arbeiten nach
Möglichkeit weiter.

Fehlerhafte require-Dateien brechen den Programmablauf ab.
```

INCLUDE_ONCE ODER REQUIRE_ONCE

```
include_once 'file.inc.php';
require_once 'class.db.php';
```

'once' stellt sicher, dass Dateien nur einmal eingebunden werden. Ein weiterer Versuch von include/require erzeugt einen Zeiger auf die erste Einbindung.

FUNKTIONEN

```
function foo($arg_1, $arg_2, ..., $arg_n) {!
  echo 'Example function.\n';!

  return $retval;!
}
```

Die Argumente sind in Anzahl und Reihenfolge bindend.
Es kann ein Rückgabewert definiert werden.

ARGUMENTE MIT DEFAULTWERT

```
function foo($arg = defaultValueForArg) {
    ...
}
```

REFERENZ ALS ARGUMENT VERWENDEN

```
function foo2 (&$st) {
    $st .= ' und etwas mehr.';
}

$str = 'Dies ist ein String';

echo $str;
-> Dies ist ein String

foo2 ($str);
echo $str;

-> Dies ist ein String und etwas mehr.
```

EINE FUNKTION ZUM ADDIEREN VON ZWEI ZAHLEN

```
function add ($a, $b) {
    $true = true;
    while ($true) {
        if (empty($a) $true = false;
        if (empty($b) $true = false;
        if (   gettype($a) !== ‚integer‘
             && gettype($a) !== ‚float‘
             && gettype($a) !== ‚real‘) {
            $true = false;
        }
        if (   gettype($b) !== ‚integer‘
             && gettype($b) !== ‚float‘
             && gettype($b) !== ‚real‘) {
            $true = false;
        }
    }
    if ($true) {
        return $a + $b;
    } else {
        return false;
    }
}
```

AUSGABEN FORMATIEREN

FORMATIERTE AUSGABE - PRINTF

```
$tag = 13;! $monat = 5;! $jahr = 1984;

$format = '%02d.%02d.%04d\n';

printf($format, $tag, $monat, $jahr);

Ausgabe: 13.05.1984
```

FORMATIERTE AUSGABE - PRINTF

```
%b  Integer     Binärzahl
%c  Integer     Zeichen mit entsprechendem ASCII-Code
%d  Integer     Dezimalzahl, ggf. mit Vorzeichen
%u  Integer     Dezimalzahl ohne Vorzeichen
%f  Double      Fließkommazahl
%o  Integer     Oktalzahl
%s  String      String
%x  Integer     Hexadezimalzahl mit Kleinbuchstaben
%X  Integer     Hexadezimalzahl mit Großbuchstaben
```

FORMATIERTE AUSGABE - PRINTF

```
Angenommen, in $lang stehe die Sprache:

if ($lang == 'de') {
  $format = 'Heute ist der %1\$02d.%2\$02d.%3\$04d.';
}
elseif ($lang == 'en') {
  $format = 'Today's date is %3\$04d-%2\$02d-%1\$02d.';
}
else {
  // unbekannte Sprache -> wir geben nur die Zahlen aus
  $format = '%3\$04d-%2\$02d-%1\$02d';
}

printf($format, $tag, $monat, $jahr);
```

NUMBER_FORMAT

```
string number_format (float number
                      [,
                          int decimals
                          [,
                              string dec_point ,
                              string thousands_separator
                          ]
                      ])

$num = number_format($num);
$num = number_format($num, 2);
$num = number_format($num, 2, ',', '.');
```

NUMBER_FORMAT

```
function Euro ($preis) {
  return sprintf('%s EUR\n',
    number_format($preis, 2, ',', '.')
  );
}

echo Euro(17392.48365);

Ausgabe: 17.392,48 EUR
```

SCHREIBEN EINES LOGINSKRIPTES

EIN PROGRAMM ENTWICKELN

CODIN

DAS ZUSAMMENSPIEL ZWISCHEN HTML/CSS UND PHP

LOGIN FORM

LOGIN FORM

```
<!doctype html>!
<html>
<head>!
    <meta charset="utf-8">
    <title>Anmeldung</title>
</head>
<body>
    <h1>Login</h1>
    <p>Loggen Sie sich mit Usernamen und Passwort ein.</p>
    <form action='login.php' method='post' >
        <label>Username:</label><br />
        <input type='text' name='username' value='' /><br />
        <label>Passwort:</label><br />
        <input type='password' name='password' value='' /><br />
        <input type='submit' name='action' value='login' /><br />
    </form>
</body>
</html>
```

SEMANTISCHE FORM MIT EINGABEFELDGRUPPE UND EINGABEKOMPONENTEN

```
<form action="login.php?test=test" method="post">
     <fieldset>
          <legend>Login</legend>

          <div class="fgroup fgroup-horizontal">
               <label for="email">E-Mail</label>
               <input  id="email" type="email"
               name="email" value="michael@zenbox.de">
               <p class="help">Hinweistextzeile</p>
          </div>

          <div class="fgroup fgroup-horizontal">
               <label for="password">Passwort</label>
               <input  id="password" type="text"
               name="password" value="geheim">
               <p class="help">Hinweistextzeile</p>
          </div>

          <div class="fgroup fgroup-horizontal">
               <input type="submit"
                    name="action" value="login">
               <input type="reset"
                    name="action" value="zurücksetzen">
          </div>


     </fieldset>
</form>
```

LOGIN.PHP - CGI VARIABLEN

```
// Entgegennehmen der Daten über das Common Gateway Interface
// $_POST, $_GET, $_REQUEST

if ( isset($_POST) ) {
   switch ($_POST['action']) {
      case 'login':
         $username = $_POST['username'];
         $password = $_POST['password'];
         $login = true;
      break;
      default:
         header('location: index.html’);
   }
}
```

Schreiben Sie eine Webseite mit einer Navigation aus Links
auf die der Server veschiedenen Antworten liefert:

```
<ul>
   <li><a href="server.php">Link 1</a></li>
   <li><a href="server.php">Link 2</a></li>
   <li><a href="server.php">Link 3</a></li>
</ul>
```

Nennen Sie die Serverdatei server.php.

Jeder Eintrag soll einen Titel/Text ausgeben:

```
<h1>Link1</h1>
<p>Lorem ipsum...</p>
```

ACHTUNG MIT $_REQUEST

- $_REQUEST
  -> array_merge($_GET, $_POST, $_COOKIE)
- Initialisierungswert 'request_order' beachten! (CGP)

PRÜFEN UND VERZWEIGEN

```
//Prüfen, ob die Eingaben gültig sind:

if ($login === true) {

   $is_valid = true;
   $is_valid = validate($username, $is_valid);
   $is_valid = validate($password, $is_valid);

   if ($is_valid === true) {
      header('location: start.html');
   }

} else {
  header('location: index.html');
}
```

DIE VALIDATE FUNKTION

```
// Prüfen, ob die formalen Voraussetzungen erfüllt wurden

function validate($value, $is_valid){
    if ($is_valid === true) {
        $length = strlen(trim($value));
        if ($length > 0) {
            return true;
        }
    }
    return false;
}
```

FORMULAREINGABEN DES USERS ENTHALTEN POTENTIELLE GEFAHREN

- code injections (unerwünschtes HTML)
- sql injections

WEITERE FUNKTIONEN ZUM VALIDIEREN VON FORMULAREINGABEN

```
// führende/anhängende Leerzeichen löschen
// -> Tabs, Leerzeichen, evt. new line
$value = trim($value);

// SQL Injection unschädlich machen
// escaped Schrägstriche und Anführungszeichen
// / -> \/
// " -> \"
// ' -> \'
$value = mysqli_real_escape_string($link, $value);

// HTML in Ausgabezeichen konvertieren
// <b> -> &lt;b&gt;
$value = htmlspecialchars($value);

// Optionen: HTML Elemente entfernen
$value = strip_tags($value, '<b><i>');
```

WEITERE FUNKTIONEN ZUM VALIDIEREN VON FORMULAREINGABEN

```
preg_replace($pattern, $replace, $data);
```

Legen sie eine allgemeine Methodensammlung "functions.php" an. Binden Sie die Methodensammlung in ihr Skript ein.

Hinweis:
"include_once" bricht die Codeausführung bei Fehlern nicht ab, sondern erlaubt die Fehleranalyse.

Legen Sie dazu folgende Ordnerstruktur an:

```
PHP
| login.php
| app
|   | includes/functions.php
|   | …
| …
```

Schreiben Sie dort eine Funktion "destroyInjectionCode()", die die mögliche Injections in den Eingaben des Users unschädlich macht.

Achten Sie darauf, dass keine Leereingaben, keine SQL-Injections und keine HTML Elemente übergeben werden können.

Schreiben Sie eine Funktion "validateUserInputLength()", die prüft, ob der User etwas eingegeben hat (Länge der Eingabe korrekt).

Der Username soll mindestens 3 und höchstens 20 Zeichen lang sein, das Passwort mindestens 6, höchstens 12 Zeichen lang sein.

Hinweis:
validateUserInputLength() bekommt sowohl den Wert, als auch die erwartete Länge als Argument mit. Achten sie ein der Funktion auf problematische Leerzeichen.

-> validateUserInputLength($value, $length)

ERROR REPORTING

FEHLERAUSGABE DES PHP PROZESSORS

FEHLERAUSGABE IM SKRIPT STEUERN

```
// Alle Fehler ausschalten
error_reporting(0);

// Einfache Laufzeitfehler ausgeben
error_reporting(E_ERROR | E_WARNING | E_PARSE);

// Ausgabe von E_NOTICE um nicht
// initialisierte Variablen auszugeben
error_reporting(E_ERROR | E_WARNING | E_PARSE | E_NOTICE);

// Alle Fehler ausser E_NOTICE ausgeben
error_reporting(E_ALL & ~E_NOTICE);

// Alle PHP Fehler ausgeben
error_reporting(E_ALL);
error_reporting(-1);

// Entspricht der obigen Anweisung
ini_set('error_reporting', E_ALL);
```

FEHLER SIND NUR FÜR ENTWICKLER

- Fehler nur in der Entwicklungsumgebung ausgeben.
- Fehlerausgabe in der Liveumgebung vermeiden - Fehler geben ungewollte Hinweise auf das System.

Schreiben Sie eine Konfigurationsdatei "config.php", die von allen Skripten verwendet werden kann. Diese soll zunächst das Fehlerreporting mit ini_set() einstellen.

Das Fehlerreporting soll nur unter "localhost" Bedingungen eingeschaltet sein.

Hinweis:
Die Servervariable $_SERVER enthält Laufzeitumgebungswerte. $_SERVER['SERVER_NAME'] liefert den
Konfigurationsdateien werden in der Regel vor allen anderen eingebunden.

PHP
| login.php
| app
| | includes/config.php
| | …
| …

ANBINDUNG AN EINE SQL DATENBANK

MIT MYSQL DATEN ARBEITEN

EINE DATENBANKVERBINDUNG AUFBAUEN

```
$host     = 'localhost'; $user='root'; $password='';
$database = 'application';
$sql      = "SELECT * FROM user WHERE userId=1;";

$mysqli = new mysqli($host, $user, $password, $database);

if ($mysqli->connect_errno) {
    echo "Failed to connect to MySQL: "
    . $mysqli->connect_error;
}
```

EINE ANFRAGE (QUERY) AN DIE DATENBANK ABSETZEN

```
$query = "SELECT userId, username, email FROM user;";

$result = $mysqli->query($query);
```

DATEN AUS DEM ERGEBNISOBJEKT $RESULT AUSGEBEN

```
$result = $mysqli->query($query);

while ($row = $result->fetch_assoc()) {
    echo " id = " . $row['id'] . "\n";
}
```

DATEN EINFÜGEN

```
INSERT INTO table_name [ (feld_name,...) ] VALUES (werte,...)
[,(werte,...), [werte,...]]
```

```
INSERT INTO
   user
   (username, email, password)
VALUES
   ('Michael','michael@zenbox.de', 'geheim'),
   ('Paula',  'paula@zenbox.de',   'auch#geheim');
```

Datenbank und Tabelle anlegen

Legen sie unter http://localhost/phpmyadmin/ eine Datenbank "application" an. (Kollation: utf-8-general-ci).

Legen Sie darin eine Tabelle "user" mit den folgenden Feldern an:

```
userId    int           primary   autoincrement
username  varchar(20)   not null
email     varchar(255) not null
password  varchar(12)   not null
```

Legen Sie drei Userdatensätze an.

Daten in Dokument ausgeben – Teil 1

Schreiben Sie ein Dokument userlist.php, das sämtliche Usernamen aus der Datenbank ausliest und in einer HTML-Navigationsliste ausgibt.

Daten in Dokument ausgeben – Teil 2

Mit Klick auf einen Eintrag sollen die Userdaten einzeln in einem Formular mit Texteingabefeldern angezeigt werden. Hier kann der Anwender Daten ändern und speichern.

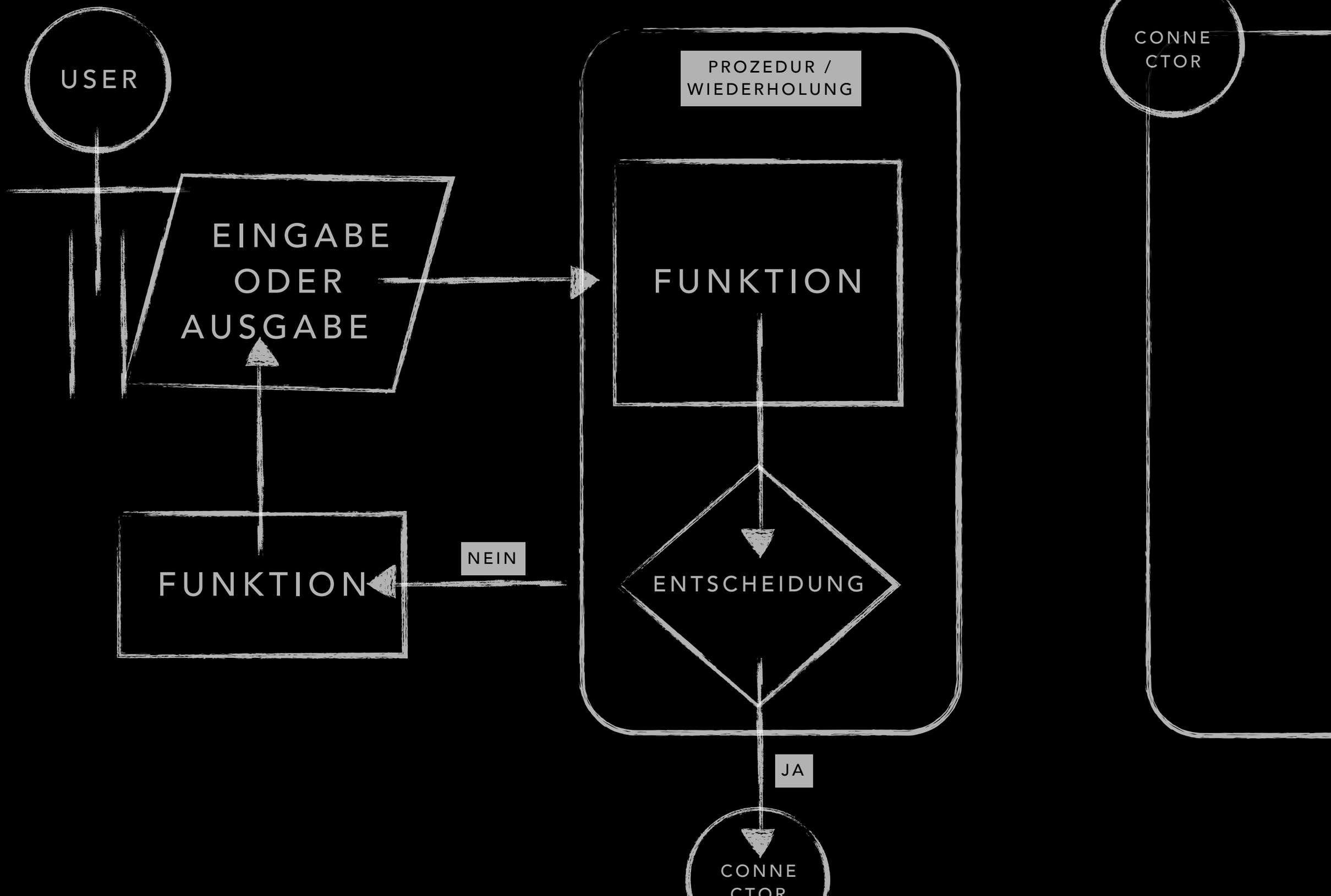
Hinweis:

Das Query zum sichern von Daten heisst
"UPDATE user SET username='Username', email='Emailadresse', password='Passwort' WHERE userID=#id;

Es muss aus den Daten dynamisch erzeugt werden.

Skizzieren Sie sich zunächst den Programmablauf.

NOTATION FÜR EINEN PROGRAMMABLAUF



DEPRECATED:
MYSQLI PROZEDURAL CHEAT SHEET

```
$host     = 'localhost'; $user='root'; $password='';
$database ='application';
$sql      = "SELECT * FROM user WHERE userId=1;";

$mysqli   = mysqli_connect($host, $user, $password, $database);

$result   = mysqli_query($mysqli,$query);
$row      = mysqli_fetch_assoc($result);

echo mysqli_num_rows($result);
mysqli_insert_id($mysqli);
```

WICHTIGE MYSQLI MEMBER

```
$mysqli->host_info    -> Host für den DB Zugriff

$mysqli->errno        -> Fehlernummer bei DB Zugriff
$mysqli->error        -> Fehlertext
$mysqli->use_result() -> Zugriff auf das Ergebnis
                         des Datenbankzugriffes


$result->fetch_assoc()-> Methode, um auf den Inhalt
                         von Result zuzugreifen.

$result->data_seek(0);-> Setzt den Zeiger in $result auf
                         bestimmten Datensatz

Viele weitere Methoden finden Sie in der Doukumentation.
```

DATEN IN BUFFER SCHREIBEN UND AUSGEBEN

```
$result = $mysqli->query($query);

echo "Reverse order...\n";
for ($row_no = $result->num_rows - 1; $row_no >= 0; $row_no--)
{
    $result->data_seek($row_no);
    $row = $result->fetch_assoc();
    echo " id = " . $row['id'] . "\n";
}

echo "Result set order...\n";
$result->data_seek(0);
while ($row = $result->fetch_assoc()) {
    echo " id = " . $row['id'] . "\n";
}
```

DATEN DIREKT AUSGEBEN (OHNE BUFFER)

```
$mysqli->real_query("$query);

$result = $mysqli->use_result();

echo "Result set order…\n";

while ($row = $result->fetch_assoc()) {
    echo " id = " . $row['id'] . "\n";
}
```

MIT EXCEPTIONS ARBEITEN

THROW -> TRY - CATCH

THROW() - FEHLER ABFANGEN UND "WERFEN"

Bauen Sie mit throw() in Ihre Funktionen Fehlerausgaben ein, damit Sie Fehlerverhalten steuern können.

```
function inverse ( $x = 0 ) {
    if ( !$x ) {
        $message = 'Division by Zero!!' . '<br>';
        throw new Exception($message);
    } else {
        return 1/$x;
    }
}
```

TRY {} CATCH (EXCEPTION $E) {}

Den Programmablauf können Sie dann mit try{} catch() {} ausführen lassen. Ein Fehler in einem try-Block wird im catch-Block aufgefangen.

```
try {
   echo inverse(5) . '<br>';
   echo inverse(0) . '<br>;
} catch ( Exception $e ) {
   echo 'Exception catched: ' . $e->getMessage() . BR;
}

echo 'continuing ...';
```

DAS FEHLEROBJEKT EXCEPTION $E

- Die Fehlermeldung selbst erhalten Sie mit der Methode $e->getMessage().
- Dateinamen und Zeilennummer zur Fehlermeldung gibt es mit $e->getLine() und $e->getFile().

FEHLER WERFEN UND FEHLER FANGEN

```php
// Eine Funktion wirft einen Fehler
    function inverse ( $x = 0 ) {
        if ( !$x ) {
            throw new Exception('Division by Zero!!' . BR);
        } else {
            return 1/$x;
        }
    }
// Der Programmablauf fängt den Fehler auf und gibt ihn aus
    try {
        echo inverse(5) . BR;
        echo inverse(0) . BR;
    } catch ( Exception $e ) {
        echo 'Exception catched: ' . $e->getMessage() . BR;
    }
    echo 'continuing ...';
```

DEN USER IDENTIFIZIEREN

MIT SESSIONS ARBEITEN

SESSION IDS VERWENDEN

- Sessions identifizieren die Verbindung von Browser und Server.
- Sie erzeugen eine Session-Id, die konstant zwischen Browser und Server transportiert wird.
- Session-Ids werden häufig mit User-Ids verküpft, so kann ein Server einen User wiedererkennen.
- Die Verknüpfung wird in der Datenbank gesichert.

EINE SESSION STARTEN

```
@session_start();

$sessionId = session_id();
$time = date('Y-m-d h:i:s');
```

Eine Session wird initiiert. Dabei wird eine neue Session-Id erzeugt oder eine vorhandene verwendet.
Zu jeder Sessioninitialisierung sollte die Zeit gespeichert werden. So kann die Session-Id später als nicht mehr gültig angesehen werden.

SESSION INFORMATIONEN WERDEN DER DATENBANK GEHALTEN.

```
if ( !hasSessionId($sessionId) ) {
    $query = "
        INSERT INTO
            sessions
            (session_id, start_time, user_id)
        VALUES
            ('" . $sessionId() . "','" . $time() . "'," .
$userId .");";
} else {
    $query = "
        UPDATE
            sessions
        SET
            time='" . $time() . "'
        WHERE
            session_id='" . $this->getSessionId() . "';";
}

$mysqli->$query($query);
```

DIESE METHODE PRÜFT, OB DIE SESSION BEREITS EXISTIERT.

```
function hasSessionId ()
{
    $query = "
        SELECT
            count(sessionId)
        AS
            count
        FROM
            sessions
        WHERE
            sessionId='" . $sessionId . "';";

    $mysqli->query($mysqli, $query);

    if ($mysqli->result) {

        $row = $$mysqli>result->fetchRow();

        if ($row['count'] == 1)
            return TRUE;
        else
            return FALSE;
    } else {
        return FALSE;
    }
}
```

DIESE METHODE ERMITTELT DIE USER-ID ZU EINER SESSION

```
$query = "
    SELECT
        user_id
    FROM
        sessions
    WHERE
        session_id='" . $this->getSessionId() . "'
    ;";

$mysqli->query($mysqli, $query);
$row = $mysqli->result->fetchRow();

$sessionId = $row['user_id'];
```

SESSION TABELLE ANLEGEN

Schreiben Sie ein Include inc.session.php, das bei jedem Seitenaufruf eine Session aufruft oder verwendet und den User ermittelt.

Legen Sie in der Datenbank eine Tabelle 'sessions' an:

```
sessionId    int   primary
userId       int   foreign key
sessionTime  datetime
```

GENERISCHES HTML

MIT TEMPLATES ARBEITEN

HTML TEMPLATES - GANZ EINFACH

- In PHP 5 wurde ein Outputbuffer eingeführt. Dieser erleichtert das Handhaben von separaten HTML Templates.

- Mit include/require eingebundene Dateien werden so nicht direkt in den Programmablauf eingebunden, sondern in eine Puffervariable.

- Dabei verhält sich die eingebundene Datei so, wie wenn Sie im Programmablauf eingebunden wäre; sie kann daher aktuelle Variablen verwenden.

TEMPLATES IN DEN BUFFER SCHREIBEN

```php
$template = ROOT . 'app/templates/userlist.html';

// Einen Ausgabepuffer anlegen
ob_start();

// Das Template aufrufen und dessen Inhalte
// in eine Variable übergeben
require $template;
$output = ob_get_contents();

// Den Ausgabepuffer wieder löschen
ob_end_clean();

// Jetzt kann der Inhalt des Templates ausgegeben werden
echo $output;
```

EIN TEMPLATE ANLEGEN

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Userlist</title>
</head>
<body>

    <h1>Userliste</h1>
    <p><?php echo $content; ?></p>

</body>
</html>
```

ALTERNATIVE VORGEHENSWEISE

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Userlist</title>
</head>
<body>

    <h1>Userliste</h1>
    <ul>
    <?php
        while ($row = $result->fetch_assoc()){
            $content =   $content
                            . '<li><a href="user.php?id='
                            . $row['userId']
                            . '">'
                            . $row['username']
                            . '</li>';
        }
    ?>
    </ul>

</body>
</html>
```

TEMPLATES IN DEN BUFFER SCHREIBEN

Schreiben Sie eine Funktion, die Templates einliest und als Ausgabevariable $content ausgibt. Verwenden Sie diese Funktion überall dort, wo HTML Ausgaben generiert werden.

Schreiben Sie alle HTML-Ausgaben als externe HTML-Templates um.

Legen Sie dazu folgende Ordnerstruktur an:

```
PHP
| …
| app
|   | templates/userlist.html
|   | templates/user.html
|   | templates/dashboard.html
|   | …
| …
```

SAUBERE CODES MIT PHP

CLEAN CODING

INDENTING AND LINE LENGTH

- Use an indent of 4 spaces, with no tabs. This helps to avoid problems with diffs, patches, SVN history and annotations.

- Vim rules:

- set expandtab set shiftwidth=4
  set softtabstop=4 set tabstop=4

- It is recommended to keep lines at approximately 75-85 characters long for better code readability. Paul M. Jones has some thoughts about that limit.

CONTROL STRUCTURES

- These include if, for, while, switch, etc. Here is an example if statement, since it is the most complicated of them:

- Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls.

- You are strongly encouraged to always use curly braces even in situations where they are technically optional. Having them increases readability and decreases the likelihood of logic errors being introduced when new lines are added.

CONTROL STRUCTURES

```
<?php
if ((condition1) || (condition2)) {
    action1;
} elseif ((condition3) && (condition4)) {
    action2;
} else {
    defaultaction;
}
?>
```

SWITCH CONDITIONS

```
switch (condition) {
case 1:
    action1;
    break;

case 2:
    action2;
    break;

default:
    defaultaction;
    break;
}
```

SPLIT LONG IF STATEMENTS ONTO SEVERAL LINES

- Long if statements may be split onto several lines when the character/line limit would be exceeded. The conditions have to be positioned onto the following line, and indented 4 characters. The logical operators (&&, ||, etc.) should be at the beginning of the line to make it easier to comment (and exclude) the condition. The closing parenthesis and opening brace get their own line at the end of the conditions.

SPLIT LONG IF STATEMENTS ONTO SEVERAL LINES

```
if (
        $condition1
    || $condition2
    || $condition3
) {
    //code here
}
```

TERNARY OPERATORS

```
$a = $condition1 && $condition2
    ? $foo : $bar;

$b = $condition3 && $condition4
    ? $foo_man_this_is_too_long_what_should_i_do
    : $bar;
```

FUNCTION CALLS

- Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon.

FUNCTION CALLS

```
$var = foo($bar, $baz, $quux);

$short         = foo($bar);
$long_variable = foo($baz);

$this->callSomeFunction('param1',     'second',         true);
$this->callSomeFunction('parameter2', 'third',          false);
$this->callSomeFunction('3',          'verrrrrrylong', true);
```

SPLIT FUNCTION CALL ON SEVERAL LINES

- The CS require lines to have a maximum length of 80 chars. Calling functions or methods with many parameters while adhering to CS is impossible in that cases. It is allowed to split parameters in function calls onto several lines.

SPLIT FUNCTION CALL ON SEVERAL LINES

```
$this->someObject->subObject->callThisFunctionWithALongName(
    $parameterOne, $parameterTwo,
    $aVeryLongParameterThree
);
```

SPLIT FUNCTION CALL ON SEVERAL LINES

```php
$this->someObject->subObject->callThisFunctionWithALongName(
    $this->someOtherFunc(
        $this->someEvenOtherFunc(
            'Help me!',
            array(
                'foo'  => 'bar',
                'spam' => 'eggs',
            ),
            23
        ),
        $this->someEvenOtherFunc()
    ),
    $this->wowowowowow(12)
);
?>
```

ALIGNMENT OF ASSIGNMENTS

```
$short  = foo($bar);
$longer = foo($baz);

$short = foo($bar);
$thisVariableNameIsVeeeeeeeeeeeryLong = foo($baz);

$GLOBALS['TSFE']->additionalHeaderData[$this-
>strApplicationName]
=$this->xajax-
>getJavascript(t3lib_extMgm::siteRelPath('nr_xajax'));
```

CLASS DEFINITIONS

```
class Foo_Bar
{

    //... code goes here

}
```

FUNCTION DEFINITIONS

```
function fooFunction($arg1, $arg2 = '')
{
    if (condition) {
        statement;
    }
    return $val;
}

Function declarations follow the "K&R style"
```

FUNCTION DEFINITIONS

- Arguments with default values go at the end of the argument list. Always attempt to return a meaningful value from a function if one is appropriate. Here is a slightly longer example:

FUNCTION DEFINITIONS

```
function connect(&$dsn, $persistent = false)
{
    if (is_array($dsn)) {
        $dsninfo = &$dsn;
    } else {
        $dsninfo = DB::parseDSN($dsn);
    }

    if (!$dsninfo || !$dsninfo['phptype']) {
        return $this->raiseError();
    }

    return true;
}
```

SPLIT FUNCTION DEFINITIONS ONTO SEVERAL LINES

- Functions with many parameters may need to be split onto several lines to keep the 80 characters/line limit. The first parameters may be put onto the same line as the function name if there is enough space. Subsequent parameters on following lines are to be indented 4 spaces. The closing parenthesis and the opening brace are to be put onto the next line, on the same indentation level as the "function" keyword.

SPLIT FUNCTION DEFINITIONS ONTO SEVERAL LINES

```
function someFunctionWithAVeryLongName(
    $firstParameter  = 'something',
    $secondParameter = 'booooo',
    $third           = null,
    $fourthParameter = false,
    $fifthParameter  = 123.12,
    $sixthParam      = true)
{
    //....
```

ARRAYS

```
$some_array = array(
    'foo'  => 'bar',
    'spam' => 'ham',
);
```

Assignments in arrays may be aligned. When splitting array definitions onto several lines, the last value may also have a trailing comma. This is valid PHP syntax and helps to keep code diffs minimal:

INCLUDING CODE

- Anywhere you are unconditionally including a class file, use require_once. Anywhere you are conditionally including a class file (for example, factory methods), use include_once. Either of these will ensure that class files are included only once. They share the same file list, so you don't need to worry about mixing them - a file included with require_once will not be included again by include_once.

PHP CODE TAGS

- Always use <?php ?> to delimit PHP code, not the <? ?> shorthand. This is required for PEAR compliance and is also the most portable way to include PHP code on differing operating systems and setups.

NAMING CONVENTIONS

GLOBAL VARIABLES AND FUNCTIONS

- If your package needs to define global variables, their names should start with a single underscore followed by the package name and another underscore. For example, the PEAR package uses a global variable called $_PEAR_destructor_object_list.

- Global functions should be named using the "studly caps" style (also referred to as "bumpy case" or "camel caps"). In addition, they should have the package name as a prefix, to avoid name collisions between packages. The initial letter of the name (after the prefix) is lowercase, and each letter that starts a new "word" is capitalized. An example:

- XML_RPC_serializeData()

CLASSES

- Classes should be given descriptive names. Avoid using abbreviations where possible. Class names should always begin with an uppercase letter. The PEAR class hierarchy is also reflected in the class name, each level of the hierarchy separated with a single underscore. Examples of good class names are:

- Log
  Net_Finger
  HTML_Upload_Error

CLASS VARIABLES AND METHODS

- Class variables (a.k.a properties) and methods should be named using the "studly caps" style (also referred to as "bumpy case" or "camel caps"). Some examples (these would be "public" members):

- $counter connect() getData() buildSomeWidget()

- Private class members are preceded by a single underscore. For example:

- $_status _sort() _initTree()

- Protected class members are not preceded by a single underscore. For example:

- protected $somevar protected function initTree()

CONSTANTS

- Constants should always be all-uppercase, with underscores to separate words. Prefix constant names with the uppercased name of the class/package they are used in. Some examples:

- DB_DATASOURCENAME
- SERVICES_AMAZON_S3_LICENSEKEY

FILE FORMATS

FILE FORMATS

- All scripts contributed to PEAR must:
- Be stored as ASCII text
- Use ISO-8859-1 or UTF-8 character encoding. The encoding may be declared using declare(encoding = 'utf-8'); at the top of the file.
- Be Unix formatted

FILE FORMATS

- "Unix formatted" means two things:

- 1) Lines must end only with a line feed (LF). Line feeds are represented as ordinal 10, octal 012 and hex 0A. Do not use carriage returns (CR) like Macintosh computers do or the carriage return/line feed combination (CRLF) like Windows computers do.

- 2) There should be one line feed after the closing PHP tag (?>). This means that when the cursor is at the very end of the file, it should be one line below the closing PHP tag.

E_STRICT-COMPATIBLE CODE

- Starting on 01 January 2007, all new code that is suggested for inclusion into PEAR must be E_STRICT-compatible. This means that it must not produce any warnings or errors when PHP's error reporting level is set to E_ALL | E_STRICT.

- The development of existing packages that are not E_STRICT-compatible can continue as usual. If however a new major version of the package is released, this major version must then be E_STRICT-compatible.

HEADER COMMENT

BLOCKS

SPACES RULES, THE PAGE LEVEL DOCBLOCK

```
/* vim: set expandtab tabstop=4 shiftwidth=4 softtabstop=4: */
/**
 * Short description for file
 * Long description for file (if any)...
 * PHP version 5
 * LICENSE: This source file is subject to version 3.01
 * of the PHP license
 * that is available through the world-wide-
 * web at the following URI:
 * http://www.php.net/license/
3_01.txt.  If you did not receive a copy of
 * the PHP License and are unable to obtain it through the web,
 please
 * send a note to license@php.net so we can mail you a copy imm
ediately.
 ...
```

THE PAGE LEVEL DOCBLOCK

```
...
*
* @category   CategoryName
* @package    PackageName
* @author     Original Author <author@example.com>
 * @author     Another Author <another@example.com>
 * @copyright  1997-2005 The PHP Group
 * @license    http://www.php.net/license/
3_01.txt  PHP License 3.01
 * @version    SVN: $Id$
 * @link       http://pear.php.net/package/PackageName
 * @see        NetOther, Net_Sample::Net_Sample()
 * @since      File available since Release 1.2.0
 * @deprecated File deprecated in Release 2.0.0
 */
```

THE PAGE LEVEL DOCBLOCK

```
/*
 * Place includes, constant defines and $_GLOBAL settings
 * here. Make sure they have appropriate docblocks to avoid
 * phpDocumentor construing they are documented by the
 * page-level docblock.
 */

Code
```

THE CLASS LEVEL DOCBLOCK

```
/**
 * Short description for class
 *
 * Long description for class (if any)...
 *
 * @category   CategoryName
 * @package    PackageName
 * @author     Original Author <author@example.com>
 * @author     Another Author <another@example.com>
 * @copyright  1997-2005 The PHP Group
 * @license    http://www.php.net/license/
 *             3_01.txt  PHP License 3.01
 * @version    Release: @package_version@
 * @link       http://pear.php.net/package/PackageName
 * @see        NetOther, Net_Sample::Net_Sample()
 * @since      Class available since Release 1.2.0
 * @deprecated Class deprecated in Release 2.0.0
 */
```