
# PHP Basics

SAE Wien
Alexander Hofbauer
hofbauer.alexander@gmail.com

# Geschichte

PHP steht rekursiv für “**PHP: Hypertext Preprocessor**”

**1995**: PHP/FI (“Personal Home Page Tools / Form Interpreter”)

**1997**: PHP/FI 2

**1997**: Neuentwicklung PHP 3

**1999**: Neuentwicklung des Kerns PHP 4

**2004**: PHP 5

**2016**: PHP 7.1

**2020**: **PHP 7.4** (v7.4.3 - Feb. 2020)

# http://php.net (!!)

+ Dokumentation
+ Diskussion, Meinungen, Tipps
+ Release Informationen

# Eigenschaften von PHP

+ Interpretersprache
+ Anleihen aus C/C++, Perl, Unix Shell, Java
+ **serverseitig**
+ Vielzahl eingebauter Funktionen
+ Modular erweiterbar

# Grundlagen

+ Dateiendung: `.php` (z.B. `index.php`)
+ PHP und HTML können gemischt werden → PHP-Tags

```php
<?php

// PHP Code

?>
```

# Kommentare

```
+  // einzlg. Kommentar
+  # einzlg. Kommentar
+  /* mehrzlg.
Kommentar */
```

**Nach Möglichkeit einheitlich kommentieren!**

Best Practice:

```
/**
 * Kommentar
 */
```

# GANZ GANZ WICHTIG!

...ganz wichtig!

In PHP wird nach jeder Anweisung genau ein **Semikolon** gesetzt;

Wird es vergessen → Syntaxfehler, das Skript wird nicht ausgeführt!

```
echo "hallo welt!";
```

!!

# Variablen und Datentypen

# Variablen

+ Beginnen mit einem Dollar-Zeichen: $
+ Gefolgt von einem Buchstaben oder Unterstrich (_)
+ Erlaubt sind Buchstaben und Ziffern (inkl. Umlaute ← nicht zu empfehlen)
+ Case-Sensitive! (Groß-/Kleinschreibung wird unterschieden)
+ Alphanumerisch & Unterstrich

# Variablen

```php
<?php

$var = "Wir";
$Var = "lernen";
$vaR = "PHP";

echo "$var $Var $vaR!"; // Wir lernen PHP!

// Variablen Namen

$2birnen = "2 Birnen"; // Ungültig!
$_2birnen = "2 andere Birnen"; // GÜLTIG :D

?>
```

# Datentypen

**PHP wählt den Typ automatisch!** ← Fluch & Segen (Verknüpfungen & Operationen)

**Im Zweifelsfall immer manuell umwandeln.**

```php
// Skalare Datentypen
$wahrheitswert = true; // boolean
$ganzzahl = 42;        // integer
$kommazahl = 3.1415    // float (= double)
$zeichenkette = "foo"; // string

// Zusammengesetzte Datentypen
$feld = array('foo', 'bar'); // array
$objekt = new stdClass();    // object
$funktion = function () {    // callable
    echo "hello world";
};

// Spezielle Datentypen
$resource = fopen("foo", "w"); // resource
$nichts = NULL;                // NULL
```

# Implizite Typumwandlung

z.B.: `$zahl = (int)$irgendeinWert;`

Dabei wird der Inhalt der Variable umgewandelt.

**Typecasting passiert in PHP bei Bedarf automatisch:**

```php
<?php

$foo = "0";             // $foo <-- string (ASCII 48)
$foo += 2;                // $foo <-- integer (2)
$foo += 1.3;            // $foo <-- float (3.3)
$foo = 5 + "10 Birnen"; // $foo <-- integer (15)

?>
```

# Strings (Zeichenketten)

- aufeinanderfolgende Zeichen
- theoretisch beliebig lang
- Anführungszeichen
- **Einfache Anführungszeichen**: String wird **nicht** vom Compiler **interpretiert**
- **Doppelte Anführungszeichen**: String wird vom Compiler **interpretiert**

```php
$i = "Das ist ein String";
$j = 'Das hier auch :D';
```

```php
<?php

$i = ‘ein String‘;
$k = ‘Das ist $i‘;
echo $k; // Das ist $i

?>
```

```php
<?php

$i = ‘ein String‘;
$k = “Das ist $i ”;
echo $k; // Das ist ein String

?>
```

# Strings escapen

sollen Anführungszeichen in Strings verwendet werden, müssen diese “escaped” werden

```php
<?php

$i = ‘ein String‘;
$k = “Das ist \”$i\””;
echo $k; // Das ist “ein String”

?>
```

werden die Anführungszeichen gemischt verwendet, müssen sie nicht escped werden.

```php
<?php

$i = ‘ein String‘;
$k = “Das ist ‘$i’”;
echo $k; // Das ist ‘ein String’

?>
```

# Strings escapen - Spezielle Zeichen

Escapen funktioniert nur in Strings mit doppelten Anführungszeichen!

```php
<?php

$zeilenumbruch  = "\n"; // "Line Feed"
$carriageReturn = "\r"; // Windows Zeilenumbruch: \r\n
$tabulator      = "\t";
$backslash      = "\\";
$singleQuote    = "\'";
$doubleQuote    = "\"";

?>
```

# Zahlen

Integers

```php
<?php

$zahl = 42;         // decimal number
$zahl = -42;        // negative number
$zahl = 052;        // octal number
$zahl = 0x2A;       // hexadecimal number
$zahl = 0b101010;// binary number

?>
```

Floats/Doubles

```php
<?php

$zahl = 3.1415;
$zahl = -2.7182;

?>
```

# Boolesche Werte

+ logischer Wahrheitswert
+ genau 2 mögliche Werte: **true** oder **false**

```php
<?php

$true = true;
$false = false;

?>
```

# Arrays/Felder

+ dienen zum Abbilden von Reihen/Listen
+ Indizes sind mehr oder weniger beliebig

```php
<?php

// Array zuweisen
$farben[0] = 'rot';
$farben[1] = 'blau';
$farben[2] = 'gelb';

// Oder
$farben = array('rot', 'blau', 'gelb');
echo $farben[1]; // blau

?>
```

# Mehrdimensionale Arrays

```php
<?php

$plants = [
    ['oak', 'hickory', 'baobab'],
    ['tulip', 'rose', 'daffodile'],
    ['small shrubbery', 'burning shrubbery with spikes']
];

echo $plants[1][0]; // tulip

?>
```

# Assoziative Arrays

```php
<?php

$plants = [
    'trees' => ['oak', 'hickory', 'baobab'],
    'flowers' => ['tulip', 'rose', 'daffodile'],
    'shrubberies' => ['small shrubbery', 'burning shrubbery with spikes']
];

echo $plants["flowers"][1]; // rose

?>
```

# Datentypen prüfen

```
is_array();
is_bool();
is_double();
is_float();
is_integer();
is_numeric();
is_string();
is_null();
```

# Konstanten

+ einmal definiert, können Konstanten nicht mehr verändert werden
+ gängige Konvention ist für Konstantennamen nur Großbuchstaben zu verwenden

```php
<?php

define('FOO', 'bar');
define('ANSWER_TO_THE_QUESTION_OF_LIFE_THE_UNIVERSE_AND_EVERYTHING', 42);

?>
```

# Operatoren

# Zuweisungsoperator

```php
// Einfache Zuweisung: =
$a = 37;

// Kombinierte Operatorn: +=, -=, *=, .=, /=, %=
$a += 5; // 42

$b = "Hallo";
$b .= ' i bims!';
echo $b; // Hallo i bims!
```

# Vergleichsoperator

```
$a == $b  // true, wenn beide Werte gleich sind
$a != $b  // true, wenn beide werte NICHT gleich sind

$a === $b // true, wenn beide Werte und Datentypen gleich sind
$a !== $b // true, wenn beide werte und Datentypen NICHT gleich sind

$a > $b   // true, wenn linker Wert größer als rechter
$a < $b   // true, wenn linker Wert kleiner als rechter
$a >= $b  // true, wenn linker Wert größer als oder gleich dem rechten
$a <= $b  // true, wenn linker Wert kleiner als oder gleich dem rechten
```

# Vergleichsoperator - Beispiele

```
42 == "42"        // true
"18" != 18.0      // false

42 === "42"       // false
"18" !== 18.0     // true

"42" > 41         // true
42 < "foobar"     // false
18 >= 18.0        // true
0b101010 <= "42" // true
```

# Arithmetische Operatoren

```php
$a = 37;
$b = 5;

$c = $a + $b; // Addition --> 42
$c = $a - $b; // Subtraktion --> 32
$c = $a * $b; // Multiplikation --> 185
$c = $a / $b; // Division --> 7.4
$c = $a % $b; // Modulo --> 2
```

# Inkrement/Dekrement Operatoren

```php
$a = 42;

++$a; // Prä-Inkrement; erhöht $a um 1 und gibt es zurück
$a++; // Post-Inkrement; gibt $a zurück und erhöht es dann um 1

--$a; // Prä-Dekrement
$a--; // Post-Dekrement
```

# String Operatoren

```php
// Verketten: .
$a = "Hallo";
$b = "Welt";
$c = $a . ' ' . $b; // Hallo Welt

// Verketten und Zuweisen: .=
$a .= $b // HalloWelt

// Zeichenindizierung: []
echo $c[1]; // a
```

# Logische Operatoren

```
$a && $b  // true, wenn beide Werte true sind
$a AND $b // true, wenn beide Werte true sind

$a || $b  // true, wenn mind. ein Wert true ist
$a OR $b  // true, wenn mind. ein Wert true ist

$a XOR $b // true, wenn beide unterschiedliche sind

!$a       // true, wenn der Wert false ist (Negation)
```

# Referenzen

+ Referenzen verweisen auf denselben Inhalt im RAM

```php
<?php

$a = 1; // a: 1
$b = 2; // b: 2

$a = &$b; // a: 2

$a = 3; // a: 3, b: 3

?>
```

# Ausgaben und Debugging

# Ausgaben - echo

```php
<?php

echo "Hallo Welt";
echo ("Hallo”, “Welt");

$a = "Hallo Welt"; echo $a; // Hallo Welt

$b = "Welt"; echo "Hallo " . $b; // Hallo Welt

?>
```

# Ausgaben - print

```php
<?php

print "Hallo Welt";
print ("Hallo Welt");

$a = "Hallo Welt"; print $a; // Hallo Welt

$b = "Welt"; print "Hallo " . $b; // Hallo Welt

?>
```

+ echo ist etwas schneller
+ nur ein Parameter möglich
+ print hat einen Rückgabewert 1

# Ausgaben - printf

```php
<?php

$everything = "Allem";
$answer = 42;

$text = "Die Antwort auf die Frage nach dem Leben, dem Universum und %s lautet %d.";

printf($text, $everything, $answer);
// Die Antwort auf die Frage nach dem Leben, dem Universum und Allem lautet 42.

// %d = positive Ganzzahl
// %s = String

?>
```

# Ausgaben - debugging: print_r

```php
<?php

$werte = ["foo", "bar"];

// rekursive Ausgabe
print_r($werte);



// Array
// (
//   [0] => foo
//   [1] => bar
// )

?>
```

# Ausgaben - debugging: var_dump

```php
<?php

$werte = ["foo", "bar"];

// rekursive Ausgabe inkl. Länge & Typ
var_dump($werte);

// array(2) {
//   [0]=>
//   string(3) "foo"
//   [1]=>
//   string(3) "bar"
// }

?>
```

# Kontrollstrukturen und Schleifen

# Schleifen - for

```php
<?php

for (Initialisierung; Bedingung; Aenderungssequenz) {
    // Code
}


for ($i = 0; $i <= 100; $i++) {
    echo $i;
}

?>
```

+ Initialisierung
+ Bedingung
+ Änderung

# Schleifen - while

```php
<?php

while (Bedingung) {
    // Code
}

while ($row = mysqli_fetch_assoc($res)) {
    echo $row['title'];
}

?>
```

+ Bedingung am Anfang
+ ist die Bedingung nicht erfüllt, wird die Schleife nicht durchlaufen

# Schleifen - do/while

```php
<?php

do {
    // Code
} while (Bedingung)


$i = 0;
do {
    echo $i;
} while ($i > 0);

?>
```

+ Bedingung am Ende
+ wird mind. 1x durchlaufen

# Schleifen - foreach

```php
<?php

foreach (Array as Wert) {
    // Anweisungen
}


foreach (Array as Key => Wert) {
    // Anweisungen
}

?>
```

+ Anweisungen können auf alle Elemente eines Arrays angewendet werden
+ kann **NUR** auf **Arrays** angewendet werden

# Schleifen - foreach

### Wert

```php
<?php

foreach ($array as $wert) {
    echo "Das Feld hat den Wert: $wert";
}

?>
```

### Schlüssel & Wert

+ häufig für assoziative Arrays verwendet

```php
<?php

foreach ($array as $key => $wert) {
    echo "Index: $key, Wert: $wert";
}

?>
```

# Bedingung - if/else

```php
<?php

if ($anzahl >= 15) {
    // something
} elseif ($anzahl == 10) {
    // something else
} else {
    // well, no condition was met -.-
}

?>
```

# Funktionen und Parameter

# Funktionen

```php
<?php

// Funktion definieren
function funktionsname (Parameter, Parameter, ...) {
    // Code
}

// Funktion aufrufen
funktionsname(Parameter);

?>
```

# Funktionen

```php
<?php

$a = 2;
$b = 3;

// “int” ist hier jeweils ein optionaler Typehint für Parameter und Rückgabewert
function calc(int $zahl1, int $zahl2): int {
    echo $zahl1 + $zahl2;
}

calc($a, $b);

?>
```

# Funktionen - Parameter

+ Vorgabewerte für Parameter

```php
<?php

function func($text = "Whoop", $zahl = 3) {
    echo "$text $zahl";
}

func("other text", 42); // other text 42
func();                 // Whoop 3

?>
```

# Funktionen - Parameter

```php
<?php

function appendString(&$string) {
    $string .= ' und ein angefuegter String';
}

$str = 'ein string';
appendString($str);
echo $str; // ein string und ein angefuegter String

?>
```

+ Verweise als Parameter
+ normalerweise werden Änderungen an Variablen nur im aktuellen Gültigkeitsbereich (Scope) durchgeführt
  ← Verweise brechen diesen Scope auf

# Funktionsumfang von PHP 7.4

# Größe von Arrays

- count()
- Anzahl der Elemente im Array
- 0 für leeres Array oder NULL
- 1 wenn das Objekt nicht gezählt werden kann

```php
<?php

$food = [
    'Obst' => ['Orange', 'Banane', 'Apfel'],
    'Gemuese' => ['Karotte', 'Kohl', 'Erbse']
];

echo count($food);          // 2
echo count($food['Obst']); // 3

echo count($food, COUNT_RECURSIVE); // 8

?>
```

# Länge von Strings

- strlen()
- Anzahl der Zeichen im String
- 0 für leeren String oder NULL

```php
<?php

$someString = "some cool string";

echo strlen($someString); // 16

?>
```

# Teilen von Strings

- explode()
- Anhand eines Parameters wird ein String in ein Array zerteilt

```php
<?php

$string = "rot blau gelb gruen";
$farben = explode(' ', $string);

$string2 = "BMW::Opel::Audi::VW";
$autos = exlode('::', $string2);

echo $autos[2]; // Audi

?>
```

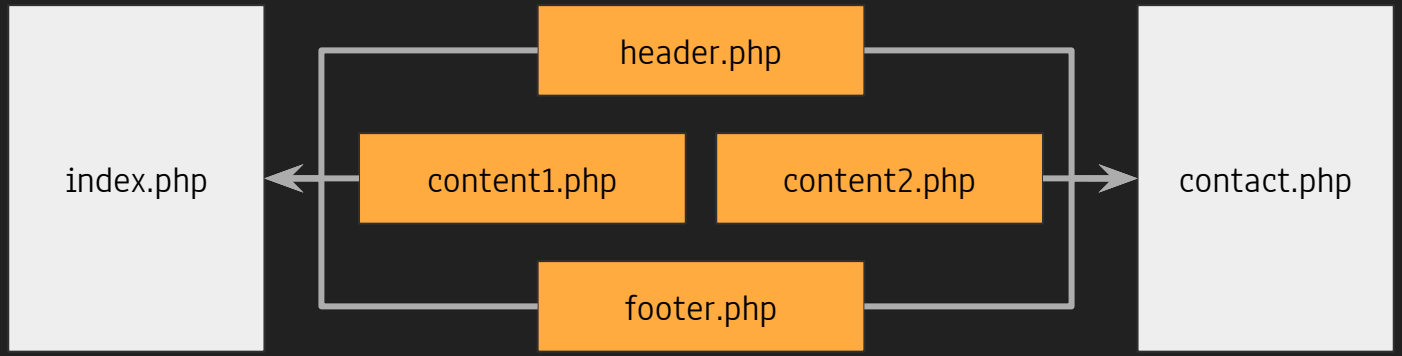
# Zusammenführen von Arrays

- implode()
- Array Elemente werden mittels eines $glue-Strings zu einem String zusammengefügt

```php
<?php

$string = "rot blau gelb gruen";
$farben = explode(' ', $string);

$farbenString = implode(', ', $farben);
echo $farbenString; // rot, blau, geld, gruen

?>
```

# Einbinden und Auslagern

+ PHP kann Dateien einbinden ⟶ dynamisches und strukturiertes Arbeiten
+ Gleiche Bereiche einer Website können ausgelagert werden ⟶ Bspw. Header und Footer



# Einbinden und Auslagern

- include() liefert **Warning**, wenn Datei nicht existiert
- require() liefert **Fatal Error**, wenn datei nicht existiert ⟶ Abbruch
- *_once() bindet nur ein, wenn die Datei nicht bereits eingebunden wurde

```php
<?php

include("datei.php");
include_once($file);

require "datei.html";
require_once($folder . 'header.php');

?>
```

# Einbinden und Auslagern

- HTML-Blöcke & Skripte auslagern
  - Übersichtlichkeit
  - Verwandten Code gruppieren
  - Wiederverwendbarkeit
- Eingebundener Code wird ausgeführt

**NIEMALS CODE AUS DEM NETZ EINFACH SO EINBINDEN! → Sicherheitslücken!!**

# require() Beispiel

```php
<?php

// Laden von Funktionen
require('functions.php');

// Aufruf einer "externen" Funktion
$filename = get_filename($bild);
echo "Das Bild heißt $filename";

?>
```

```php
<?php // functions.php

function get_filename($path) {
    // some code
}

?>
```

# include() Beispiel

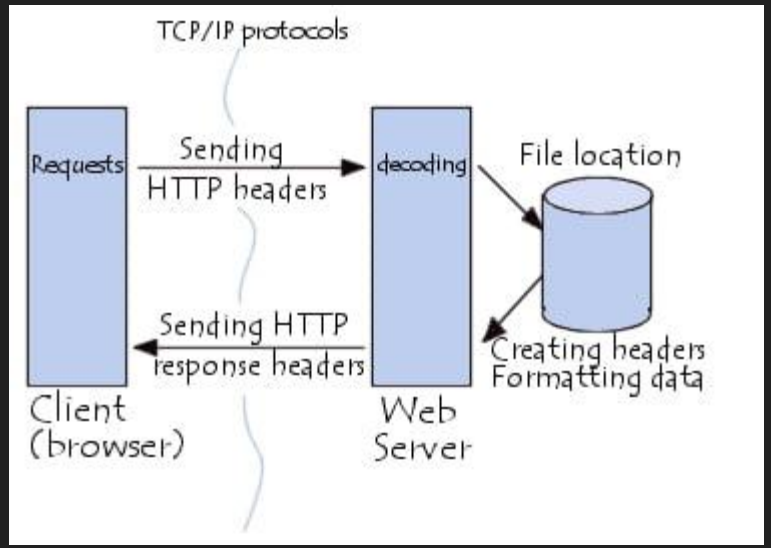
```php
<?php

// Auslagern von Skriptteilen
if ($style === 1) {
    include("style_1.php");
} elseif ($style === 2) {
    include("style_2.php");
} else {
    include("style_default.php");
}
?>
```

# Superglobals

- Built-In
- IMMER in ALLEN Gültigkeitsbereichen verfügbar

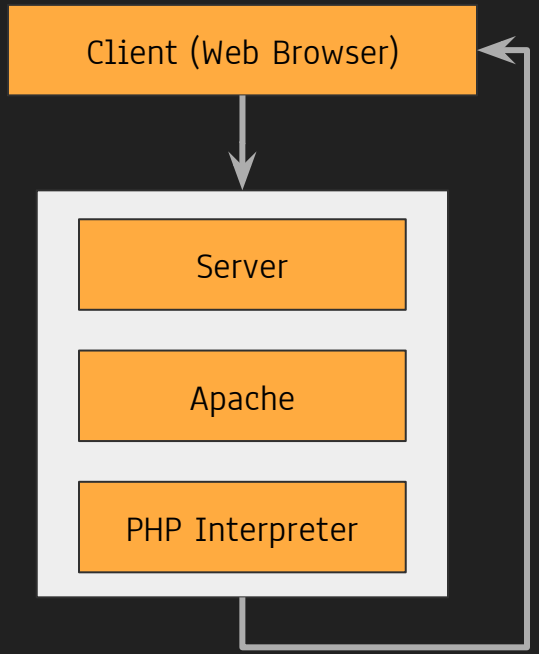
```
$GLOBALS // alle Superglobals

$_GET | $_POST | $_REQUEST // Request-Parameter
$_FILES                    // Datei-Upload
$_COOKIE | $_SESSION
$_SERVER                   // Server Informationen
```

# HTTP Basics




https://de.ccm.net/contents/44-das-http-protokoll

# HTTP Basics




HTTP Header

HTTP Body (HTML Dokument)

# GET-Request

https://example.com/index.php
https://example.com/index.php?pid=1&preview=true

+ ruft Script “index.php” auf
+ übermittelt alles nach dem “?” an Server

GET-Daten werden in Superglobals gespeichert. **$_GET[] wird pro Skriptaufruf neu belegt.**

```php
<?php
echo $_GET['pid'];     // '1'
echo $_GET['preview']; // 'true'
?>
```

# GET-Request

https://example.com/blog.php?article_id=2&number_of_comments=40

```php
<?php

$article_id = $_GET['article_id'];
$article = get_article_from_db($article_id); // custom function

$comment_limit = DEFAULT_COMMENT_LIMIT;
if ($_GET['number_of_comments'] <= DEFAULT_COMMENT_LIMIT) {
    $comment_limit = $_GET['number_of_comments'];
}

?>
```

# Aufgabe 1

Teil 1

**Projekt:** /projektphp/
**Seiten:** Startseite, Kontakt

Lege eine Startseite und eine Kontaktseite an.

Wichtige und wiederverwendbare Teile einer Datei sollen ausgelagert werden (bspw. Header).

# Aufgabe 1

Teil 2

**Projekt:** /projektphp/
**Seiten:** Startseite, Kontakt

Auch die beiden Inhalte sollen ausgelagert werden. (Ordner: content/)

Die Inhalte sollen über den GET-Parameter “page” aufgerufen werden können:

index.php?page=home
index.php?page=contact

# POST-Request

- \+ Sehr oft für **Formulare** verwendet
- \+ Daten werden in Superglobal $_POST gespeichert
- \+ das "name"-Attribute im <input>-Tag definiert den Index
- \+ Daten sind in der URL NICHT sichtbar
- \+ Gut für größere Datenmengen geeignet
- \+ **$_POST wird pro Skriptaufruf neu belegt**

```
<form method="post">
    <input type="text" name="name">
    <input type="text" name="email">
    <input type="submit" name="do-submit">
</form>

// handle.php
<?php

echo $_POST['name'];
echo $_POST['email'];

?>
```

# POST-Request

```php
<?php
/**
 * $_POST ist erst NACH dem Absenden des Formulars
 * befüllt. Der PHP-Code kann also vor dem HTML-Code
 * stehen.
 */
if (isset($_POST['do-submit'])) {
    echo $_POST['email'];
}
?>

<form method="post">
    <input type="text" name="name">
    <input type="text" name="email">
    <input type="submit" name="do-submit">
</form>
```

# String Funktionen

```php
// Laenge eines Strings
strlen($string);

// erste Position eines Substrings; sonst FALSE
strpos($haystack, $needle);

// String zerteilen
explode($delimiter, $string);

// String in Kleinbuchstaben umwandeln
strtolower($string);

// String in Grossbuchstaben umwandeln
strtoupper($string);
```

# Aufgabe 2

Teil 1

**Beim Absenden des Formulars auf der Kontakt-Seite sollen folgende Validierungen durchgeführt werden:**

+ Name mind. 2 Zeichen lang?
+ Email enthält @ und Punkt?
+ Anrede darf nicht der Standardwert sein
+ Geschlecht ist verpflichtende Einfachauswahl
+ Nachricht mind. 10 Zeichen?
+ Newsletter optional, wenn checked, dann Statusmeldung

Bei Validierungsfehlern sollen Meldungen ausgegeben werden.

# Aufgabe 2

Teil 2

Erweitere die Fehlermeldung.

Für jedes Feld soll es eine eigene Fehlermeldung geben (z.B.: ungültige E-Mail).

Schreibe eine Funktion die überprüft, ob ein String nur Zahlen beinhaltet und überprüfe damit das Formularfeld “Telefon”.

# Mailversand

+ mail()
+ Text, HTML, ...

```
// Funktionsaufbau
mail(Empfaenger, Betreff, Nachricht, Header);
---------------
<?php
$message = "Test nachricht";
$header = "From: Alexander <hofbauer.alexander@gmail.com>";

mail("office@sae.edu", "Test Betreff", $message, $header);
?>
```

# Mailversand

Der letzte Parameter kann diverse Header enthalten:

+ **Content-Type** (z.B.: text/html)
+ **Content-Transfer-Encoding**
+ **Reply-To**
+ **CC/BCC**
+ decoded Attachments

Layout von HTML-Emails: **Tabellen und inline-CSS :'(**

# Sessions

+ HTTP-Protokoll ist **stateless** ⟶ Zustand nicht über einen Seitenaufruf hinweg möglich
+ Sessions bieten die Möglichkeit Clients zu identifizieren
+ **Session-ID** wird zwischen Server und Client als **Cookie** hin und her gereicht
+ Server speichert Daten zur Session-ID und kann somit einen Zustand wiederherstellen ⟶ bspw. Login-Status
+ PHP verwaltet Sessions automatisch
+ Sessiondaten am Server, Client-Cookie enthält nur die Session-ID

# Arbeiten mit Sessions

```php
<?php

session_start(); // startet Session
// muss vor allen Ausgaben aufgerufen werden

// Daten in Superglobal $_SESSION[] schreiben
$_SESSION['logged_in'] = true;
$_SESSION['username'] = "arthurdent42";

session_destroy(); // zerstör eine Session
session_regenerate_id(); // erzeugt neue Session-ID

?>
```

# Weiterleitungen

```
<!-- via HTML -->
<meta http-equiv="refresh" content="0; URL=http://sae.edu">

---------------

<?php
// via HTTP Header
header('Location: http://sae.edu/');

// MUSS VOR JEGLICHER AUSGABE AUFGERUFEN WERDEN!!
?>
```

# Aufgabe 3

**Sessions und Header**

Füge mehrere Links zu externen Seiten unter dem Menüpunkt Links ein.

Die Links sollen in neuen Tabs angezeigt werden.

Besuchte Links (in dieser Sessione) sollen in der Liste mit dem text “wurde besucht” markiert werden.

# Zeit

+ Sekunden seit Beginn der UNIX-Epoche (01.01.1970 - 00:00:00)

```php
// Ausgabe von Timestamp
echo time();

// Ausgabe des aktuellen Datums
echo date('d.m.Y');  // DD.MM.YYYY

// Umwandlung Timestamp in Datum
$timestamp = '258244200';
echo date('d-m-Y', $timestamp);
```

# $_SERVER Superglobal

```php
<?php

$_SERVER[‘HTTP_HOST‘];            // Domain
$_SERVER[‘SERVER_ADDR‘];          // IP Adresse
$_SERVER[‘HTTP_ACCEPT_LANGUAGE‘]; // Benutzer Sprache
$_SERVER[‘REQUEST_URI‘];          // URI or Filename
$_SERVER[‘HTTP_USER_AGENT‘];      // UA Daten

?>
```

# $_COOKIE Superglobal

- Gegenstück zu Sessions
- Informationen über längeren Zeitraum speichern
- Ablaufdatum (expire)
- Browser muss Cookies akzeptieren
- Cookies müssen vor jeglicher Ausgabe an den Browser gesetzt werden

```php
<?php

// Cookie setzen
setcookie($name, $value, $expire);

// Cookie setzen
setcookie('chocolate_cookie', 'awesome', time() + (3600*24));

// Cookie auslesen
echo $_COOKIE['chocolate_cookie']; // awesome

?>
```

# Dateioperationen - Speichern

```php
<?php

$content = ["Zeile 1", "Zeile 2", "Zeile 3"];

$handle = fopen("file.txt", "w");

foreach ($content as $line) {
    fputs($handle, $line."\n");
}

fclose($handle); // Resourcen IMMER schließen

?>
```

# Dateioperationen - Lesen

```php
<?php

$file = "file.txt";

$content = file_get_contents($file);

?>
```

# Datei-Uploads

- HTML-Formulare
- Attribut: `enctype=”multipart/form-data”`
- Datei wird vom PHP Interpreter temporär zwischengespeichert
- Dateiinformationen über `$_FILES` Superglobal
- Datei muss in PHP an den Bestimmungsort kopiert werden

```
<form enctype="multipart/form-data"
      action=”#" method="POST">

    <!-- Empfehlung an den Browser -->
    <input type="hidden" name="MAX_FILE_SIZE"
value="30000" />

    Send this file:
    <input name="uploaded_file" type="file" />

    <input type="submit" value="Send File" />!
</form>
```

# Datei-Uploads

- Datei muss durch `move_uploaded_file()` gesichert werden
- Entfernen von Pfadinformationen mittels `basename()`
- PHP löscht temporäre Datei mit Ende des Skripts
- Bildgrößen: `getimagesize()`

```php
// Originaldateiname
$_FILES['uploaded_file']['name'];

// MIME-Type (bspw. image/png, application/pdf)
$_FILES['uploaded_file']['type'];

// Dateigröße in Byte
$_FILES['uploaded_file']['size'];

// temorärer Dateiname
$_FILES['uploaded_file']['tmp_name'];

// ggf. Fehlercode des Uploads
$_FILES['uploaded_file']['error'];
```

# Datei-Uploads

## Upload Errors

+ Wert 0: Es liegen keine Fehler vor
+ Wert 1: überschreitet die festgelegte Größe (php.ini)
+ Wert 2: überschreitet die festgelegte Größe (HTML Formular)
+ Wert 3: Datei wurde nur teilweise hochgeladen
+ Wert 4: Es wurde keine Datei hochgeladen
+ Wert 6: Fehlender temporärer Ordner
+ Wert 7: Speichern der Datei auf der Festplatte fehlgeschlagen
+ Wert 8: Eine PHP Erweiterung hat den Upload gestoppt

# Arrays serialisieren

- serialize()
- unserialize()
- Array <-> String
- Abspeichern von Arrays

```php
<?php

$array = [
    "a" => 1,
    "b" => 2,
    "c" => [3, 4, 5]
];

$serialized = serialize($array);

?>
```

# Error Reporting

```php
<?php

// Error Reporting abschalten
error_reporting(0);

// Alle Fehler melden
error_reporting(E_ALL);

// Notizen, die keine fatalen Fehler sind
error_reporting(E_NOTICE);

?>
```

# Copyright

Referenzen: http://php.net

Autor: Alexander Hofbauer <hofbauer.alexander+sae@gmail.com>
basierend auf Ausarbeitungen von Guillermo Neugebauer