



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 (v7.4.10 - 3. Sept. 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 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

$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**.

```
// 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.

Typcasting passiert in PHP bei Bedarf automatisch:

```
<?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**

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

```
<?php  
  
$i = 'ein String';  
$k = 'Das ist $i';  
echo $k; // Das ist $i  
  
?>
```

```
<?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
$i = 'ein String';
$k = "Das ist \"$i\"";
echo $k; // Das ist "ein String"

?>
```

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

```
<?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
```

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

```
?>
```

Zahlen

Integers

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

?>
```

Floats/Doubles

```
<?php
$zahl = 3.1415;
$zahl = -2.7182;

?>
```


Boolesche Werte

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

```
<?php  
  
$true = true;  
$false = false;  
  
?>
```

Arrays/Felder

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

```
<?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

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

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

?>
```

Assoziative Arrays

```
<?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  
  
define('FOO', 'bar');  
define('ANSWER_TO_THE_QUESTION_OF_LIFE_THE_UNIVERSE_AND_EVERYTHING', 42);  
  
?>
```

Operatoren

Zuweisungsoperator

```
// Einfache Zuweisung: =
```

```
$a = 37;
```

```
// Kombinierte Operatoren: +=, -=, *=, .=", /=, %="
```

```
$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

```
$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

```
$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

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

```
// Verkettten 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
```

```
$a = 1; // a: 1
```

```
$b = 2; // b: 2
```

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

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

```
?>
```

Ausgaben und Debugging

Ausgaben - echo

```
<?php

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

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

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

?>
```

Ausgaben - print

```
<?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

$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

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

// rekursive Ausgabe
print_r($werte);

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

?>
```

Ausgaben - debugging: var_dump

```
<?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
```

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

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

```
?>
```

- + Initialisierung
- + Bedingung
- + Änderung

Schleifen - while

```
<?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

do {
    // Code
} while (Bedingung)
```

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

```
?>
```

- + Bedingung am Ende
- + wird mind. 1x durchlaufen

Schleifen - foreach

```
<?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

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

?>
```

Schlüssel & Wert

+ häufig für assoziative Arrays verwendet

```
<?php

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

?>
```

Bedingung - if/else

```
<?php

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

?>
```

Funktionen und Parameter

Funktionen

```
<?php

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

// Funktion aufrufen
funktionsname(Parameter);

?>
```

Funktionen

```
<?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

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

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

?>
```


Funktionen - Parameter

```
<?php

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

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

$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

$someString = "some cool string";

echo strlen($someString); // 16

?>
```

Teilen von Strings

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

```
<?php

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

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

echo $autos[2]; // Audi

?>
```

Zusammenführen von Arrays

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

```
<?php

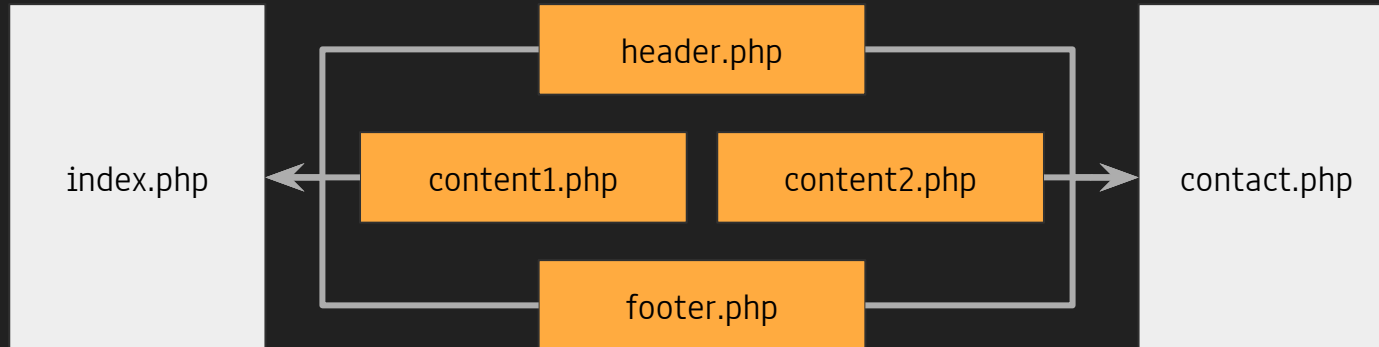
$string = "rot blau gelb gruen";
$farben = explode(' ', $string);

$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

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

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

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

?>
```

```
<?php // functions.php

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

?>
```

include() Beispiel

```
<?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
```

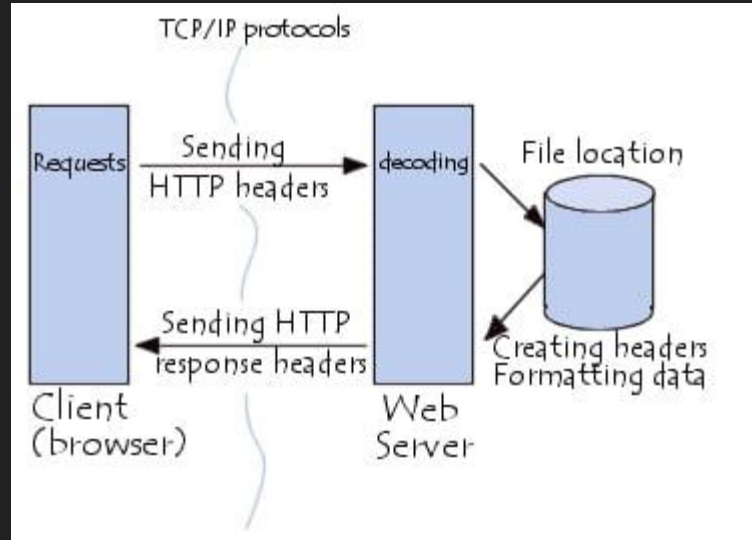
```
$_GET | $_POST | $_REQUEST // Request-Parameter
```

```
$_FILES // Datei-Upload
```

```
$_COOKIE | $_SESSION
```

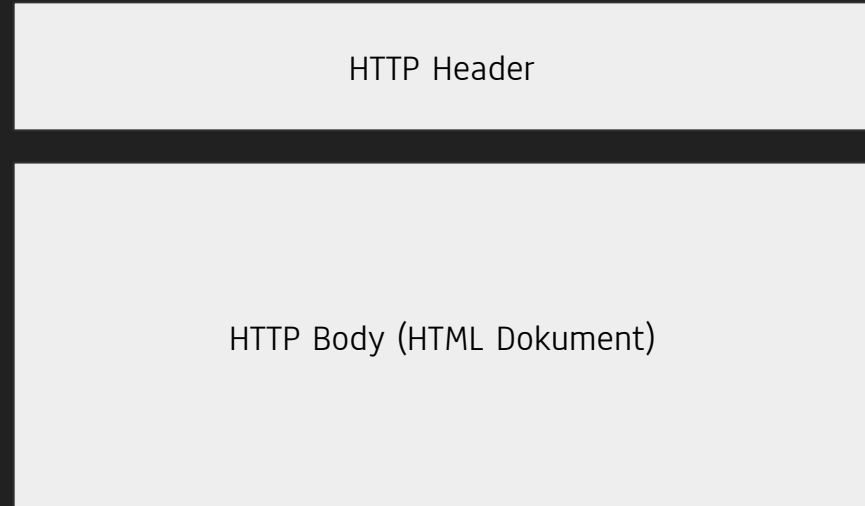
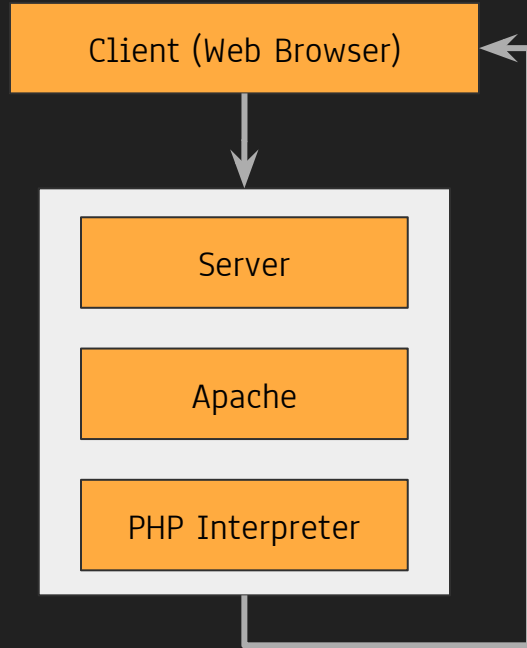
```
$_SERVER // Server Informationen
```

HTTP Basics



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

HTTP Basics



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

GET-Request

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

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

<form method="post">
    <input type="text" name="name">
    <input type="text" name="email">
    <button type="submit">Submit</button>
</form>
```

String Funktionen

```
// 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?
- + E-Mail enthält @ und Punkt?
- + 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-E-mails: **Tabellen und inline-CSS** :'(