Vorlesung

Webbasierte Informationssysteme

(CS4130)

PHP Hypertext Preprocessor

Professor Dr. rer. nat. habil. Sven Groppe
https://www.ifis.uni-luebeck.de/index.php?id=groppe

Chronologische Übersicht über die Themen

PHP Hypertext Preprocessor

- Open Source
- Server-basierte Skriptsprache
- dokumentenzentrierte HTML Programmierung
  - Einbettung von PHP Code in HTML mit HTML als Ergebnis
  - aber auch General-Purpose Programmiersprache
- prozedurale Sprache mit objektorientierten Erweiterungen
- dynamisch typisiert mit wenigen einfachen Typen
- Notation orientiert sich an Perl und C
- große Funktionsbibliothek
  - Z.B. umfangreiche Unterstützung von Datenbanken
- über 80% von durch server-seitige Programmierspachen generierte Webseiten verwenden PHP*

* W3Techs: Usage of server-side programming languages for websites

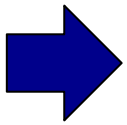
PHP - Anwendungen und Standardkonfiguration

- kleine private bis mittelgroße kommerzielle Projekte
- Schwerpunkt:
  - Generation von Webseiten basierend auf Daten in Datenbanken
- Beispielanwendungen
  - Content Management Systeme
    - Typo3, Wordpress, MediaWiki, Joomla!, Drupal, …
- Standardkonfiguration mit freier Software (bis auf Windows)

Komponente	LAMP	WAMP
Betriebssystem	Linux	Windows
Web Server	Apache Web Server	
Datenbankmanagementsystem	MySQL	
Serverseitige Skriptsprache	PHP	

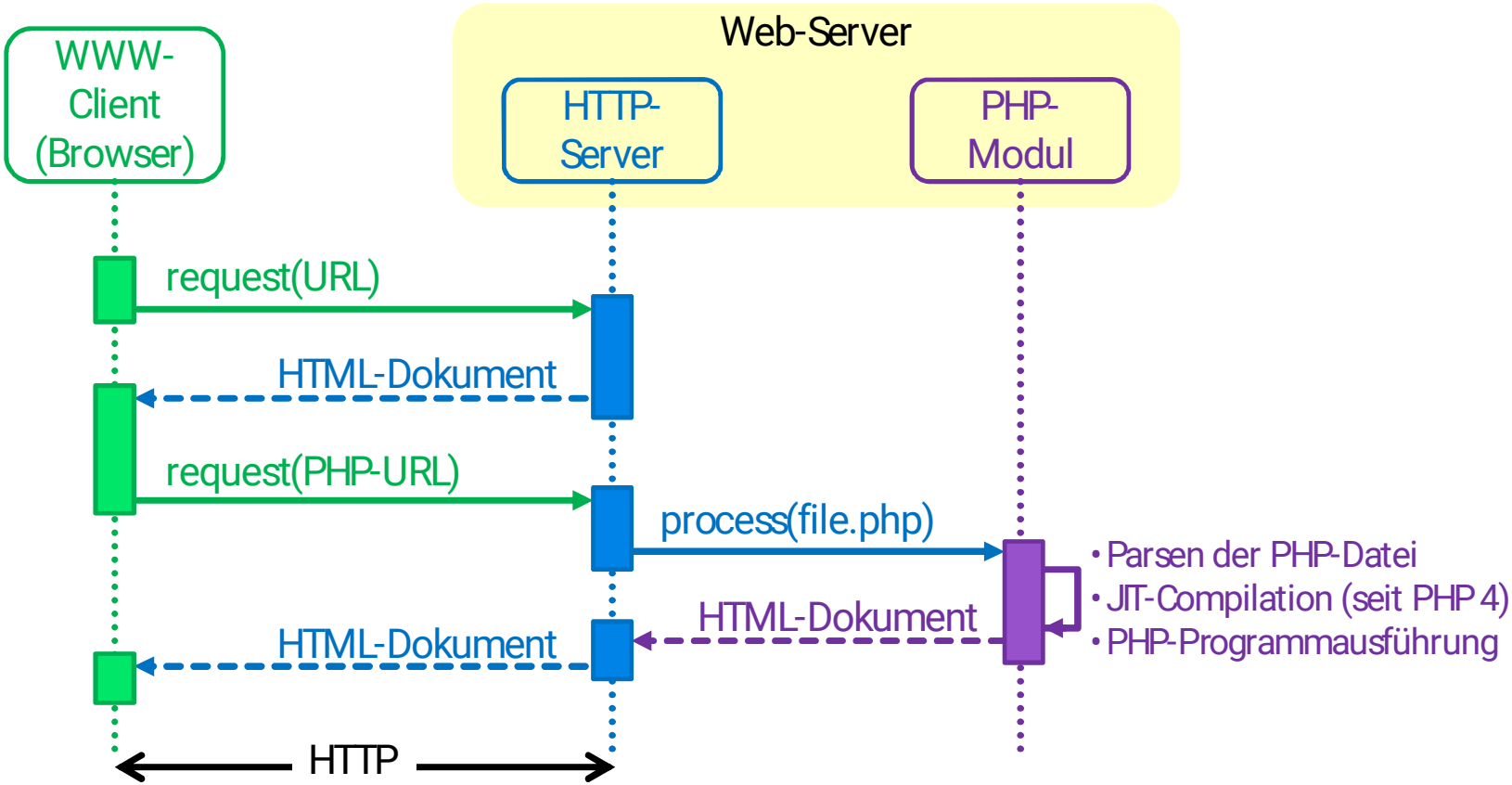
Beispieleinbettung

```
<!DOCTYPE html>
<html>
  <head><title>Tree</title></head>
  <body>
    <pre>
      <?php
        $middle = 6;
        $width = 1;
        while ($middle >= 0) {
          $i=0;
          while($i<$middle){
            echo ' ';
            $i++;
          }
          $i=0;
          while($i<$width){
            echo '*';
            $i++;
          }
          $middle--;
          $width += 2;
          echo "\n";
        }
      ?>
    </pre>
  </body>
</html>
```



```
     *
    ***
   *****
  *******
 *********
***********
*************
```

Anfordern einer PHP-Seite



PHP – Einbettung in HTML-Dokumente

- HTML-Anteile ohne PHP-Code werden nicht interpretiert und unverändert in das Ergebnis aufgenommen
- Interpretation nur von ausgezeichneten PHP-Code
- Syntaxalternativen der Auszeichnung
  - Standard-Tags: `<?php … ?>`
  - Sprachspezifische Script-Deklaration: `<script language="php"> … </script>`
  - ASP-Stil: `<% … %>`
  - Kurzschreibweise: `<? … ?>`
- Konfiguration im PHP-Modul notwendig bei Verwendung des ASP-Stils oder der Kurzschreibweise

Grundlagen der Syntax

Bezeichner

```
identifier = [a-zA-Z_][a-zA-Z_0-9]*
```

- **Variablen**namen beginnen immer **mit** `$`
- **Konstanten- und Funktion**snamen **ohne** `$`
- Groß-/Kleinschreibung
  - bei Variablennamen (& Konstanten via `const`) unterschieden
  - bei Funktionsnamen (& Konstanten via `define`) **nicht** unterschieden

Anweisungen

```
<?php echo "Hello!";$i++; ?> ≡ <?php echo "Hello!";$i++ ?>
```

- Ende einer Anweisung bei **Semikolon** oder **schließendes Tag** `?>`

Bedingte Anweisung

```
if($a < $b) { $min = $a; } else { $min = $b; }
```

- Bei einzelnen Anweisungen sind die `{}`-Klammern optional

while-Schleife

```
$i=0; while($i<$width) { echo '*'; $i++; }
```

for-Schleife

```
for($i=0; $i<$width; $i++) { echo '*'; }
```

return-Anweisung

```
return $n*42;     return '+';
```

Parameterübergabe

- standardmäßig mittels **call-by-value**
- optional: **call-by-reference**
  - Übernahme der Änderungen des Wertes der Parametervariablen innerhalb der Funktion zu der Variablen des Funktionsaufrufes
  - Kennzeichnung in der Funktionsdefinition durch Notation von „&“

Kommentare

- Mehrzeilig: `/* Kommentar … */`
- Einzeilig: `// Kommentar`

Parameterübergabe

- call-by-value

```php
<?php
 $a = 1;
 function calltest( $a ) {
  $a = 2;
  echo $a;
 }
 calltest($a);
 echo $a;
?>
```

Ausgabe: ???

- call-by-reference

```php
<?php
 $a = 1;
 function calltest( &$a ) {
  $a = 2;
  echo $a;
 }
 calltest($a);
 echo $a;
?>
```

Ausgabe: ???

Variablen

- Definition durch Initialisierung
- Annahme von Werten beliebigen Typs
- Unterscheidung zwischen
  - lokalen Variablen
  - globalen und superglobalen (vordefinierte) Variablen
    - gelten im ganzen Programm
    - superglobale Variablen sind immer sichtbar
    - Definition von globalen Variablen
      - außerhalb des Bindungsbereiches einer Funktion
      - Verwendung des Schlüsselwortes `global` innerhalb des Bindungsbereiches einer Funktion
  - statischen Variablen
    - existieren nur im Bindungsbereich einer Funktion
    - Wert geht beim Verlassen dieses Bereichs nicht verloren

Illustration von Geltungsbereichen


```php
<?php
      $a = 1;
      function globaltest() {
       $a = 2;
       echo $a;
      }
      globaltest();
      echo $a;
?>
```

Ausgabe: ???

```php
<?php
      $a = 1;
      function globaltest() {
       global $a;
       $a = 2;
       echo $a;
      }
      globaltest();
      echo $a;
?>
```

Ausgabe: ???

Konstanten

- Gültigkeitsbereich entspricht dem von superglobalen Variablen
- Definition durch
  - die Funktion `define()`

    Beispiel:

```php
define("HELLO", "Hallo Welt.");
echo HELLO; // gibt "Hallo Welt." aus
```

  - das Schlüsselwort **const** (ab PHP 5.3.0)

    Beispiel:

```php
const HELLO = "Hallo Welt.";
echo HELLO; // gibt "Hallo Welt." aus
```

Besondere Konzepte von Variablen

- variable Variablen: `$$VarName`
  - Verwendung des Inhaltes von `$VarName` als Variablenname
- Referenzen: `$VarName2 = &$VarName1`
  - Neuer (Alias) Variablenname für den Inhalt von `$VarName1`
  - Durch Referenzen Call-By-Reference bei der Parameterübergabe möglich

Built-In Funktion	Bedeutung
`boolean isset($VarName)`	Überprüfung, ob eine Variable definiert ist
`void unset($VarName)`	Löschen einer Variablen
`boolean print_r($VarName)`	Ausgabe von Informationen über eine Variable in lesbarer Form

Primitive Datentypen

integer
float

- Dezimale, hexadezimale oder oktale Notation von Ganzzahlen
- Bei Überlauf Konvertierung nach `float`

string

- Zeichenkettenliterale mit
  - einfachen Anführungszeichen
    - Alle Zeichen stehen für sich selbst (nur `'` muss „escaped“ werden: `\'`)
  - doppelten Anführungszeichen
    - Parsen des Strings und Ersetzen von vorkommenden Variablen und Escape-Folgen, Bsp.: `"Sum of $year = \t$sum EUR"`
- Konkatenation mit Punkt: `"Hello" . "world!"`
- Ausgabe durch **echo** (von durch Kommata getrennte Ausdrücke) oder `print` (1 Parameter)

boolean

- Literale: **`true`** und **`false`** (keine Unterscheidung von Groß-/Kleinschreibung)
- Operatoren: Konjunktion `&&` bzw. **and** (kleinere Präzedenz), Disjunktion `||` bzw. **or** (k.P.), Negation `!`, Exklusiv-Oder **xor**

Datentypen: Arrays

Bedeutung	Abbildung von Indizes auf Werte
Einträge	Paare (k, v) mit k ganzzahl. (evtl. von anderen primitiven Typen konvertiert) oder String-Schlüssel und zugeordneter Wert v
	Erzeugung von Arrays
Liste von Werten	`// indiziert von 0 an:` `$monthName = array("", "Jan", "Feb", "Mar");`
Liste von Index-Wert-Paaren	`$monthName = array(1 => "Jan", 2 => "Feb", 3 => "Mar");`
Explizite Indizierung	`$monthName[1]= "Jan"; $monthName[2]= "Feb"; …`
Assoziatives Array	`$monthName = array("Jan" => 1, "Feb" => 2, "Mar" => 3);`
	Zugriff
Aufzählung aller Elemente	`foreach($monthName as $k => $v){ echo $k." => ".$v." "; }`

Spezielle Werte

null

- Bedeutung: Variable hat keinen Wert
- keine Unterscheidung von Groß-/Kleinschreibung
- Einzig möglicher Wert des Typs `NULL`
- Interpretation des Wertes einer Variablen als `NULL`, falls
  - ihr die Konstante `NULL` als Wert zugewiesen wurde
  - ihr bis zum aktuellen Zeitpunkt kein Wert zugewiesen wurde, oder
  - sie mit `unset()` gelöscht wurde

Superglobale Variablen (vordefinierte Arrays)

Superglobale Variable	Bemerkung
`$GLOBALS`	(Name, Wert)-Paare von globalen Variablen
`$_SERVER`	Informationen über Server und Ausführungsumgebung
`$_GET`	HTTP GET-Variablen
`$_POST`	HTTP POST-Variablen
`$_FILES`	HTTP Dateiupload-Variablen
`$_COOKIE`	HTTP Cookies
`$_SESSION`	Sessionvariablen
`$_REQUEST`	Ein assoziatives Array, das standardmäßig den Inhalt von `$_GET`, `$_POST` und `$_COOKIE` enthält
`$_ENV`	Umgebungsvariablen

- Vordefinierte Variablen können **nicht** als variable Variablen verwendet werden

Unterstützung von Sitzungen (Sessions)

- Möglichkeit, Daten während einer Folge von Aufrufen einer Website festzuhalten (bis der Browser geschlossen wird)
  - Session-Daten werden in `$_SESSION` gespeichert
  - Zuordnung einer eindeutigen Session-ID zu einem Besucher beim Aufruf der Website
  - Bei Aufruf der Website prüft PHP je nach Konfiguration automatisch oder durch Aufruf von `session_start()`, ob eine Session-ID mitgesendet wurde

    $\Rightarrow$ Wiederherstellung einer zuvor gespeicherten Session-Umgebung

Beispiel zu Sessions: Login-Skript

HTML-Seite für den Login

```
<form action="login.php" method="post">
  User Name:<br/>
  <input type="text" size="24" maxlength="50" name="name"/><br/><br/>
  Password:<br/>
  <input type="password" size="24" maxlength="50" name="passwd"/><br/>
  <input type="submit" value="Login"/>
</form>
```

User Name:

Password:

Login

login.php

```
<?php
session_start();
$name = $_POST["name"];
$passwd = $_POST["passwd"];
$hash = md5($passwd);
// following condition can be replaced
// with database requests or file lookup
if($name == 'Peter' && $hash == 'XYZ'){
  $_SESSION['username'] = $name;
  echo "Login successful. <a href=\"secret.php\">Secret area</a>";
} else { echo "Access Denied <a href=\"login.html\">Back</a>"; }
?>
```

Erfolgreiche Anmeldung:

Login successful. Secret area

Falscher Benutzer/Passwort:

Access Denied Back

Anmerkungen zum Login-Skript

- Kommunikation zu `login.php` und nachfolgenden Seiten sollte gesichert über HTTPS geschehen
  - Ansonsten Mitlesen der Session-ID und damit Übernahme der Session möglich
- Auf nachfolgenden PHP-Seiten (wie `secret.php`) muss jeweils überprüft werden, ob `$_SESSION['username']` gesetzt ist
  - Ansonsten ist der Benutzer nicht eingeloggt!

Objektorientierte Programmierung in PHP

- Vergleichbare Mächtigkeit von OOP-Sprachkonstrukten von PHP zu anderen OO-Programmiersprachen
- Details differieren

```php
abstract class Mammal {
  public function getNumberOfLegs() {
    return $this -> legs; // Achtung: Zugriff auf Variable der Kindsklasse
  }
}
class Dog extends Mammal {
  protected $legs = 4;
}
$waldi = new Dog;
echo $waldi -> getNumberOfLegs() . " legs";
```

Funktionsbibliothek

- ausgereift
- > 700 Funktionen
- Beispiele der Abdeckung der PHP-Funktionsbibliothek:
  - Arrays
  - Protokolle
  - Datenbanken
  - Datum/Uhrzeit
  - Dateiverzeichnisse
  - Dateien
  - Grafik
  - HTTP
  - IMAP
  - LDAP
  - Mathematik
  - MCAL
  - Mcrypt
  - Mhash
  - PDF
  - POSIX
  - **reguläre Ausdrücke**
  - Strings
  - Variablenmanipulation
  - XML

Reguläre Ausdrücke

- Hauptanwendungen
  - Validieren von Benutzereingaben
  - Suchen (& Ersetzen) in Zeichenketten
    (ausdrucksstärker als nur Schlüsselwortsuche)
- 2 „Engines“
  zum Verarbeiten regulärer Ausdrücke
  - POSIX (vor PHP 7)
  - Perl Compatible Regular Expressions (PCRE)
    - Perl-kompatibel
    - 200x schneller als POSIX-Engine
      $\Rightarrow$ hier Vorstellung von PCRE*

Definition Grammatik und erzeugte Sprache

Definition Grammatik: Viertupel $G = (N, \Sigma, P, S)$ mit

- N endliche Menge von Nichtterminalsymbolen
- Σ endliche Menge von Terminalsymbolen, $N \cap \Sigma = \varnothing$
- P endliche Menge von Produktionen bzw. Regeln $P \subset (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$
- S Startsymbol, $S \in N$

Beispiel:

$N = \{S\}$

$\Sigma = \{0, 1\}$

$P = \{S \to \epsilon, S \to 0S1\}$

Definition erzeugte Sprache: Worte w der erzeugten Sprache $L(G) \subseteq \Sigma^*$ einer Grammatik $G = (N, \Sigma, P, S)$ können vom Startsymbol aus mit einer endlichen Anzahl von Schritten abgeleitet werden: $L(G) := \{w \in \Sigma^* | S \to_G^* w\}$ wobei $\to_G^*$ die reflexiv-transitive Hülle der Transitionsrelation (d.h. Anwendung einer Regel) $\to_G$ repräsentiert

Bsp.: $S \to_G 0S1 \to_G 00S11 \to_G 0011, L(G) = \{0^a 1^a | a \in \mathbb{N}_0\}$

Chomsky-Hierarchie:

Typ 3 $\subset$ Typ 2 $\subset$ Typ 1 $\subset$ Typ 0

Typ der Sprache/ Grammatik	Für alle Regeln $w \to w' \in P$ gilt:	Ausführungs-modell	Komplexität des Wortproblems $w \in L(G)$?
0	keine Einschränkung	Turingmaschine	unentscheidbar
1 kontextsensitiv	$\lvert w\rvert \leq \lvert w'\rvert$	linear beschränkte Turingmaschine	exponentiell, NP-hart
2 kontextfrei	$w \in N$ (linke Seite w enthält eine einzelne Variable)	Kellerautomat	$O(n^3)$
3 regulär	$w \in N \wedge w' \in (\Sigma \cup \Sigma N)$ (rechte Seite besteht aus einem Terminal optional gefolgt von einem Nicht-Terminal)	deterministischer/ nicht-deterministischer endlicher Automat	$O(n)$

- Typ 3-Grammatik $\equiv$ reguläre Ausdrücke $\equiv$ endliche Automaten, reg. Ausdrücke spezifizieren reg. Sprachen, endliche Automaten akzeptieren Worte der reg. Sprachen

Aufbau eines Perl-kompatiblen regulären Ausdruckes

- Syntax: `"D R D [M]"`
  - `D` (für Delimiter) muss ein nicht-alphanumerisches Zeichen sein
  - `R` für Regulärer Ausdruck
  - Beeinflussung der Match-Strategie durch optionale Modifizierer `M`
- Beispiele
  - `"/def/"`
  - `"=def="`

Funktionen für reguläre Ausdrücke

Funktion	Beschreibung
`int preg_match(string $pattern, string $string [,array &$matches [, int $flags = PREG_PATTERN_ORDER [, int $offset = 0 ]]])`	Durchsucht `$string` nach der ersten Übereinstimmung mit dem durch `$pattern` definierten regulären Ausdruck, gibt `1` bei mind. einer Übereinstimmung zurück, ansonsten `0`
`int preg_match_all(…)`	Sucht nach allen Übereinstimmungen (& legt sie in `$matches` ab)
`mixed preg_replace(…)`	Suchen & Ersetzen von Übereinstimmungen
`mixed preg_replace_callback(…)`	Suchen & Ersetzen von Übereinstimmungen, wobei der Return-Wert einer Callback-Funktion den Ersetzungstext bestimmt

Aufbau eines regulären Ausdruckes R

R	Bedeutung
a	Das Zeichen a
FG	Zusammenfügen von zwei Worten
$F\|G$	Alternative
$F?$	optionales F (gleiche Semantik wie $F\|\epsilon$ mit ϵ leeres Wort)
(F)	Klammerung
$F+$	nicht-leere Folge von Worten aus $L(F)$
$F*$	beliebig lange Folge (auch 0-mal) von Worten aus $L(F)$
$F\{n\}$	Folge von n Worten aus $L(F)$

R	Bedeutung
$F\{n,m\}$	Folge von mind. n und max. m Worten aus $L(F)$
$[abc]$	alternativ ein Zeichen aus der Klammer
$[\hat{}abc]$	alternativ ein anderes Zeichen als die in der Klammer
$[a-zA-Z]$	alternativ ein Zeichen aus Zeichenbereichen
.	beliebiges Zeichen
^	Anfang der Zeichenfolge (nichts darf vorangehen)
$	Ende der Zeichenfolge (nichts darf darauf folgen)

F, G reguläre Ausdrücke — n, m ganze Zahlen

$L(F)$ definierte Sprache des regulären Ausdruckes F — a, b, c, A, Z Zeichen

Bemerkungen zu regulären Ausdrücken

- Escape-Zeichen: \
- Standardeinstellung für die Match-Bildung: „gierig“ (greedy)
  - Suche nach dem längsten Match
  - Umschaltung in den „nicht gierig“-Modus: Ungreedy-Modifizierer `U`
- Beispiele
  - Bezeichnung von Gleisabschnitten eines Bahnhofes mit den Gleisen `1` bis `9`, `a` bis `f`: `"/^[1-9][a-f]?$/"`
  - Bezeichner: `"/^[a-zA-Z][a-zA-Z_0-9]*$/"`

Regulärer Ausdruck für EMail-Adressen

- Entwickeln Sie einen regulären Ausdruck für eine deutsche Email-Adresse der Form

  `Bezeichner@Bezeichner.de`

  - `Bezeichner` kann dabei mit einem Klein- oder Großbuchstaben beginnen und anschließend beliebig viele Buchstaben, Ziffern, Unterstriche oder Punkte enthalten*

* Dieser reguläre Ausdruck soll nicht dafür verwendet werden, um jedwede deutsche Email-Adressen zu erkennen (viel mehr ist erlaubt!), sondern kann eingesetzt werden, um bei der Email-Vergabe nur einigermaßen sinnvolle Email-Adressen zu vergeben…

Quellen zum Nachschlagen und Nacharbeiten

- PHP Dokumentation (The PHP Group)
- SELFPHP (SELFPHP OHG)
- PHP Tutorial (W3 Schools)
- PHP 101: PHP For the Absolute Beginner (Zend)
- Reguläre Sprachen, reguläre Ausdrücke (Leibniz-Rechenzentrum)

Zusammenfassung PHP

- Hauptanwendung: Generation von Webseiten basierend auf Daten in Datenbanken
  - Dafür ausgelegte Sprachkonzepte & Funktionsbibliothek
- Meist verwendete Serverseitige Programmiersprache
- Standard-Webserver-Architektur:
  L(W)**AMP = L**inux(Win)**+A**pache Webserver**+M**ySQL**+P**HP
- Einführung in imperative Sprachkonstrukte von PHP (mit einem Bsp. für OOP in PHP)
- Sessions
- Reguläre Ausdrücke in PHP