# Programmieren in PHP

Christian RAINER

# Software

- Webserver: Apache 2.4


- PHP: 8.2

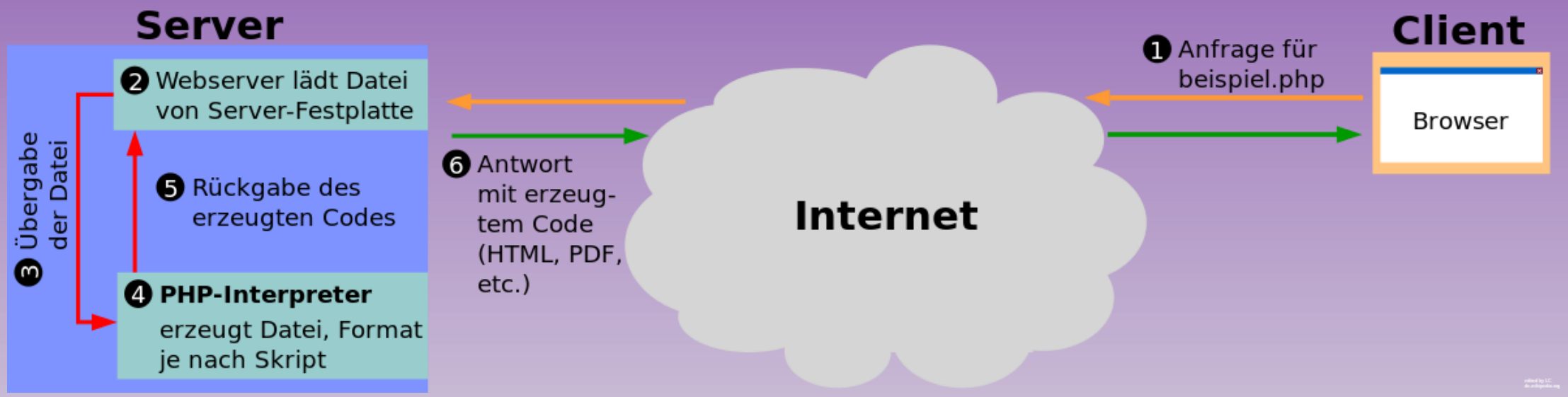
# Programmieren in PHP

- Variablen und Datentypen
- Funktionen
- Kontrollstrukturen & Schleifen
- Praxisbeispiel

# Warum gerade PHP?

- Dynamische generierte, auf den Anwender zugeschnittene Webseiten
- Betriebssystem unabhängig
- Quelloffen und ständig weiterentwickelt
- Kostenlos, auch für den kommerziellen Einsatz
- Einsteigerfreundlich

# Funktionsweise von PHP



# Die Syntax von PHP

- Starten / beenden eines PHP Blocks
  - Standard Tag: `<?php`
  - Short Tag: `<?`
  - Beenden: `?>`
- Ausgaben an den Browser mit echo
  - `<?php echo "Hallo Welt!"; ?>`
- Trennung der Befehle per Semikolon ;
  - `<?php echo "Hallo "; echo "Welt!"; ?>`

# Beispiel: Hallo Welt!

- Text „Hallo Welt“ ausgeben
- Text mit zwei getrennten Befehlen ausgeben

# Kommentare

- Einzeilige Kommentare
  - `// Einzeiliger Kommentar`
  - `# Alternative Schreibweise (früher)`
- Mehrzeilige Kommentare
  - `/* Kommentar */`
  - 
```
/*
Dies ist ein
 mehrzeiliger Kommentar.
 */
```

# Beispiel: Hallo Welt!

- Programm Kommentieren

# Was passiert?

```php
<?php
$antwort = 42;
echo $antwort;
?>
```

# Variablen

- Sind Platzhalter für Werte
- Variablen müssen vor der Verwendung nicht definiert werden
- Der *Datentyp*, wird bei der ersten Verwendung automatisch definiert

# Verwendung von Variablen

- Eine Variable setzt sich aus einem $, gefolgt von dem Namen zusammen
  - `$ganzzahl = 5;`

- Konventionen für den Variablennamen:
  - Gültige Zeichen: a-z, A-Z, 0-9, _
  - Das erste Zeichen darf keine Zahl sein.

# Datentypen: Numerisch

- Integer: Ganzzahl
  - Systemabhänginger Wertebereich, mind. 2^32

```
$ganzzahl1 = 1234;
$ganzzahl2 = -1234;
```

- Float: Fließkommazahl
  - Begrenzte Genauigkeit

```
$reellezahl1 = 43.12;
$reellezahl2 = 2.413e2;
```

# Beispiel: Integer und Float

- Einen Integer definieren und ausgeben
- Einen Float definieren und ausgeben

# Datentypen: Text

- String: Zeichenkette
  - `$zeichenkette1 = "Willkommen im WIFI";`
  - `$zeichenkette2 = 'Herzlich willkommen';`
- Heredoc und Nowdoc Notifikation
  - ```
    $heredoc = <<<ANYWORD
    Mein name ist $name
    Danke für Ihren Besuch!
    ANYWORD;
    ```
  - ```
    $nowdoc = <<<'AWORD'
    Hello!
    AWORD;
    ```

# Datentypen: Text

- Steuerzeichen (Escape-Sequenzen)
  - Verwendbar bei Strings mit “ oder *Heredoc* Notifikation
  - Zeilenvorschub: `\n`
  - Horizontaler Tabulator: `\t`
  - Dollarzeichen: `\$`
  - Backslash: `\\`
  - Anführungszeichen: `\"`

# Beispiel: String

- Einen String in einer Variable merken und ausgeben
- Verwendung der Variablen in Verknüpfung mit fixem Text

# Datentypen: Spezielle

- Boolean
  - Kann nur zwei Werte annehmen:
    **`true`** oder **`false`**
- NULL
  - Repräsentiert “nichts“, bzw. “nicht definiert“
- Object
  - Ein Objekt mit *Eigenschaften* und *Methoden*

# Konstanten

- Werden wie folgt definiert:
  - `define ("alter",  43);`
- Können im weiteren Skriptverlauf nicht geändert werden
- Sinnvoll für z.B. Datenbankverbindungs-Werte (DB-Benutzer, DB-Name, …)

# Beispiel: Bool, Null, Konstante

- Typ Boolean in Variable definieren und ausgeben
- Null in einer Variable definieren und ausgeben
- Eine Konstante definieren und ausgeben

# Datentypen: Array

- Array sind Datenfelder, die mehrere Werte in sich beherbergen können:
  - `$daten = array("Wert 1", 2, 3.3, "Wert 4");`
- Jeder Werte im Array hat einen eigenen Datentyp
- Jedem Wert wird ein eindeutiger Index zugeordnet

# Beispiel: Array

- Ein Array mit Namen definieren und einen Wert daraus ausgeben

# Datentypen: Array

- Die Werte werden durch einen Index angesprochen der …
  - *numerisch* (Integer), oder
  - *assoziativ* (String) ist

# Beispiel: Array

- Ein assoziatives Array mit Daten einer Person definieren und verwenden

# Datentypen: Array

- Wir wissen:

  Ein Array beinhaltet Werte mit beliebigen Datentypen

- Dann müssen doch…

  …ein Wert in einem Array auch vom Datentyp “array“ sein dürfen

# Beispiel: Array

- Ein mehrdimensionales Array erstellen und verwenden

# Zuweisungs-Operatoren

- Eine Zuweisung besteht aus Ausdrücken.
- Einfache Ausdrücke sind z.B.:
  - `21`
  - `$a`
- Bei einer Zuweisung wird ein Wert einer Variable zugewiesen. Dies geschieht mit dem Gleichheitszeichen =
  - `$a = 21;`

# Mathematische Operatoren

- Grundrechenarten
  - `$c = 5 + 3;`
  - `$c = $a - $b;`
  - `$c = $a * $b;`
  - `$c = $a / $b;`
- Modulo: Rest einer Division
  - `$c = 5 % 2`

# Modulo-Operatoren

- Wechselnde Zeilenmarkierungen
- Wenn: `$zeile % 2 == 0`, dann grau. Sonst weiß.

```
1)  1 % 2 = 1
2)  2 % 2 = 0
3)  3 % 2 = 1
4)  4 % 2 = 0
5)  5 % 2 = 1
```

→ Welche Farbe?

# Komplexe Zuweisungs-Operatoren

- Zuweisung eines komplexen Ausdrucks
  - `$a = 9 / 4 + 27 * 2.5;`
- Addition und Zuweisung in einem
  - ```
    $c = 5;
    $c += 3;
    echo $c;
    ```
  - Funktioniert mit -=, *=, /=, %=
    ```

# Komplexe Zuweisungs-Operatoren

- Zeichenkette an  String anhängen

```
$str = "Baum";
$str .= " fällt!";
echo $str;
```

# Inkrement-Operatoren

- Zu einem Integer 1 addieren

```
$a++;
```

- Prä-Inkrement: Erst erhöhen, dann verwenden

```
$zahl = 3;
echo ++$zahl;      // Ausgabe: 4
echo $zahl;        // Ausgabe: 4
```

- Post-Inkrement: Erst verwenden, dann erhöhen

```
$alter = 3;
echo $alter++;     // Ausgabe: 3
echo $alter;       // Ausgabe: 4
```

# Dekrement-Operatoren

- Von einem Integer 1 subtrahieren

```
$a--;
```

- Prä-Dekrement: Erst  verringern, dann verwenden

```
$zahl = 3;
echo --$zahl;      // Ausgabe: 2
echo $zahl;        // Ausgabe: 2
```

- Post-Dekrement: Erst verwenden, dann verringern

```
$alter = 3;
echo $alter--;     // Ausgabe: 3
echo $alter;       // Ausgabe: 2
```

# Was passiert?

```php
<?php
$frucht = "Äpfel";
$obst = "Birnen";

if ($frucht == $obst) {
    echo "Äpfel sind Birnen";
} else {
    echo "Äpfel und Birnen kann man nicht
 vergleichen";
}
?>
```

# Kontrollstrukturen: if und else

- Ein Ausdruck wird auf Wahrheit überprüft
- Je nach Ergebnis wir ein bestimmter Programmteil ausgeführt

```
if ($a == $b) {
    echo ”Die Variable a entspricht b.”;
} else {
    echo ”Die Variablen sind ungleich.”;
}
```

# Vergleichs-Operatoren

- Verwendung üblicherweise in `if`-Strukturen
- Ergeben immer **`false`** oder **`true`**
  - Gleichheit:
    ```
    5 == 5      ← true oder false?
    7 == 3
    ```
  - Ungleichheit:
    ```
    "3" != 3
    "zwei" != "2"
    ```
  - Typgleich:
    ```
    "4" === 4
    "eins" === "eins"
    ```
  - Typungleich:
    ```
    3.0 !== 3
    "?" !== "?"
    ```

# Vergleichs-Operatoren

- Verwendung üblicherweise in `if`-Strukturen
- Ergeben immer **`false`** oder **`true`**
  - Größer-gleich:

```
3 >= 3      ← true oder false?
7 >= 3
```

  - Kleiner-gleich:

```
-5 <= -3
5 <= 0
```

# Boolesche Operatoren

- Verknüpft zwei Vergleiche miteinander
  - Und-Operator:

    ```
    "Apfel" == "Apfel" && "Birne" == "Birne"
    ```
  - Oder-Operator:

    ```
    "Apfel" == "Gras" || "Birne" == "Birne"
    ```
  - Negation:

    ```
    !("Apfel" == "Apfel")
    ```
  - Exklusives Oder:

    ```
    "Apfel" == "Apfel" xor "Birne" == "Birne"
    ```

# Kontrollstrukturen: else if

- Wenn ein vorangehender **`if`**-Ausdruck **`false`** liefert, kann mit **`else if`** ein weiterer Ausdruck geprüft werden

```
$a = "foo";
if ($a == "bar") {
    echo "Var a ist: bar.";
} else if ($a == "foo") {
    echo "Var a ist: foo";
} else {
    echo "Nichts von beidem.";
}
```

# Beispiel: if-Abfragen

- Den Besucher nach der Tageszeit begrüßen
- Eine Variable erstellen, in der wir uns eine Zahl zwischen 0 und 23 merken. Enthält später automatisch die aktuelle Stunde.
- Mithilfe von if, else if und else diese Logik erstellen:
  - Von Stunde 0 bis 5 ”Schlaf gut” ausgibt
  - Von Stunde 6 bis 9 ”Guten Morgen” ausgibt
  - Bei Stunde 12 und 18 ”Mahlzeit” ausgibt
  - Von Stunde 19 bis 23 ”Gute Nacht” ausgibt
  - Bei allen anderen Stunden ”Hallo!” ausgibt

# Funktionen

- Sind Programmteile, die eine bestimmte Aufgabe lösen
- Können wiederverwendet (öfter aufgerufen) werden
- Vermeidet kopieren von Programmcode
- Unterschied zwischen:
  - Von PHP vordefinierten Funktionen
    z.B. date(), str_replace()
  - Selbstdefinierte Funktionen

# Funktionen

- Eine Funktion verarbeitet übergebene Daten
- Das Ergebnis wird zurückgegeben und kann im weiteren Programm verwendet werden
- Beispiel einer kaufmännischen Rundung:

```
$geld = 5.989;
$gerundet = round($geld);
echo $gerundet; // gibt 6 aus
```

# Funktionen für Strings

- Großbuchstaben in Kleinbuchstaben umwandeln

```
$klein = strtolower($string);
```

- Leerzeichen vor und nach einem gegebenen String kürzen

```
$kurz = trim($string [, $zeichen]);
```

- HTML-Tags aus Text entfernen

```
$ohne = strip_tags($string [, $erlaubt]);
```

# Funktionen für Strings

- Die Zeichenlänge eines Strings ermitteln

```
$laenge = strlen($string);
```

- Einen Teiltext an bestimmter Position aus Text raus kopieren

```
$teil = substr($string, $start [, $laenge]);
```

- Zeilenumbrüche in <br /> umwandeln

```
$br = nl2br($string);
```

# Funktionen für Strings

- Wichtige Zeichen in ihre HTML-Entities umwandeln

```
$kodiert = htmlspecialchars($eingabe);
```

  - & nach &
  - “ nach "
  - ‘ nach '
  - < nach <
  - > nach >

# Beispiel: String-Funktionen

- String in Kleinbuchstaben umwandeln
- Leerzeichen vor / nach einem String entfernen
- HTML-Tags aus einem String entfernen
- Länge eines Strings zurückgeben
- Einen Teil eines String extrahieren
- Zeilenumbrüche in <br /> umwandlen

# Funktionen für Arrays

- Zähle die Elemente im Eingabe-Array

```
$anzahl = count($array);
```

- Zufälligen, existierenden Index erhalten

```
$index = array_rand($array);
```

- Doppelte Elemente aus dem Array entfernen

```
$einmalig = array_unique($array);
```

- Prüfen, ob ein Wert in einem Array vorkommt

```
$bool = in_array("Wert", $array);
```

# Funktionen für Arrays

- Ein Array aufsteigend alphabetisch sortieren

```
asort($array);
```

- Einen Wert im Nachhinein hinzufügen

```
array_push($array, "Neuer Wert");
```

- oder -

```
$array[] = "Neuer Wert";
```

- Ein Array aufsteigend alphabetisch sortieren und die Schlüssel verwerfen und neu zuordnen

```
sort($array);
```

# Beispiel: Array-Funktionen

- Elemente in einem Array zählen
- Zufälligen Eintrag eines Arrays ausgeben
- Doppelte Werte entfernen
- Prüfen, ob ein Wert in einem Array existiert
- Ein Array alphabetisch aufsteigend sortieren
- Einen Wert einem bestehenden Array anfügen
- Ein Array alphabetisch aufsteigend sortieren und neue Indizes zuweisen

# Was passiert?

```php
<?php
function meine_formel($zahl1, $zahl2) {
    $ausgabe = $zahl1 * 4 + $zahl2;
    return $ausgabe;
}

echo meine_formel(4, 1.5);

?>
```

# Gültigkeitsbereich von Funktionen

- Getrennter Gültigkeitsbereich
  - In einer Funktion hat man keinen Zugriff auf Variablen von außerhalb
  - Außerhalb hat man keinen Zugriff auf Variablen in einer Funktion
- Die Kommunikation geschieht über
  - Eingabeparameter
  - Rückgabewert
- Nach einem abgeschlossenen Funktionsaufruf werden die Variablen darin verworfen

# Gültigkeitsbereich von Funktionen

- Sollte es nötig sein eine Variable von Außerhalb in einer Funktion zu verwenden:
  - ```
    $var = 3;
    function global_beispiel() {
      global $var;
    }
    ```
- Wenn eine Variable über mehrere Aufrufe hinweg bestehen bleiben soll:
  - ```
    function static_beispiel() {
      static $variable_bleibt;
    }
    $variable_bleibt += 3;
    ```

# Beispiel: Eigene Funktionen

- Grad Celsius in Grad Fahrenheit umrechnen
  - Formel: °F = °C * 1.8 + 32
- Datum formatieren
  - 2022-04-17 nach 17.04.22
- Zeichenkette abschneiden und "…" anhängen
  - Ab 10 Zeichen schneiden
  - Unter 10 Zeichen keine "…" anhängen

# Kontrollstrukturen: switch

- Alternative zu vielen aufeinander folgenden **`else if`**‘s
- Findet Verwendung wenn man eine Variable auf unterschiedliche Inhalte prüfen will
- Bietet auch den Fall einer Standardbehandlung (entspricht einem else)
- 
```
switch ($var) {
    case 1: echo "Var ist 1"; break;
    case 2: echo "Var ist 2"; break;
    default: echo "Var ist nichts"; break;
}
```

# Schleifen

- Wiederholen einen Skriptabschnitt mehrfach
- Verwendet man, um gewisse Codeabschnitte mehrfach ausgeben zu können
- Es gibt verschiedene Schleifenarten:
  - Eine die läuft, bis ein Vergleich `false` ergibt
  - Zum Durchlaufen aller Werte eines Arrays
  - Zum Durchzählen eines bestimmten Wertebereichs

# Was passiert?

```php
<?php
$zahl = 3;
while ($zahl <= 6) {
    $zahl++;
}

echo $zahl;

?>
```

# Schleifen: while

- Die **`while`**-Schleife läuft so lange, bis eine Bedingung (ein Vergleich) einmal **`false`** ergibt
- ```
  $zahl = 3;
  while ($zahl <= 6) {
      $zahl += 1;
  }
  echo $zahl;  // gibt 7 aus
  ```
- Ist die Zahl zu Beginn >= 10, wird der Skriptteil innerhalb der **`while`**-Schleife nicht ausgeführt
- Vorsicht vor Endlos-Schleifen!
  ```

# Beispiel: while

- **`while`**-Schleife die 1 bis 10 ausgibt

# Schleifen: do while

- Bei der do **while**-Schleife wird immer mindestens ein Schleifendurchlauf ausgeführt
- ```
  $zahl = 15;
  do {
      $zahl += 1;
  } while ($zahl <= 9);
  echo $zahl;  // gibt 16 aus
  ```
- Vorsicht vor Endlos-Schleifen!
  ```

# Was passiert?

```php
<?php
$staedte = array(
      ”Salzburg”, ”Wien”, ”Linz”, ”Bregenz”
);
foreach ($staedte as $stadt) {
     echo $stadt . ” ”;
}

?>
```

# Schleifen: foreach

- **`foreach`**-Schleifen finden Verwendung wenn man die Werte eines Arrays durchlaufen möchte
- Dabei werden die Einzelnen Array-Werte in eine Separate Variable gespeichert
- Zugehörige Indizes können auch mit durchlaufen werden
  - ```
    foreach ($staedte as $index => $stadt){
        echo $stadt . " ";
    }
    ```
- Kann nicht endlos sein
    ```

# Beispiel: foreach

- **`foreach`**-Schleife verwenden, welche die Werte eines Arrays ausgibt

# Schleifen: for

- **for**-Schleifen sind vereinfachte Formen von while-Schleifen, wenn ein Wertebereich durchgezählt werden soll
- Besteht aus Startwert, Stop-Bedingung und einer de- / inkrementierung
  - ```
    for ($i=1; $i <= 9; $i++ ){
        echo $i . " ";
    }
    ```
    ```

# Steuerung von Schleifen

- Will man eine Schleife während des Durchlaufs abbrechen, kann man **`break`** verwenden
- Um den aktuellen Durchlauf zu überspringen und mit dem Nächsten zu beginnen ist **`continue`** zu verwenden
- 
```php
for ($i = 1; $i < 50; $i++ ){
    if ($i == 3) continue; // 3 überspringen
    if ($i >= 10) break; // nach 10 aufhören
    echo $i;
}
```

# Beispiel: For-Schleife

- 1 bis 10 in einer HTML-Tabelle darstellen
- 1x1 in der Tabelle darstellen
- Alle durch 7 teilbare Zahlen ausblenden
- Die 6.Zeile überspringen

# E-Mails und Dateien

- Eine E-Mail senden:
  - `mail("rc@wifi.at", "Betreff", "E-Mail Inhalt");`
- Einen String in eine Datei schreiben:
  - ```
    $text = "Text für die Datei";
    file_put_contents("daten/text.txt", $text);
    ```
- Eine Datei auslesen:
  - ```
    $inhalt = file_get_contents("daten/text.txt");
    echo $inhalt;
    ```

# Includes

- Auslagerung von Skriptteilen in eine eigene Datei
- Sinnvoll wenn Inhalte einer Datei in mehreren anderen PHP-Dateien benötigt werden
- Vermeidet das Kopieren von Code
  - Nachteil / Vorteil:
    Ein Fehler tritt an mehreren Stellen auf, muss allerdings nur an einer Stelle behoben werden

# Includes

- Beide Möglichkeiten binden den Inhalt einer Datei an der aktuellen Stelle ein
- Unterschiede existieren in der Fehlerbehandlung bei nicht existierender Datei
  - **`include`** `"datei.php";`
    Existiert sie nicht, wird ein Fehler ausgegeben, jedoch mit dem restlichen Code fortgefahren.
  - **`require`** `"datei.php";`
    Existiert sie nicht, wird ein Fehler ausgegeben und sofort abgebrochen.

# include_once / require_once

- Selbe Unterschiede in der Fehlerbehandlung
- Wurde die geforderte Datei bereits früher einmal eingebunden, wir der Aufruf ignoriert
  - `include_once "datei.php";`
  - `require_once "datei.php";`

# Funktionen: RegExp

- Reguläre Ausdrücke werden verwendet, um komplexe Suchmuster auf Strings anwenden zu können
- Es wird auf ein gewisses Format des Textes überprüft
- Beispiele:
  - Validierung eines Datums
  - Überprüfung auf gültige Zeichen eines Usernamens
  - Validierung einer E-Mail Adresse

# Funktionen: RegExp

- Bespiel einer Datumsvalidierung
  - ```
    $datum = "24.12.2022";
    if (preg_match(
          "/^[0-9]+\.[0-9]+\.20[0-9]+$/", $datum
    )) {
          echo "Datum ist korrekt";
    }
    ```
    ```

# Beispiel: RegExp

- Einen Benutzernamen in einer Variable merken
- Prüfen, ob darin nur Buchstaben, Zahlen und Punkte vorkommen
- Bei ungültigen Zeichen eine Fehlermeldung anzeigen

# Programmsteuerung

- Wenn ein Skript sofort abgebrochen werden soll, können folgende Funktionen verwendet werden
  - `exit ([$nachricht]);`
  - `die ([$nachricht]);`

# Superglobale Variablen

- $_GET
  - Variablen, die per Adresszeile vom Browser mitgeschickt werden können
  - Üblicherweise in Links platziert
  - z.B.: index.php**?**anzeige=5**&**name=Max
  - **Achtung!** Diesen Werten grundsätzlich misstrauen und Inhalte überprüfen – sie können vom Benutzer manipuliert werden

# Superglobale Variablen

- $_POST
  - Variablen, die im Hintergrund eines Aufrufes vom Browser mitgeschickt werden
  - Üblicherweise verwendet in Formularen
  - Enthalten Eingaben des Benutzers
  - **Achtung!** Diesen Werten grundsätzlich misstrauen und Inhalte überprüfen – sie können vom Benutzer manipuliert werden

# Superglobale Variablen

- $_SESSION
  - Werte darin werden einem Benutzer für die Dauer seiner Sitzung zugeordnet
  - Die Zuordnung wird mit einem Cookie gewährleistet
  - Benötigt vor Verwendung `session_start();`
  - Kann wie ein normales Array verwendet werden
  - Hat im Normalfall einen Ablaufzeit (siehe php.ini)

# Superglobale Variablen

- $_SERVER
  - Informationen zur Client/Server/Ausführungs-Umgebung
  - Beispiele sind:
    - REMOTE_ADDR: IP-Adresse des Clients
    - HTTP_REFERER: Wenn vorhanden, die Herkunfts-URL (nicht sicher)
    - HTTP_USER_AGENT: Kennung von Browser / Betriebssystem des Clients (nicht sicher)
    - HHTP_HOST und REQUEST_URI: Vom Browser angefragte Domain und Pfad zur Datei