Web 2.0 Technologien 2

Kapitel 2:

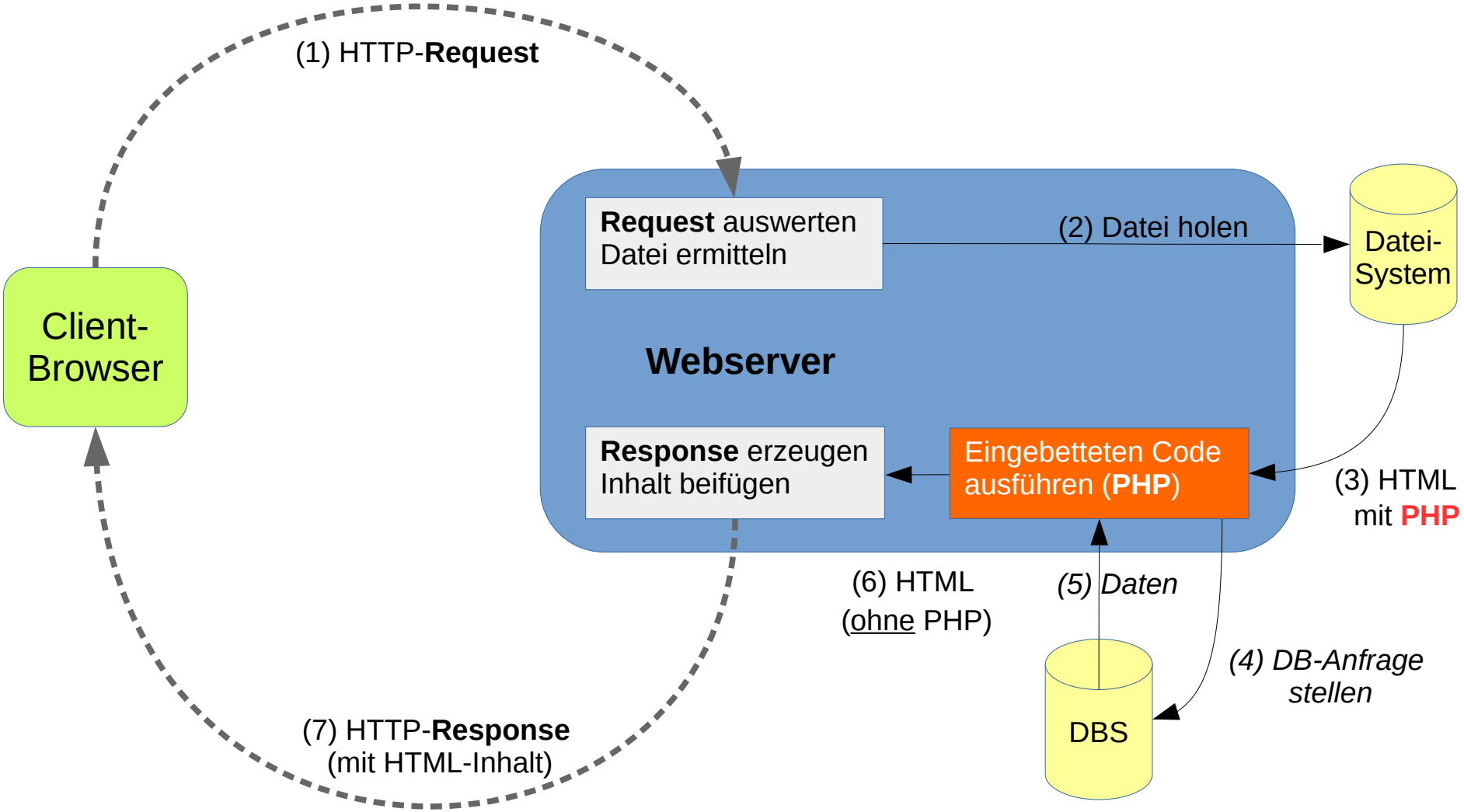
Serverseitige Techniken: PHP

Serverseitige Techniken

- **Nächste Schritte: Serverseitige Techniken**
  - **Statische Inhalte** ausliefern
    - **Webserver** z.B. **Apache**, nginx, IIS, ...
  - **Dynamische Inhalte** erzeugen, komplexe Anfragen verarbeiten
    - Serverseitige **Programmierung** z.B. **PHP**, ASP, JSP, .NET, Node.js, …
  - **Datenmanagement** (Daten speichern, abrufen, verknüpfen)
    - **Datenbank**-Schnittstelle z.B. zu **MySQL**, Postgres, DB2, ...
- **Verbreitete Standard-Lösung: LAMP**
  - Betriebssystem **Linux** (oder Windows → XAMPP)
  - Webserver **Apache**, Datenbank **MySQL**, Sprache **PHP**

Webserver: intern generierte Webseite

- **Abruf einer dynamisch erzeugten Webseite**



PHP-Grundlagen

- **PHP = „PHP: Hypertext Preprocessor“**
- **Idee:**
  - *Man fügt in eine fertige HTML-Seite genau dort Code ein, wo ein berechneter Inhalt landen werden soll.*
  - *Beispiel*:

```
<body> 1 + 2 = <?php  echo 1+2; ?> </body>
```

    - blau ist hier HTML,
    - rot ist PHP-Code,
    - **lila+fett** ist die syntaktische Klammerung des PHP-Codes
    - Lädt man die Webseite, erhält man die Ausgabe

      1 + 2 = 3

    - Genauer: Der vom Server gelieferte HTML-Text lautet hier

```
<body> 1 + 2 = 3 </body>
```

PHP-Grundlagen

- **Idee**
  - Der PHP-Code wird also auf dem Server ausgeführt und durch die Ausgabe *(echo / print)* des Codes ersetzt.
    - Außerhalb des Webservers ist kein PHP-Quellcode mehr zu finden.
  - Der resultierende Text darf alles enthalten was HTML/CSS/... kann
    - also auch **HTML**-Tags, **CSS**-Styledefinitionen, **Javascript**, ...
  - Der PHP-Code muss nicht vorab übersetzt werden.
    - Er wird bei jedem Aufruf durch den Webserver direkt ausgeführt.
- **Vorteil:**
  - Sehr effiziente Software-Entwicklung (kurze Zyklen)
  - Leichter Einstieg, schnelle Erfolge
- **Nachteile:**
  - fehleranfällig, u.a. keine statische Typ-Prüfung

PHP-Grundlagen

- **PHP ist leicht zu lernen**
  - Syntax ähnelt C++ / Java / Javascript
  - Etwas ungewohnt: Variablennamen fangen mit „$“ an.

- **Beispiel: Von 1 bis 10 zählen**

HTML

```
<h1>PHP-Schleife</h1>
```

PHP

```
<?php
    $limit = 10;
    $i = 1;
    while ($i <= $limit) {
      echo "Loop number " . $i . "<br> \n";
      $i = $i + 1;
    }
?>
```

```
<h1>PHP-Schleife</h1>

Loop number 1 <br>
Loop number 2 <br>
Loop number 3 <br>
Loop number 4 <br>
Loop number 5 <br>
Loop number 6 <br>
Loop number 7 <br>
Loop number 8 <br>
Loop number 9 <br>
Loop number 10 <br>
```

Die Ausgaben in PHP (echo) ersetzen später den PHP-Code

PHP-Grundlagen

- **PHP und HTML können stark verzahnt werden**
  - Wird von PHP auf HTML zurück geschaltet, so wirkt das, als ob der HTML-Code dort ausgegeben würde. Das funktioniert auch innerhalb von Kontrollstrukturen!
- **Beispiel: Von 1 bis 10 zählen**

HTML
```
<h1>PHP-Schleife</h1>
```
PHP
```
<?php
    $limit = 10;
    $i = 1;
    while ($i <= $limit) {
?>
```
HTML
```
 Loop number <?php echo $i; ?> <br>
```
PHP
```
<?php
        $i = $i + 1;
    }
?>
```

Trick: Statement-Block („{“) noch offen bei „?>“

```
<h1>PHP-Schleife</h1>

Loop number 1 <br>
Loop number 2 <br>
Loop number 3 <br>
Loop number 4 <br>
Loop number 5 <br>
Loop number 6 <br>
Loop number 7 <br>
Loop number 8 <br>
Loop number 9 <br>
Loop number 10 <br>
```

Die (HTML-) Zeile wird in der (PHP-) while-Schleife 10 mal ausgegeben.

PHP-Grundlagen

- **PHP-Sprachreferenz**
  - **php.net** *(offizielle Referenz)*
    - http://php.net/manual/de/langref.php
    - Sehr detailliert (vollständige Sprachbeschreibung)
- **PHP-Tutorials**
  - php.net:
    - http://de2.php.net/manual/de/
    - Einstieg: http://de2.php.net/manual/de/intro-whatis.php
    - Leider sehr kurz
  - w3schools.com:
    - http://www.w3schools.com/php/
    - Hinweis: Die englische Fassung hat weniger Fehler
  - **Quakenet/#php Tutorial**
    - http://unix.oppserver.net/php-tut/
    - Sehr ausführlich und gut gemacht!

PHP-Grundlagen: Ausführung

- Standard: **PHP-Scripte durch Webserver ausführen**
  - PHP-Scripte werden auf dem Webserver als Dateien abgelegt.
    - Sie enthalten verzahnt HTML-Quelltext und eingebettete PHP-Scripte
  - Der Webserver erkennt PHP-Dateien an der Datei-Namensendung „`.php`“ und behandelt sie entsprechend
    - Der Webserver muss dazu ggf. konfiguriert bzw. erweitert werden (Z.B. Apache: mod-php, z.B. aus Debian-Paket libapache2-mod-php)
      - Der Webserver kann auch konfiguriert werden, z.B. alle HTML-Dateien als PHP-Dateien zu behandeln.
  - Rufen wir die Datei über den Webserver ab, werden …
    - die PHP-Scripte ausgeführt und durch ihre Ausgabe ersetzt
      - Es ist von außen nicht mehr zu erkennen, welche HTML-Teile aus PHP-Code stammen.
    - die resultierende HTML-Datei wie gewohnt an den aufrufenden Browser geschickt und dort dargestellt. (→ Inhalt von W2T-1)
      - Das erzeugte HTML-Dokument muss als Ganzes syntaktisch korrekt sein, soll den gewünschten Inhalt (→ Elementbaum) haben und ggf. im Quelltext gut formatiert sein.

PHP-Grundlagen: Ausführung

- **PHP-Scripte durch Webserver ausführen** (Beispiel)
  - Beispiel: (Übungs-Testserver, hier *scilab-0100.cs.uni-kl.de*)
    - Datei test.php unter `~/htdocs/` ablegen:

```
<!DOCTYPE html>
<html>
 <body>
   Hallo <?php echo "John\n<b>Doe</b>"; ?>!
 </body>
</html>
```

    - Aufruf der URL http://scilab-0100.cs.uni-kl.de/test.php im Browser
    - Anzeige im Browser:

Hallo John **Doe**!

    - Quelltext-Anzeige im Browser (Ctrl-U):

```
<!DOCTYPE html>
<html>
 <body>
   Hallo John
<b>Doe</b>!
   </body>
</html>
```

"\n" erzeugt **Zeilenumbruch**

Übungsfrage: *Warum sieht man den Umbruch nicht im Browser?*

PHP-Grundlagen: Ausführung

- Zum Testen: **PHP-Scripte manuell ausführen**
  - Das Ausführen der PHP-Anteile einer PHP-Datei kann auch manuell erfolgen.
  - Dazu rufen wir den php-Interpreter mit dem Kommando „`php -f` *`Dateiname`*“ auf
    - Datei test.php:

```
<body>
  Hallo <?php echo "John\n<b>Doe</b>"; ?>!
</body>
```

    - Aufruf von php:

```
php -f test.php
```

    - Ausgabe:

```
<body>
  Hallo John
<b>Doe</b>!
</body>
```

  - Die Kommandozeilenversion von PHP muss dazu ggf. installiert werden (Z.B. Debian-Paket php-cli)

PHP-Grundlagen: Ausführung

- Praxistipp: **PHP als lokale Scriptsprache**
  - Man kann PHP auch als Scriptsprache benutzen, um Kommandozeilen-Tools zu realisieren
    - Das ist z.B. nützlich, um kleine Datenbank-Werkzeuge zu bauen.
  - Beispiel:
    - Datei „myscript.php“ anlegen:

```
#!/usr/bin/php -f
<?php
    echo "Script-Demo -- übergebener Parameter:\n";
    print_r($argv);
?>
```

Sorgt für den Aufruf von “/usr/bin/php -f myscript.php“. Zeile wird nicht ausgegeben.

    - Datei als Ausführbar kennzeichnen:

```
chmod u+x myscript.php
```

    - Script aufrufen:

```
./myscript.php xxx yyy
```

```
Script-Demo -- übergebener Parameter:
Array
(
    [0] => ./php-script.php
    [1] => xxx
    [2] => yyy
)
```

PHP-Grundlagen

- **PHP-Kommentare**
  - Einzeilige Kommentare: Ab „//“ oder „#“ bis zum Ende der Zeile
  - Mehrzeilige Kommentare: Zwischen „/*“ und „*/“

```
<!DOCTYPE html>
<html>
<body>
<?php
  // A single line comment    echo "1";
  #  A single line comment    echo "2";
  /* A multi line             echo "3";
       comment */             echo "4";
?>
</body>
</html>
```

- Übungsfrage: Welche der echo-Anweisungen werden ausgegeben?
- PHP-Scripte werden auch innerhalb von HTML-Kommentaren ausgeführt.

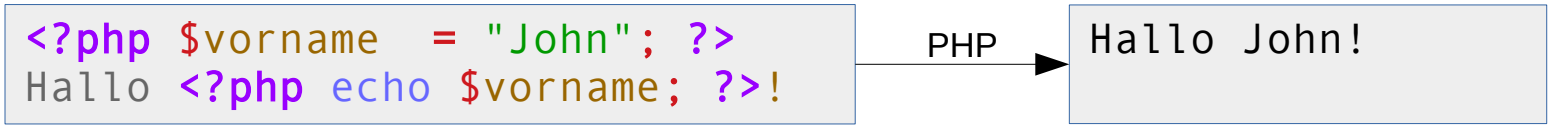
```
<!-- HTML Kommentar mit <?php echo "PHP"; ?> -->
```

PHP →

```
<!-- HTML Kommentar mit PHP -->
```

PHP-Grundlagen

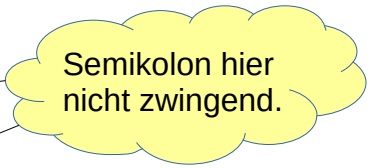
- **Variablen**
  - Variablennamen beginnen immer mit einem **$**-Zeichen
    - Vergisst man das, wird eine Konstante mit dem Namen (erfolglos?) gesucht
    - Variablennamen sind Case-sensitive
  - Variablen müssen nicht deklariert werden
    - **Variablen haben keinen Typ** (aber ihr jeweils aktueller **Wert** hat einen Typ!)
    - Man kann Variablen einfach verwenden, z.B. einen Wert zuweisen
- **Zuweisung von Werten zu Variablen**
  - Variablen, die noch keine Zuweisung erhalten haben, haben den Wert NULL
  - Einen Wert zuweisen kann man mit dem Zuweisung-Operator **=**



PHP-Grundlagen

- **Sequenzen von Anweisungen**
  - Anweisungen werden mit Semikolon („;“) voneinander getrennt
    - Kommt dahinter keine Anweisung, kann (aber muss kein) Semikolon stehen

```php
<?php
    $vorname  = "John";
    $nachname = "Doe";
?>
<body>
Hallo <?php echo $vorname; ?>!
</body>
```



- **Groß-Klein-Schreibung**
  - Schlüsselworte (z.B. „if“), Funktionsnamen und Konstanten (z.B. „true“) sind nicht Case-Sensitive
    - Sie können gemischt groß oder klein geschrieben werden
  - Variablennamen sind aber Case-Sensitiv (s.o.)
    - $a und $A sind zwei verschiedene Variablen!

PHP-Grundlagen

- **Kontrollstrukturen**
  - PHP kennt typische Kontrollstrukturen wie in anderen (C-/Java-artigen) imperativen Programmiersprachen
    - `if (audruck) anweisung`
    - `if (audruck) anweisung else anweisung`
    - `if (audruck) anweisung elseif (ausdruck) anweisung`
    - `while (audruck) anweisung`
    - `foreach (array_expression as $value) anweisung`
    - `foreach (array_expression as $key => $value) anweisung`
    - `for (expr1; expr2; expr3) anweisung`
    - `...`
  - Beispiel
    - ```
      foreach (array(3,5,7) as $k => $v) {
         echo "<li>Wert für $k ist $v\n";
      }
      ```
  - https://www.php.net/manual/de/language.control-structures.php
      ```

PHP-Grundlagen

- **Mehrere Arten von String-Literalen**
  - Einfache Anführungszeichen: `'Text'`
    - Inhalt wird 1:1 übernommen
  - Doppelte Anführungszeichen: `"Text"`
    - Bestimmte Zeichenketten werden interpretiert und durch Sonderzeichen ersetzt
      - `"\n"` → LF (Linefeed = Neue Zeile),
        `"\r"` → CR (Carriage Return),
        `"\\"` → `'\'`,
        `"\$"` → `'$'`,
        `"\""` → `'"'`,
        ...
    - Es gibt noch weitere Sequenzen:
      https://www.php.net/manual/de/language.types.string.php#language.types.string.syntax.double
    - Variablennamen (`$name` und `{$name}`) werden durch ihren Wert ersetzt

Der Vollständigkeit halber:
Es gibt noch 2 weitere Arten,
Heredoc und *Nowdoc*

Nützlich wenn nach dem Var.-Namen z.B. ein Buchstabe steht. (Warum?)

Der Mechanismus ist sehr komplex, z.B. ist zur Ausgabe eines Array-Wertes auch so etwas möglich:
`"Ergebnis: $arr[0]"`

```php
<?php
    $vorname  = "John";
    $nachname = "Doe";
    echo "Hallo $vorname {$nachname}!"
?>
```

PHP

```
Hallo John Doe!
```

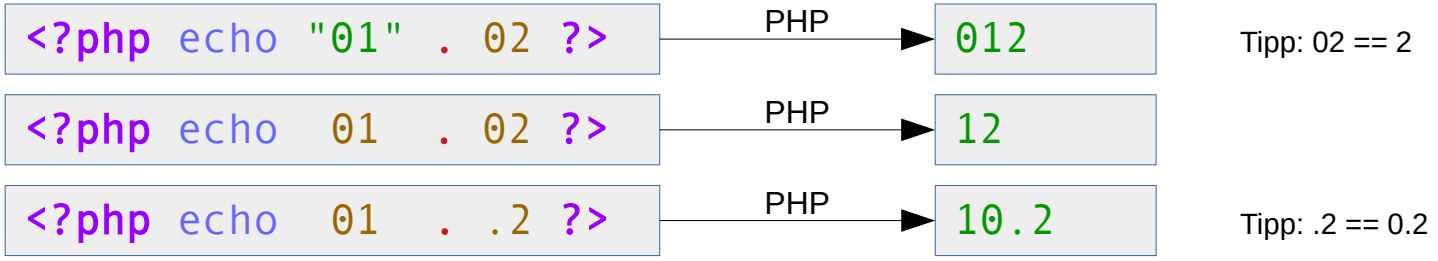
PHP-Grundlagen

- **String-Werte können mit „ . “ verkettet werden**
  - Der .-Operator verkettet zwei Strings
    - Das folgende Script erzeugt die Ausgabe „Hallo John Doe!“

```
<?php
    $vorname  = "John";
    $nachname = "Doe";
?>
Hallo <?php echo $vorname . " " . $nachname ?>!
```

- **Der .-Operator kann aber nur Strings verarbeiten**
  - Andere Typen werden ggf. implizit in Strings umgewandelt.
  - Das kann teilweise verblüffende Effekte haben.

Die Ergebnisse haben jeweils den Typ String



- **Typkonvertierungen** (Beispiel String → Integer)
  - Allgemein werden Typen in PHP bei Bedarf flexibel konvertiert
    - Bsp.: Numerische Operatoren (wie +, -, *) brauchen Zahlen als Parameter
      - Übergibt man z.B. einen String, wird dieser in eine Zahl konvertiert

```
<?php echo  "1" +  "2" ?>   PHP → 3
<?php echo "01" +  "2" ?>   PHP → 3
<?php echo  10  + ".2" ?>   PHP → 10.2
```

Die Ergebnisse haben hier jeweils den Typ integer oder float

      - Die Konvertierung String nach Zahl endet, sobald der Rest nicht mehr als Zahl interpretiert werden kann

```
<?php echo "1x" + "2y" ?>   PHP → 3
```

      - Der leere String wird bei Umwandlung in eine Zahl als 0 interpretiert

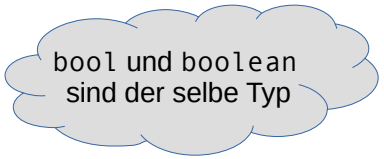
```
<?php echo  10 + "" ?>   PHP → 10
<?php echo "x" + "y" ?>   PHP → 0
```

PHP-Grundlagen

- **Datentypen**
  - Skalare Typen:
    - boolean Werte: true, false
    - integer z.B. 123, -5
    - float z.B. 1.0, 5.7e3 == 5700.0, 1e-3 == 0.001
    - string z.B. "Hallo"
  - Strukturierte Typen
    - array z.B. array(2, 3, 5, 7)
      z.B. array("Hund", "Katze", 123)
      z.B. array("matnr"=>12345, "name"=>"Müller")
    - Object, Callable, Iterable
  - Spezielle Typen
    - Ressource
    - NULL Wert: NULL
  - Siehe https://www.php.net/manual/de/language.types.php

Zahlen-Literale:
führende 0
bei Oktalzahlen,
0x bei Hex-Zahlen

- **Explizite Typkonvertierungen**
  - Typen können auch explizit konvertiert werden
    - Dazu wird der Typname in Klammern vor den Ausdruck gestellt
    - z.B. `(bool) 1` (Ergebnis: true)
    - z.B. `(boolean) 0` (Ergebnis: false)



  - Es gibt folgende Konvertierungen
    - (int), (integer) - cast to integer
    - (bool), (boolean) - cast to boolean
    - (float), (double), (real) - cast to float
    - (string) - cast to string
    - (array) - cast to array
    - (object) - cast to object
    - (unset) - cast to NULL

PHP-Grundlagen

- **Typkonvertierungen**
  - Einige Typkonvertierungswerte sind besonders wichtig:
  - Folgende Werte werden als boolean false interpretiert
    - boolean false selbst
    - integer 0
    - float 0.0
    - Der leere string, und der string "0"
    - Ein array ohne Elemente
    - Der spezielle Typ NULL (also auch nicht initialisierte Variablen)

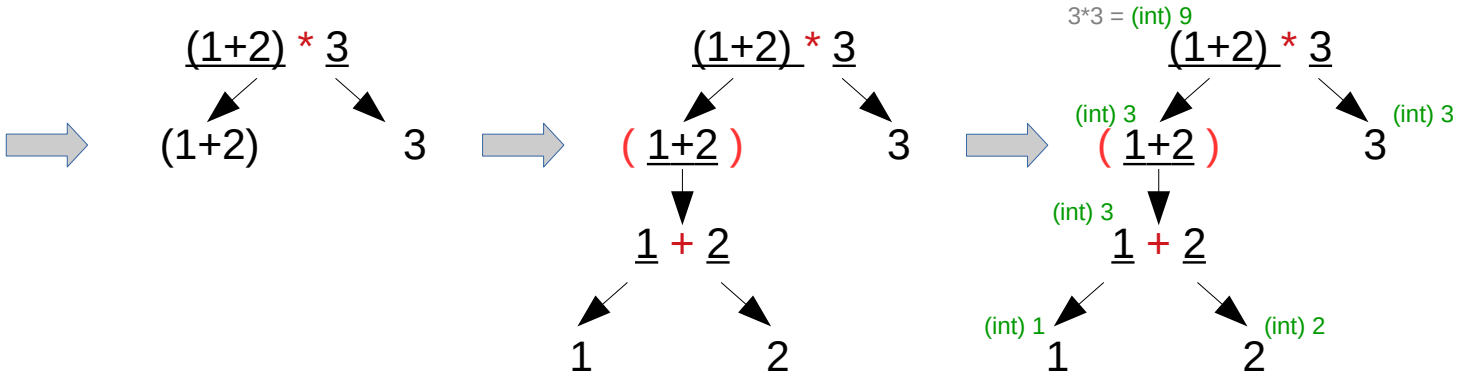
    Wir können alle diese Type z.B. direkt als if-Bedingung nutzen
    - z.B.: `$list = array(1,2,3); /* … */ if ($list) { … }`
  - Diese Werte werden bei der Konvertierung zu integer entsprechend auch als 0 interpretiert.
    - Beachten Sie die Anomalie bei den beiden String-Werten!
  - https://www.php.net/manual/de/language.types.type-juggling.php

- **Ausdrücke (Expression): Zerlegung**
  - Durch Operatoren (z.B. +) werden Ausdrücke (z.B. 1+2) gebildet
  - **Ausdruck-Analyse**
    - Ausdrücke werden rekursiv (von oben nach unten) in Teilausdrücke zerlegt

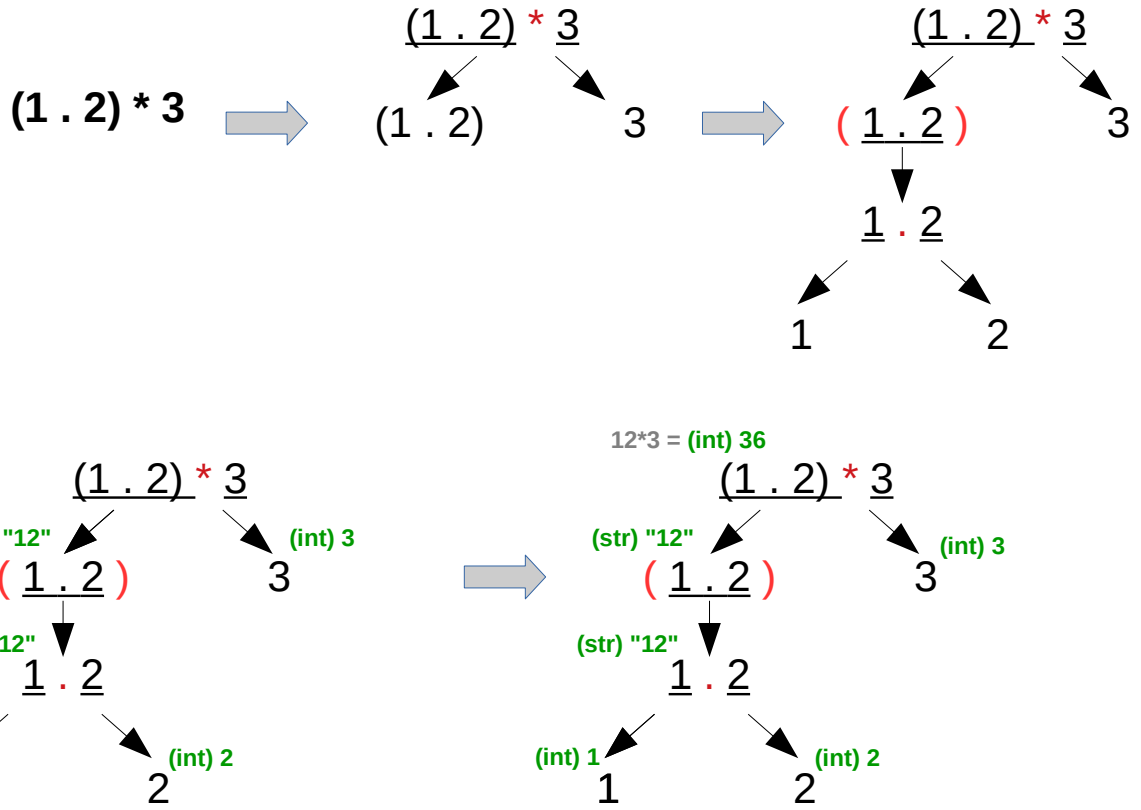
      Ziel: Baumstruktur zur Ausdrucksanalyse
      - Knoten sind Ausdrücke, die nach unten zerlegt werden
      - Die Blätter sind schließlich nur noch atomare Ausdrücke (nicht mehr zerlegbar)
    - die Werte und Typen der atomaren Ausdrücke (z.B. Variablen) bestimmt
    - schrittweise (von unten nach oben) Werte u. Typen der Teilbäume bestimmt

      Ziel: Wert und Typ des gesamten Baums bestimmen.
  - Beispiel

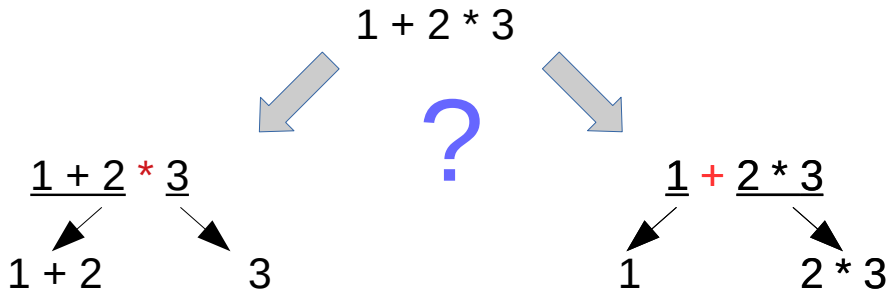
(1+2)*3



- **Ausdrücke (Expression): Typkonvertierungen**
  - Bei der Berechnung der Ergebnisse muss man auf implizite Typkonvertierungen achten
  - Beispiel



- **Ausdrücke (Expression): Präzedenzen**
  - Die Zerlegung scheint manchmal nicht eindeutig zu sein



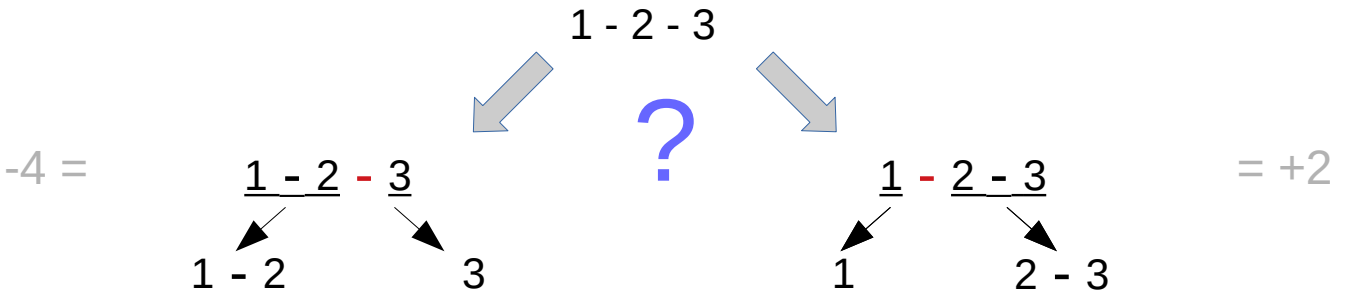
  - Die Operatoren haben dazu eine Rangfolge (**Präzedenz**)
    - https://www.php.net/manual/de/language.operators.precedence.php
    - In der Tabelle höher stehende Werte „binden stärker“
      → die schwächeren Operatoren werden immer zuerst zerlegt

Die Zeile „+ - .“ ist auf php.net leider um eine Spalte „verrutscht“.

  - Ergebnis im obigen Beispiel:
    - * bindet stärker als +

1 + 2 * 3 → 1 + 2 * 3 | 1 | 2 * 3

- **Ausdrücke (Expression): Assoziativität**
  - Die Zerlegung scheint *weiterhin* manchmal nicht eindeutig zu sein



  - Der Vorrang Operatoren gleicher Präzedenz (gleiche Zeile) wird über die **Assoziativität** (erste Spalte) geregelt
    - https://www.php.net/manual/de/language.operators.precedence.php
    - links-assoziativ bedeutet der linke Operator „bindet stärker“
  - Ergebnis im obigen Beispiel:
    - **-** (minus) ist links-assioziativ
      → linker Operator bindet stärker