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

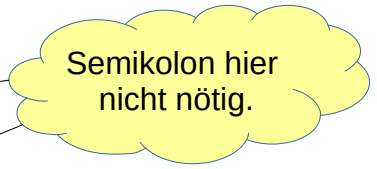
PHP →

```
Hallo John!
```

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. in Strings umgewandelt.
  - Das kann teilweise verblüffende Effekte haben.

Die Ergebnisse haben jeweils den Typ String

```
<?php echo "01" . 02 ?>   PHP →  012
```

Tipp: 02 == 2

```
<?php echo  01  . 02 ?>   PHP →  12
```

```
<?php echo  01  . .2 ?>   PHP →  10.2
```

Tipp: .2 == 0.2

PHP-Grundlagen

- **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

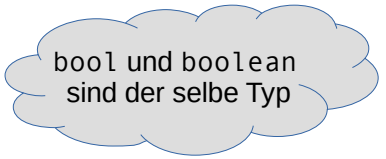
```
<?php echo  10 + "" ?>   PHP →  10
<?php echo "x" + "y" ?>   PHP →  0
```

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

PHP-Grundlagen

- **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

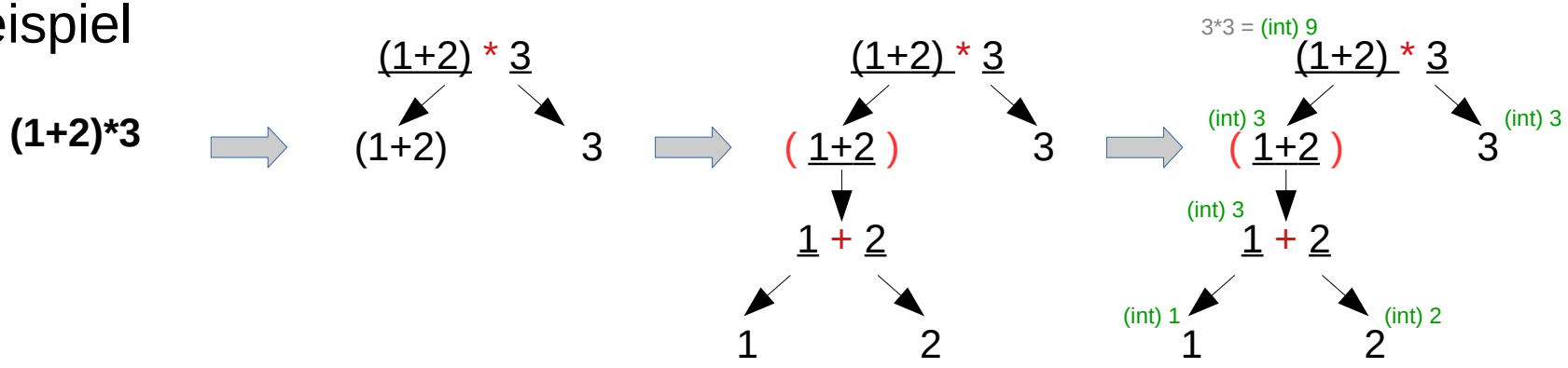
- **Typkonvertierungen**
  - Einige Typkonvertierungswerte sind besonders wichtig:
  - Folgende Werte werden als boolean false interpretiert
    - boolean false selbst
    - integer 0
    - float 0.0
    - Der leere string, und der string "0"
    - Ein array ohne Elemente
    - Der spezielle Typ NULL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

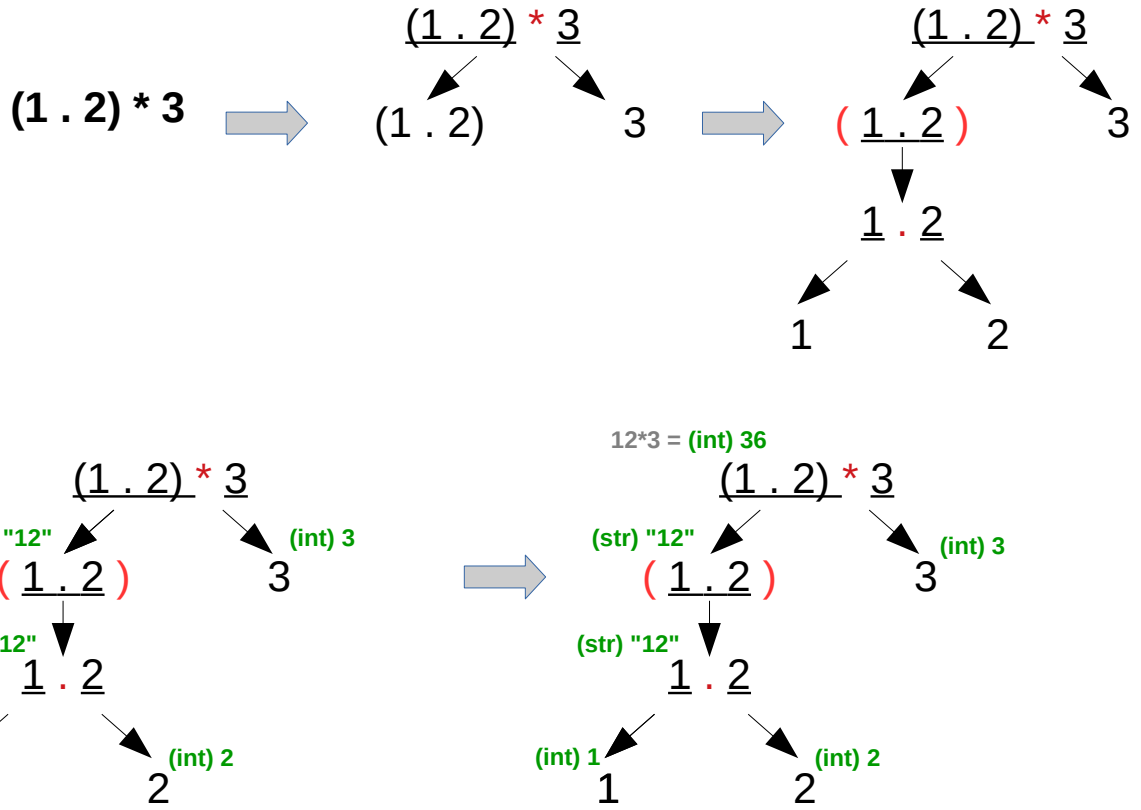
- **Ausdrücke (Expression): Zerlegung**
  - Durch Operatoren (z.B. +) werden Ausdrücke (z.B. 1+2) gebildet
  - **Ausdruck-Analyse**
    - Ausdrücke werden rekursiv (von oben nach unten) in Teilausdrücke zerlegt

      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

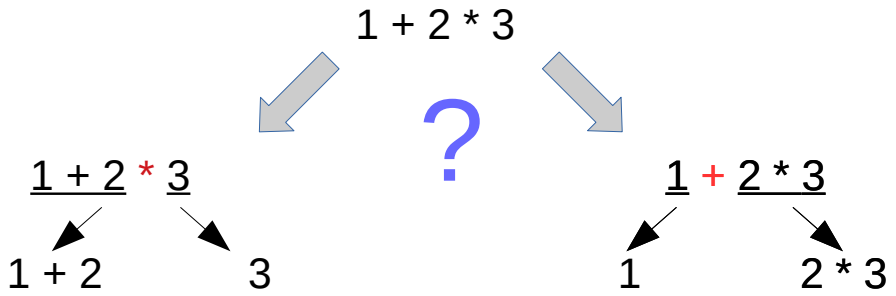
      Ziel: Wert und Typ des gesamten Baums bestimmen.
  - Beispiel



- **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



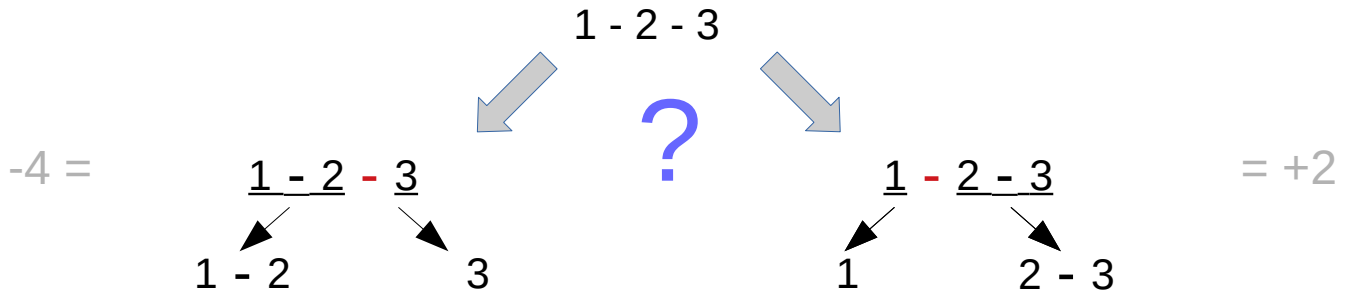
  - Die Operatoren haben dazu eine Rangfolge (**Präzedenz**)
    - https://www.php.net/manual/de/language.operators.precedence.php
    - In der Tabelle höher stehende Werte „binden stärker“
      → die schwächeren Operatoren werden immer zuerst zerlegt

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