Westfälische Wilhelms-Universität Münster

Erstellen dynamischer Webseiten mit PHP

WS 2016/2017

wissen.leben
WWU Münster

Dozent: Patrick Förster, Michael Hasseler

Übersicht der heutigen Inhalte

6. Vorlesung: Funktionen
    - Motivation
    - Aufruf und Definition
    - Gültigkeitsbereich von Variablen
    - Übergabe per „call by value“ bzw. „call by reference“
    - Anonyme Funktionen

Motivation (b01.php)

- Folgender Anweisungsblock entfernt aus einem Array alle leeren Einträge:

```
$meinArray = …;
foreach($meinArray as $key => $value) {
    if (empty($value)) {
        unset ($meinArray[$key]);
    }
}
```

- Es werden keinerlei Annahmen über die Struktur von `&myarray` getroffen
  - ➔ Dieser Block kann auf jeden beliebigen Array angewandt werden
  - ➔ Hohes Potential für Wiederverwendbarkeit dieses Blockes
- Zur Wiederverwendung muss der Code dupliziert werden
- Die Variable, die den zu verarbeitenden Array hält muss „`myarray`“ heißen
- Oder der Name der aktuellen Situation angepasst werden

Motivation (Beispiel)

```php
// Stelle 1 im Code:
$meinArray = …;
foreach($meinArray as $key => $value) {
    if (empty($value)) {
        unset ($meinArray[$key]);
    }
}
```

```php
// Stelle 2
$anderer = …;
foreach($anderer as $key => $value) {
    if (empty($value)) {
        unset ($anderer[$key]);
    }
}
```

Motivation (Schlussfolgerung)

- Duplikation birgt Gefahr:
  - Feste Bindung der Variablennamen begünstig Flüchtigkeitsfehler
  - Was wenn der verwendete Code sich als fehlerhaft herausstellt?
    - Alle Duplikate müssen angepasst werden
  - Erweiterungen müssten ebenfalls überall nachgepflegt werden

➔Der Code wird durch Duplikation zudem unnötig aufgebläht

➔Verlust der Übersichtlichkeit

➔Insgesamt gilt das **DRY**-Prinzip: *Don‘t repeat yourself.*

Funktionen

• Um die DRY-Problematik zu verhindern benötigt es einer neuen Programmierstruktur:

➔ **Funktionen**

• **Wiederverwendbarkeit**: Nur einmal schreiben und testen

• **Wartbarkeit**: Fehlerkorrekturen und Änderungen können schneller vorgenommen werden

• **Lesbarkeit**: Problemstellung kann in Teilprobleme aufgeteilt werden, die dann von den jeweiligen Funktionen erfüllt werden

  • sinnvolle Aufteilung der Teilprobleme

  • sinnvolle Benennung der Funktionen

Eigene Funktion definieren

```
function nameDerFunktion([parameter_1 [,…, parameter_n]]) {
        ANWEISUNGEN;
        [return [WERT];]
}                                                    // […] = optional
```

- Der Funktionsname unterliegt den gleichen Vorschriften wie Variablen mit der Ausnahme, dass *kein* `$` vorangestellt wird
- Per Konvention beginnt eine Funktion mit kleinem Buchstaben, danach Binnenmajuskel für jedes weitere Wort
- Funktionen haben eine beliebige Anzahl an Parametern (auch `0`)
- Parameter sind nur für die Funktion gültige Variablen
- Die **`return`** Anweisung ist optional und liefert das Ergebnis der Funktion
- Aufruf/Ausführen:

```
nameDerFunktion(Wert_1, …, Wert_N);
```

Beispiel (b02.php)

- Berechne die Summe der ersten n natürlichen Zahlen:
  - Funktionsname: „aufsummieren“
  - Parameter-Anzahl: 1
  - Parameter-Name: „n“
  - Rückgabe: Die Summe

```
function aufsummieren($n) {
    $summe = 0;
    for ($i = 1; $i <= $n; $i++) {
        $summe += $i;
    }

    return $summe;
}
```

```
$summe = aufsummieren(10);
echo $summe;
➔ 55



echo aufsummieren(100)
➔ 5050
```

Funktion ohne `return` (b03.php)

- Enthält eine Funktion keine **`return`**-Anweisung, so interessieren die „Nebenwirkungen“ dieser Funktion

```
function test($errechnet, $erwartet) {
  if ($errechnet === $erwartet) {
    echo "OK: " . $errechnet . " === " . $erwartet . "<br />";
  }
  else {
    echo "FEHLER: " . $errechnet . " !== " . $erwartet . "<br />";
  }
}
```

- Diese Funktion liefert kein Ergebnis
- Nebenwirkung: Falls der Wert des Parameters `errechnet` **typ**- und **wertegleich** (`===`) dem Wert von `erwartet` ist, wird *OK:...* ansonsten *FEHLER:...* in die HTML-Ausgabe geschrieben

Optionale Parameter

- Bei der Funktionsdefinition können in der Parameterliste Standardwerte festgelegt werden
- Standardwerte dürfen bei den Parametern nur von rechts nach links gesetzt werden, wobei keine Lücken erlaubt sind

```
function umtausch($menge, $waehrung='Dollar', $zielwaehrung='Euro') {
        …
}
```

- 3 mögliche Parameteranzahlvarianten:

Aufruf	$waehrung	$zielwaehrung
umtausch(1);	'Dollar'	'Euro'
umtausch(1, 'Yen');	'Yen'	'Euro'
umtausch(1, 'Euro', 'Dollar');	'Euro'	'Dollar'

Bedingte Rückgabe (b04.php)

- Die **`return`**-Anweisung beendet die Funktion
  ➔jeglicher Code nach einem **`return`** wird nicht ausgeführt!
- Mehrfache **`return`** Anweisungen innerhalb einer Funktion aber durchaus erwünscht:
  - Rückgabe in Abhängigkeit einer Bedingung
- Beispiel: Die Funktion **`umtausch`** unterstütze: „`Dollar`“, „`Euro`“, „`Yen`“

```
if (!($waehrung==="Dollar" || $waehrung==="Euro" || $waehrung==="Yen")) {
  return -1;
}
if (!($zielwaehrung==="Dollar" …) // wie oben für $zielwaehrung

…                                 // Umrechungsvorschrift
return …                          // Rückgabe des umgerechneten Wertes
```

- Falls **`$waehrung`** oder **`$zielwaehrung`** fehlerhaft sind, wird **`-1`** zurückgegeben
- Die Aufrufende kann die **`-1`** als Fehler interpretieren und entsprechend handeln

Namensraum von Variablen

- Jede Variable in PHP hat einen **Geltungsbereich**
- Nur in diesem Bereich gilt die Variable als „definiert“

- Allgemein zwei mögliche Bereiche:
  - Außerhalb einer Funktion
  - Innerhalb einer Funktion

- Es gilt:
  - Innerhalb einer Funktion kann nicht auf Variablen zugegriffen werden, die außerhalb definiert wurden
  - vice versa

Lokale Variablen (b05.php)

```php
$ausserhalb = "Ich bin außerhalb";

function f() {
    echo "Innerhalb der Funktion";

    $innerhalb = "Ich bin innerhalb";

    echo $innerhalb;
    echo $ausserhalb;
}

echo "Außerhalb der Funktion";
echo $innerhalb;
echo $ausserhalb;

f();
```

```
Außerhalb der Funktion

Notice: Undefined variable: innerhalb
Ich bin außerhalb

Innerhalb der Funktion

Ich bin innerhalb
Notice: Undefined variable: ausserhalb
```

Globale Variablen

- Mit **`global`** können Variablen als „global“ ausgezeichnet werden
  - Sie sind inner- und außerhalb jeder Funktion gültig

```php
global $ausserhalb;
$ausserhalb = "Ich bin außerhalb";

function f() {
  $innerhalb = "Ich bin innerhalb";
  echo $innerhalb;

  global $ausserhalb;    // !
  echo $ausserhalb;
}

echo $innerhalb;
echo $ausserhalb;
f();
```

```
Notice: Undefined variable: innerhalb
Ich bin außerhalb

Ich bin innerhalb
Ich bin außerhalb
```

Statische Variablen

- Mit dem Ende der Gültigkeit ist auch der Wert der Variablen verloren
- Mit **`static`** können Werte über Funktionsaufrufe hinweg gespeichert werden

```php
function f() {
  static $anzahl_an_aufrufen = 0;
  $anzahl_an_aufrufen++;

  echo "'f' Aufruf: " . $anzahl_an_aufrufen;
}

for ($i = 0; $i < 10; $i++) {
  f();
}
```

```
'f' Aufruf: 1
'f' Aufruf: 2
'f' Aufruf: 3
'f' Aufruf: 4
'f' Aufruf: 5
'f' Aufruf: 6
'f' Aufruf: 7
'f' Aufruf: 8
'f' Aufruf: 9
'f' Aufruf: 10
```

- Die initiale Zuweisung nach **`static`** wird nur ein Mal durchgeführt!
- **`static`** ist nur innerhalb von Funktionen gültig (später auch in Klassen)

Superglobale Variablen

- PHP-eigene Variablen, die *ohne* **`global`** von überall aus zugreifbar sind
- Theoretisch überschreibbar, aber praktisch zu handhaben wie Schlüsselwörter

```
$GLOBALS
$_SERVER
$_GET
$_POST
$_FILES
$_COOKIE
$_SESSION
$_REQUEST
$_ENV
```

```
// vgl. b03.php aus Vorlesung 4
…
if (!empty($_POST['express'])) $betrag += 5;
if (!empty($_POST['versicherung'])) $betrag += 5;
```

Wertübergabe bei Parameter

1. Wertübergabe per Wert (**call by value**)
   - Wird **standardmäßig** verwendet
   - Lokale Variable wird angelegt und Übergabewert kopiert
   - Änderungen an lokaler Variable beeinflussen nicht den Originalwert

2. Wertübergabe per Referenz (**call by reference**)
   - Erkennbar am **&** vor einer Variable in der Parameterliste bei der Funktionsdefinition
   - Es wird eine Referenz auf den Originalwert übergeben und Änderungen in der Funktion beeinflussen auch den Originalwert

„Call by value“ Beispiel

- Das Eingangsbeispiel „entferne leere Werte“ als Funktion (naiver Ansatz):

```
function entferneLeereWerte($array) {
    foreach($array as $key => $value) {
        if (empty($value)) unset ($array[$key]);
    }

    ausgeben($array);
}
```

```
$zahlen= array("NULL", "EINS", "", "ZWEI", "DREI", "VIER", "FÜNF", "");
entferneLeereWerte($zahlen);
ausgeben($zahlen);

➔ 0: NULL, 1: EINS, 3: ZWEI, 4: DREI, 5: VIER, 6: FÜNF
➔ 0: NULL, 1: EINS, 2: "", 3: ZWEI, 4: DREI, 5: VIER, 6: FÜNF, 7: ""
```

- Aufruf „entferneLeereWerte“ hat keinen Einfluss auf den Array „zahlen“

„Call by reference“ Beispiel (b07.php)

- Der Parameter **`$array`** wurde als „call by value“ definiert
- D.h. bei Aufruf der Funktion wird der übergebene Wert kopiert
  - Hier der **`$zahlen`** Array
  - Die Funktion arbeitet auf dieser Kopie, **`$zahlen`** bleibt unverändert

- Um **`$zahlen`** per *Referenz* zu übergeben, muss dem Parameter **`$array`** ein **`&`** vorangestellt werden

```
function entferneLeereWerte(&$array) {
    foreach($array as $key => $value) {
        if (empty($value)) unset ($array[$key]);
    }

    ausgeben($array);
}
```

„Call by reference“ Beispiel (Fortsetzung)

- Referenz: Der Wert der übergebenen Variable wird geändert
- D.h. insbesondere: Ist ein Funktionsparameter als „call by reference“ definiert, können diesem beim Aufruf nur Variablen zugewiesen werden:

```
entferneLeereWerte(array("NULL", "EINS", "", "ZWEI", "DREI", "VIER"));

➔ Fatal error: Only variables can be passed by reference in …
```

- Auch „primitive“ Variablen können per Referenz übergeben werden:

```
function f(&$var) {
    $var = 10;
}
```

```
$zahl = 0;
echo $zahl;

f($zahl);
echo $zahl;
```

```
➔ 0



➔ 10
```

Funktionen als Werte

- Neben den primitiven Datentypen und Arrays sind auch Funktionen als Werte für Variablen erlaubt
- Die Zuweisung erfolgt per Name

```
function quadrat ($x) {
    return $x * $x;
};

$f = "quadrat";
echo $f(10);
```

➔ 100

- Erst bei der Auswertung der Variablen **$f** „merkt“ der PHP Interpreter, dass diese als Funktionsaufruf benutzt wird (Klammerung)
- Jetzt wird überprüft, ob eine Funktion mit dem zugewiesenen Namen existiert und wenn ja diese aufgerufene (Ansonsten: **Fatal error**)

Anonyme Funktionen/Lambda-Funktion

- Die Zuweisung kann auch direkt geschehen
- In diesem Fall spricht man von anonymen Funktionen, da kein Name vergeben wird

```php
$f = function ($x) {
    return $x * $x;
};

echo $f(10);
```

➔ 100

- Der Variablen **f** ist nun eine Funktion zugewiesen
- Diese hat keinen Namen und ist daher nicht anders als über **$f** aufrufbar

- Funktionen können damit auch als Parameter übergeben werden
- Solche Parameter werden häufig als *Callbacks* bezeichnet

Callback Beispiel (b07.php)

```
function aufsummieren($n) {…}

function test($errechnet, $erwartet, $callback) {
    if ($callback($errechnet, $erwartet)) {
        echo "OK: " . $errechnet . " ist gleich " . $erwartet;
    }
    else
        echo "FEHLER: " . $errechnet . " ist NICHT gleich" . $erwartet;
}

$vergleicheTypWert = function ($a, $b) {return $a === $b;};
$vergleicheWert = function ($a, $b) {return $a == $b;};

test(aufsummieren(10), 55, $vergleicheTypWert);
test(aufsummieren(10), "55", $vergleicheTypWert);
test(aufsummieren(10), "55", $vergleicheWert);
```

➔ OK: 55 ist gleich 55
➔ FEHLER: 55 ist NICHT gleich 55
➔ OK: 55 ist gleich 55

Zusammenfassung

- Wie werden Funktionen definiert und aufgerufen?
- Was ist der Gültigkeitsbereich/Namensraum einer Variablen?
  - Lokal, global, superglobal
- Was sind optionale Parameter?
- Parameterübergabe per Wert oder per Referenz
- Funktionen als Wert-Elemente

Aufgaben

1. Auf Folie 18 wird die Funktion **`ausgeben`** aufgerufen, die einen beliebigen *Array* in die HTML-Ausgabe schreibt.
   Schreiben Sie diese Funktion. Nutzen Sie hierbei den Codeschnipsel a01.php.
2. Schreiben Sie die Funktion **`wendeAn`**, die einen *Array* und eine *Callbackfunktion* als Parameter erwartet. Innerhalb der Funktion soll der Callback auf jeden Wert im Array angewandt werden und das Resultat den Wert im Array ersetzen.
3. Schreiben Sie zwei Funktionen **`berechneKreisflaeche`** und **`berechneKreisumfang`**, die beide als Parameter den *Radius* haben. Eine *HTML-Seite* soll ein Eingabefeld für den Radius haben. Das korrespondierende PHP-Skript soll diese Eingabe als Pflichtfeld prüfen. Bei erfolgreicher Pflichtfeldeingabe sollen die beiden Methoden aufgerufen und deren Ergebnisse mit **`echo`** ausgegeben werden.

Anhang: ZIVinteraktiv-Abstimmung

- QR-Code zur anonymen Abstimmung:

- https://www.uni-muenster.de/ZIV/anw/zivinteraktiv/abstimmung.php?code=5804a66a