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

- Arrays
  - Motivation
  - Initialisierung
  - Zugriff
    - indiziert
    - assoziiert
  - mehrdimensional

Motivation von Arrays

- Variablen können einen Wert speichern
- Mehrere Werte müssen in mehreren Variablen abgespeichert werden
- $wert1 = 2;
- $wert2 = 3;
- $wert3 = 5;

$wert1	$wert2	$wert3
2	3	5

- Arrays ermöglichen eine zusammenhängende Abspeicherung von Werten

Array		
wert_1	*wert_2*	*wert_3*

$werte		
2	3	5

Vgl. PHP Grundlagen – Erstellung dynamischer Webseiten, RRZN, 9.Aufl., Seite 52

Array-Arten

- Jedem Wert in einem Array ist ein eindeutiger Schlüssel zugewiesen
- Über diesen Schlüssel erfolgt der Zugriff auf den Wert
- Es gibt zwei verschiedene „Arten“ von Arrays

  - **Indiziert**: Ein Schlüssel ist eine nicht negative natürlich Zahl 0, 1, 2, …
    - Die Schlüssel sind konsekutiv

  - **Assoziativ**: Ein Schlüssel ist entweder eine Zahl oder ein String („Wort“)
    - Auch negative ganze Zahlen
    - Schlüsseltypen können gemischt werden
    - Keine Reihenfolge in der Schlüsselmenge

Indizierte Arrays

- Ein Array (indiziert oder assoziativ) wird in **php** mit dem Befehl **array(…)** erzeugt.
- Bei einem indizierten Array werden die Werte komma-getrennt aufgezählt:

```
$planeten = array(‘Merkur‘, ‘Venus‘, ‘Jupiter‘);
```

- Die Schlüssel werden automatisch vergeben, angefangen bei 0:

Index	0	1	2
Wert	Merkur	Venus	Jupiter

- Der Zugriff erfolgt über den sogenannten Klammer-Operator:

```
echo $planeten[0];
```

Indizierte Arrays

- Ein Array kann jederzeit um einen neuen Eintrag erweitert werden:

```
$planeten[] =  ‘Mars‘;
```

- Der Index ergibt sich automatisch aus dem zuvor höchstem + 1

Index	0	1	2	3
Wert	Merkur	Venus	Jupiter	Mars

```
$test = ($planeten[3] == ‘Mars‘);
```

```
$test == true
```

- Per Zuweisung eines Wertes über einen bestimmten Index, kann ein Wert ersetzt werden:

```
$planeten[0] = ‘Erde‘;
```

Index	0	1	…
Wert	**Erde**	Venus	…

Indizierte Arrays (Iteration)

- Iteration über einen indizierten Array
  - Iteration ≙ „Betrachten“ eines jeden Elementes (in einem Array)
  - Die Schlüssel sind aufsteigend ab 0 geordnet ➔ for-Schleife

```
for (Initialisierung; Bedingung; Reinitialisierung) { Anweisungsblock; }
```

  - Die for-Anweisung benötigt eine Abbruch-Bedingung
  - Deskriptiv: Wenn das Ende des Arrays erreicht wurde, aufhören
  - Das Ende des Arrays ist erreicht, wenn das Element mit dem höchsten Index betrachtet wurde

```
Höchster Index == Länge(Array) -1
```

  - Die Länge eines Arrays erhält man mit dem php-Befehl **count**

```
$laenge = count($array)
```

Indizierte Arrays (Iteration)

- Beispiel:

```
$planeten = array('Merkur', 'Venus', 'Jupiter');

for ($i = 0; $i < count($planeten); $i++) {                    // 1
        echo $planeten[$i] . ', ';                             // 2
}
```

Ausgabe ➔ Merkur, Venus, Jupiter,

1) Initialisierung der Variablen $i beginnend beim ersten Index 0
   Iteration solange $i != count($array) ≙ Länge des Arrays
2) Ausgabe des Wertes im Array mit dem aktuell in $i gespeicherten Index

Assoziative Arrays

- Bei assoziativen Arrays müssen die Schlüssel explizit verteilt werden
- Initial mit **array(…)** durch Auflistung von Schlüssel => Wert-Paaren

```
$mondanzahl = array('Merkur'=>0, 'Venus'=>0, 'Erde'=>2);
```

Schlüssel	Merkur	Venus	Erde
Wert	0	0	2

- Zugriff: `$mondanzahl['Merkur']`
- Neuzuweisung: `$mondanzahl['Erde'] = 1;`
- Hinzufügen: `$mondanzahl['Mars'] = 2;`

Schlüssel	Merkur	Venus	Erde	Mars
Wert	0	0	1	2

Iteration bei assoziativen Arrays

- count(…) liefert auch auf einem assoziativen Array seine Länge (Größe):

```
echo count($mondanzahl); ➔ 4
```

- Die for-Schleife ist hier allerdings problematisch
- Die Indizes sind nicht mehr automatisch ab 0 und schon gar nicht aufsteigend definiert
- Für assoziative Arrays wird die **foreach**-Schleife benutzt:

```
foreach (Array as Schluessel => Wert) {Anweisungsblock;}
```

- *Array* muss eine Variable vom Typ Array sein
- In jedem Iterationsschritt beinhaltet *Schluessel* den aktuellen Schlüssel und *Wert* den aktuellen Wert

Iteration bei assoziativen Arrays

```
$mondanzahl = array('Merkur'=>0, 'Venus'=>0, 'Erde'=>1, 'Mars'=>2);

foreach ($mondanzahl as $schluessel => $wert) {
        echo $schluessel . ': ' . $wert . ', ';
}
```

Ausgabe ➔ Merkur: 0, Venus: 0, Erde: 1, Mars: 2,

- Die Schlüsselzuweisung bei foreach kann ausgelassen werden

```
foreach ($mondanzahl as $wert) {
        echo $wert . ', ';
}
```

Ausgabe ➔ 0, 0, 1, 2,

foreach-Schleife bei Feldern

- indiziert:

```
foreach ($feld as $wert) {
        Anweisungsblock;
}
```

Nacheinander werden alle Werte des Arrays auf die Variable zugewiesen

- assoziativ:

```
foreach ($feld as $index => $wert) {
        Anweisungsblock;
}
```

Nacheinander werden alle Paare Index/Wert des Arrays auf die beiden Variablen zugewiesen

Vgl. PHP Grundlagen – Erstellung dynamischer Webseiten, RRZN, 9.Aufl., Seite 57

Besonderheiten

- Assoziative Arrays können ganze Zahlen als Index beinhalten

```
$array = array('Schluessel 1' => 'Wert 1', 10 => 'Wert 2');
```

- Auch das Erweitern über [] ist möglich

```
$array[] = 'Wert 3';
```

- Achtung 1: Die Zuweisung ohne Angabe zählt nicht die Anzahl der Elemente, sondern sucht den höchsten nicht negativen ganzzahligen Index

Schlüssel	Schluessel 1	10	11
Wert	Wert 1	Wert 2	Wert 3

- Achtung 2:

```
$array = array(-6 => 'Wert 1');
$array[] = 'Wert 2';
```

Schlüssel	-6	0
Wert	Wert 1	Wert 2

Besonderheiten

- Das Mischen von indiziert und assoziativ ist schlecht wartbar, da nicht intuitiv lesbar
- Es gibt allerdings eine praktische Anwendung:
  - Definition eines Start-Index != 0

```
$array = array(1 => ‘eins‘, ‘zwei‘, ‘drei‘, ‘vier‘)
```

Schlüssel	1	2	3	4
Wert	eins	zwei	drei	vier

Besonderheiten

- Der Zugriff auf nicht-existierende Array Inhalte liefert immer den Spezialwert **NULL**
- Je nach Einstellung wird ein solcher Zugriff zudem mit einer „Notice“-Meldung in der HTML-Ausgabe quittiert:

Notice: Undefined offset: 100 in **….php** on line **26**

- Umwaldung von Schlüsseltypen:
  - Gleitkommzahlen (z.B 0.5) ➔ Ganzzahl (durch Abschneiden 0.5 ➔ 0)
  - true, false ➔ 1 bzw. 0
  - NULL ➔ „“ (leeres Wort/String)
  - Achtung: Wörter die ganze Zahl repräsentieren werden zu dieser Zahl!

    $array[‘3‘] = ‘Wert‘ ≙ $array[3] = ‘Wert‘

Löschen von Array-Inhalten

- Mit dem php-Befehl unset(…) können Array-Inhalte gelöscht werden

```
$mondanzahl = array('Merkur'=>0, 'Venus'=>0, 'Erde'=>1, 'Mars'=>2);
```

Schlüssel	Merkur	Venus	Erde	Mars
Wert	0	0	1	2

```
unset($mondanzahl['Merkur'])
```

Schlüssel	Venus	Erde	Mars
Wert	0	1	2

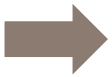
$mondanzahl['Merkur'] ➔ NULL (+ Notice)

Löschen von Array-Inhalten

- Achtung: unset funktioniert auch bei indizierten Arrays

```
$planeten = array('Merkur', 'Venus', 'Jupiter');
unset($planeten[1]);
```

Index	0	1	2
Wert	Merkur	Venus	Jupiter



Index	0	2
Wert	Merkur	Jupiter

- An den übrigen Indizes ändert dies allerdings nichts
- Die Schlüsselmenge ist damit nicht mehr konsekutiv und der Array damit assoziativ und nicht mehr indiziert

Besonderheit beim Löschen

- Das Löschen in indizierten Arrays ändert nicht den nächsten Auto-Index!

```
$planeten = array('Merkur', 'Venus', 'Jupiter');
```

Index	0	1	2
Wert	Merkur	Venus	Jupiter

```
unset($planeten[0]);
unset($planeten[1]);
unset($planeten[2]);
```

Index	
Wert	

```
$planeten[] = 'Saturn';
```

Index	3
Wert	Saturn

Arrays vergleichen

- Wie schon bei den primitiven Datentypen können Arrays entweder mit == oder mit === verglichen werden
- Auch hier ist der Unterschied wichtig:
  - $a == $b liefert true, falls $a und $b die gleichen Schlüssel-Werte-Paare beinhalten, jeweils verglichen mit ==
  - $a === $b liefert true, falls $a und $b die gleichen Schüssel-Werte-Paare beinhalten, jeweils verglichen mit === und die Reihenfolge übereinstimmt

```
$array1 = array("1", "2", "3");
$array2 = array(1, 2, 3);
$array3 = array(1=>2, 2=>3, 0=>1);
```

```
$array1 == $array2    ➔ true
$array1 === $array2   ➔ false
$array1 == $array3    ➔ true
$array2 === $array3   ➔ false
```

Arrays vereinen

- Mit dem + Operator können zwei Arrays zu einem neuen vereint werden
- Falls in beiden Arrays Werte mit dem gleichen Index vorkommen, so werden die linksseitigen Arrays übernommen

```
$array1 = array(1 => “eins”, “zwei”, “drei”);
$array2 = array(3 => “DREI”, “vier”, “fünf”);
$array3 = $array1 + $array2;
```

Index	1	2	3
Wert	eins	zwei	**drei**

Index	3	4	5
Wert	DREI	vier	fünf

Index	1	2	3	4	5
Wert	eins	zwei	**drei**	vier	fünf

Mehrdimensionales Feld

- Elemente eines Arrays können wieder Arrays sein
- Man spricht dann von mehrdimensionalen Arrays
- **`array()`** liefert ein leeres Array zurück
- Sind alle Zeilen gleich lang, so handelt es sich um rechteckige Matrizen
- Elemente brauchen nicht denselben Datentyp zu besitzen
- **`is_array()`** hilft bei der Identifizierung von Unter-Arrays.
- Unter-Arrays brauchen nicht gleich lang zu sein
- **`count()`** liefert die Anzahl der Elemente eines Arrays

Beispiel: Mehrdimensionales Array

```
$person1 = array(
  “vorname” => “Otto”,
  “nachname” => “Normalverbraucher”,
  “wohnort” => “12345 Musterstadt”
);
```

```
$person2 = array(
  “vorname” => “Erika”,
  “nachname” => “Mustermann”,
  “wohnort” => “67890 Musterhausen”
);
```

```
$personen = array(
        $person1,
        $person2
)
```

Index	0		1	
Wert	vorname	Otto	vorname	Erika
	nachname	Normal…	nachname	Muster…
	wohnort	12345…	wohnort	67890 …

Mehrdimensionale Arrays

- Der Zugriff kann „klassisch“ durch auseinanderpflücken erfolgen:

```
$personen = array(….);
$person = $personen[0];
echo $person[“vorname”] . “ “ . $person[“nachname”]
```

- oder per doppeltem Klammer-Operator

```
$personen = array(….);

echo $personen[0][“vorname”] . “ “ . $personen[0][“nachname”];
```

- Natürlich sind auch Dimensionen größer als 2 erlaubt

Zusammenfassung

- Wozu werden Arrays verwendet?
- Was ist der Unterschied zwischen indizierten und assoziativen Arrays?
- Wozu verwendet man die foreach-Schleife?
- Wie werden mehrdimensionale Arrays definiert?

Aufgaben

1. Das Ausgabekonstrukt auf Folie 8 sowie 11 ist verbesserungswürdig. Schreiben Sie die for/each-Schleife so um, dass die Ausgabezeile nicht auf “, ” endet.
2. Speichern Sie die Anzahl der Kalendertage für ein Schaltjahr zu jedem Monat in ein Array und geben Sie die Monate mit der jeweiligen Tagesanzahl in einer foreach-Schleife aus. Des Weiteren soll die Summe der Tage des ganzen Jahres berechnet werden und als Gesamtwert ausgegeben werden.
3. Sehen Sie sich noch einmal das HTML-Tabellen Konstrukt der ersten Vorlesung (Folie 31ff) an und nutzen Sie dieses um einen mehrdimensonalen Array wie bspw. auf Folie 22 in eine Tabellenausgabe zu transferieren.