

Letzte Woche

Themen der letzten Vorlesung:

- HTML-Formulare
- CGI-Programme
- Zustandskodierung bei CGI-Programmen
- Sessions
- Cookies

Webprogrammierung mit PHP

CGI im Rückblick

CGI bietet eine Möglichkeit Web-Applikationen zu schreiben.

Allerdings:

- Viele technische Details zu beachten
- Sessiondaten müssen selbst verwaltet werden
- Ende der Ausgabe bedeutet Ende des CGI-Programms
- Für jede Anfrage neuer Start des CGI-Programms

⇒ Idee: Eine Programmiersprache speziell für die Entwicklung von Web-Applikationen.

PHP: Hypertext Preprocessor

Spezialisiert auf Programmierung von Web-Applikationen:

- Code wird in HTML-Seite eingebettet
- Automatische Verwaltung von Sessiondaten
- Einfacher Zugriff auf Formular-Daten
- Interpreter als interne Funktion des Webserverns

- C-ähnliche Syntax
- dynamisches Typsystem
- automatische Speicherverwaltung
- Anbindung für viele (C-)Bibliotheken

PHP-Interpreter im Webserver

Der Interpreter für PHP ist eine interne Funktion des Webservers:

- Server ruft Interpreter auf, wenn Datei mit Endung `.php` aufgerufen wird
- Anfrage steht über Variablen zur Verfügung
- Ausgabe des PHP-Programms ergibt HTTP-Antwort

⇒ Effektiver als Aufruf eines CGI-Programms

Skalare Datentypen

PHP kennt die skalaren Datentypen:

Ganzzahlen	1	23	-456
Fließkommazahlen	1.2	3.456	-5.0
Strings	"	"abc"	"6\x2A7\n"
Booleans	TRUE	FALSE	
NULL	(Kein Literal)		

Toplevel-Syntax

Syntax eines PHP-Programms:

- PHP-Programm ist eine Folge von `<statement>`
- Ein Statement: `<statement> ::= <expression> ";"`
- Mehrere Statements zusammenfassen:
`"{" <statement> ";" <statement> ";" ... "}"`
- Kommentare: einzeilig mit `//`, mehrzeilig mit `/* ... */`

Interpreter interpretiert nur Code, der innerhalb von

```
<?php ... ?> oder <? ... ?>
```

steht. Alles außerhalb geht direkt an den Client.

Operatoren

Bitweise	<code>&</code> , <code> </code> , <code>^</code> , <code>~</code> , <code><<</code> , <code>>></code>
Vergleich	<code>==</code> und <code>!=</code> testen auf strukturelle Gleichheit <code>===</code> und <code>!==</code> strukturelle Gleichheit mit Typprüfung Beispiel: <code>0===FALSE</code> aber <code>0!==FALSE</code>
Logisch	<code>&&</code> , <code> </code> , <code>!</code> , <code>and</code> , <code>or</code> , <code>xor</code>
Arithmetisch	- (unär und binär), <code>+</code> , <code>/</code> , <code>*</code> , <code>%</code>
Inkrement	<code>++</code> und <code>--</code> unär, jeweils präfix und postfix
Zuweisung	<code>=</code> , <code>+=</code> , <code>-=</code> , <code>*=</code> , <code>/=</code> , <code>%=</code> , <code>&=</code> , <code> =</code> , <code>^=</code> , <code><<=</code> , <code>>>=</code> , <code>.=</code>
Strings	<code>.</code> verbindet Strings: <code>"ab"."cd"=="abcd"</code>

Für Präzedenzregeln: siehe Handbuch

Kontrollstrukturen

Die wichtigsten Kontrollstrukturen:

- `if (<expression>) <statement>`
- `if (<expression>) <statement> else <statement>`
- `while (<expression>) <statement>`
- `do <statement> while (<expression>)`
- `for (<expression>; <expression>; <expression>) <statement>`

Außerdem: Kontrollstrukturen in "alternativer Syntax"

Testausdrücke werden in Booleans konvertiert

Variablen (2)

Für Funktionen und Variablen gilt:

- In Funktionen "deklarierte" Variablen gelten bis Ende des Rumpfes
- Funktionen müssen bei Zugriff auf globale Variable das Schlüsselwort `global` verwenden:

```
global $a; $b = $a * 2;
```

Variablen können innerhalb von Strings referenziert werden:

```
$a = "Hallo";  
"$a Welt!" == "Hallo Welt!"
```

Variablen (1)

Variablennamen in PHP sind case-sensitive beginnen immer mit `$`

Beispiele: `$a`, `$abc`, `$ab_cd`

- Keine Möglichkeit Variablen zu deklarieren
- Erste Benutzung (Referenzieren oder Zuweisen) bindet Variable
- Nicht zugewiesene Variablen haben den Wert `NULL` vom Typ `NULL`
Test mit `is_null()`
- Sichtbarkeitsbereich ist Kontext der Definition
- Allerdings erzeugt `{ ... }` keinen neuen Kontext
- Mögliche Kontexte: Rumpf einer Funktion, global

Automatische Typkonversion (1)

Viele Operatoren führen automatisch eine Typkonversion durch, wenn der Typ nicht passt:

- `"12a" + 3 ⇒ 15`
- `3.3 % 3 ⇒ 0`
- `if (1) 2; else 3 ⇒ 2`
- `if (0.0) 2; else 3 ⇒ 3`
- `if ("0.0") 2; else 3 ⇒ 2`
- `if ("1.0") 2; else 3 ⇒ 2`
- ...

Automatische Typkonversion (2)

Eine grobe Übersicht der automatischen Typkonversion:

Zahl → String	gemäß Darstellung
Fließkomma → Ganzzahl	durch Rundung
String → Zahl	Ziffern am Anfang des Strings, sonst NULL
Boolean → X	Boolean → Zahl → X Dabei: TRUE → 1 und FALSE → 0
X → Boolean	X → Zahl → Boolean 0.0 → FALSE 0 → FALSE 1 → TRUE

Konstanten

Syntax:

```
define(<const-name>, <expression>);
```

Definiert eine Konstante, die überall sichtbar ist.

Konstanten sind *superglobal*, d.h. Funktionen können die Konstante verwenden, ohne `global` zu verwenden.

Automatische Typkonversion (3)

Fortsetzung der Übersicht zur automatischen Typkonversion:

String → Boolean	"" → FALSE "0" → FALSE "0.0" → TRUE sonst → FALSE
NULL → Boolean	FALSE
NULL → Integer	0
NULL → String	""

Explizites Typcasting ist möglich:

```
(int),(string),(double)...
```

Funktionen

Syntax:

```
function <fun-name> "("<arg-list>"<statement>
```

wobei `<arg-list>` eine durch Kommata getrennte Liste der Parameter ist.

Beispiel:

```
function foo($a, $b)
{
...
}
```

Rückgabewert mit `return <expression>;`

Funktionsaufrufe (1)

Syntax: `<fun-name>(<args>)`

Beispiel:

```
function f ($p) {  
    p[0]="d";  
}
```

```
$a = "abc";
```

```
f($a);
```

```
⇒ $a[0] == "a"
```

Argumente werden call-by-value übergeben

Funktionsaufrufe (2)

Durch ein `&` vor den Parametern ist außerdem ist noch Call-by-reference möglich:

```
function f(&$p) {  
    p[0] = "d";  
}
```

```
$a = "abc";
```

```
f($a);
```

```
⇒ $a[0] == "d"
```

Vorgabewerte

Es können Vorgabewerte für Parameter festgelegt werden:

```
function f($a, $b=3) {  
    return $a + $b;  
}
```

```
⇒ f(5) 8
```

Vorsicht:

```
function f($a=3, $b) {  
    return $a + $b;  
}
```

```
⇒ f(5)
```

Arrays

Arrays sind die wichtigsten Datenstrukturen in PHP.

Literale: `array()`, `array(1,2)`, `array("a",12)`,
`array("de" => "Deutschland")`,
`array(1 => "Jan", "Feb", "Mär")`

Unterschiede zu klassischen Arrays:

- Größe kann verändert werden
- Array kann Löcher haben
- Index kann beliebiges Objekt sein
- Ungültiger Index liefert `NULL`
- Elemente können unterschiedlichen Typ haben

Zugriff auf Arrays

Zugriff auf ein Element des Arrays:

```
$a[0]   $a["de"]
```

Setzen eines Elementes

```
$a[0] = 3   $a["be"] = "Belgien"
```

- Die Funktion `count()` liefert Anzahl der Elemente zurück
- Mehrdimensionale Arrays haben Arrays als Elemente

Iteration über Arrays (1)

Falls ein Array nur mit Zahlen indiziert wurde, kann mit `for` darüber iteriert werden:

```
$arr = array ("a", "b", "c");
$arr[] = "d";
$arr[4] = "e";

$num_elements = count ($arr);
print (" $num_elements\n");

for ($idx = 0; $idx < $num_elements; $idx = $idx + 1) {
    echo "$arr[$idx]\n";
};
```

Vorsicht: Elemente, deren Index keine Zahl ist werden übersprungen.

Iteration über Arrays (2)

Syntax:

```
foreach (<array-expr> as <var>) <statement>
```

Für jedes Element: Binde `<var>` an das Element und führe `<statement>` aus.

Beispiel:

```
$arr=array("erstes","zweites");
$arr["bla"]="bei bla";
$arr[]="drittes";
```

```
foreach ($arr as $inhalt)
    print (" $inhalt\n");
```

Iteration über Arrays (3)

Syntax:

```
foreach (<array-expr> as <var-1> => <var-2>) <statement>
```

Diese Syntax bindet neben dem Element auch den Index.

Beispiel:

```
$arr=array("erstes","zweites");
$arr["bla"]="bei bla";
$arr[]="drittes";
```

```
$num_elements = count($arr);
```

```
foreach ($arr as $index => $inhalt)
    print (" $index points to $inhalt\n");
```

Elemente prüfen

- `isset()` überprüft, ob ein Element im Array gesetzt wurde
- `unset()` löscht eine Element aus dem Array

Beispiel:

```
$a = array(1,2,3);

function pp_bool($b) { return $b ? "TRUE" : "FALSE"; }

function print_arr($a) {
    for ($i = 0; $i < count($a); $i++)
        echo "$i (".pp_bool(isset($a[$i])).") -> $a[i]\n";
}

echo pp_bool(isset($a[2]))."\n";
echo pp_bool(isset($a[10]))."\n";
unset($a[2]);
echo pp_bool(isset($a[2]))."\n";
print_arr($a);
```