



Jan Teriete

# *Grundlagen der PHP7-Programmierung*

**Ein Webmasters Press Lernbuch**

Version 7.2.0 vom 23.04.2019

Autorisiertes Curriculum für das Webmasters Europe Ausbildungs- und Zertifizierungsprogramm

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	17
1.1	Für wen dieses Lernbuch gedacht ist	17
1.2	Vorkenntnisse	17
1.3	Aufbau der Lektionen	18
1.3.1	Code-Listings im Text	18
1.3.2	»Testen Sie Ihr Wissen!«	18
1.3.3	»Übungen«	18
1.3.4	»Optionale Übungen«	18
1.4	Formale Anforderungen	19
1.4.1	Systemanforderungen	19
1.4.2	PHP-fähiger Webserver	19
1.4.3	Code-Editor	19
1.4.4	Webbrowser	20
1.5	Zusammenfassung	20
<b>2</b>	<b>Dynamische Webseiten mit PHP</b>	21
2.1	Was sind dynamische Webseiten?	21
2.1.1	Statische Webseiten	21
2.1.2	Dynamische Webseiten	21
2.1.3	Vorteile dynamischer Webseiten	22
2.1.4	Nachteile dynamischer Webseiten	22
2.1.5	Die Auslieferung von Webseiten	22
2.2	PHP	23
2.2.1	Stärken von PHP	23
2.2.2	Geschichte von PHP	24
2.3	Zusammenfassung	26
2.4	Testen Sie Ihr Wissen!	27
<b>3</b>	<b>HTML und PHP</b>	28
3.1	Das Problem	28
3.2	PHP in eine HTML-Seite einbetten	28
3.2.1	Eine statische HTML-Seite erstellen	28
3.2.2	PHP-Code einfügen	28
3.2.3	PHP erzeugt HTML	29
3.3	Die sieben berühmten Fehler	30
3.3.1	PHP-Code im HTML-Bereich schreiben	30
3.3.2	HTML-Code im PHP-Bereich schreiben	30
3.3.3	Die PHP-Datei hat die Endung .html	31
3.3.4	Die PHP-Datei liegt nicht im Zugriff des Webserver	31
3.3.5	Die PHP-Datei wurde direkt im Browser geöffnet	31
3.3.6	Die echo-Anweisung fehlt	32
3.3.7	Das Leerzeichen wurde vergessen	32
3.4	Mehrzeilige PHP-Anweisungen	33
3.4.1	Das Semikolon	33
3.4.2	Mehrzeilige PHP-Anweisungen	33
3.5	Stilfragen	34
3.5.1	Saubere Trennung von HTML und PHP	34
3.5.2	Einrückungen	36

3.6	Zusammenfassung	37
3.7	Testen Sie Ihr Wissen!	37
3.8	Übungen	37
3.9	Optionale Übungen	38
<b>4</b>	<b>Variablen und Operatoren</b>	<b>39</b>
4.1	Das Problem	39
4.2	Variablen	39
4.2.1	Grundlagen	39
4.2.2	Der Zuweisungsoperator	39
4.2.3	Zahlen	40
4.2.4	Zeichenketten	40
4.2.5	Der null-Wert	40
4.2.6	Variableninhalte auslesen	41
4.2.7	Benennung von Variablen	41
4.3	Kommentare in PHP einbetten	43
4.3.1	Grundlagen	43
4.3.2	Einzeilige Kommentare	43
4.3.3	Mehrzeilige Kommentare	43
4.4	Operatoren	44
4.4.1	Grundlagen	44
4.4.2	Rechenoperatoren	44
4.4.3	Weitere Rechenoperatoren	46
4.4.4	Der Verknüpfungs-Operator	46
4.4.5	Kombinierte Operatoren	47
4.4.6	Inkrement und Dekrement-Operatoren	48
4.5	Zusammenfassung	49
4.6	Testen Sie Ihr Wissen!	49
4.7	Übungen	50
4.8	Optionale Übungen	51
<b>5</b>	<b>PHP-Funktionen</b>	<b>52</b>
5.1	Das Problem	52
5.2	Aufgabe von Funktionen	52
5.3	Funktionen aufrufen	52
5.3.1	Grundlagen	52
5.3.2	Fehlerhafter Funktionsaufruf	53
5.3.3	Fehlerausgabe aktivieren	53
5.4	Parameter an Funktionen übergeben	54
5.5	Rückgabewerte von Funktionen	54
5.5.1	Grundlagen	54
5.5.2	Rückgabewerte ausgeben	55
5.5.3	Rückgabewerte in Variablen speichern	55
5.6	Funktionen mit mehreren Parametern	55
5.7	Die PHP-Funktionsreferenz	56
5.7.1	Grundlagen	56
5.7.2	Suchen und browsen auf php.net	57
5.7.3	Funktionsdefinitionen lesen	57
5.8	Zusammenfassung	60
5.9	Testen Sie Ihr Wissen!	60
5.10	Übungen	60
5.11	Optionale Übungen	61

<b>6</b>	<b>PHP-Datentypen</b>	62
6.1	Das Problem	62
6.2	Die Funktion gettype()	62
6.3	NULL	62
6.4	Boolean	63
6.4.1	Grundlagen	63
6.4.2	isset() und unset()	63
6.4.3	empty()	64
6.5	Integer	67
6.5.1	Grundlagen	67
6.5.2	is_int()	68
6.5.3	intval()	69
6.6	Float/Double	70
6.6.1	Grundlagen	70
6.6.2	is_float()	71
6.6.3	floatval()	71
6.7	String	71
6.7.1	Grundlagen	71
6.7.2	Bedeutung der Anführungszeichen	71
6.7.3	is_string()	73
6.7.4	strval()	73
6.7.5	strlen()	73
6.8	Weitere Datentypen	74
6.8.1	Array	74
6.8.2	Object	74
6.8.3	Resource	74
6.9	Type Casting	74
6.10	Dynamische Typumwandlung	75
6.10.1	Grundlagen	75
6.10.2	Mathematische Operatoren	75
6.10.3	Der Verknüpfungs-Operator	77
6.10.4	Nicht-boolesche Wahrheitswerte	77
6.11	Zusammenfassung	78
6.12	Testen Sie Ihr Wissen!	78
6.13	Übungen	78
6.14	Optionale Übungen	79
<b>7</b>	<b>Arrays</b>	80
7.1	Das Problem	80
7.2	Grundlagen	80
7.3	Mit Arrays arbeiten	80
7.3.1	Erzeugen mit eckigen Klammern	80
7.3.2	Auslesen von Werten	81
7.3.3	Erzeugen mit explizitem Index	82
7.3.4	Erzeugen mittels array()	83
7.3.5	Erzeugen mittels kurzer Array-Syntax	84
7.4	Assoziative Arrays	85
7.4.1	Grundlagen	85
7.4.2	Der Doppelpfeil-Operator (=>)	85
7.5	Nützliche Array-Funktionen	86
7.5.1	is_array()	86
7.5.2	count()	87
7.5.3	var_dump()	87

7.5.4	explode()	88
7.5.5	implode()	88
7.5.6	shuffle()	89
7.5.7	in_array()	90
7.5.8	array_reverse()	91
7.5.9	array_keys()	92
7.5.10	array_values()	93
7.5.11	array_pop()	94
7.5.12	array_shift()	95
7.6	Zusammenfassung	95
7.7	Testen Sie Ihr Wissen!	95
7.8	Übungen	98
7.9	Optionale Übungen	98
<b>8</b>	<b>Formulare und Links</b>	<b>99</b>
8.1	Das Problem	99
8.2	Informationen an PHP-Skripte übergeben	99
8.3	Parameter über HTML-Links übergeben	99
8.3.1	Aufbau von URLs	99
8.3.2	URL-Parameter in PHP auslesen	100
8.3.3	Zweck von URL-Parametern	101
8.4	Parameter durch HTML-Formulare übergeben	102
8.4.1	Grundlagen	102
8.4.2	Der Form-Tag	103
8.4.3	Formulardaten auslesen	105
8.4.4	Formularelemente	105
8.5	Das Array \$_REQUEST	113
8.5.1	Grundlagen	113
8.5.2	\$_REQUEST oder \$_POST?	114
8.6	Zusammenfassung	114
8.7	Testen Sie Ihr Wissen!	114
8.8	Übungen	115
8.9	Optionale Übungen	115
<b>9</b>	<b>Funktionsentwurf</b>	<b>117</b>
9.1	Das Problem	117
9.2	Funktionen definieren	117
9.2.1	Funktionskopf	117
9.2.2	Funktionskörper	118
9.2.3	Selbst geschriebene Funktionen aufrufen	118
9.3	Funktionen in Funktionen aufrufen	119
9.4	Namenskonventionen	119
9.5	Zusammenfassung	120
9.6	Testen Sie Ihr Wissen!	120
9.7	Übungen	120
<b>10</b>	<b>Strukturierung von PHP-Skripten</b>	<b>121</b>
10.1	PHP-Code strukturieren	121
10.2	Dreiteilung des PHP-Codes	122
10.2.1	Grundlagen	122
10.3	Namensgebung	125
10.3.1	Variablen	125
10.3.2	Funktionen	126

10.4	Auslagern von Funktionen	126
10.4.1	Konzept	126
10.4.2	require_once	127
10.4.3	require	127
10.4.4	include_once	128
10.4.5	Namens- und andere Konventionen	129
10.5	Zusammenfassung	130
10.6	Testen Sie Ihr Wissen!	130
10.7	Übungen	132
10.8	Optionale Übungen	133
<b>11</b>	<b>Verzweigungen</b>	<b>134</b>
11.1	Das Problem	134
11.2	Boolesche Logik	134
11.2.1	Konzept	134
11.2.2	Vergleichsoperatoren	134
11.2.3	Boolesche Ausdrücke modifizieren	142
11.2.4	Wahrheitswerte aus anderen Quellen	147
11.3	if-Anweisung	148
11.3.1	Grundlagen	148
11.3.2	Reine if-Anweisung	149
11.3.3	if-else	150
11.3.4	if-elseif-else	151
11.4	Alternativen zur if-Anweisung	154
11.4.1	Grundlagen	154
11.4.2	switch-Anweisung	154
11.4.3	Der ternäre Operator	158
11.4.4	Der Null-Coalesce-Operator	160
11.5	Zusammenfassung	162
11.6	Testen Sie Ihr Wissen!	162
11.7	Übungen	163
<b>12</b>	<b>Schleifen</b>	<b>164</b>
12.1	Das Problem	164
12.2	Schleifen	165
12.3	foreach-Schleife	165
12.3.1	Grundlagen	165
12.3.2	foreach zur formatierten Ausgabe von HTML	166
12.3.3	Über Schlüssel und Werte iterieren	167
12.4	for-Schleife	168
12.4.1	Grundlagen	168
12.4.2	Die Startanweisung	169
12.4.3	Die Abbruchbedingung	169
12.4.4	Der Modifikator	169
12.4.5	Der Schleifenkörper	170
12.4.6	Anwendungen	170
12.5	while-Schleife	173
12.5.1	Grundlagen	173
12.5.2	Anwendung	174
12.6	Endlos-Schleifen	175
12.7	Zusammenfassung	176
12.8	Testen Sie Ihr Wissen!	176

12.9	Übungen	176
12.10	Optionale Übungen	177
<b>13</b>	<b>Funktionen mit Parametern und Rückgabewerten</b>	<b>178</b>
13.1	Das Problem	178
13.2	Parameter an Funktionen übergeben	178
13.2.1	Geltungsbereiche von Variablen	178
13.2.2	Übergabe von Parametern	179
13.2.3	Übergabe mehrerer Parameter	180
13.2.4	Parameter mit Standardwerten	182
13.3	Werte aus einer Funktion zurückgeben	183
13.3.1	Konzept	183
13.3.2	return	183
13.3.3	Einen Wert zurückgeben	184
13.3.4	Mehrere Werte als Array zurückgeben	185
13.4	Zusammenfassung	186
13.5	Testen Sie Ihr Wissen!	186
13.6	Übungen	188
13.7	Optionale Übungen	189
<b>14</b>	<b>Mehrdimensionale Arrays</b>	<b>190</b>
14.1	Das Problem	190
14.2	Konzept	190
14.3	Zweidimensionale Arrays	190
14.3.1	Zweidimensionale Arrays erzeugen	190
14.3.2	Zweidimensionale Arrays auslesen	191
14.3.3	Formatierung	191
14.4	Mehrdimensionale Arrays	193
14.5	Verschachtelte foreach-Schleifen	194
14.6	Zusammenfassung	196
14.7	Testen Sie Ihr Wissen!	196
14.8	Übungen	197
14.9	Optionale Übungen	197
<b>15</b>	<b>Stringbehandlung</b>	<b>199</b>
15.1	Das Problem	199
15.2	Strings formatieren	199
15.2.1	Grundlagen	199
15.2.2	trim()	199
15.2.3	strtoupper()	200
15.2.4	strtolower()	200
15.2.5	ucfirst()	202
15.2.6	lcfirst()	202
15.2.7	htmlspecialchars()	202
15.2.8	strip_tags()	206
15.2.9	nl2br()	208
15.2.10	vprintf()	208
15.2.11	vsprintf()	211
15.3	Datum und Uhrzeit formatieren	211
15.3.1	Das Problem	211
15.3.2	Der Timestamp	211
15.3.3	time()	212
15.3.4	strftime()	212

15.3.5	mktime()	214
15.3.6	strptime()	216
15.4	Strings durchsuchen	218
15.4.1	Grundlagen	218
15.4.2	strlen()	218
15.4.3	strpos()	219
15.5	Strings bearbeiten	221
15.5.1	Grundlagen	221
15.5.2	substr()	221
15.5.3	str_replace()	222
15.6	lfirst() selbst programmieren	224
15.7	Zusammenfassung	225
15.8	Testen Sie Ihr Wissen!	225
15.9	Übungen	226
15.10	Optionale Übungen	226
<b>16</b>	<b>Persistente Daten</b>	<b>228</b>
16.1	Das Problem	228
16.2	Dateien als Speichermedium	228
16.2.1	file_put_contents()	228
16.2.2	file_get_contents()	229
16.3	Speichern komplexer Datenstrukturen	229
16.3.1	serialize()	230
16.3.2	unserialize()	230
16.4	Serialisierte Daten aktualisieren	232
16.4.1	Das Problem	232
16.4.2	Vorgehensweise	232
16.5	Zusammenfassung	233
16.6	Testen Sie Ihr Wissen!	233
16.7	Übungen	235
16.8	Optionale Übungen	236
<b>17</b>	<b>HTTP und PHP-Sessionverwaltung</b>	<b>237</b>
17.1	Das Problem	237
17.2	Grundlagen des HTTP-Protokolls	237
17.2.1	Anfrage des Clients (Request)	239
17.2.2	Antwort des Servers (Response)	241
17.3	Mit PHP HTTP-Header auslesen und manipulieren	243
17.3.1	Grundlagen	243
17.3.2	\$_SERVER	243
17.3.3	header()	245
17.4	Einschränkungen des HTTP-Protokolls	247
17.5	Sessions	248
17.6	Die PHP-Sessionverwaltung	248
17.6.1	Sessions eröffnen	248
17.6.2	Daten in Sessions speichern	249
17.6.3	Sessions beenden	250
17.7	Zusammenfassung	251
17.8	Testen Sie Ihr Wissen!	251
17.9	Übungen	252
17.10	Optionale Übungen	252



<b>18</b>	<b>Einige Tipps und Tricks aus der PHP-Kiste</b>	254
18.1	Das Problem	254
18.2	Konstanten	254
18.2.1	Prinzip	254
18.2.2	Alternative Syntax	255
18.2.3	Anwendung	255
18.3	error_reporting()	256
18.3.1	Das Problem	256
18.3.2	Den Fehlerlevel einstellen	257
18.3.3	Fehlermeldungen abschalten	257
18.4	Alternative PHP-Syntax für HTML-Templates	258
18.4.1	Warum eine andere Syntax?	258
18.4.2	Die Syntax	259
18.4.3	Alternativ-Syntax Variablen-Ausgabe	259
18.5	Zusammenfassung	260
18.6	Testen Sie Ihr Wissen!	260
18.7	Übungen	260
18.8	Optionale Übungen	261
<b>19</b>	<b>Wir programmieren ein Weblog</b>	262
19.1	Das Projekt	262
19.1.1	Grundlagen	262
19.1.2	Ein wenig Make-up	262
19.1.3	CSS	265
19.2	Die Funktionen	269
19.3	Testdaten	270
19.3.1	Benutzer	270
19.3.2	Einträge	270
19.4	Die Startseite	273
19.4.1	Inhalt	273
19.4.2	Einträge auslesen	274
19.4.3	Einträge formatiert anzeigen	274
19.4.4	Menü oder Login-Formular	275
19.5	Einloggen	276
19.5.1	Inhalt	276
19.5.2	Der Login	276
19.5.3	Weiterleitung per header()	277
19.6	Ausloggen	277
19.7	Eintrag schreiben	278
19.7.1	Inhalt	278
19.7.2	Logintest	279
19.7.3	Eintrags-Formular	279
19.7.4	Hauptmenü	279
19.8	Eintrag speichern	280
19.8.1	Inhalt	280
19.8.2	Login und POST-Test	281
19.8.3	Eintrag-Array erstellen	281
19.8.4	Neuen Eintrag speichern	281
19.8.5	Eintrag formatiert anzeigen	282
19.9	Zusammenfassung	282
19.10	Testen Sie Ihr Wissen!	282
19.11	Übungen	283
19.12	Optionale Übungen	284

<b>20</b>	<b>Einführung in PDO</b>	285
20.1	Das Problem	285
20.2	Einführung in PDO	285
20.3	Datenbank-Verbindung aufbauen	286
20.3.1	Das PDO-Objekt	286
20.3.2	Der DSN	286
20.4	SQL-Anweisungen mit PDO ausführen	287
20.4.1	PDO#query()	287
20.4.2	PDO#errorInfo()	288
20.4.3	Ergebnisse verarbeiten: Die Klasse PDOStatement	289
20.5	PDO-Attribute	293
20.5.1	PDO-Attribute beim Erzeugen des PDO-Objekts setzen	293
20.5.2	Fehlerverhalten einstellen (PDO::ATTR_ERRMODE)	293
20.5.3	Fetch-Modus (PDO::ATTR_DEFAULT_FETCH_MODE)	294
20.5.4	Gepufferte Abfragen (PDO::MYSQL_ATTR_USE_BUFFERED_QUERY)	295
20.5.5	Vorschlag für eine Standard-PDO-Verbindung	297
20.6	MySQL und Unicode	298
20.6.1	Unicode-konforme Tabellen erzeugen	298
20.6.2	Unicode-Daten aus MySQL auslesen	298
20.7	Testen Sie Ihr Wissen!	299
20.8	Übungen	299
<b>21</b>	<b>PDO in der Praxis</b>	301
21.1	SQL-Übersicht	301
21.1.1	CREATE TABLE	301
21.1.2	DROP TABLE	302
21.1.3	INSERT	302
21.1.4	SELECT	304
21.1.5	UPDATE	305
21.1.6	DELETE	305
21.2	Prepared Statements	306
21.2.1	Einführung in Prepared Statements	306
21.2.2	Mehrfaches Ausführen von Prepared Statements	307
21.2.3	Limitierungen von Platzhaltern in Prepared Statements	308
21.2.4	Prepared Statements mit benannten Platzhaltern	308
21.2.5	PDO und Sicherheit	310
21.3	CRUD	313
21.3.1	Vorbereitungen	313
21.3.2	READ: index.php	314
21.3.3	CREATE: anlegen.php	316
21.3.4	DELETE: loeschen.php	318
21.3.5	UPDATE: bearbeiten.php	319
21.4	Zusammenfassung	321
21.5	Testen Sie Ihr Wissen!	321
21.6	Übungen	322
21.7	Optionale Übungen	322
<b>22</b>	<b>Tricks rund um Arrays</b>	323
22.1	Das Problem	323
22.2	Array-Sortierung	323
22.2.1	Benutzerdefinierte Vergleichsfunktionen	327
22.2.2	Der Spaceship-Operator	328
22.2.3	Anonyme Funktionen	329

22.3	Der Splat-Operator	330
22.4	Variadische Funktionen	331
22.5	Dereferenzierung	333
22.6	Destructuring	333
22.7	Zusammenfassung	336
22.8	Testen Sie Ihr Wissen!	336
<b>23</b>	<b>Anhang 1: XAMPP</b>	<b>339</b>
23.1	Was XAMPP ist	339
23.2	Installation	339
23.2.1	Windows	340
23.2.2	Mac OS X	340
23.2.3	Linux	341
23.3	Zusammenfassung	341
<b>24</b>	<b>Anhang 2: Editoren und IDEs</b>	<b>342</b>
24.1	Notepad++	342
24.2	Alternativen zu Notepad++	342
	<b>Lösungen der Übungsaufgaben</b>	<b>344</b>
	<b>Lösungen der Wissensfragen</b>	<b>348</b>
	<b>Index</b>	<b>378</b>

## 3

# HTML und PHP

## In dieser Lektion lernen Sie

- ▶ wie Sie PHP in Ihre HTML-Dateien einbetten können.
- ▶ was Sie dabei beachten müssen.
- ▶ die sechs Fehler kennen, die jeder PHP-Anfänger macht.
- ▶ die `echo`-Anweisung kennen.

## 3.1 Das Problem

Wenn Sie die letzte Lektion durchgelesen haben, wissen Sie nun, wie eine dynamische Webseite prinzipiell funktioniert. Jetzt wird es jedoch Zeit, dass Sie selbst eine PHP-Seite bauen, daher lassen Sie uns auch gleich loslegen. Sie haben inzwischen doch den Webserver und den Code-Editor installiert, wie es in [Lektion 1](#) verlangt war, oder?

## 3.2 PHP in eine HTML-Seite einbetten

### 3.2.1 Eine statische HTML-Seite erstellen

Öffnen Sie Ihren bevorzugten Code-Editor und erstellen Sie nachfolgendes HTML-Dokument. Speichern Sie es im **document root** Ihres Webserver im Unterordner *phpschulung* unter dem Namen *erste\_seite.php* ab. Wenn Sie beispielsweise eine Standardinstallation von XAMPP (siehe [Lektion 23](#)) durchgeführt haben, ist Ihr **document root** das Verzeichnis `c:\xampp\htdocs`. Bei MAMP unter Mac OS X ist der Standard `/Applications/MAMP/htdocs` und auf Linux-Servern finden Sie dieses Verzeichnis oft unter `/var/www`. Vergessen Sie nicht die korrekte Dateiergung `.php` zu benutzen, diese ist sehr wichtig.

### Beispiel

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title>Hallo Welt</title>
7 </head>
8
9 <body>
10     Hallo Welt
11 </body>
12
13 </html>
```

**Codebeispiel 1** *erste\_seite.php (Version 1)*

Rufen Sie die Datei nun in Ihrem Browser unter der URL `http://localhost/phpschulung/erste_seite.php` auf.

### 3.2.2 PHP-Code einfügen

So weit sollte das für Sie nichts Neues gewesen sein. Das Besondere an PHP ist allerdings, dass Sie es, ähnlich wie JavaScript, direkt in Ihre HTML-Dokumente einfügen können.

Um dem Webserver zu zeigen, wo PHP-Code in HTML-Dokumenten steht, wurde ein spezieller Tag entwickelt, der jeden PHP-Code umgeben muss. Kein PHP-Code darf außerhalb dieser Begrenzung stehen, oder der Webserver hält ihn für normales HTML und zeigt ihn einfach an.

### Beispiel

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8" />
6   <title><?php echo "Hallo Welt"; ?></title>
7 </head>
8
9 <body>
10  <?php echo "Hallo Welt"; ?>
11 </body>
12
13 </html>
```

**Codebeispiel 2** *erste\_seite.php (Version 2)*

Was hat sich hier geändert? Wenn Sie sich die geänderte Seite in Ihren Browser ansehen, werden Sie keinen Unterschied feststellen. Die Seite hat immer noch den Titel `Hallo Welt` und gibt im `body`-Tag denselben Text noch einmal aus.

Der Unterschied ist, dass der Text `Hallo Welt` nun mittels PHP in die Seite geschrieben wurde. Wann immer Sie PHP-Code einfügen wollen, müssen Sie zuerst `<?php` schreiben, um zu zeigen, dass nun PHP-Code folgt. Wenn Sie mit dem PHP-Block fertig sind, zeigen Sie das durch den schließenden Tag `?>`. Um die Bedeutung des eigentlichen PHP-Codes müssen Sie sich im Moment noch nicht kümmern.<sup>4</sup>

Jeder PHP-Code muss zwischen `<?php` und dem schließenden Tag `?>` stehen. Ansonsten wird er nicht ausgewertet, sondern unverändert angezeigt. Dies sind die sogenannten **PHP-Tags**, analog zu den bekannten HTML-Tags.



### 3.2.3 PHP erzeugt HTML

Wenn Sie sich den Quelltext der HTML-Seite anzeigen lassen (z. B. im *Firefox*-Browser mit Strg+U), werden Sie feststellen, dass dort nichts mehr von dem PHP-Code zu sehen ist. Bei genauerer Betrachtung werden Sie sogar erkennen, dass die erzeugte Seite mit der ersten, statischen Version aus [Abschnitt 3.2.1](#) identisch ist. Vermutlich sind nicht alle Zeilenumbrüche da, wo Sie es erwarten. Zum Beispiel könnte der schließende `body`-Tag hinter das "Hallo Welt" gerutscht sein. Doch dies soll uns nicht weiter stören.

Exakt das ist damit gemeint, wenn man von PHP als einer serverseitigen Programmiersprache spricht. Der Webserver hat die PHP-Datei geöffnet, den PHP-Code durch den Interpreter auswerten lassen und, was als Ergebnis übrig blieb, an den Browser gesendet. Der Browser sieht nichts mehr von den ursprünglichen PHP-Tags und das ist auch gut so, denn Webbrowser haben keine Ahnung, was PHP überhaupt ist!

JavaScript wird vom Webbrowser ausgewertet. Also ist JavaScript-Code für den Browser sichtbar. PHP wird schon auf dem Webserver ausgewertet, lange bevor der Browser die Seite erhält. Also sollte ein Webbrowser niemals PHP-Code sehen.



4. Er sorgt dafür, dass der Text in den Anführungszeichen in der Webseite angezeigt wird.

## 3.3 Die sieben berühmten Fehler

Wenn Sie den Beispielen aus dem vorherigen Abschnitt problemlos folgen konnten, gratulieren Sie sich. Sie haben die Beispiele fehlerfrei abgetippt. Wenn Sie hingegen seltsame Meldungen oder schlimmer, gar nichts in Ihrem Browser gesehen haben, sind Sie wahrscheinlich in einen der sechs Fehler gelaufen, die jeder PHP-Anfänger macht. Sollte es Sie erwischt haben, grämen Sie sich nicht. Das bedeutet nur, dass Sie es schon hinter sich haben! Jeder PHP-Anfänger macht diese Fehler!

### 3.3.1 PHP-Code im HTML-Bereich schreiben

Einer der beliebtesten Fehler ist es, PHP-Code außerhalb des durch `<?php ?>` markierten Bereiches zu schreiben. Das bedeutet, dass dieser Code für HTML gehalten und einfach an den Browser geschickt wird.

#### Beispiel

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title><?php echo "Hallo Welt"; ?></title>
7 </head>
8
9 <body>
10     echo "Hallo Welt";
11 </body>
12
13 </html>
```

**Codebeispiel 3** *php\_im\_html.php*

Die erste `echo`-Anweisung in Zeile 6 wird korrekt ausgewertet. Im Browser steht an dieser Stelle nur noch `Hallo Welt`. In Zeile 10 jedoch haben wir vergessen, den PHP-Code in den passenden Tags zu verbergen. Das Ergebnis ist, dass der Browser an dieser Stelle das komplette `echo "Hallo Welt";` ausgibt.

### 3.3.2 HTML-Code im PHP-Bereich schreiben

Das Gegenteil des letzten Fehlers ist genauso verhängnisvoll. Wenn Sie HTML-Tags in den PHP-Bereich schreiben, versucht der PHP-Interpreter, aus den für ihn seltsamen Zeichen schlau zu werden und gibt schließlich mit einer Fehlermeldung auf.

#### Beispiel

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <?php <meta charset="utf-8" /> ?>
6     <title><?php echo "Hallo Welt"; ?></title>
7 </head>
8
9 <body>
10     <?php echo "Hallo Welt"; ?>
11 </body>
12
13 </html>
```

**Codebeispiel 4** *html\_im\_php.php*

Jetzt liegt der Fehler in Zeile 5. Dort wird zwar ein PHP-Tag geöffnet, aber sein Inhalt besteht aus purem HTML. Bestraft wird dieses Verhalten von PHP mit einer hässlichen Fehlermeldung. Wenn Sie die Seite aufrufen, sehen Sie folgenden Text:

```
Parse error: syntax error, unexpected '<' in (...)html_im_php.php on line 5
```

Genauso wenig, wie der Browser Ahnung von PHP hat, weiß der PHP-Interpreter etwas mit den HTML-Formatierungen anzufangen. Dementsprechend sagt er uns auch, dass er mit dem unerwarteten (unexpected) Zeichen '<' nichts anfangen kann.

### 3.3.3 Die PHP-Datei hat die Endung .html

Selbst wenn Sie das ursprüngliche Beispiel fehlerfrei abgetippt haben, kann es dennoch sein, dass Sie den kompletten Code mit allen PHP-Tags im Browser sehen. Das kann mehrere Gründe haben. Der häufigste ist jedoch, dass Sie die Datei mit der Endung `.html` versehen haben.

Wenn eine Datei, die PHP-Code enthält, nicht auf `.php` endet, weiß der Webserver für gewöhnlich nicht, dass sich PHP darin befindet, und schickt die Datei unverändert an den Browser.

Welche Dateiendungen der Webserver als PHP-Dateien ansieht, kann natürlich konfiguriert werden. Häufig funktionieren neben `.php` auch versionsspezifische Endungen wie beispielsweise `.php55`. Solange Sie keinen triftigen Grund haben, etwas anderes zu tun, sollten Sie immer die Endung `.php` verwenden, da diese immer funktioniert und Sie auf die genaue Konfiguration bei Ihrem Webhoster oft keinen Einfluss haben.

### 3.3.4 Die PHP-Datei liegt nicht im Zugriff des Webserver

Ein Webserver darf nur auf Dateien in einem vorher festgelegten Verzeichnis, genannt **document root**, zugreifen. Wenn Sie Ihre PHP-Dateien außerhalb dieses Verzeichnisses speichern, wird der Webserver sie nicht finden und einen Fehler melden.

Dies ist ein, gerade von PHP-Anfängern, häufig gemachter Fehler, da die meisten Windows-Anwender es beispielsweise gewohnt sind, Dateien in *Eigene Dateien* oder auf dem *Desktop* abzulegen, wo der Webserver sie nicht finden kann.

Wenn Sie häufiger in diese Falle tappen, legen Sie sich eine Verknüpfung zum **document root** auf Ihrem *Desktop* an. Dadurch sieht es so aus, als würden Sie die Dateien direkt in einem Ordner auf dem *Desktop* ablegen, und es funktioniert trotzdem.



### 3.3.5 Die PHP-Datei wurde direkt im Browser geöffnet

Kommen wir nun zum wahrscheinlich häufigsten Anfängerfehler. Anstatt die PHP-Datei über den Webserver aufzurufen, haben Sie die Datei direkt vom Dateisystem aus geöffnet. Das bedeutet, Ihr Webbrowser erhält direkt den PHP-Code und hat natürlich keine Ahnung, was er damit anfangen soll. Moderne Browser blenden meist alles zwischen `<?php ... ?>` einfach aus, aber gerade ältere Browser haben den Inhalt direkt angezeigt.

Sie merken am besten an der URL-Zeile im *Firefox*-Browser, dass Sie die Datei direkt geöffnet haben. Wenn Sie dort zu Anfang `file://` lesen, sind Sie in die Falle getappt. Gerade am Anfang Ihrer PHP-Karriere sollte der erste Blick der URL gelten, wenn etwas nicht so funktioniert, wie Sie es erwarten. In anderen Browsern wie beispielsweise Chrome wird eine PHP-Datei in einem solchen Fall übrigens gar nicht geöffnet, sondern zum Download angeboten.

### 3.3.6 Die echo-Anweisung fehlt

In den vorherigen Abschnitten haben Sie in die PHP-Tags das Wort `echo`, gefolgt von Text in Anführungszeichen, geschrieben. Das Ergebnis war, dass das Wort `echo` und die Anführungszeichen verschwunden sind und nur der Text selbst im Browser angekommen ist.

Dieses Wort `echo` ist für PHP eine Anweisung und veranlasst den PHP-Interpreter, den folgenden Text auszugeben. Im Gegensatz zu HTML, wo Sie jegliche Ausgabe einfach in die Datei schreiben, müssen Sie PHP ausdrücklich sagen, wenn Sie etwas auf dem Bildschirm ausgeben wollen. Für diesen Zweck gibt es in PHP eine ganze Reihe von Anweisungen, für den Anfang soll uns `echo` allerdings genügen, da Sie mit dieser Anweisung fast jede Situation meistern können.

#### Beispiel

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title><?php echo "Hallo Welt"; ?></title>
7 </head>
8
9 <body>
10    <?php "Hallo Welt"; ?>
11 </body>
12
13 </html>
```

**Codebeispiel 5** *echo\_fehlt.php*

Fehlt die `echo`-Anweisung, so hat der PHP-Interpreter keine Ahnung, was er mit dem Text in Anführungszeichen anfangen soll, und dieser verfällt nach Ausführung von Zeile 10 ungenutzt. In diesem Fall erhalten wir somit eine leere Seite, da das `body`-Tag keinen weiteren Inhalt hat.

### 3.3.7 Das Leerzeichen wurde vergessen

Fehlt zwischen dem öffnenden Tag `<?php` und der nachfolgenden Anweisung das trennende Leerzeichen, so kann der PHP-Interpreter den PHP-Code nicht mehr erkennen.

#### Beispiel

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title><?php echo "Hallo Welt"; ?></title>
7 </head>
8
9 <body>
10    <?phpecho "Hallo Welt"; ?>
11 </body>
12
13 </html>
```

**Codebeispiel 6** *leerzeichen\_fehlt.php*

Im Browser bekommen wir beim Aufruf dieser PHP-Datei eine leere Seite angezeigt. Es gibt auch keine Fehlermeldung des PHP-Interpreters, da dieser den PHP-Code ja gar nicht erkannt hat. Schaut man sich jedoch den Quelltext der HTML-Seite im Browser an, so kann man das Problem an dem im Quelltext enthaltenen PHP-Code erkennen. Eine PHP-IDE würde die fehlerhafte Zeile 10 übrigens sofort im Code markieren.



## 3.4 Mehrzeilige PHP-Anweisungen

### 3.4.1 Das Semikolon

Bisher haben Sie in jedem Beispiel am Ende einer PHP-Anweisung ein **Semikolon**, zu Deutsch Strichpunkt, gesehen. Sollten Sie eines vergessen haben, werden Sie keinen Unterschied bemerkt haben. Der PHP-Code hat weiterhin genau dasselbe getan.

Wofür brauchen wir dieses Semikolon dann überhaupt? Es ist für die Trennung einzelner PHP-Anweisungen zuständig. Wenn Sie also mehr als eine Anweisung für PHP haben, müssen Sie dazwischen ein Semikolon setzen.

#### Beispiel

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8" />
6   <title><?php echo "Hallo Welt"; ?></title>
7 </head>
8
9 <body>
10  <?php echo "Hallo Welt "; echo "und willkommen beim PHP-Seminar!"; ?>
11  Das ist normales HTML.
12  <?php echo "Das ist korrekt." echo "Das nicht mehr!"; ?>
13 </body>
14
15 </html>
```

**Codebeispiel 7** *mehrere\_php\_anweisungen.php*

Der Unterschied zwischen den Zeilen 10 und 12 besteht darin, dass in Zeile 12 vor dem zweiten `echo` kein Semikolon steht. PHP denkt also, hier stünde nur eine einzige lange Anweisung anstatt zwei. Da eine Anweisung mit zwei `echo`-Aufrufen für PHP kein gültiger Code ist, wird hier ein Fehler erzeugt:

```
Parse error: syntax error, unexpected 'echo' (T_ECHO), expecting ',' or ';' in
(...) \mehrere_php_anweisungen.php on line 12
```

PHP sagt uns hier, dass in Zeile 12 entweder ein Komma oder ein Semikolon erwartet wird. Tragen Sie also das in diesem Fall fehlende Semikolon nach, damit der Code funktioniert.

### 3.4.2 Mehrzeilige PHP-Anweisungen

Auch wenn es funktioniert, mehrere PHP-Anweisungen hintereinander zu schreiben, so ist es nicht wirklich übersichtlich. Vergleichen Sie im folgenden Listing die beiden PHP-Blöcke:

#### Beispiel

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8" />
6   <title>Hallo Welt</title>
7 </head>
8
9 <body>
10  <?php echo "Hallo Welt "; echo "und willkommen beim PHP-Seminar! "; echo
"Langsam wir es unübersichtlich. "; echo "Jetzt ist aber wirklich Schluss!"; ?>
```

```

11 <br />
12 <?php
13
14 echo "So sieht das schon viel besser aus. ";
15 echo "Wenn Sie jede Anweisung auf eine Zeile schreiben, ";
16 echo "behalten Sie leichter den Überblick. ";
17 echo "Trotzdem müssen Sie immer noch ";
18 echo "ein Semikolon nach jeder Anweisung schreiben, ";
19 echo "da PHP eine neue Zeile nicht als das Ende ";
20 echo "einer Anweisung sieht.";
21
22 ?>
23 </body>
24
25 </html>

```

**Codebeispiel 8** *mehrzeiliges\_php.php*

Der erste Abschnitt in Zeile 10 ist, obwohl PHP ihn versteht, nicht sehr übersichtlich. Sie müssen gezielt nach den Semikolons suchen, um den Beginn einer neuen Anweisung zu sehen. Der zweite Abschnitt in den Zeilen 12 bis 22 bringt jede Anweisung auf eine eigene Zeile, was es Ihnen leichter macht, den Code zu verstehen. Für PHP macht es allerdings keinen Unterschied.



Schreiben Sie jede Anweisung auf eine eigene Zeile, um die Lesbarkeit des Codes zu erhöhen. Wenn Sie mehrzeilige PHP-Anweisungen haben, schreiben Sie den öffnenden und den schließenden PHP-Tag auf jeweils eigene Zeilen.

## 3.5 Stilfragen

### 3.5.1 Saubere Trennung von HTML und PHP

In diesem Abschnitt werden Sie eines der wenigen Codebeispiele in diesem Lernbuch sehen, das eindeutig schlechten Stil darstellt. Ich bemühe mich natürlich, Ihnen nur Dinge beizubringen, die Sie später auch so benutzen sollten. Manchmal müssen Sie dafür aber auch Beispiele sehen, wie Sie es nicht machen sollten. Hier ist das erste:

#### Beispiel

```

1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8" />
6   <title>Hallo Welt</title>
7 </head>
8
9 <body>
10  <?php
11
12  echo '<form action="test.php" method="post">';
13  echo '<input type="text" name="vorname" id="vorname" value="Arthur" />';
14  echo '<input type="text" name="nachname" id="nachname" value="Dent" />';
15  echo '<input type="submit" value="abschicken" />';
16  echo '</form>';
17
18  ?>
19 </body>
20
21 </html>

```

**Codebeispiel 9** *stillos.php*

Vergleichen Sie das Beispiel eben mit dem folgenden, das exakt die gleiche Ausgabe erzeugt:

### Beispiel

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta charset="utf-8" />
6   <title>Hallo Welt</title>
7 </head>
8
9 <body>
10  <form action="test.php" method="post">
11    <input type="text" name="vorname" id="vorname" value="<?php echo 'Arthur';
?>" />
12    <input type="text" name="nachname" id="nachname" value="<?php echo 'Dent';
?>" />
13    <input type="submit" value="abschicken" />
14  </form>
15 </body>
16
17 </html>
```

**Codebeispiel 10** *stilvoll.php*

Im ersten Listing wird ein komplettes HTML-Formular mit mehreren `echo`-Anweisungen ausgegeben. Das ist prinzipiell möglich, da `echo` jede Art von Inhalten ausgeben kann, auch HTML-Formatierungen. Diese Herangehensweise hat allerdings mehrere Nachteile, von denen einige bereits jetzt offensichtlich sind:

- ▶ Da Sie wahrscheinlich (hoffentlich) mit einem Editor arbeiten, der Syntax-Highlighting beherrscht, wird auch der HTML-Code farblich hervorgehoben, z. B. werden Attribute und ihre Werte unterschiedlich dargestellt. Wenn Sie einen HTML-Tag in eine `echo`-Anweisung verpacken, verlieren Sie in einigen Editoren diesen Vorteil. Alles wird in einer Farbe dargestellt, was das Auffinden von Fehlern erschwert.
- ▶ Sie müssen alle Texte, die Sie mit `echo` ausgeben, in Anführungszeichen einschließen. Ich werde später noch im Detail darauf eingehen, aber Sie dürfen dieselben Anführungszeichen nicht mehr im Text verwenden, die Sie zum Einschließen verwendet haben. In unserem Beispiel wäre Folgendes also nicht erlaubt: `<?php echo 'Arthur's'; ?>`
- ▶ Jedes Zeichen, das Sie statt direkt im HTML mit `echo` ausgeben, sorgt dafür, dass die Seite langsamer lädt, da PHP mehr Daten verarbeiten muss. Der Unterschied mag bei kleinen Seiten nicht ins Gewicht fallen, aber auch dort gibt es keinen Grund, den Server unnötig zu belasten.
- ▶ Es ist schlicht und ergreifend hässlich!

Wenn Sie sich fertige PHP-Programme aus dem Internet herunterladen, werden Sie gerade bei älteren Projekten noch viel Code sehen, der wie die erste Version aussieht. Inzwischen ist man in der PHP-Community allerdings komplett von dieser Vorgehensweise abgerückt. Viele Projekte jüngerer Datums akzeptieren gar keinen Code mehr, der irgendwelche HTML-Formatierungen mit PHP-Anweisungen ausgibt. Sie werden später noch wesentlich mehr zu diesem Thema hören, also gewöhnen Sie es sich am besten gar nicht erst falsch an:

Versuchen Sie, wo immer möglich, HTML-Code und PHP-Code getrennt zu halten. HTML-Tags mit PHP-Anweisungen auszugeben, ist extrem schlechter Stil und behindert die Übersichtlichkeit Ihres Codes.



### 3.5.2 Einrückungen

Ein zweiter Punkt beim Thema Stil sind Code-Einrückungen. Vergleichen Sie die beiden Beispiele:

#### Beispiel

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title>Hallo Welt</title>
7 </head>
8
9 <body>
10     <?php
11
12         echo "So sieht das schon viel besser aus. ";
13 echo "Wenn Sie jede Anweisung auf eine Zeile schreiben, ";
14     echo "behalten Sie leichter den Überblick. ";
15     echo "Trotzdem müssen Sie immer noch ";
16         echo "ein Semikolon nach jeder Anweisung schreiben, ";
17     echo "da PHP eine neue Zeile nicht als das Ende ";
18 echo "einer Anweisung sieht.";
19
20     ?>
21 </body>
22
23 </html>
```

**Codebeispiel 11** schlechter\_stil.php

#### Beispiel

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <meta charset="utf-8" />
6     <title>Hallo Welt</title>
7 </head>
8
9 <body>
10     <?php
11
12     echo "So sieht das schon viel besser aus. ";
13     echo "Wenn Sie jede Anweisung auf eine Zeile schreiben, ";
14     echo "behalten Sie leichter den Überblick. ";
15     echo "Trotzdem müssen Sie immer noch ";
16     echo "ein Semikolon nach jeder Anweisung schreiben, ";
17     echo "da PHP eine neue Zeile nicht als das Ende ";
18     echo "einer Anweisung sieht.";
19
20     ?>
21 </body>
22
23 </html>
```

**Codebeispiel 12** guter\_stil.php

Sie sollten auf den ersten Blick sehen, was im oberen Beispiel falsch gelaufen ist. In PHP gelten im Prinzip dieselben Regeln wie in HTML: Sie sollten alles einheitlich formatieren und einrücken. Wenn Sie mehrere `echo`-Anweisungen nacheinander schreiben, sollten alle gleich eingerückt sein.

Wenn Sie sich jetzt fragen, warum ich das überhaupt erwähne, seien Sie versichert: Solange Ihnen das Einrücken nicht in Fleisch und Blut übergegangen ist, werden Sie sich immer wieder dabei ertappen, dass Ihr Code nicht einheitlich formatiert ist.

Sobald Sie mehr PHP-Anweisungen beherrschen, werde ich noch mehr Beispiele zu diesem Thema bringen. Merken Sie sich aber schon mal:

Formatieren Sie Ihren Code immer auf die gleiche Weise. Der (in)offizielle PHP-Standard ist es, jede Ebene um zusätzliche **vier Leerzeichen** einzurücken. Viele Editoren unterstützen Sie dabei, indem Sie einen einzelnen Tab in vier Leerzeichen umwandeln. Die einzige Ausnahme bildet bei einer mehrzeiligen PHP-Anweisung die erste Ebene, welche wir nicht (weiter) einrücken. Diese ist somit bündig zum öffnenden und schließenden PHP-Tag.



Noch besserer Stil ist es übrigens, wenn man statischen Text gar nicht erst mit `echo` ausgibt, aber zu den variablen Inhalten kommen wir erst in der nächsten Lektion.

## 3.6 Zusammenfassung

In dieser Lektion haben Sie die ersten Gehversuche in PHP unternommen. Sie haben gelernt, dass Sie mit den Tags `<?php` und `?>` PHP-Code direkt in HTML einbetten können. Sie haben erfahren, dass Sie mehrere Anweisungen mit dem Semikolon trennen müssen, und welchen Zweck die `echo`-Anweisung erfüllt.

Zusätzlich haben Sie aber auch schon jetzt die ersten Hinweise erhalten, wie Sie sauberen Code schreiben, der Ihnen später keinen Ärger macht. Je früher Sie sich an diese Tipps halten, desto besser, denn einmal falsch gelernte Angewohnheiten sind nur schwer wieder loszuwerden. Auch wenn Ihnen das im Moment vielleicht übertrieben erscheint, werden Sie später sehr dankbar sein, sich nicht umgewöhnen zu müssen.

## 3.7 Testen Sie Ihr Wissen!

1. Wie können Sie PHP-Code in HTML einbetten?
2. Wofür ist die `echo`-Anweisung gut?
3. Was müssen Sie bei der Kombination von HTML und PHP beachten?
4. Wie können Sie mehrere Anweisungen in einem PHP-Tag unterbringen?
5. Warum sollten Sie HTML-Tags nicht mit `echo` ausgeben?
6. Welchen Vorteil hat es, Code einheitlich einzurücken?

## 3.8 Übungen

### Übung 1:

Legen Sie einen Unterordner *lektion03* im Ordner *phpschulung* an. Speichern Sie dort ein HTML-Dokument namens *spass.php*, das im `<body>`-Tag den Text `PHP macht Spaß!` stehen hat.

**Hinweis:** Legen Sie fortan für alle Übungen einer Lektion zunächst einen entsprechenden Unterordner an.

### Übung 2:

Verändern Sie das Dokument. Der Text `PHP macht Spaß!` soll nun mit der PHP-Anweisung `echo` ausgegeben werden.

### Übung 3:

Öffnen Sie nun die Datei `spass.html` direkt im *Firefox*-Browser über **Datei** → **öffnen** und beobachten Sie, was passiert.

**Hinweis:** Seit Version 4 ist im *Mozilla Firefox* die Menüleiste standardmäßig deaktiviert. Sie kann jedoch durch Drücken der (linken) Alt-Taste eingeblendet werden. Alternativ lässt sich die Datei auch einfach auf den Anzeigebereich des Browsers ziehen, wodurch sie im Browser geöffnet wird.

### Übung 4:

Öffnen Sie die Datei nun über den Webserver, also über `http://localhost/phpschulung/lektion03/spass.php`, und beobachten Sie, was passiert.

### Übung 5:

Benennen Sie die Datei in `spass.html` um, öffnen Sie sie über `http://localhost/phpschulung/lektion03/spass.html` und beobachten Sie den Unterschied in der Darstellung.

## 3.9 Optionale Übungen

### Übung 6:

Schreiben Sie ein HTML-Dokument mit dem Namen `formular.php`, das ein Formular mit den folgenden Feldern enthält:

- ▶ `<input type="text" name="benutzername" value="" />`,
- ▶ `<input type="password" name="passwort1" value="" />`,
- ▶ `<input type="password" name="passwort2" value="" />` und
- ▶ `<input type="submit" value="registrieren" />`.

### Übung 7:

In dem Feld `benutzername` soll nun standardmäßig "Mr. X" stehen, was über das `value`-Attribut möglich ist. Der Text soll dort mittels einer PHP-`echo`-Anweisung eingetragen werden.

### Übung 8:

In den beiden `password`-Feldern sollen jeweils fünf Sterne (`*****`) stehen. Auch diese sollen dort mit `echo` eingetragen werden. Wie werden die Sterne im Browser dargestellt?