

## S3. PHP

### S3.1 Einleitung

S

EWS-4.1

#### Entstehung und Einsatzziele:

- ursprünglich 1994 von Rasmus Lerdorf entwickelt; PHP: **Personal Home Page Tools**
- **Skriptsprache, vieles übernommen von Perl**, einiges vereinfacht
- PHP-Programme werden **eingebettet** in Texte anderer Sprachen - meist HTML
- wichtigster Einsatz: Programme auf Web-Servern (siehe Beispiel Telefonbuch)
- **Interpretation mit interner Analyse** der Programmfragmente vor der Ausführung

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 401

#### Ziele:

Sprache PHP einordnen

#### in der Vorlesung:

Hinweise auf

- Historie von Sprachen (Folie 2.21),
- Eigenschaften von Skriptsprachen (Folie 2.18),
- Konsequenzen der Interpretation mit Analyse für Spracheigenschaften (Folie 2.16)

## Charakteristika der Sprache PHP

S

EWS-4.2

- Notation an C und Perl orientiert
- **einfache Programmstrukturen**
- einfache Regeln zur **statischen Bindung von Bezeichnern**
- wenige, einfache Typen, **dynamisch typisiert**
- wichtigste **Operationen auf Zeichenreihen, Zahlwerten und Arrays**
- Definition und Aufruf **parametrisierter Funktionen**
- sehr große Menge **vordefinierter nützlicher Funktionen**
- **Klassen und Objekte** zur übersichtlichen Formulierung von komplexeren Strukturen

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 402

#### Ziele:

Spracheigenschaften einordnen

#### in der Vorlesung:

Hinweise auf

- statische und dynamische Semantik (Folie 2.13, 2.14),
- kommende Themen: Programmstrukturen, Typen und Operationen, Arrays, Funktionen und Aufrufe

## Ausführung von PHP-Programmen

Ein PHP-Interpreter verarbeitet jeweils eine Datei, in die ein **PHP-Programm eingebettet** ist. Es kann aus **mehreren Stücken** bestehen. Sie werden jeweils vom umgebenden Text **abgegrenzt** durch:

```
<?php ... PHP-Programmstück ... ?>
```

auch mehrzeilige Programmstücke, auch mehrere Stücke:

### eingebettete Programmstücke:

Warnung auf Englisch:

```
<?php
echo "Beware of the dog!"
?>
```

Warnung auf Deutsch:

```
<?php
echo "Bissiger Hund!"
?>
Aufpassen!
```

### Ergebnis der Ausführung:

Warnung auf Englisch:

Beware of the dog!

Warnung auf Deutsch:

Bissiger Hund!

Aufpassen!

Die umgebenden Texte werden übernommen; an den Stellen der **Programmstücke** wird deren **Ausgabe** eingesetzt.

### PHP-Interpreter

- kann **direkt mit der Programmdatei** aufgerufen werden,
- ist **auf Web-Server** installiert, verarbeitet eine Seite beim Anfordern der URL

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 403

### Ziele:

Einbettung und Ausführung verstehen

### in der Vorlesung:

An Beispiel wird erklärt:

- Kennzeichnung von Programmstücken,
- andere, erlaubte Kennzeichnungen erklären wir hier nicht,
- Programm besteht aus mehreren Stücken, die zusammengehören,
- Ausgabe im umgebenden Text,
- 2 Arten PHP-Programme auszuführen,
- Beispiel Warnung vor dem Hunde,
- Beispiel Dreieck aus Sternen.

## Ein zweites Beispiel als Eindruck von PHP

### PHP-Programm:

```
<?php
// ein Dreieck aus *-Zeichen
$line = 1;
while ($line < 16) {
    $col = 1;
    while ($col <= $line) {
        echo "*";
        $col = $col + 1;
    }
    echo "\n";
    $line = $line + 1;
}
?>
```

### Ausgabe dazu:

```
*
**
***
****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 404

### Ziele:

Vorschau auf PHP-Programmkonstrukte

### in der Vorlesung:

Am Beispiel wird hingewiesen auf:

- Variable und Zuweisungen,
- Schleifen mit Bedingungen,
- Ausgabeanweisungen.
- Die Ausführung dieses Programmes wird gezeigt.

## S3.2 Variable und Zuweisungen

EWS-4.5

Eine **Variable speichert Werte** während der **Ausführung** eines Programmes.

Eine **Variable** wird präzise beschrieben durch 3 Dinge

- **Name** im Programm
- **Speicherstelle** bei der Ausführung des Programmes
- **Wert** an der Speicherstelle zu einem bestimmten Zeitpunkt der Programmausführung

`$line`

42

### Notation von Variablenamen im Programm:

\$ als Kennzeichen einer Variablen - zur Unterscheidung von Funktionen und Typen - gefolgt von einem Namen der Form `[a-zA-Z_][a-zA-Z_0-9]*`

- Eine Variable kann **Werte beliebiger Typen** aufnehmen.
- Der Wert einer Variablen wird durch Ausführen von Zuweisungen verändert (s. 4.6)  
`$line = 1;`
- In Ausdrücken werden Werte von Variablen für Berechnungen benutzt.  
`$line < 16`
- Variable brauchen in PHP nicht deklariert zu werden

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 405

### Ziele:

Den Begriff der Variablen verstehen

### in der Vorlesung:

An Beispielen wird erklärt:

- die Bedeutung von Name, Stelle, Wert,
- Notation,
- Eigenschaften von Variablen in PHP.
- Gültigkeitsregeln werden später besprochen.

## Zuweisungen

EWS-4.6

Eine Zuweisung hat die Form:

`Variable = Ausdruck;`

z. B. `$line = 1;`

Eine **Zuweisung wird ausgeführt** durch folgende Schritte

- die **Speicherstelle** der Variablen wird bestimmt,
- der Ausdruck wird ausgewertet und liefert einen **Wert**,
- der **Wert wird** an der Speicherstelle der Variablen **gespeichert** (ersetzt den Wert, der bisher dort stand).

Auf der **linken Seite einer Zuweisung** bezeichnet die Variable die **Speicherstelle**, an die zugewiesen wird.

`$line = 1;`

In einem **Ausdruck** steht die Variable für den **Wert**, den ihre Speicherstelle gerade enthält. `$line < 16`

Ausführung einer Folge von Zuweisungen

Werte im Speicher an der Stelle von a b

```
$a = 1;           1  null
                 1  7
$b = 7;           1  7
                 9  7
$a = $b + 2;     9  7
                 9  8
$b = $b + 1;     9  8
                 18 8
$a = $a * 2;     18 8
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 406

### Ziele:

Zuweisungen verstehen

### in der Vorlesung:

An Beispielen wird erklärt:

- Struktur von Zuweisungen,
- Ausführungsschritte,
- Variable in 2 verschiedenen Rollen,
- Zustände beim Ausführen einer Folge von Zuweisungen.
- Anfangs hat jede Variable den speziellen Wert null.

### S3.3 Ausdrücke

S EWS-4.7

Die **Auswertung eines Ausdrucks** liefert einen **Wert eines bestimmten Typs**.

`$anzahl + 5`      `$i * 3 <= 100`

#### Elementare Ausdrücke:

- **Literal:** Notation gibt den Wert an, z. B. `42`    `"Hello"`
- **Variable:** hat einen Wert, z. B. `$anzahl`
- **Aufruf einer Funktion:** liefert einen Wert als Ergebnis, z. B. `abs($konto)`

#### Ausdrücke mit Operatoren

- **2-stellig** mit 2 Operanden (Ausdrücke), z. B. `$anzahl + 5`    `$zeile . ""`
- **1-stellig** mit 1 Operanden (Ausdruck), z. B. `! $gefunden`

Der Operator verknüpft die Werte der Operanden zum Wert des Ausdrucks.

### Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 407

#### Ziele:

Auswertung von Ausdrücken verstehen

#### in der Vorlesung:

An Beispielen wird erklärt:

- Formen von Ausdrücken,
- rekursiver Aufbau,
- Auswertung von Teilausdrücken

### Präzedenz und Assoziativität von Operatoren

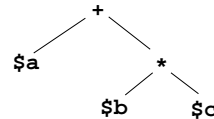
S EWS-4.8

Ein Operator mit höherer Präzedenz bindet seine Operanden stärker als ein Operator mit niedrigerer Präzedenz.

z. B. `*` hat höhere Präzedenz als `+`; deshalb hat der Ausdruck `$a + $b * $c` die Struktur:

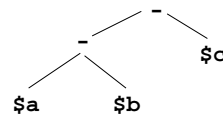
Wenn man trotzdem erst `$a + $b` verknüpfen wollte, müsste man dies durch Klammerung ausdrücken:

`( $a + $b ) * $c`



Ein Operator ist **linksassoziativ**, wenn beim Aufeinandertreffen von Operatoren gleicher Präzedenz der linke seine Operanden stärker bindet als der rechte. (Entsprechendes gilt für **rechtsassoziativ**.)

Beispiel: `-` ist linksassoziativ; deshalb hat `$a - $b - $c` die Struktur:



### Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 408

#### Ziele:

Präzedenz und Assoziativität verstehen

#### in der Vorlesung:

Die Begriffe werden an Beispielen erklärt:

- Präzedenz,
- Assoziativität,
- i.a. bestimmt die Struktur auch den Wert des Ausdrucks.

## Präzedenz und Assoziativität aller Operatoren in PHP

Präzedenz	Assoziativität	Operatoren
1	left	,
2	left	or
3	left	xor
4	left	and
5	right	print
6	right	= += -- *= /= .= &=  = ^= <<= >>=
7	left	?:
8	left	
9	left	&&
10	left	
11	left	^
12	left	&
13	non-ass.	== != === !==
14	non-ass.	< <= > >=
15	left	<< >>
16	left	+ - .
17	left	* / %
18	right	! ~ ++ -- @ (int) (float) (string) ...
19	right	[
20	non-ass.	new

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 409

### Ziele:

Zum Nachschlagen

### in der Vorlesung:

Die Liste dient zum Nachschlagen. Nicht alle Operatoren werden besprochen.

## Arithmetische Datentypen und Operationen

<b>Datentypen:</b>	<code>int</code>	ganze Zahlen
	<code>float</code>	Gleitpunktzahlen, d. h. reelle Zahlen mit begrenzter Genauigkeit
<b>int-Literale:</b>	<code>-127</code>	<b>dezimal</b> ; Schreibweise: <code>[+-]?([1-9][0-9]* 0)</code>
	<code>064</code>	<b>oktal</b> <code>[+-]?(0[0-7]+)</code>
	<code>0x1A</code>	<b>hexadezimal</b> <code>[+-]?(0[xX][0-9a-fA-F]+)</code>
<b>float-Literale:</b>	<code>1.234</code>	
	<code>1.2e3</code>	Wert ist $1.2 * 10^3 = 1200$
	<code>7E-3</code>	Wert ist $7 * 10^{-3} = 0.007$

**Operatoren:** + - \* / %

% bedeutet **modulo**, d. h. `$a % $b` ergibt den Rest der Division von `$a` durch `$b`.  
z. B. `37 % 10` ergibt 7.

Verknüpfung **zweier int-Werte** liefert einen `int`-Wert, z. B. `7 * 5` liefert 35;

Ausnahmen: / liefert immer einen `float`-Wert,  
wenn der Wert nicht mehr als `int`-Wert darstellbar ist

Ist mindestens **ein Operand ein float-Wert**, so ist das Ergebnis ein `float`-Wert,  
z. B. `2.5 * 2` ergibt (etwa) 5.0

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 410

### Ziele:

Literale und Operatoren anwenden können

### in der Vorlesung:

An Beispielen wird erklärt:

- Bedeutung der Datentypen,
- Schreibweise der Literale,
- Bedeutung der Operatoren.

## Datentyp `string` für Zeichenreihen

EWS-4.11

Ein Wert vom Typ `string` ist eine beliebig lange Folge von Zeichen des ASCII-Zeichensatzes, auch **Zeichenreihe** genannt.

3 Notationen für **Literale**:

- mit `'` geklammert:  
Alle Zeichen stehen für sich selbst; nur für `'` muss `\'` geschrieben werden, z. B.

```
'Hello World!'    'Mary\'s car'
```

- mit `"` geklammert:  
Es können darin Umschreibungen und Variable vorkommen, z. B.

```
"Summe $jahr =\t${betrag}EUR"
```

```
$jahr      Variable, ihr Wert wird eingesetzt  
${betrag}  ebenso; zur Abgrenzung des Namens vom umgebenden Text  
\n         Zeilenwechsel  
\n         "                und viele mehr ...
```

- „Heredoc“ für nicht-leere Folgen von Zeilen als Zeichenreihe; Umschreibungen wie bei `"`

```
print <<<EOS  
  Frohe  
  Weihnachten  
EOS;
```

`EOS` ist ein **frei wählbarer Bezeichner**; `<<<EOS` steht am Zeilenende, das schließende `EOS` am Zeilenanfang, danach höchstens ein `;`

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 411

**Ziele:**

Notation von `string`-Literalen

**in der Vorlesung:**

`string`-Literale und Umschreibungen werden an Beispielen erklärt.

## Operationen mit Zeichenreihen

EWS-4.12

**Konkatenationsoperator** `.` verknüpft zwei Zeichenreihen zu einer neuen, z. B.

```
"Hello " . "world!"    $Sterne = $Sterne . "!!";
```

Wenn die Variable `$str` einen `string`-Wert hat, **indiziert** `$str{$i}` das **(i+1)-te Zeichen** darin, z. B.

```
$Sterne{3} = "!!";
```

**Ausgabe von Zeichenreihen:**

```
echo Folge von Ausdrücken (durch Kommata getrennt),  
deren Auswertung Zeichenreihen liefert;
```

```
print Ausdruck, dessen Auswertung eine Zeichenreihe liefert;
```

```
echo $Sonne, $Mond, $Sterne;
```

```
print "Summe $jahr =\t${betrag}EUR";
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 412

**Ziele:**

`string`-Operationen kennenlernen

**in der Vorlesung:**

Konkatenation, Indizierung und Ausgabe werden an Beispielen erklärt.

## Datentyp bool für logische Operationen

Der **Datentyp bool** (oder auch **boolean**) hat die beiden **Literale true** und **false** (geschrieben mit Groß- oder Kleinbuchstaben).

Logische Werte werden insbesondere für Bedingungen in Schleifen und bedingten Anweisungen benötigt, z. B.

```
if ($alter < 14 or $alter > 65) print "Rabatt";
```

### Logische Operatoren:

**\$a and \$b** true, falls beide **\$a** und **\$b** true sind

**\$a && \$b** ebenso

**\$a or \$b** true, falls **\$a** oder **\$b** oder beide true sind

**\$a || \$b** ebenso

**\$a xor \$b** true, falls genau einer von **\$a** und **\$b** true ist

**!\$a** true, falls **\$a** false ist

### Vergleichsoperatoren liefern Werte vom Typ bool:

==	gleich	<	kleiner
!=	ungleich	>	größer
<>	ungleich	<=	kleiner oder gleich
		>=	größer oder gleich

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 413

### Ziele:

Logische Operationen verstehen

### in der Vorlesung:

Die Operationen werden an Beispielen erklärt.

## Konversion: Umwandlung von Werten

### Konversion:

Ein **Wert eines Typs** wird in einen „entsprechenden“ **Wert eines anderen Typs** umgewandelt.

- **explizite Konversion (engl. type cast):**

Der **Zieltyp**, in den der Wert eines Ausdruckes umgewandelt werden soll, wird **explizit angegeben**, z. B.

```
(string)(5+1) liefert die Zeichenreihe "6"
```

- **implizite Konversion (engl. coercion):**

Wenn der Typ eines Wertes nicht zu der darauf angewandten Operation passt, wird **versucht, ihn anzupassen**, z. B.

```
$sum = 42; print "Summe = " . $sum;
```

Die ganze Zahl 42 wird in die Zeichenreihe "42" konvertiert.  
(Der Wert der Variablen **\$sum** bleibt unverändert.)

In PHP kommt man mit impliziter Konversion weitgehend aus.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 414

### Ziele:

Das Prinzip Konversion verstehen

### in der Vorlesung:

Konversion wird an Beispielen erklärt.

## Verfügbare Konversionen

`int -> float` ganze Zahl in Gleitpunktzahl mit Exponent,  
ggf. Genauigkeitsverlust:  
`2000000001*2000000001` liefert `4.000000004E+18`

`float -> int` abrunden (Richtung 0): `(int)-3.6` liefert `-3`

`int -> string` Wert als Zeichenreihe: `echo 42, 3.2;`  
`float -> string`

`string -> int` aus dem Anfang der Zeichenreihe wird versucht,  
`string -> float` einen Zahlwert zu lesen: `1 + "10.5"` liefert `11.5`  
`1 + "Meier8"` liefert `1`

`bool -> int` `false -> 0`, und `true -> 1`  
`int -> bool` `0 -> false`, alle anderen Werte `-> true`

weitere durch Zusammensetzen

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 415

### Ziele:

Konversionen verstehen

### in der Vorlesung:

Die Konversionen werden an Beispielen erklärt.

## S3.4 Ablaufstrukturen

Ausführbare Programmteile werden **aus Anweisungen zusammengesetzt**. Durch **Bedingungen** und **Verzweigungen** können je nach dem Ergebnis von Berechnungen **unterschiedliche Abläufe** durch die Programmstruktur ausgeführt werden.

Zu folgenden **algorithmischen Grundelementen** gibt es in jeder imperativen Programmiersprache jeweils einige Anweisungsformen:

Beispiele aus PHP

- **Zuweisung** `$st = $st . "*";`
- **Anweisungsfolge** `{ $st = $st . "*"; $i = $i+1; }`
- **Alternativen** `if ($i > 100) echo "zu groß"; else echo "ok";`
- **Schleife** `while ($i < 20) { $st = $st . "*"; $i = $i+1; }`
- **Funktionsaufruf** `fclose ($out);`

Meist gibt es

- mehrere Formen für Schleifen und Alternativen,
- unterschiedliche Notationen der Anweisungen und
- Anweisungsformen für weitere Ablaufstrukturen.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 416

### Ziele:

Übersicht über elementare Ablaufstrukturen

### in der Vorlesung:

Grundelemente und Zusammensetzen an den Beispielen erklärt.



## Grammatik für elementare Anweisungsformen in PHP

### Statement

```

::= Variable '=' Expression ';'
|   '{' Statement* '}'
|   'if' '(' Expression ')' Statement [ 'else' Statement ]
|   'while' '(' Expression ')' Statement
|   FunctExpr '(' [ Parameters ] ')' ';'

```

FunctExpr ::= FunctName | ...

Parameters ::= Parameters ',' Expression | Expression

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 417

### Ziele:

Notation der Anweisungsformen verstehen

### in der Vorlesung:

An Beispielen wird erklärt:

- Notation der Anweisungsformen,
- Rolle der geklammerten Anweisungsfolge,
- rekursiver Aufbau von Ablaufstrukturen.
- Diese Notation entspricht der in C, C++ und Java.

## Bedingte Anweisung

einseitig: `if ( Bedingung ) Then-Teil`

zweiseitig: `if ( Bedingung ) Then-Teil else Else-Teil`

Die **Bedingung** wird ausgewertet. Wenn sie `true` liefert, wird der **Then-Teil** ausgeführt; liefert sie `false`, wird in der zweiseitigen Form der **Else-Teil**, in der einseitigen **nichts** ausgeführt.

### Beispiele:

```
if ($a < 0) { $a = -$a; }
```

```
if ($a > $b) {
    echo "a ist größer als b";
} else {
    echo "a ist nicht größer als b";
}
```

### Stilregel:

Die Alternativen der bedingten Anweisung immer als **geklammerte Folge** schreiben - auch wenn es nur einzelne Anweisungen sind!

Verzweigung über mehrere Bedingungen in verschiedene Fälle, `if`-Kaskade:

```
if      ($jahr % 4 != 0)  { $tage = 365; }
else if ($jahr % 100 != 0) { $tage = 366; }
else if ($jahr % 400 != 0) { $tage = 365; }
else                                     { $tage = 366; }
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 418

### Ziele:

Bedingte Anweisung anwenden können

### in der Vorlesung:

An Beispielen wird erklärt:

- die Ausführung,
- Schachtelung,
- Klammerung und
- Probleme, wenn Sie fehlt.

## Iterative Berechnungen mit `while`-Schleifen

`while` ( Bedingung ) Schleifenrumpf

Die **Bedingung** wird **ausgewertet**; wenn sie `true` liefert, wird der **Schleifenrumpf** **ausgeführt** und dann die Bedingung erneut geprüft. Erst wenn sie `false` liefert, wird die **Iteration beendet**.

### Beispiel:

```
$s = 0;
while ($s < $n) {
    echo "*";
    $s = $s + 1;
}
echo "\n";
```

### Überlegungen zum Entwurf der Schleife:

`$s` gibt an, wieviele Sterne schon ausgegeben wurden.

Wenn  $0 \leq \$n$  gilt, dann ist immer  $\$s \leq \$n$ .

Nach der Schleife gilt  $\$s \leq \$n$  und  $\$s \geq \$n$  also  $\$s == \$n$

Also wurden `$n` Sterne ausgegeben.

Jede Ausführung des Schleifenrumpfes **ändert Variable in der Bedingung**.

Die Bedingung muss irgendwann `false` liefern - sonst **terminiert die Schleife** nicht.

Hinter der Schleife gilt die **Negation der Bedingung**; hier `!( $s < $n )` also `( $s >= $n )`

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 419

### Ziele:

`while`-Schleifen anwenden können

### in der Vorlesung:

Am Beispiel wird erklärt:

- Die Regeln zur Ausführung von `while`-Schleifen,
- Überlegungen zur Terminierung,
- Negation der Schleifenbedingung,

## Anwendungsmuster: Iterationen zählen

Eine **Variable** zählt die Ausführungen des Schleifenrumpfes mit.

Varianten:

- aufwärts oder abwärts zählen,
- versetzt zählen, mit Startwert  $\neq 0$ ,
- mit Schrittweite  $\neq 1$  zählen
- auch als `for`-Schleife formulierbar

**Beispiel:** Sterne ausgeben auf der vorigen Folie.

**Beispiel:** Celsius in Fahrenheit umrechnen:

```
echo "\tC \tF\n";
```

```
$c = 10;
```

```
while ($c <= 20) {
```

```
    echo "\t" . $c . "\t" .
```

```
        round ($c*9.0/5 + 32) . "\n";
```

```
    $c = $c + 1;
```

```
}
```

C	F
10	50
11	52
12	54
13	55
14	57
15	59
16	61
17	63
18	64
19	66
20	68

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 420

### Ziele:

Einfaches Iterations-Paradigma kennenlernen

### in der Vorlesung:

Am Beispiel wird erklärt:

- Charakteristika des Paradigmas,
- Variation der Zählung,
- `round` rundet kaufmännisch.

## Anwendungsmuster: Iterieren bis Zielbedingung gilt

EWS-4.21

Das Ziel ist es, dass nach **Abschluss der Iteration** eine bestimmte **Zielbedingung** gilt.

Wir zerlegen die Zielbedingung logisch in zwei Teile: **Inv** and **Halt**, sodass **Inv** vor, während und nach der Schleife gilt; die Negation von **Halt** wird zur Schleifenbedingung.

Dann entwerfen wir den Schleifenrumpf.

**Beispiel:** Dualdarstellung einer Zahl berechnen.

Methode: Iterativ durch 2 dividieren und die Reste aufschreiben.

**Zielbedingung:** `$zahl == 0` zerlegt in `$zahl >= 0 and $zahl <= 0`  
`$zahl >= 0` gilt unverändert; `!($zahl <= 0)` wird Schleifenbedingung

```
$zahl = 42;
$dual = "";
echo "$zahl dual ist ";
while ($zahl > 0) {
    $dual = (int)($zahl%2) . $dual;
    $zahl = (int)($zahl/2);
}
echo "$dual\n";
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 421

**Ziele:**

Schleifenbedingungen systematisch herleiten

**in der Vorlesung:**

Am Beispiel wird erklärt:

- das Paradigma,
- die Idee zur Berechnung der Dualzahlen,
- die Umformung der Bedingung,
- die Schleife,
- Beispielrechnungen.

## Anwendungsmuster: Suchschleife

EWS-4.22

Die **Elemente einer Folge** werden durchsucht, bis das **Gesuchte gefunden** ist oder **alle Elemente vergeblich untersucht** wurden.

Die Schleife hat **2 verschiedenartige Ergebnisse**:

gefunden oder Folge ist durchsucht

Sie müssen nach der Schleife unterschiedlich behandelt werden.

Die **Schleifenbedingung** lautet:

nicht gefunden und Folge ist noch nicht durchsucht

**Beispiel:**

Position des ersten

Auftretens eines bestimmten Zeichens in einer Zeichenreihe suchen.

```
$str = "Ein # und noch ein #";
$lg = strlen($str); $zeichen = "#";
$i = 0; $gefunden = false;

while (!$gefunden and $i < $lg) {
    if ($str{$i} == $zeichen)
        { $gefunden = true; }
    else { $i = $i + 1; }
}

if ($gefunden)
    { echo "$zeichen an Position $i\n"; }
else { echo "$zeichen kommt nicht vor\n"; }
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 422

**Ziele:**

Suchschleifen schreiben können

**in der Vorlesung:**

Am Beispiel wird erklärt:

- 2 Ergebnisse,
- Konsequenzen daraus,
- Indizierung von Zeichenreihen,
- entsprechend auch für die Suche in Dateien, Eingabe oder Arrays.

## Funktionsaufrufe

S

EWS-4.23

Eine **Funktion definiert eine Berechnung**, die mit bestimmten (formalen) **Parametern** ausgeführt werden kann. Die Berechnung kann ein **Ergebnis** liefern.

Der **Aufruf einer Funktion** führt die definierte Berechnung aus mit den (aktuellen) **Parameterwerten**, die beim Aufruf angegeben sind.

### Beispiel:

Die Funktion `strlen` berechnet die Länge einer Zeichenreihe, die als Parameter angegeben wird. Der Aufruf `strlen("abc")` liefert als Ergebnis 3

Funktionen, die ein Ergebnis liefern, können in Ausdrücken aufgerufen werden:

```
$lg = strlen("abc");    $p = strpos($str, $zeichen);
```

Aufrufe haben die **Form**: `FuncExpr ( [ Parameters ] )`

Ein **Aufruf wird in folgenden Schritten ausgewertet**:

1. **FuncExpr auswerten** (ist meist nur ein Name) liefert eine Funktion.
2. **Parameter auswerten** und an die Funktion **übergeben**.
3. **Berechnung der Funktion** mit den übergebenen Parameterwerten ausführen; **Ergebnis** an die Aufrufstelle **zurückgeben**.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 423

### Ziele:

Aufrufe von Funktionen verstehen

### in der Vorlesung:

An Beispielen wird erklärt:

- parametrisierte Berechnungen,
- Rolle der Parameter: formal - aktuell,
- Aufruf auswerten,
- unterscheiden: Funktion(sdefinition) und Aufruf,
- PHP: viele, nützliche Funktionen sind vordefiniert.

## Bibliothek von Funktionen zu PHP

S

EWS-4.24

Zu PHP gibt es eine sehr große **Bibliothek mit zahlreichen nützlichen Funktionen** zu vielen Themen. Sie können in jedem PHP-Programm aufgerufen werden.

**Beschreibungen** der Funktionen findet man z. B. im PHP-Manual (siehe URL).

Einige der **Themen** sind:

### Arrays

Konfiguration und Protokolle

Datenbanken

Datum/Uhrzeit

Dateiverzeichnisse

### Dateien

Grafik

HTTP

IMAP

LDAP

### Mathematik

MCAL

Mcrypt

Mhash

PDF

POSIX

### Strings

Variablenmanipulation

XML

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 424

### Ziele:

Funktionsbibliothek kennenlernen

### in der Vorlesung:

- Hinweis auf einige der Themen und auf das Manual:
- PHP Manual (englisch)
- PHP Handbuch (deutsch)

## string-Funktionen aus der Bibliothek

EWS-4.25

Beschreibung einer Bibliotheksfunktion:

<b>Name und Zweck</b> der Funktion:	<code>strtr</code> -- Tauscht bestimmte Zeichen aus
<b>Typen der Parameter</b> und des <b>Ergebnisses</b> :	<code>string strtr ( string str, string from, string to)</code>
Beschreibung der <b>Wirkung</b> :	Diese Funktion erzeugt aus <code>str</code> einen neuen String als Ergebnis, indem alle Vorkommen von Zeichen aus <code>from</code> in die entsprechenden Zeichen aus <code>to</code> umgesetzt werden. Sind <code>from</code> und <code>to</code> von unterschiedlicher Länge werden die überzähligen Zeichen ignoriert.
<b>Beispiel</b> für einen Aufruf:	<code>\$addr = strtr(\$addr, "äöü", "aou");</code>

Weitere `string`-Funktionen:

`str_replace` -- Ersetzt alle Vorkommen eines Strings in einem anderen String

`strcmp` -- lexikographischer Vergleich zweier Strings

`strpos` -- Sucht erstes Vorkommen des Suchstrings und liefert die Position

`strtolower` -- Setzt einen String in Kleinbuchstaben um

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 425

**Ziele:**

Funktionsbeschreibungen verstehen

**in der Vorlesung:**

Am Beispiel `strtr` wird erklärt:

- Funktionsbeschreibung,
- Typen der Parameter und des Ergebnisses,
- Suche von passenden Funktionen in der Bibliothek

## S3.5 Ein- und Ausgabe mit Dateien

EWS-4.26

1. Eine **Datei speichert Daten** im Dateisystem des Rechners.
2. Dateien werden bei der **Ausführung von Programmen** geschrieben und gelesen.
3. Die Dateien sind **persistent**, d. h. sie **bleiben erhalten** - auch nach Ausführung des Programms, das sie geschrieben hat.
4. Der **Inhalt** einer Datei kann als eine (sehr lange) **Zeichenreihe** (`string`) aufgefasst werden. Zeichen für Zeilenwechsel **gliedern sie in Zeilen**. (Wir betrachten hier nur solche Dateien; es gibt auch andere: sog. Binär-Dateien, ihr Inhalt ist speziell strukturiert und codiert).
5. **Im Dateisystem** wird eine Datei durch den **Pfad** und den **Dateinamen** identifiziert, z. B. `/home/rasmus/file.txt`
6. **Im Programm** wird eine Datei durch einen **Dateizeiger** identifiziert. Er wird beim **Öffnen der Datei** erzeugt.
7. Der **Dateizeiger** gibt auch die **Position innerhalb der Datei** an, wo gerade gelesen oder geschrieben wird.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 426

**Ziele:**

Grundbegriffe zu Dateien verstehen

**in der Vorlesung:**

Die Begriffe werden erklärt.

## Funktionen zum Öffnen und Schließen von Dateien

EWS-4.27

**fopen** -- Öffnet eine Datei oder URL

```
resource fopen (string filename, string mode...)
```

**mode** spezifiziert den gewünschten Zugriffstyp:

"r" zum Lesen vom Anfang der Datei

"w" zum Schreiben vom Anfang der Datei

"a" zum Weiterschreiben am Ende der Datei

```
$handle = fopen ("/home/rasmus/file.txt", "r");
```

**feof** -- Prüft, ob der Dateizeiger am Ende der Datei steht

```
bool feof (resource handle)
```

Gibt **TRUE** zurück, falls der Dateizeiger am Ende der Datei steht oder ein Fehler aufgetreten ist, andernfalls **FALSE**.

**fclose** -- Schließt einen offenen Dateizeiger

```
bool fclose (resource handle)
```

Die Datei, auf die **handle** zeigt, wird geschlossen.  
Gibt bei Erfolg **TRUE** zurück, im Fehlerfall **FALSE**.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 427

**Ziele:**

Öffnen und Schließen von Dateien verstehen

**in der Vorlesung:**

Konzepte und Funktionen werden erklärt:

- Öffnen und Schließen,
- Zugriffstypen,
- Erzeugen von Ausgabedateien,
- Position des Dateizeigers.

## Schema: eine Ausgabedatei schreiben

EWS-4.28

**fputs** -- Schreibt Daten an die Position des Dateizeigers

```
int fputs (resource handle, string str [, int length])
```

**fputs** schreibt den Inhalt einer Zeichenkette **string** in die Datei, auf welche der Dateizeiger **handle** zeigt. Wenn der **length** Parameter gegeben ist, wird das Schreiben nach **length** Bytes beendet, oder wenn das Dateiende (EOF) erreicht ist, je nachdem, was eher eintritt.

**fputs** gibt bei Erfolg die Anzahl der geschriebenen Bytes zurück, andernfalls **FALSE**.

```
// ein dreieck aus *-Zeichen schreiben
$out = fopen ("Sterne.txt", "w");
if (!$out) { echo "Sterne.txt nicht geöffnet"; exit; }
$line = 0;
while ($line < 15) {
    $col = 0; $str = "";
    while ($col < $line) {
        $str = $str . "*"; $col = $col + 1;
    }
    $str = $str . "\n"; $line = $line + 1;
    fputs ($out, $str);
}
fclose ($out);
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 428

**Ziele:**

Ausgabeschema anwenden können

**in der Vorlesung:**

Am Beispiel wird erklärt:

- Öffnen und Schließen,
- Schreiben in Datei,
- Schema wiederverwenden,

## Schema: Eingabedatei lesen

S

EWS-4.29

`fgets` -- Liest eine Zeile von der Position des Dateizeigers

```
string fgets (resource handle [, int length])
```

Gibt eine Zeile bis zu (length -1) Bytes Länge zurück, welche aus der Datei von der aktuellen Position des Dateizeigers `handle` aus gelesen wird.

Beispiel: Eine Datei Zeile für Zeile einlesen

```
$handle = fopen ("/tmp/inputfile.txt", "r");
if (!$handle)
    { echo "/tmp/inputfile.txt nicht geöffnet"; exit; }

while (!feof($handle)) {
    $buffer = fgets($handle, 4096);
    echo $buffer;
}

fclose ($handle);
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 429

**Ziele:**

Eingabeschema anwenden können

**in der Vorlesung:**

Am Beispiel wird erklärt:

- Öffnen und Schließen,
- Lesen aus Datei,
- Schema wiederverwenden,

## Schema: eine Datei ändern

S

EWS-4.30

`copy` -- Kopiert eine Datei

```
bool copy (string source, string dest)
```

**Beispiel:** Aus einer Datei mit je einem Vereinsnamen pro Zeile sollen alle Zeilen gelöscht werden, die „FC“ im Vereinsnamen enthalten, d. h. „1.FC Köln“ verschwindet und „SV Werder Bremen“ bleibt drin.

**Vorgehen:** Datei kopieren; die Kopie lesen; das Original überschreiben.

```
copy ("vereine", "vereine.bak");
$in = fopen ("vereine.bak", "r");
if (!$in) { echo "vereine.bak nicht geöffnet"; exit; }
$out = fopen ("vereine", "w");
if (!$out) { echo "vereine nicht geöffnet"; exit; }

while (!feof ($in)) {
    $buffer = fgets ($in);
    if (strpos ($buffer, "FC") === false) {
        fputs ($out, $buffer);
    }
}

fclose ($in);
fclose ($out);
```

Dateien kopieren  
und öffnen

Datei durchlesen  
Teil-string suchen  
Zeile ausgeben

Dateien schließen

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 430

**Ziele:**

Schema anwenden können

**in der Vorlesung:**

Am Beispiel wird erklärt:

- Datei kopieren,
  - mehrere offene Dateien,
  - Leseschema anwenden.
- `strpos` liefert `false`, wenn die Zeichenreihe nicht gefunden wurde. Das Ergebnis muss OHNE Konversion mit `false` verglichen werden (`=== false`), damit beim Finden an Position 0, nicht 0 auf `false` konvertiert wird.

## S3.6 Arrays

S

EWS-4.31

Ein **Array** ist eine **Abbildung von Indizes (oder Schlüsseln) auf Werte**.

Jedes **Element eines Arrays** ist ein **Paar aus Index** und zugeordnetem **Wert**.

**Beispiele:** Index Wert

Monatsnr.	Monatsname	Bundesligatabelle
-----------	------------	-------------------

```
array ( 1 => "Januar",  
        2 => "Februar",  
        ...  
        12 => "Dezember")
```

```
array ( 1 => "FC Bayern München",  
        2 => "Hamburger SV",  
        3 => "SV Werder Bremen",  
        ...  
        18 => "1.FC Kaiserslautern")
```

**Schlüssel** können ganze **Zahlen** (`int`) oder **Zeichenreihen** (`string`) sein,  
**Werte** können **beliebigen Typ** haben.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 431

**Ziele:**

Arrays verstehen

**in der Vorlesung:**

An den Beispielen wird erklärt:

- Array als Abbildung,
- Notation von Array-Werten,

## Zuweisung von Array-Referenzen

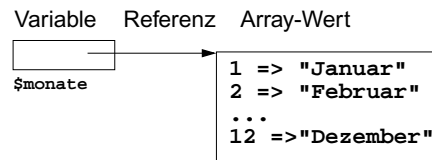
S

EWS-4.31a

Jeder **Array-Wert** wird bei der Ausführung des Programms zusammenhängend im Speicher untergebracht.

Die **Referenz auf die Stelle des Array-Wertes** im Speicher kann man Variablen zuweisen.

```
$monate =  
array ( 1 => "Januar",  
        2 => "Februar",  
        ...  
        12 => "Dezember");
```



Man kann auch die **Elemente einzeln** an Indexpositionen der Variable zuweisen, z. B.

```
$monate[1] = "Januar";  
$monate[2] = "Februar";  
...  
$monate[12] = "Dezember";
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 431a

**Ziele:**

Arrays verstehen

**in der Vorlesung:**

An den Beispielen wird erklärt:

- Array-Werte im Speicher,
- Array-Referenzen,
- Zuweisung von Arrays,
- Indizierung einer Variablen, die ein Array als Wert hat,
- elementweise Zuweisung.



## Array-Variable indizieren

S

EWS-4.32

Eine **Variable**, die eine Array-Referenz enthält, kann man **indizieren**, um den Wert eines bestimmten **Elementes zu lesen** oder zu (über)schreiben.

```
echo $monate[5];      gibt "Mai" aus
$monate[1] = "Jan";  überschreibt das 1. Element
```

**Beispiel:** In einer Klausur wurden maximal 10 Punkte vergeben. Die von den Teilnehmern erreichten Punktzahlen werden je eine pro Zeile von einer Datei gelesen. Mit einem Array von 11 Zählern wird die **Häufigkeit jeder Punktzahl ermittelt**:

```
$i = 0;
while ($i<=10) {
    $punkte[$i]=0; $i=$i+1;
}
$in = fopen("Klausur", "r");
if (!$in) { echo "Klausur nicht geöffnet"; exit; }

while (!feof($in)) {
    $buffer = fgets($in);
    if (strlen($buffer)>0) {
        $punkte[(int)$buffer] += 1;
    }
}
fclose ($in);

echo "Punkte\tAnzahl\n";
$i = 0;
while ($i <= 10) {
    echo $i, "\t",
        $punkte[$i], "\n";
    $i = $i + 1;
}
```

© 2006 bei Prof. Dr. Uwe Kastens

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 432

### Ziele:

Array-Zugriffe anwenden können

### in der Vorlesung:

Am Beispiel wird erklärt:

- Array-Element lesen,
- Array-Element schreiben,
- ein Array von Zählern,
- erste Schleife initialisiert, zweite liest und erhöht Zähler, dritte gibt Zähler aus.
- $Sa += 1$ ; ist eine Abkürzung für  $Sa = Sa + 1$ ;
- $strlen(\$buffer) > 0$  ist bei leeren Zeilen erfüllt, z. B. am Ende,
- $(int)\$buffer$  konvertiert den gelesenen Zeileninhalt in einen ganzzahligen Index.

## Zeichenreihen als Array-Schlüssel

S

EWS-4.33

Auch **Zeichenreihen** können als **Indizes** verwendet werden, man spricht dann von **Schlüsseln**.

**Beispiele:** Monatsname => Monatsnummer oder Vereinsname => Punktestand

Neue **Notation für Schleifen zum Durchlaufen aller Elemente eines Arrays**

```
foreach ($arr as $key => $value){
    ... Benutzung der Variablen $key und $value
}
```

```
$liga = array (
    "VfB Stuttgart" => 22,
    "1.FC Köln" => 12,
    ...
    "SV Werder Bremen" => 35);

arsort ($liga);
echo "Tabellenstand\n\n";

foreach ($liga as $verein => $punkte) {
    echo $verein, "\t", $punkte, "\n";
}
```

Tabellenstand	
FC Bayern München	41
Hamburger SV	37
SV Werder Bremen	35
FC Schalke 04	31
Hertha BSC Berlin	25
Borussia M'gladbach	22
VfB Stuttgart	22
Borussia Dortmund	21
Hannover 96	20
VfL Wolfsburg	18
Bayer 04 Leverkusen	18
Eintracht Frankfurt	18
DSC Arminia Bielefeld	17
1.FSV Mainz 05	15
1.FC Nürnberg	13
1.FC Köln	12
MSV Duisburg	11
1.FC Kaiserslautern	9

© 2006 bei Prof. Dr. Uwe Kastens

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 433

### Ziele:

Arrays mit Schlüsseln verstehen

### in der Vorlesung:

Am Beispiel wird erklärt:

- Nutzen von Zeichenreihen als Schlüssel von Abbildungen, z. B. in Wörterbüchern,
- neue Schleifennotation,
- $Skey$  und  $Svalue$  stehen für den Schlüssel bzw. den Wert des in dieser Iteration betrachteten Elementes,
- $arsort (\$liga)$  sortiert das array nach fallenden Werten der Elemente.

## Weitere Schleifenformen

S

EWS-4.34

### foreach-Schleifen über Arrays in 2 Formen

- für jedes Element soll auf Index/Schlüssel **und** Wert zugegriffen werden:

```
$monate = array (1 => "Jan", 2 => "Feb", ...12 => "Dez");  
foreach ($monate as $nr => $name)  
{ echo $nr, "\t", $name, "\n";}
```

- für jedes Element soll **nur auf den Wert** zugegriffen werden:

```
foreach ($monate as $name) {echo $name, "\n";}
```

### Allgemeine for-Schleife

```
for (Initialisierung; Bedingung; Fortschaltung) { Rumpf }
```

hat die **gleiche Bedeutung** wie die while-Schleife

```
Initialisierung; while (Bedingung) {Rumpf; Fortschaltung}
```

wird meist als **Zählschleife** benutzt

```
for ($i = 1; $i <= 12; $i++) {echo $i, "\t", $monate[$i], "\n";}
```

**\$i++ erhöht den Wert von \$i um 1.** Wird **\$i++ als Ausdruck** verwendet, so hat er den **Wert von \$i vor dem Erhöhen**, z. B. in `$monate[$i++]`.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 434

### Ziele:

Alle wichtigen Schleifenformen kennen

### in der Vorlesung:

Die Schleifenformen werden an Beispielen erklärt:

- Zugriff auf Array-Elemente,
- Schreibweise von for-Schleifen,
- Bezug zu while-Schleifen,
- Notation i++

## S3.7 Funktionsdefinitionen

S

EWS-4.35

**Funktion:** Rechenvorschrift mit einem **Namen** und ggf. **formalen Parametern**, die an mehreren Stellen im Programm mit unterschiedlichen **aktuellen Parametern aufgerufen** werden kann.

Beispiel für die Definition einer Funktion:

```
function prTabellenZelle ($v) { print "<td><b>$v</b></td>"; }
```

Aufrufe der Funktion:

```
prTabellenZelle ("Hallo!");          prTabellenZelle ($x * $y);
```

Zweck von Funktionen:

- Wiederholung** gleicher Berechnungen an mehreren Stellen des Programmes **vermeiden**
- abstrahieren:** das **Was** soll berechnet werden? vom **Wie** soll das geschehen?  
also die **Aufgabe** von der **Lösung im Detail**  
im Programmtext: Name und akt. Parameter im **Aufruf** Rumpf in der **Definition**

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 435

### Ziele:

Zweck von Funktionsdefinitionen erkennen

### in der Vorlesung:

Am Beispiel wird erklärt:

- formale - aktuelle Parameter,
- Abstraktion

## Syntax von Funktionsdefinitionen

S

EWS-4.36

```
FuncDef ::= 'function' FuncName '(' [FormParams] ')'
         '{' Statement* '}'

FormParams ::= FormParams ',' FormParam | FormParam

FormParam ::= VariableName                call-by-value
            | VariableName = Literal       call-by-value, optional
            | '&' VariableName            call-by-reference

Statement ::= 'return' [Expression] ';'


```

Es darf **keine 2** Funktionsdefinitionen geben,  
die den **gleichen Funktionsnamen definieren**.

Das heißt insbesondere, dass der Name verschieden  
von allen Funktionsnamen der Bibliothek sein muss.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 436

### Ziele:

Notation lernen

### in der Vorlesung:

Es wird erklärt:

- Notation der Parameter,
- Hinweis: Aussagekräftige Funktionsnamen wählen.
- Rolle der return-Anweisung auf folgender Folie,
- Erklärung von "call-by-value" folgt später,
- "call-by-reference" und optionale Parameter werden hier nicht erklärt. (Interessierte mögen das nachlesen.)

## Funktionen mit und ohne Ergebnis

S

EWS-4.37

**Funktionen ohne Ergebnis** werden als **Anweisungen** aufgerufen. Der Aufruf bewirkt einen Seiten-Effekt (Veränderung von Werten von Variablen, Ausgabe, Eingabe), z. B.

```
function prTabellenZelle ($v) { print "<td><b>$v</b></td>"; }
```

Aufrufe der Funktion als Anweisungen:

```
prTabellenZelle ("Hallo!");      prTabellenZelle ($x * $y);
```

**Funktionen** können durch Ausführen einer **return-Anweisung ein Ergebnis liefern**. Dann können ihre Aufrufe als **Ausdruck** verwendet werden, z. B.

```
function TabellenZelle ($v) { return "<td><b>$v</b></td>"; }
```

Aufrufe der Funktion als Ausdrücke:

```
print TabellenZelle ("Hallo!");   $z = TabellenZelle ($x * $y);
```

Die Ausführung einer **return-Anweisung** liefert den **Wert des Ausdrucks als Ergebnis** und **beendet die Ausführung des Funktionsrumpfes**. Eine **return-Anweisung ohne Ausdruck** beendet die Ausführung des Funktionsrumpfes ohne Ergebnis.

Funktionen **mit Ergebnis und ohne Seiten-Effekt** sind meist **allgemeiner einsetzbar**.

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 437

### Ziele:

Ergebnis von Funktionen verstehen

### in der Vorlesung:

Am Beispiel wird erklärt:

- Ergebnis einer Funktion,
- Wirkung einer return-Anweisung,
- Begriff Seiten-Effekt,
- Vergleich mit - ohne Ergebnis

## Parameterübergabe call-by-value

S

EWS-4.38

### Parameterübergabe:

Bezug zwischen **aktuellem Parameter im Aufruf** und **formalem Parameter in der Funktionsdefinition**.

**Call-by-value:** wichtigste Art der Parameterübergabe (auch in C, C++, Java, Pascal, Ada, ...)

Der **formale Parameter ist eine Variable**, die nur während der Ausführung des Funktionsrumpfes existiert.

Der **aktuelle Parameter ist ein Ausdruck**. Er wird beim Aufruf ausgewertet. Mit seinem Wert wird der formale Parameter initialisiert.

**Zuweisungen an den formalen Parameter** im Funktionsrumpf **wirken sich nicht** auf den **aktuellen Parameter** aus.

Beispiel:

```
function fak ($n) {  
    $sum = 1;  
    while ($n > 1) {  
        $sum = $sum * $n;  
        $n = $n - 1;  
    }  
    return $sum;  
}  
  
$k = 5;  
print fak ($k);  
print " ist Fakultät von $k\n";
```

© 2016 bei Prof. Dr. Uwe Kastens

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 438

### Ziele:

Call-by-value verstehen

### in der Vorlesung:

Am Beispiel wird erklärt:

- Parameterübergabe allgemein,
- call-by-value speziell,
- call-by-reference wird hier nicht erklärt,
- im Beispiel wird durch den Aufruf von fak der Wert der Variablen \$k nicht verändert, obwohl im Rumpf von fak an den formalen Parameter \$n zugewiesen wird.
- Optionale Parameter mit Initialisierung werden hier auch nicht erklärt.

## Globale und lokale Variable

S

EWS-4.39

Wir unterscheiden global und lokale Variable:

### Globale Variable:

- wird durch Benutzung des Namens **außerhalb von Funktionsdefinitionen eingeführt**;
- ihr **Name gilt im ganzen Programm**; aber **in Funktionsrumpfen nur, wenn** er dort als **global** deklariert wird;
- sie **existiert** während der **gesamten Ausführung** des Programms.

### Lokale Variable:

- wird durch Benutzung des Namens **in einer Funktionsdefinition eingeführt**;
- ihr **Name gilt im Rumpf dieser Funktion**; er **kann mit Namen anderer Variable** in anderen Funktionen oder globaler Variable **übereinstimmen**;
- sie **existiert jeweils während eines Aufrufes** dieser Funktion

```
function namesOut ($v) {  
    global $out, $lineCnt;  
  
    $lg = strlen ($v);  
    fputs ($out, $v);  
    $lineCnt++;  
}  
  
$out = fopen ("names", "w");  
$lineCnt = 0;  
$sum = 0;  
  
while ( ... ) {  
    namesOut (...);  
}  
  
print $lineCnt;  
fclose ($out);  
  
global: $out, $lineCnt, $sum  
  
lokal in namesOut: $v, $lg
```

© 2003 bei Prof. Dr. Uwe Kastens

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 439

### Ziele:

Global und lokal verstehen

### in der Vorlesung:

An dem Beispiel wird erklärt:

- Ein Name gilt in bestimmten Abschnitten des Programmes: sein Gültigkeitsbereich;
- Variable existieren im Speicher für eine bestimmte Zeit: ihre Lebensdauer;
- Unterschied global - lokal für Gültigkeitsbereich und Lebensdauer;
- Bedeutung und Nutzen der global-Deklaration.

## Eine HTML-Seite erzeugen

S

EWS-4.40

```
<?php
function headOut ($title) {
    global $out;
    fputs ($out, <<<KOPF
<html><head>
    <title>$title</title>
</head><body>
    <h3>$title</h3>\n
KOPF
    );
}

function footOut () {
    global $out;
    fputs ($out,
        "</body></html>\n");
}

$out = fopen ("such.html" ,"w");
$tel = fopen ("tele.txt" ,"r");
headOut ("Suche im Telefonbuch");
$name = "Thies";
fputs ($out,
    "<h4>Ergebnisse f&uuml;r $name".
    "</h4><p>\n<pre>\n");

while (!feof($tel)) {
    $line = fgets($tel, 64);
    if (preg_match("/$name/i",$line)) {
        fputs ($out, "\t$line");
    }
}
fputs ($out, "</pre></p>\n");
footOut();fclose($tel);fclose($out);
?>
```

```
<html><head>
    <title>Suche im Telefonbuch</title>
</head><body>
<h3>Suche im Telefonbuch</h3>
<h4>Ergebnisse f&uuml;r Thies</h4><p>
<pre>
    Thies, Michael, Dr. 6682
</pre></p>
</body></html>
```

HTML-Datei:

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 440

Ziele:

Das Gelernte zusammensetzen

in der Vorlesung:

Am Beispiel wird erklärt:

- Funktionsdefinitionen und Aufrufe,
- Dateien durchsuchen und schreiben,
- String-Literale und String-Operationen,
- Ablaufstrukturen,
- globale und lokale Variable,
- Vergleich mit Telefonbuch-Beispiel.

## S3.8 PHP-Eingabe mit HTML-Formularen

S

EWS-4.41

**HTML-Formulare** enthalten grafische Elemente für **interaktive Eingaben** (siehe S2.3)

Ein Formular-Element liefert ein **Paar von Zeichenreihen** **name** und **value**. **name** ist der Name des Formular-Elementes, **value** der eingegebene Wert.

Ein einzeliges Textfeld mit der gezeigten Eingabe

```
<input type="text" name="Zuname" size="10">
```

Zuname:  
Kastens

liefert das Paar "Zuname" und "Kastens".

Ein PHP-Programm nimmt die Eingabe aller Elemente eines Formulars als **Schlüssel-Wert-Paare im vordefinierten, globalen Array** **\$\_REQUEST** entgegen.

Im obigen Beispiel hat dann `$_REQUEST["Zuname"]` den Wert "Kastens".

Mit einer **foreach-Schleife** über das Array **\$\_REQUEST** kann man die **Werte aller Formular-Elemente** ermitteln:

```
foreach ($_REQUEST as $name => $value) {
    echo "$name => $value\n";
}
```

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 441

Ziele:

Zusammenhang verstehen

in der Vorlesung:

Am Beispiel wird erklärt:

- Name und Eingabewert eines Formular-Elementes;
- gilt für alle Arten von Formular-Elementen;
- Zugriff auf Elemente des Arrays `$_REQUEST`
- `$_REQUEST` ist ÜBERALL global, es braucht in Funktionen nicht als global deklariert zu werden.

## Programmstruktur für Formular-Eingabe

EWS-4.42

Das PHP-Programm wird **zweimal ausgeführt** und läuft dabei in verschiedene Zweige:

Am Inhalt des Arrays wird die Entscheidung getroffen

Beim ersten Mal wird das **Formular angeboten**.

Beim zweiten Mal wird die **Eingabe verarbeitet**.

```
<html><head>
  <title>PHP Formular-Eingabe</title>
</head>
<body>
<?php
  if (!isset($_REQUEST['submit'])) {
    // HTML-Formular ausgeben
    <form action="http://..." method="POST">
    // Formular-Elemente ...
    </form>
  } else {
    // Formular-Eingabe aus $_REQUEST
    // entnehmen, verarbeiten und
    // Ergebnisse ausgeben
  }
?>
</body></html>
```

© 2004 bei Prof. Dr. Uwe Kastens

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 442

Ziele:

Typische Programmstruktur kennen

in der Vorlesung:

Am Beispiel wird erklärt:

- die Verzweigung;
- das action-Attribut muss die URL des PHP-Programmes als Wert haben, damit das Programm zum zweiten Mal aufgerufen wird.
- Alternative Struktur: Die Datei enthält nur den ersten Programmzweig; der zweite Zweig ist in einer anderen Datei enthalten; ihre URL wird im action-Attribut in der ersten Datei angegeben.

## Ein komplettes Beispiel

EWS-4.43

```
<html><head><title>PHP Formular-Eingabe</title></head><body>
<?php
if (!isset($_REQUEST['submit'])) {
echo <<<FORMULARANZEIGE
<form action="http://..." method="POST">
  <p>Zuname:<br>
  <input type="text" name="Zuname" size="10"></p>
  <p>G&auml;stebuch:<br>
  <textarea name="eintrag" rows="3" cols="10"></textarea></p>
  <p>Versand:<br>
  <input type="checkbox" name="speed" value="fast">eilig</p>
  <p>Zahlung:<br>
  <input type="radio" name="pay" value="cash">Bar<br>
  <input type="radio" name="pay" value="card">Karte<br></p>
  <p>Wochentag:<br>
  <select name="tag" size="3">
    <option value="freitag">Freitag
    ...
  </select></p>
  <input type="reset" value="l&ouml;sch&uuml;n"><br>
  <input type="submit" value="abschicken" name="submit"><br>
</form>
FORMULARANZEIGE;
} else {
echo "<h4>Ihre Eingabe:</h4><p>\n<pre>";
foreach ($_REQUEST as $name => $value) {
  echo "$name => $value\n";
}
echo "</pre>";
}
?>
</body></html>
```

Zuname: Kastens

Gästebuch: Das sieht noch recht mager aus!

Versand:  eilig

Zahlung:  Bar  Karte

Wochentag: Freitag

Ihre Eingabe:

```
Zuname => Kastens
eintrag => Das sieht
noch recht
mager aus!
speed => fast
pay => cash
tag => samstag
submit => abschicken
```

© 2006 bei Prof. Dr. Uwe Kastens

## Vorlesung Einführung in Web-bezogene Sprachen WS 2006 / Folie 443

Ziele:

Formulareingabe anwenden können

in der Vorlesung:

Am Beispiel wird erklärt:

- das **Beispiel ausführen**.
- es muss auf einem Web-Server installiert sein,
- Programmstruktur,
- Ausgabe der Formular-Elemente,
- Name-Wert-Paare,
- Verarbeitung der Formular-Eingabe,
- Einbettung in HTML-Datei;
- das Beispiel Telefonbuch.