

Kapitel WT: V (Fortsetzung)

I. Einführung

II. Rechnerkommunikation und Protokolle

III. Dokumentsprachen

IV. Client-Technologien

V. Server-Technologien

- Web-Server
- Common Gateway Interface CGI
- Java Servlets
- Java Server Pages JSP
- Active Server Pages ASP
- Exkurs: reguläre Ausdrücke
- PHP Hypertext Preprocessor
- Perl, Python, Ruby

VI. Architekturen und Middleware-Technologien

VII. Web-Technologien für besondere Anwendungen

VIII. Semantic Web

Exkurs: reguläre Ausdrücke

Grundlagen: Grammatik

- Alphabet Σ .
- Wort w .
- Sprache L .
- Grammatik G .

Exkurs: reguläre Ausdrücke

Grundlagen: Grammatik

□ Alphabet Σ .

Ein Alphabet Σ ist eine nicht-leere Menge von Zeichen bzw. Symbolen.

□ Wort w .

Ein Wort w ist eine endliche Folge von Symbolen aus Σ . Die Länge eines Wortes $|w|$ ist die Anzahl seiner Symbole.

ε bezeichnet das leere Wort; es hat als einziges Wort die Länge 0.

Σ^* bezeichnet die Menge aller Worte über Σ .

□ Sprache L .

Eine Sprache L ist eine Menge von Worten über einem Alphabet Σ .

□ Grammatik G .

Eine Grammatik G ist ein Kalkül, um eine Sprache zu definieren – also eine **Menge von Regeln**, mit denen man Worte ableiten kann. Die zu G gehörende Sprache besteht aus allen ableitbaren, terminalen Worten.

Bemerkungen:

- Bei der Definition von Spracheigenschaften unterscheidet man verschiedene Ebenen [vgl. [Exkurs: Programmiersprachen](#)]. Die Ebene 1 behandelt die Notation von Grundsymbolen, die Ebene 2 behandelt die syntaktische Struktur der Sprache. Zur Unterscheidung, auf welcher Ebene der Grammatikanwendung man sich befindet, werden auch folgende Begriffe verwendet:
 - Ebene 1: Alphabet, Zeichen, Wort, Sprache
 - Ebene 2: Vokabular, Symbol, Satz, Sprache
- Die Worte {Alphabet, Vokabular}, {Zeichen, Symbol} und {Wort, Satz} sind die jeweiligen Entsprechungen der Grundsymbolebene und der syntaktischen Ebene.

Exkurs: reguläre Ausdrücke

Grundlagen: Grammatik (Fortsetzung)

Definition 1 (Grammatik)

Eine Grammatik ist ein Viertupel $G = (N, \Sigma, P, S)$ mit

N endliche Menge von Nichtterminalsymbolen

Σ endliche Menge von Terminalsymbolen

P endliche Menge von Produktionen bzw. Regeln

$$P \subset (N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$$

S Startsymbol, $S \in N$

$$\square N \cap \Sigma = \emptyset$$

Bemerkungen:

- Eine Regel besteht aus einer linken Seite (Prämisse) und einer rechten Seite (Konklusion), die jeweils ein Wort bestehend aus Terminalen und Nichtterminalen sind. Die linke Seite muss mindestens ein Nichtterminal beinhalten und die rechte Seite kann dabei im Gegensatz zur linken Seite auch das leere Wort sein. [\[Wikipedia\]](#)
- Eine Regel kann auf ein Wort, bestehend aus Terminalen und Nichtterminalen, angewendet werden, wobei ein beliebiges Vorkommen der linken Seite der Regel im Wort durch die rechte Seite der Regel ersetzt wird: $w \rightarrow w'$
- Für w, w' gilt die so genannte *Transitionsrelation*; eine Folge von Anwendungen von Regeln bezeichnet man als *Ableitung*.

Exkurs: reguläre Ausdrücke

Grundlagen: Grammatik (Fortsetzung)

Definition 2 (erzeugte Sprache)

Die von einer Grammatik $G = (N, \Sigma, P, S)$ erzeugte Sprache $L(G)$ enthält genau die Worte, die nur aus Terminalsymbolen bestehen und vom Startsymbol aus mit einer endlichen Anzahl von Schritten abgeleitet werden können:

$$L(G) := \{w \in \Sigma^* \mid S \rightarrow_G^* w\}$$

\rightarrow_G^* steht für die beliebige Anwendung der Produktionen in G , also die reflexiv-transitive Hülle der Transitionsrelation \rightarrow_G .

Exkurs: reguläre Ausdrücke

Grundlagen: Grammatik (Fortsetzung)

Definition 2 (erzeugte Sprache)

Die von einer Grammatik $G = (N, \Sigma, P, S)$ erzeugte Sprache $L(G)$ enthält genau die Worte, die nur aus Terminalsymbolen bestehen und vom Startsymbol aus mit einer endlichen Anzahl von Schritten abgeleitet werden können:

$$L(G) := \{w \in \Sigma^* \mid S \rightarrow_G^* w\}$$

\rightarrow_G^* steht für die beliebige Anwendung der Produktionen in G , also die reflexiv-transitive Hülle der Transitionsrelation \rightarrow_G .

Beispiel:

$G = (N, \Sigma, P, S)$ mit $N = \{S, A, B\}$, $\Sigma = \{a, b\}$ und folgenden Produktionen:

$$\begin{array}{ll} S \rightarrow ABS & BA \rightarrow AB \\ S \rightarrow \varepsilon & BS \rightarrow b \\ & Bb \rightarrow bb \\ & Ab \rightarrow ab \\ & Aa \rightarrow aa \end{array}$$

Bemerkungen:

- ❑ Es ist Konvention, die Nichtterminalsymbole mit Großbuchstaben und die Terminalsymbole mit Kleinbuchstaben zu bezeichnen.
- ❑ Zur Erzeugung einer Sprache existieren beliebig viele Grammatiken.
- ❑ Eine andere Grammatik, die die gleiche Sprache wie im Beispiel erzeugt, ist:

$$N = \{S, A, B\}, \Sigma = \{a, b\}, P = \{S \rightarrow aSb, S \rightarrow \varepsilon\}$$

Exkurs: reguläre Ausdrücke

Grundlagen: Chomsky-Hierarchie

Grammatiken werden hinsichtlich der Komplexität der Sprachen, die sie erzeugen, in vier Klassen eingeteilt.

- Typ 0.
- Typ 1 ~ kontextsensitiv.
- Typ 2 ~ kontextfrei.
- Typ 3 ~ regulär.

Exkurs: reguläre Ausdrücke

Grundlagen: Chomsky-Hierarchie

Grammatiken werden hinsichtlich der Komplexität der Sprachen, die sie erzeugen, in vier Klassen eingeteilt.

- Typ 0.

Für die Regeln in P existieren keine Einschränkungen.

- Typ 1 \sim kontextsensitiv.

Für alle Regeln $w \rightarrow w' \in P$ gilt: $|w| \leq |w'|$

- Typ 2 \sim kontextfrei.

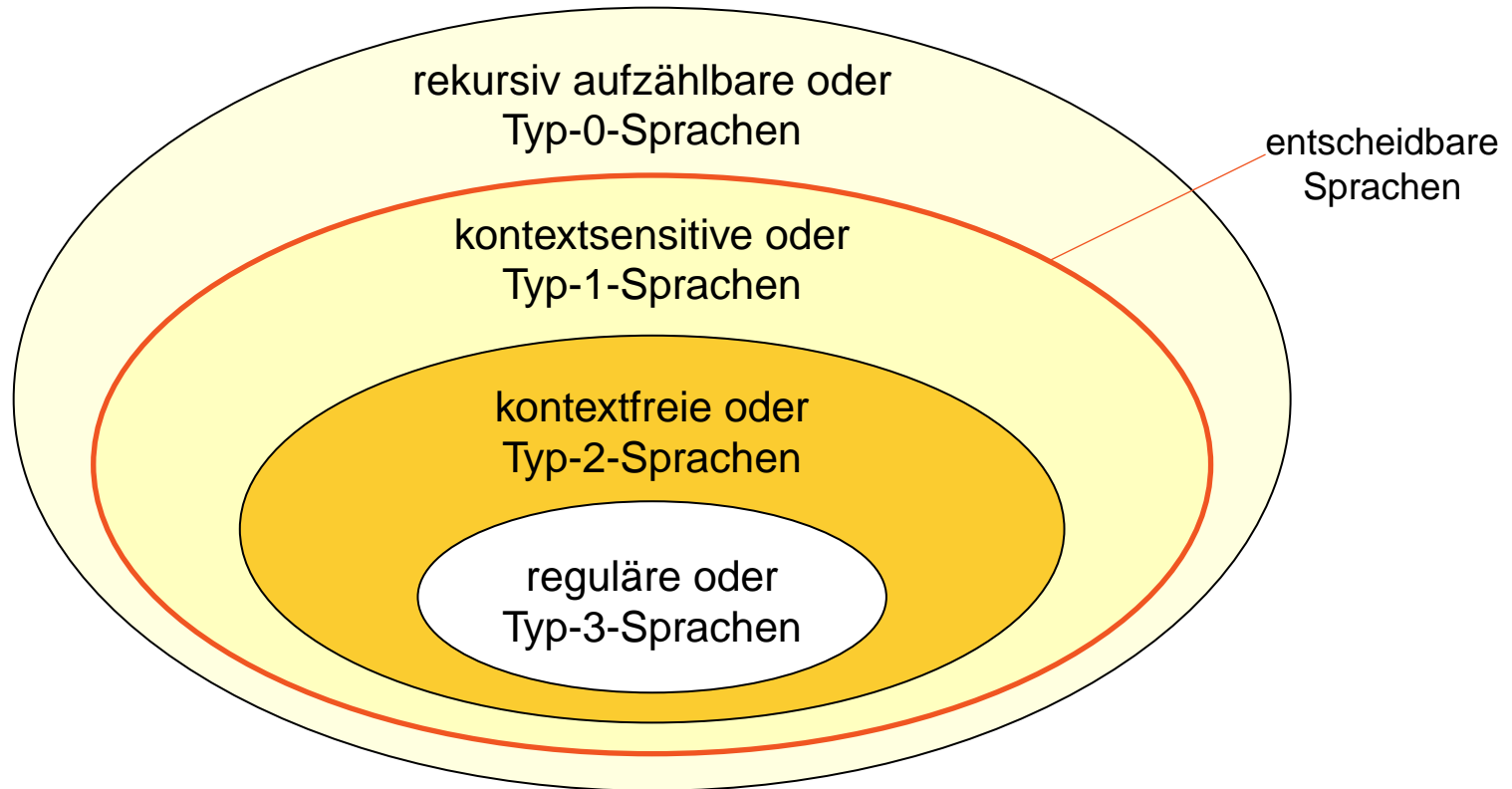
Für alle Regeln $w \rightarrow w' \in P$ gilt: w ist eine einzelne Variable; d. h., $w \in N$.

- Typ 3 \sim regulär.

Die Grammatik ist vom Typ 2 und zusätzlich gilt: $w' \in (\Sigma \cup \Sigma N)$, d. h., die rechten Seiten der Regeln sind entweder einzelne Terminalzeichen oder ein Terminalzeichen gefolgt von einem Nichtterminal.

Exkurs: reguläre Ausdrücke

Grundlagen: Chomsky-Hierarchie (Fortsetzung)



Definition 3 (Sprache vom Typ)

Eine Sprache $L \subseteq \Sigma^*$ wird Sprache vom Typ 0 (Typ 1, Typ 2, Typ 3) genannt, falls es eine Grammatik G vom Typ 0 (Typ 1, Typ 2, Typ 3) gibt, mit $L(G) = L$.

Bemerkungen:

- ❑ Die Chomsky-Hierarchie stellt eine Hierarchie mit echten Teilmengenbeziehungen dar:
Typ 3 \subset Typ 2 \subset Typ 1 \subset Typ 0
- ❑ Alle Sprachen vom Typ 1, 2 oder 3 sind entscheidbar;
d.h., das Wortproblem für alle Sprachen vom Typ 1, 2 oder 3 ist entscheidbar;
d.h., es gibt einen Algorithmus, der bei Eingabe einer Grammatik G und einem Wort w in endlicher Zeit feststellt, ob $w \in L(G)$ gilt oder nicht.
- ❑ Die Menge der Typ-0-Sprachen ist identisch mit der Menge der rekursiv aufzählbaren oder semi-entscheidbaren Sprachen. Daher gibt es Typ-0-Sprachen, die nicht entscheidbar sind.
- ❑ Im Übersetzerbau spielen Sprachen bzw. Grammatiken vom Typ 3 (lexikalische Analyse, Tokenisierung) und Typ 2 (syntaktische Strukturanalyse) die zentrale Rolle.

Exkurs: reguläre Ausdrücke

Kalküle für reguläre Sprachen

Verschiedene Kalküle zur Bildung von Worten einer regulären Sprache:

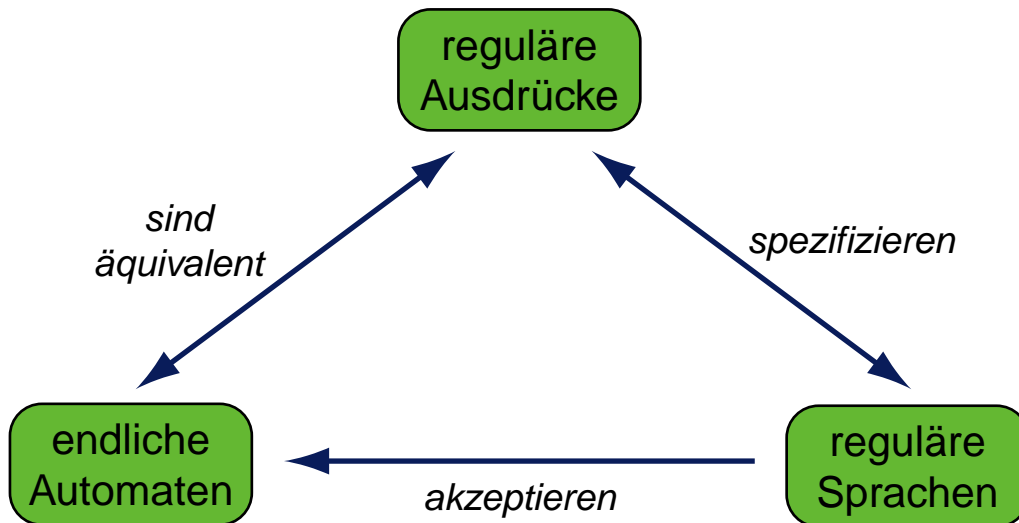
- (a) endlichen Akzeptor bzw. Automat
- (b) regulärer Ausdruck
- (c) Typ-3-Grammatik
- (d) Angabe endlich vieler Äquivalenzklassen

Exkurs: reguläre Ausdrücke

Kalküle für reguläre Sprachen

Verschiedene Kalküle zur Bildung von Worten einer regulären Sprache:

- (a) endlichen Akzeptor bzw. Automat
- (b) regulärer Ausdruck
- (c) Typ-3-Grammatik
- (d) Angabe endlich vieler Äquivalenzklassen



[vgl. Haenelt 2005]

Exkurs: reguläre Ausdrücke

Grundlagen: Zusammenfassung

Kalküle zur Spracherzeugung

Typ 0	Typ-0-Grammatik allgemeine Turingmaschine
Typ 1	kontextsensitive Grammatik linear beschränkte Turingmaschine
Typ 2	kontextfreie Grammatik Kellerautomat
Typ 3	reguläre Grammatik (Typ-3-Grammatik) deterministischer/nicht-deterministischer endlicher Automat regulärer Ausdruck

Exkurs: reguläre Ausdrücke

Grundlagen: Zusammenfassung

Kalküle zur Spracherzeugung

Typ 0	Typ-0-Grammatik allgemeine Turingmaschine
Typ 1	kontextsensitive Grammatik linear beschränkte Turingmaschine
Typ 2	kontextfreie Grammatik Kellerautomat
Typ 3	reguläre Grammatik (Typ-3-Grammatik) deterministischer/nicht-deterministischer endlicher Automat regulärer Ausdruck

Komplexität des Wortproblems

Typ 0	unentscheidbar
Typ 1	exponentielle Komplexität, NP-hart
Typ 2	$O(n^3)$
Typ 3	lineare Komplexität

Exkurs: reguläre Ausdrücke

Konstruktion [vgl. Kastens 2005] [vgl. [PHP](#)]

Ein regulärer Ausdruck R kann wie folgt rekursiv zusammengesetzt sein. F und G bezeichnen gegebene reguläre Ausdrücke.

	R	Erklärung
1.	a	das Zeichen a
2.	FG	Zusammenfügen von zwei Worten
3.	$F \mid G$	Alternativen
4.	ε	das leere Wort
5.	(F)	Klammerung
6.	F^+	nicht-leere Folge von Worten aus $L(F)$
7.	F^*	beliebig lange Folge von Worten aus $L(F)$
8.	F^n	Folge von n Worten aus $L(F)$

Exkurs: reguläre Ausdrücke

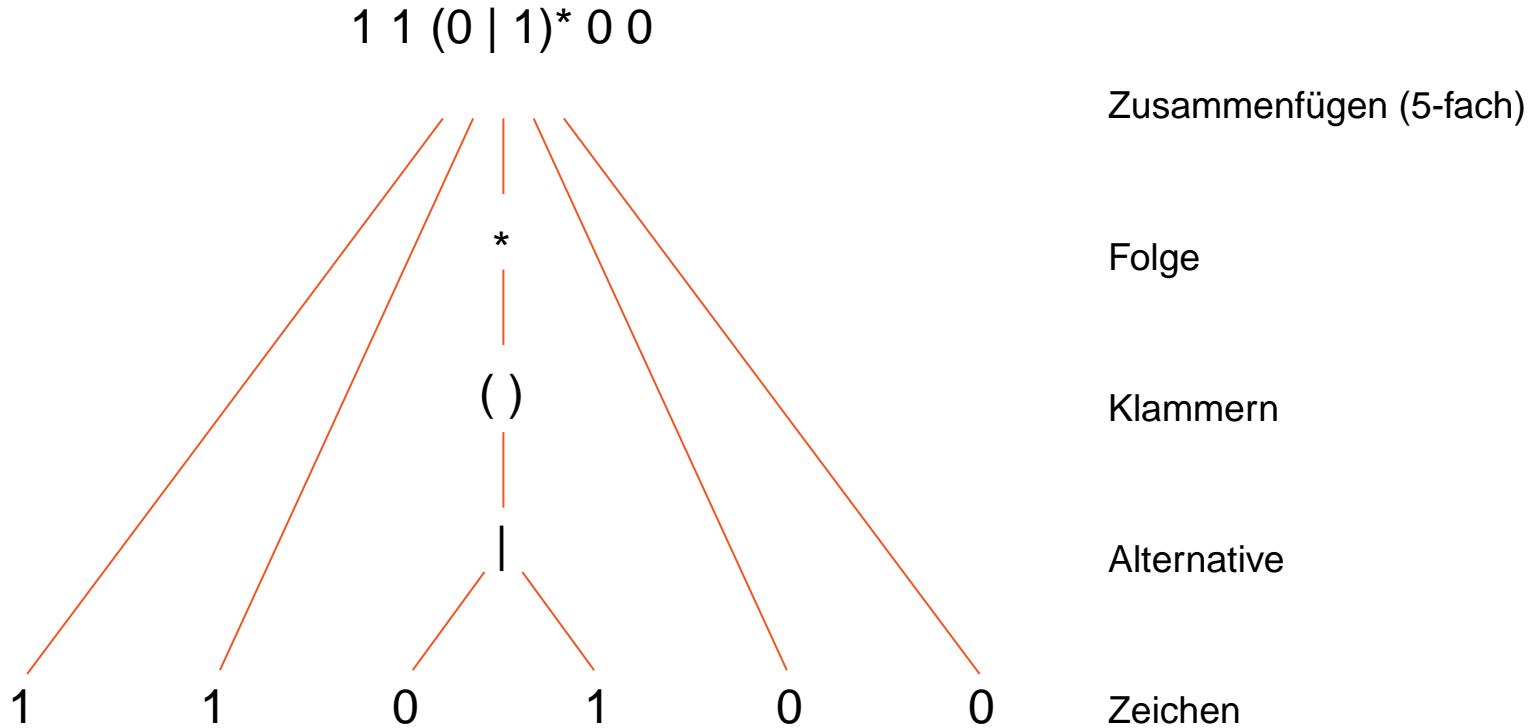
Konstruktion [vgl. Kastens 2005] [vgl. [PHP](#)]

Ein regulärer Ausdruck R kann wie folgt rekursiv zusammengesetzt sein. F und G bezeichnen gegebene reguläre Ausdrücke.

	R	Sprache $L(R)$	Erklärung
1.	a	$\{a\}$	das Zeichen a
2.	FG	$\{fg \mid f \in L(F), g \in L(G)\}$	Zusammenfügen von zwei Worten
3.	$F \mid G$	$\{f \mid f \in L(F)\} \cup \{g \mid g \in L(G)\}$	Alternativen
4.	ε	$\{\varepsilon\}$	das leere Wort
5.	(F)	$L(F)$	Klammerung
6.	F^+	$\{f_1 f_2 \dots f_n \mid f_i \in L(F), n \geq 1, i = 1, \dots, n\}$	nicht-leere Folge von Worten aus $L(F)$
7.	F^*	$\{\varepsilon\} \cup L(F^+)$	beliebig lange Folge von Worten aus $L(F)$
8.	F^n	$\{f_1 f_2 \dots f_n \mid f_i \in L(F), i = 1, \dots, n\}$	Folge von n Worten aus $L(F)$

Exkurs: reguläre Ausdrücke

Illustration [vgl. Kastens 2005]



Jedes Wort aus der Sprache dieses regulären Ausdrucks besteht aus zwei Einsen, gefolgt von beliebig vielen Nullen oder Einsen, gefolgt von zwei Nullen.

Exkurs: reguläre Ausdrücke

Beispiele [vgl. Kastens 2005]

<i>R</i>	Name der Sprache $L(R)$	Worte aus $L(R)$
$(a b) (c d \varepsilon)$	<i>Abc</i>	ac, bc, ad, bd, a, b
Sehr geehrte(r ε) (Frau Herr)	<i>Anrede</i>	Sehr geehrte Frau
$0 1 \dots 9$	<i>Digit</i>	7
$a b \dots z$	<i>sLetter</i>	x
$A B \dots Z$	<i>cLetter</i>	B
<i>sLetter</i> <i>cLetter</i>	<i>Letter</i>	m, N
<i>Letter</i> (<i>Letter</i> <i>Digit</i>)*	<i>Bezeichner</i>	Maximum, min3, a
<i>Digit</i> ⁺ . <i>Digit</i> ²	<i>GeldBetrag</i>	23.95, 0.50
$(cLetter cLetter^2 cLetter^3) -$ $(cLetter cLetter^2) -$ $(Digit Digit^2 Digit^3 Digit^4)$	<i>KFZ</i>	KR-AX-123
$1^3 (1 0)^* 0^3$	<i>Dual</i>	1111000, 1111101010000

Exkurs: reguläre Ausdrücke

Spezifikation von Textmustern

Ein wichtiger Einsatz von regulären Ausdrücken in Sprachen, die zur Textverarbeitung eingesetzt werden, ist die Spezifikation von Textmustern.

Beispiel: Darstellung aller Dateinamen der Form

„webtec(0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)³.html“

□ Unix-Shell.

```
ls webtec[0-9][0-9][0-9].html
```

□ PHP.

```
$d = "[0-9]";
```

```
preg_match("/webtecI$d$d$d\.html/", $files)
```

Bemerkungen:

- ❑ Wenn Namen von regulären Ausdrücken in anderen regulären Ausdrücken verwendet werden, müssen sie als Teil der Meta-Sprache kenntlich gemacht werden.
Hier: Verwendung der kursiven Schreibweise.
- ❑ Jede Skriptsprache zur Textverarbeitung verwendet eine andere Syntax zur Spezifikation regulärer Ausdrücke; die Konstruktionsprinzipien und die Mächtigkeit sind vergleichbar.
- ❑ Die Spezifikation regulärer Ausdrücke in PHP ist aus der Skriptsprache Perl übernommen.

PHP Hypertext Preprocessor

PHP Hypertext Preprocessor

Einführung

Charakteristika:

- ❑ Wie JSP: dokumentenzentrierte (HTML) Programmierung
- ❑ prozedurale Sprache mit objektorientierten Erweiterungen
- ❑ wenige einfache Typen, dynamisch typisiert
- ❑ Notation an C und Perl orientiert
- ❑ große Funktionsbibliothek
- ❑ umfassende Datenbankunterstützung
- ❑ **Open Source**

Anwendung:

- ❑ kleine private bis mittelgroße kommerzielle Projekte
- ❑ Schwerpunkt auf Datenbanken

PHP Hypertext Preprocessor

Einführung (Fortsetzung)

PHP-Code wird in die HTML-Datei eingebettet:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>

  <head> <title>PHP Dreieck</title> </head>

  <body>

    <?php
      $line = 1;
      while ($line < 16) {
        $col = 1;
        while ($col <= $line) {
          echo "*";
          $col = $col + 1;
        }
        echo "<br>\n";
        $line = $line + 1;
      }
    ?>

  </body>
</html>
```

PHP Hypertext Preprocessor

Einführung (Fortsetzung)

[Demo]

PHP-Code wird in die HTML-Datei eingebettet:

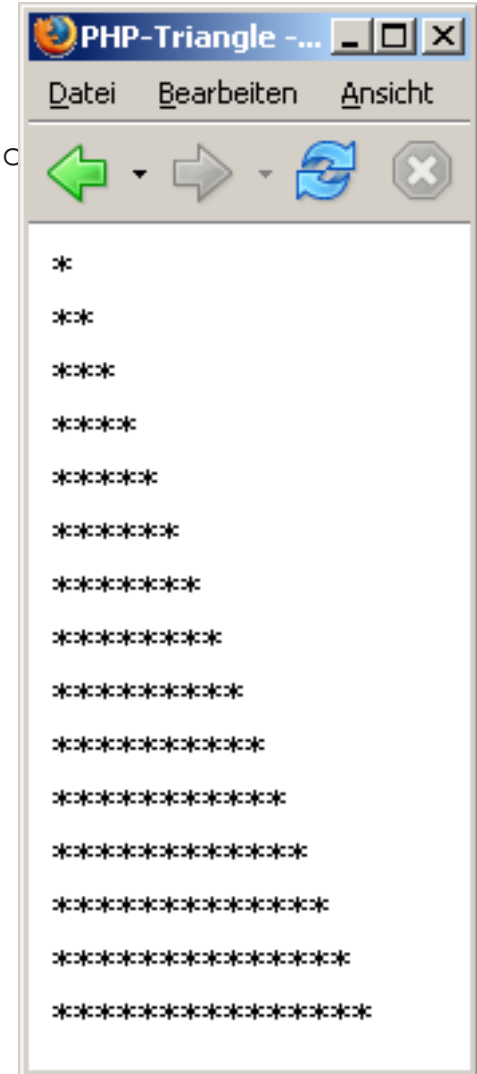
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN"
<html>

<head> <title>PHP Dreieck</title> </head>

<body>

  <?php
    $line = 1;
    while ($line < 16) {
      $col = 1;
      while ($col <= $line) {
        echo "*";
        $col = $col + 1;
      }
      echo "<br>\n";
      $line = $line + 1;
    }
  ?>

</body>
</html>
```



PHP Hypertext Preprocessor

Einführung (Fortsetzung)

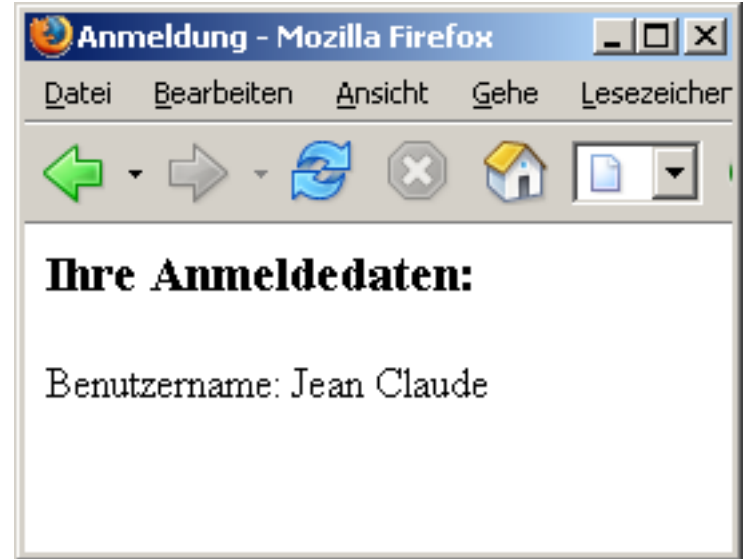
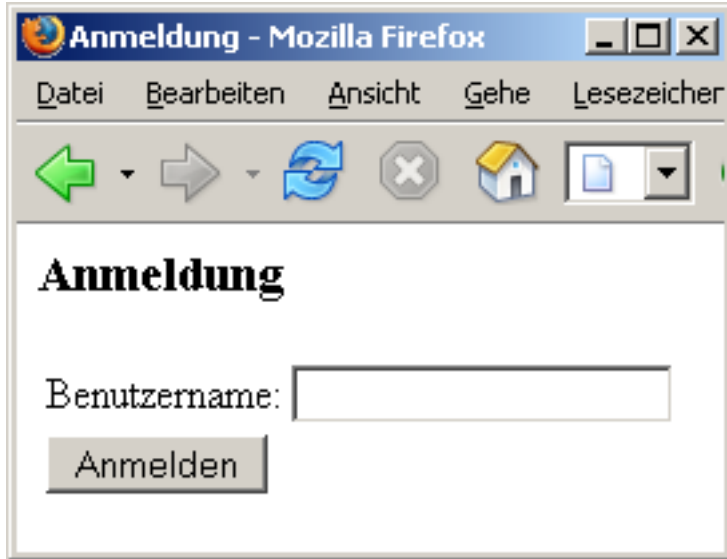
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<html>
  <head> <title>Registration</title> </head>
  <body>
    <?php
      $user = $_REQUEST['user'];
      if(trim($user) == ""){
    ?>
    <form method="get" action="registration.php">
      <table>
        <tr><td>Benutzername:</td><td><input type="text"
          name="user"></td></tr>
        <tr><td><input type="submit" value="Anmelden"></td></tr>
      </table>
    </form>
    <?php } else { ?>
    Benutzername: <?php echo $user ?>
    <?php } ?>
  </body>
</html>
```

Vergleiche hierzu die [JSP-Realisierung](#).

PHP Hypertext Preprocessor

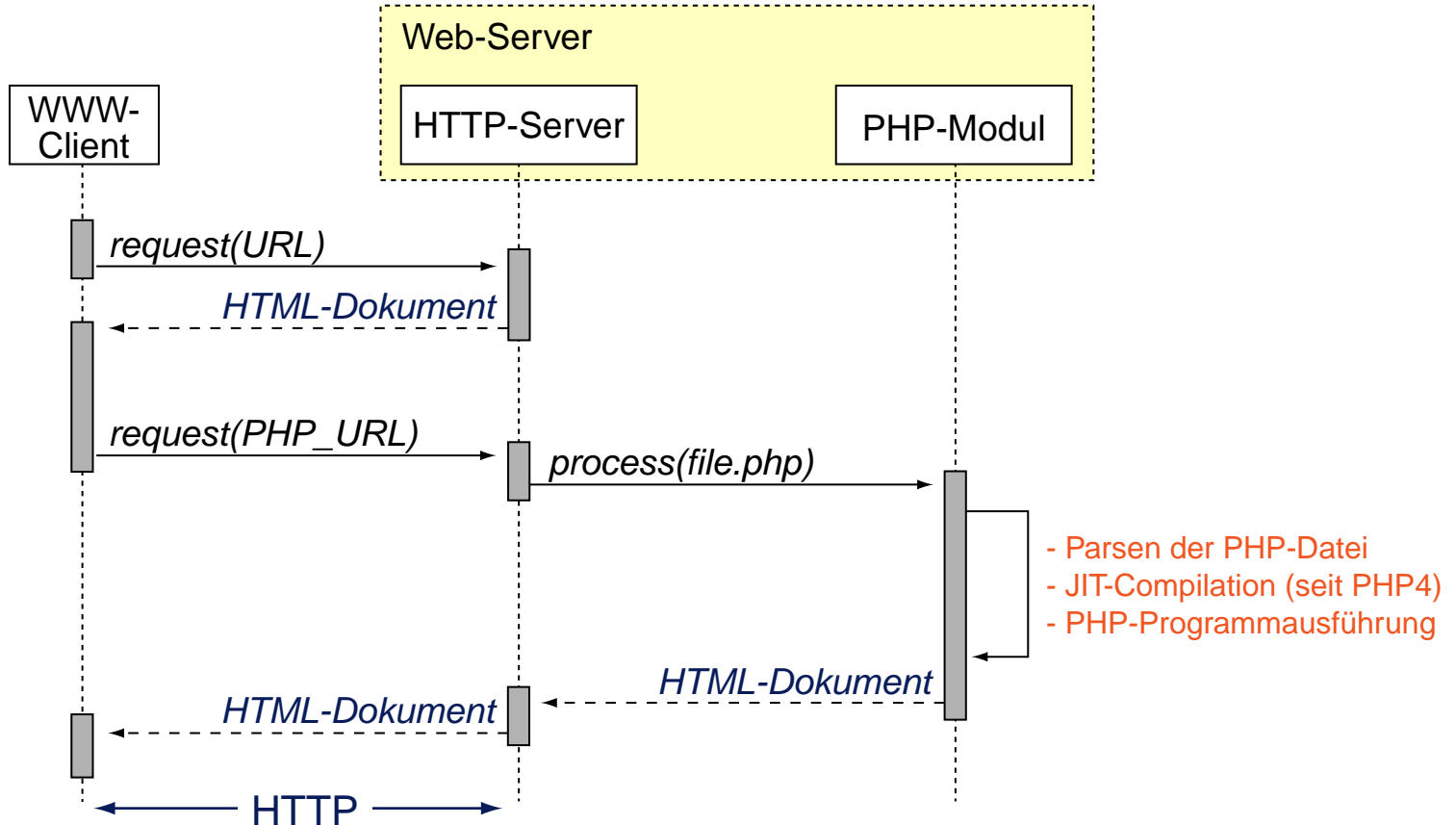
Einführung (Fortsetzung)

[Demo]



PHP Hypertext Preprocessor

Ablauf einer PHP-Interaktion



Vergleiche hierzu den Ablauf einer JSP-Interaktion.

Bemerkungen:

- ❑ Die `.php`-Datei kombiniert HTML und PHP-Code im Sinne eines Programms zur Erzeugung einer HTML-Seite.
- ❑ Das Action-Attribut `<... action="registration.php">` im Beispiel ist eingegraut gezeigt, obwohl es laut Referenz obligatorisch ist. Als Default-Wert für das Action-Attribut wird die Datei (hier: `registration.php`) selbst genommen.
- ❑ Wie bei JSP gibt es auch hier kein isoliertes CGI-Programm mehr: HTML-Form, HTML-Antwort, Programm zur Verarbeitung – alles befindet sich in derselben Datei.

PHP Hypertext Preprocessor

Historie [\[php.net\]](http://php.net)

- 1994 Rasmus Lerdorf entwickelt erste Version der „Personal Home Page Tools“ PHP zur Erfassung der Zugriffe auf seine Web-Seiten.
- 1995 PHP/FI 1. Neuer Parser und Erweiterung auf HTML-Forms (FI = Form Interpreter).
- 1996 PHP/FI 2. Open Source Gemeinde steigt in die Weiterentwicklung ein.
- 1997 PHP 3. Neuentwicklung unter Leitung von Andi Gutmans und Zeev Suraski. Konsistente Syntax, objektorientierte Konzepte. Wird von mehr als 50.000 Entwicklern verwendet. Umbenennung in „PHP Hypertext Preprocessor“.
- 1999 Gründung von Zend ([Ze]ev Suraski und A[nd]i Gutmanns) Technologies. [\[zend.com\]](http://zend.com)
- 2000 PHP 4. Leistungsfähiger Parser (Zend-Engine), modularer Basiscode, HTTP-Sessions, Ausgabepufferung, sicherere Benutzereingaben, umfangreiche Datenbankunterstützung.
- 2004 PHP 5. Zend Engine 2.0, neues Objektmodell mit public-/private-/protected-Modifizierer, deutlich verbesserte Unterstützung der XML-Konzepten DOM, SAX, XSLT.
- 2009 PHP 5.3. Namespaces, Late Static Bindings, Closures und Lambda-Kalkül.
- 2011 Aktuelle Version von PHP: [\[php.net\]](http://php.net)

PHP Hypertext Preprocessor

Einbindung in HTML-Dokumente

Der PHP-Prozessor erhält das gesamte HTML-Dokument, interpretiert aber nur die Anweisungen, die als PHP-Code ausgezeichnet sind. Der übrige Text wird unverändert zum Client gesendet.

Syntaxalternativen zur Auszeichnung:

1. Standard-Tags:

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

2. Sprachspezifische Script-Deklaration:

```
<script language="php"> ... </script>
```

3. ASP-Stil:

```
<% ... %>
```

4. Kurzschreibweise einer SGML-Verarbeitungsanweisung:

```
<? ... ?>
```

Bemerkungen:

- ❑ Statistik zur Verbreitung von PHP: [\[php.net\]](http://php.net)
- ❑ Die Kurzschreibweisen mit „<?“ bzw. „<%“ müssen in der Konfiguration von PHP aktiviert sein. Aus Sicht der Portabilität von PHP-Dateien sind sie nicht sinnvoll; insbesondere wird „<?“ in Zukunft nicht mehr unterstützt. [\[php.net\]](http://php.net)
- ❑ Es ist eine Frage des Stils, ob PHP-Code in mehrere Abschnitte aufgeteilt wird, zwischen denen HTML-Code steht, oder ob HTML-Code durch die PHP-Funktion `echo()` ausgegeben wird. Die erste Variante ist performanter.
- ❑ `echo()` ist keine (Built-in-)Funktion sondern ein Element der Sprache und wird hinsichtlich ihrer Argumente besonders behandelt. [\[php.net\]](http://php.net)

PHP Hypertext Preprocessor

Grundlagen der Syntax

Bezeichner [php.net]

- ❑ der Grundaufbau von Namen folgt der Form $[a-zA-Z_][a-zA-Z_0-9]^*$
- ❑ Variablennamen beginnen immer mit `$`.
Groß-/Kleinschreibung wird unterschieden (*case sensitive*).
- ❑ Konstantennamen werden ohne `$` geschrieben.
- ❑ Funktionsnamen sind *case insensitive*.

PHP Hypertext Preprocessor

Grundlagen der Syntax

Bezeichner [\[php.net\]](http://php.net)

- ❑ der Grundaufbau von Namen folgt der Form `[a-zA-Z_][a-zA-Z_0-9]*`
- ❑ Variablennamen beginnen immer mit `$`.
Groß-/Kleinschreibung wird unterschieden (*case sensitive*).
- ❑ Konstantennamen werden ohne `$` geschrieben.
- ❑ Funktionsnamen sind *case insensitive*.

Anweisungen [\[php.net\]](http://php.net)

- ❑ Jede Anweisung wird mit einem Semikolon beendet; auch der schließende Tag „`?>`“ beendet eine Anweisung.

```
<?php  
    echo "Hello world!";    ≈    <?php echo "Hello word!" ?>  
?>
```

- ❑ `//` kommentiert bis Zeilenende aus.
- ❑ balancierte Kommentarklammerung: `/* Kommentar */`

PHP Hypertext Preprocessor

Variablen [vgl. [JavaScript](#)]

- ❑ Variablen werden durch ihre Initialisierung definiert.
- ❑ Eine Variable kann Werte beliebigen Typs annehmen.
- ❑ Es wird zwischen **lokalen**, **globalen**, **statischen** und vordefinierten (superglobalen) Variablen unterschieden. [[php.net](#)]
- ❑ Eine Variable ist global, wenn Sie außerhalb des Bindungsbereiches einer Funktion steht.
- ❑ Globale und vordefinierte Variablen gelten im ganzen Programm. Im Bindungsbereich einer Funktion sind globale Variablen mit dem Schlüsselwort `global` sichtbar zu machen; vordefinierte Variablen sind immer sichtbar.
- ❑ Statische Variablen existieren nur im Bindungsbereich einer Funktion; ihr Wert geht beim Verlassen dieses Bereichs nicht verloren.
- ❑ Der Gültigkeitsbereich von Konstanten entspricht dem von vordefinierten Variablen. Konstanten werden durch Funktion `define()` definiert.

PHP Hypertext Preprocessor

Variablen: Illustration von Geltungsbereichen

```
<?php
    $a = 1; // globaler Bereich

    function test() {
        echo $a; // Referenz auf den Bindungsbereich von test()
    }

    test();
?>
```

PHP Hypertext Preprocessor

Variablen: Illustration von Geltungsbereichen

```
<?php
    $a = 1; // globaler Bereich

    function test() {
        echo $a; // Referenz auf den Bindungsbereich von test()
    }

    test();
?>
```

```
<?php
    $a = 1;
    $b = 2;

    function Summe() {
        global $a, $b;
        $b = $a + $b;
    }

    Summe();
    echo $b;
?>
```

PHP Hypertext Preprocessor

Variablen: Illustration statischer Variablen

```
<?php
function fak($n) {
    static $m = 1; // Initialisierung der statischen Variable (einmalig)
    if ($n == 1) {
        echo $m; // Ausgabe des Ergebnisses
        $m=1; // Zurücksetzen der statischen Variable
    }
    else {
        $m *= $n;
        fak(--$n);
    }
}

fak(10);

?>
```

PHP Hypertext Preprocessor

Variablen: besondere Konzepte

- variable Variablen: `$$VarName`

Der Inhalt von `$VarName` wird als Variablenname verwendet.

- Referenzen: `$VarName2 = & $VarName1`

Neuer (Alias)Variablenname für den Inhalt von `$VarName1`. [php.net]

- Überprüfung, ob eine Variable definiert ist:

```
boolean isset($VarName)
```

- Löschen einer Variablen:

```
void unset($VarName)
```

- Zeigt Informationen über eine Variable in lesbarer Form an:

```
boolean print_r($VarName)
```

- Die vordefinierten (superglobalen) Variablen:

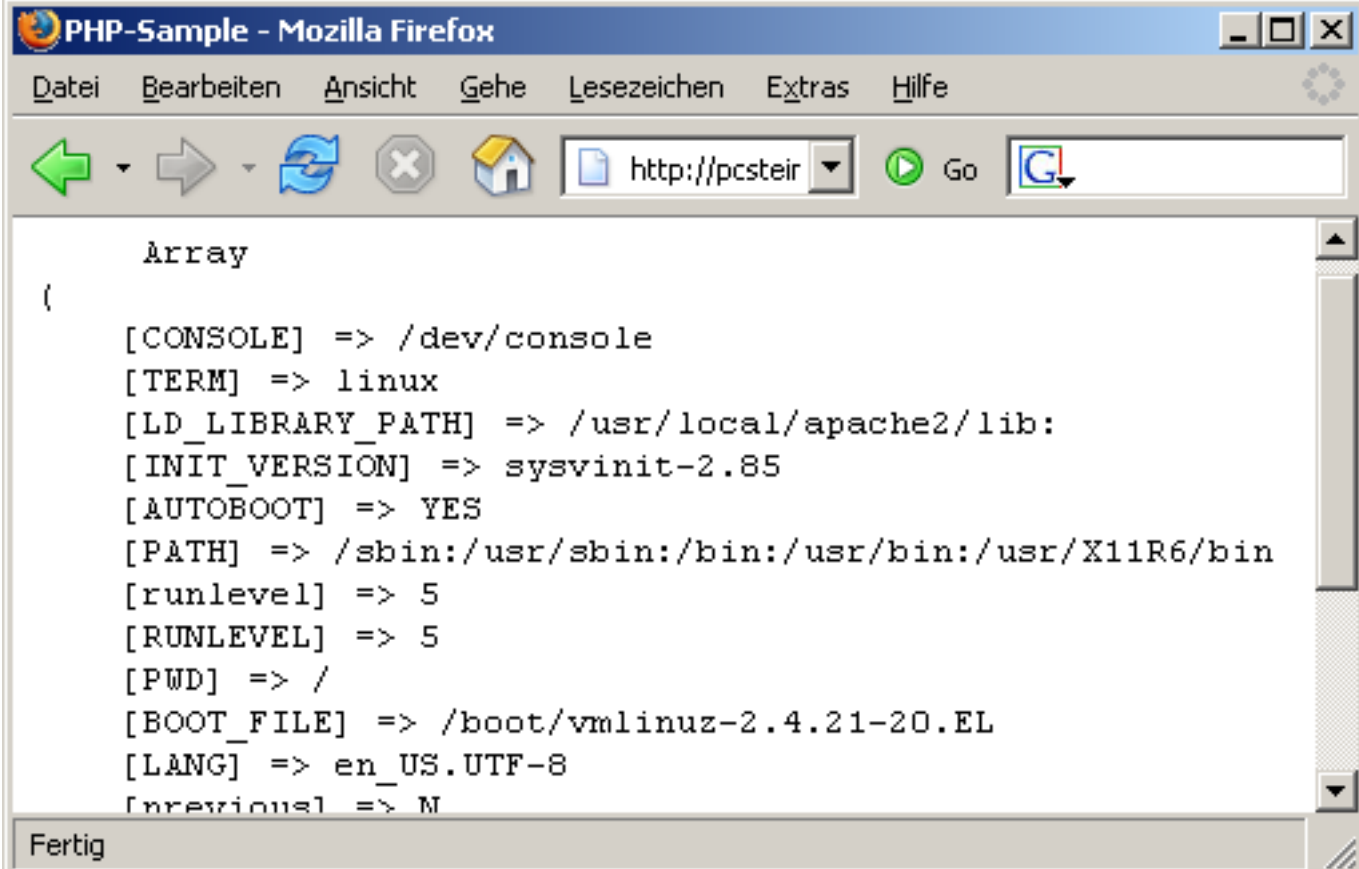
```
$GLOBALS, $_SERVER, $_GET, $_POST, $_FILES, $_COOKIE, $_SESSION,  
$_REQUEST, $_ENV [php.net]
```

PHP Hypertext Preprocessor

Variablen: besondere Konzepte (Fortsetzung)

```
<pre>
  <?php print_r($_REQUEST); ?>
  <?php print_r($_ENV); ?>
</pre>
```

[\[Demo\]](#)



The screenshot shows a Mozilla Firefox browser window titled "PHP-Sample - Mozilla Firefox". The address bar contains "http://pcsteir". The main content area displays the output of a PHP script, which is a PHP array of environment variables. The output is as follows:

```
Array
(
    [CONSOLE] => /dev/console
    [TERM] => linux
    [LD_LIBRARY_PATH] => /usr/local/apache2/lib:
    [INIT_VERSION] => sysvinit-2.85
    [AUTOBOOT] => YES
    [PATH] => /sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin
    [runlevel] => 5
    [RUNLEVEL] => 5
    [PWD] => /
    [BOOT_FILE] => /boot/vmlinuz-2.4.21-20.EL
    [LANG] => en_US.UTF-8
    [previous] => M
)
```

The status bar at the bottom of the browser window shows the word "Fertig".

Bemerkungen:

- ❑ Vordefinierte Variablen können nicht als variable Variablen verwendet werden. [[php.net](#)]
- ❑ Referenzen sind ein Mechanismus, um verschiedene Namen für den gleichen Inhalt von Variablen zu ermöglichen. Sie sind nicht mit Zeigern in C zu vergleichen, sondern Alias-Definitionen in der Symboltabelle: der gleiche Variableninhalt kann unterschiedliche Namen besitzen, ähnlich dem Konzept der Hardlinks im Unix-Dateisystem. [[php.net](#)]

PHP Hypertext Preprocessor

Datentypen [vgl. [JavaScript](#)]

`integer`, `float`

- Ganzzahlen können dezimal, hexadezimal oder oktal notiert werden. Bei Überlauf findet eine Konvertierung nach `float` statt.

`string` [[php.net](#)]

- Einfachen Anführungszeichen. Alle Zeichen stehen für sich selbst; nur `'` muss „escaped“ werden: `\'`
- Doppelte Anführungszeichen. Der String wird geparkt und eventuell vorkommende Variablen und Escape-Folgen ersetzt. Beispiel:

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

- Konkatenation mit Punkt: `"Hello" . "world!"`
- Ausgabe von Ausdrücken, deren Rückgabewert eine Zeichenkette ist:

<code>echo</code>	durch Kommata getrennte Folge von Ausdrücken
<code>print</code>	einzelner Ausdruck

PHP Hypertext Preprocessor

Datentypen

boolean

- **Literale:** `true` und `false`.
Groß-/Kleinschreibung wird nicht unterschieden.
- **Operatoren:** Konjunktion `&&` bzw. `and`, Disjunktion `||` bzw. `or`, Negation `!` und Exklusiv-Oder `xor`.

NULL

- Der spezielle Wert `NULL` steht dafür, dass eine Variable keinen Wert hat.
- `NULL` ist der einzig mögliche Wert des Typs `NULL`. Groß-/Kleinschreibung wird nicht unterschieden.
- Eine Variable wird als `NULL` interpretiert, wenn (a) ihr die Konstante `NULL` als Wert zugewiesen wurde, (b) ihr bis jetzt kein Wert zugewiesen wurde, oder (c) sie mit `unset()` gelöscht wurde.

PHP Hypertext Preprocessor

Datentypen: Arrays [vgl. [JavaScript](#)]

Ein Array ist eine Abbildung von Indizes auf Werte. Jedes Element eines Arrays ist ein Paar bestehend aus numerischem oder String-Index und zugeordnetem Wert.

Erzeugung von Arrays:

- (a) Mit der `array()`-Funktion als Liste von Werten, indiziert von 0 an:
- (b) Durch explizite Indizierung:
- (c) Mit der `array()`-Funktion als Liste von Index-Wert-Paaren oder als assoziatives Array:

PHP Hypertext Preprocessor

Datentypen: Arrays [vgl. [JavaScript](#)]

Ein Array ist eine Abbildung von Indizes auf Werte. Jedes Element eines Arrays ist ein Paar bestehend aus numerischem oder String-Index und zugeordnetem Wert.

Erzeugung von Arrays:

- (a) Mit der `array()`-Funktion als Liste von Werten, indiziert von 0 an:

```
$monatsName = array("", "Jan", ..., "Dez");
```

- (b) Durch explizite Indizierung:

```
$monatsName[1]= "Jan"; monatsName[2]= "Feb"; ...
```

- (c) Mit der `array()`-Funktion als Liste von Index-Wert-Paaren oder als assoziatives Array:

```
$monatsName = array(1 => "Jan", ..., 12 => "Dez");
```

```
$monatsName = array("Jan" => 1, ..., "Dez" => 12);
```

PHP Hypertext Preprocessor

Datentypen: Arrays [vgl. [JavaScript](#)]

Ein Array ist eine Abbildung von Indizes auf Werte. Jedes Element eines Arrays ist ein Paar bestehend aus numerischem oder String-Index und zugeordnetem Wert.

Erzeugung von Arrays:

- (a) Mit der `array()`-Funktion als Liste von Werten, indiziert von 0 an:

```
$monatsName = array("", "Jan", ..., "Dez");
```

- (b) Durch explizite Indizierung:

```
$monatsName[1]= "Jan"; monatsName[2]= "Feb"; ...
```

- (c) Mit der `array()`-Funktion als Liste von Index-Wert-Paaren oder als assoziatives Array:

```
$monatsName = array(1 => "Jan", ..., 12 => "Dez");
```

```
$monatsName = array("Jan" => 1, ..., "Dez" => 12);
```

Aufzählung aller Elemente **mit Schlüssel**:

```
foreach ($monatsName as $key => $value) {  
    echo "Index-Wert-Paar: " . $key . "=>" . $value . "<br/>";  
}
```

PHP Hypertext Preprocessor

Datentypen: Konversion

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

- **explizite** Konversion (*Type Cast*)

Der Zieltyp, in den der Wert eines Ausdruckes umgewandelt werden soll, wird explizit angegeben. Beispiel:

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

- **implizite** Konversion (*Coercion*)

Wenn der Typ eines Wertes nicht zu der darauf angewandten Operation passt, wird versucht, den Typ anzupassen. Beispiel:

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

Die ganze Zahl 42 wird in die Zeichenreihe "42" konvertiert. Der Wert der Variablen `$sum` bleibt unverändert.

PHP Hypertext Preprocessor

Kontrollstrukturen [vgl. [JavaScript](#)]

- Anweisungsfolge:
- Bedingte Anweisung:
- `while`-Schleife:
- `do-while`-Schleife:
- `for`-Schleife:

PHP Hypertext Preprocessor

Kontrollstrukturen [vgl. [JavaScript](#)]

□ Anweisungsfolge:

```
{ $i = $i+1; print "Hello world!"; }
```

□ Bedingte Anweisung:

```
if ($a < $b) {$min = $a;} else {$min = $b;}
```

Bei einzelnen Anweisungen sind die {}-Klammern optional.

□ while-Schleife:

□ do-while-Schleife:

□ for-Schleife:

PHP Hypertext Preprocessor

Kontrollstrukturen [vgl. [JavaScript](#)]

□ Anweisungsfolge:

```
{ $i = $i+1; print "Hello world!"; }
```

□ Bedingte Anweisung:

```
if ($a < $b) {$min = $a;} else {$min = $b;}
```

Bei einzelnen Anweisungen sind die {}-Klammern optional.

□ while-Schleife:

```
$i = 0; while ($i < 42) {$stars = $stars . "*"; $i = $i+1;}
```

□ do-while-Schleife:

```
$i = 0; do {$stars = $stars . "*"; $i = $i+1;} while ($i < 42);
```

□ for-Schleife:

```
for ($i = 0; $i < 12; $i++) {echo $i, $monatsName[$i], "\n";}
```

PHP Hypertext Preprocessor

Kontrollstrukturen [vgl. [JavaScript](#)]

□ Anweisungsfolge:

```
{ $i = $i+1; print "Hello world!"; }
```

□ Bedingte Anweisung:

```
if ($a < $b) {$min = $a;} else {$min = $b;}
```

Bei einzelnen Anweisungen sind die {}-Klammern optional.

□ while-Schleife:

```
$i = 0; while ($i < 42) {$stars = $stars . "*"; $i = $i+1;}
```

□ do-while-Schleife:

```
$i = 0; do {$stars = $stars . "*"; $i = $i+1;} while ($i < 42);
```

□ for-Schleife:

```
for ($i = 0; $i < 12; $i++) {echo $i, $monatsName[$i], "\n";}
```

□ Parameterübergabe standardmäßig mittels **call-by-value**.

Bemerkungen:

- ❑ call-by-value: Der formale Parameter (Funktionsdefinition) ist eine Variable, die mit dem Wert des aktuellen Parameters (Funktionsaufruf) initialisiert wird.
- ❑ Notiert man ein „&“ vor dem formalen Parameter (Funktionsdefinition) oder vor dem aktuellen Parameter (Funktionsaufruf), geschieht die Übergabe für diesen Parameter durch call-by-reference. [php.net]

PHP Hypertext Preprocessor

Funktionsbibliothek [vgl. [JavaScript](#)]

Wesentliches Merkmal von PHP ist die große, sehr leistungsfähige Funktionsbibliothek (> 700 Funktionen), die in jedem PHP-Programm zur Verfügung steht. Anwendungsbereiche sind u.a.:

- ❑ Arrays
- ❑ Protokolle
- ❑ **Datenbanken**
- ❑ Datum/Uhrzeit
- ❑ Dateiverzeichnisse
- ❑ Dateien
- ❑ Grafik
- ❑ HTTP
- ❑ IMAP
- ❑ LDAP
- ❑ Mathematik
- ❑ MCAL
- ❑ Mcrypt
- ❑ Mhash
- ❑ PDF
- ❑ POSIX
- ❑ **reguläre Ausdrücke**
- ❑ Strings
- ❑ Variablenmanipulation
- ❑ XML

PHP Hypertext Preprocessor

Funktionsbibliothek: reguläre Ausdrücke [\[php.net\]](#)

Es gibt zwei „Engines“ zum Verarbeiten regulärer Ausdrücke:

1. POSIX-Engine.
2. PCRE-Engine. Perl-kompatibel und 200x schneller als POSIX.

Aufbau des Perl-kompatiblen Strings zur Definition regulärer Ausdrücke [\[php.net\]](#) :

Delimiter *Regular_Expression* *Delimiter* [*Modifiers*]

- Der Delimiter muss ein nicht-alphanumerisches Zeichen sein.
- Optionale Modifizierer beeinflussen die Match-Strategie. [\[php.net\]](#)

Beispiele:

```
echo preg_match( "/def/", "defabcdef" );
```

```
echo preg_match( "=def=", "defabcdef" );
```

PHP Hypertext Preprocessor

Funktionsbibliothek: reguläre Ausdrücke (Fortsetzung)

- ❑ `int preg_match(string Pattern, string String [, array Matches [, PREG_OFFSET_CAPTURE [, int Offset]])`

Durchsucht *String* nach der ersten Übereinstimmung mit dem durch *Pattern* definierten regulären Ausdruck. [\[php.net\]](http://php.net)

- ❑ `int preg_match_all(...)`

Sucht nach allen Übereinstimmungen. [\[php.net\]](http://php.net)

- ❑ `mixed preg_replace(...)`

Suchen und Ersetzen von Übereinstimmungen. [\[php.net\]](http://php.net)

- ❑ `mixed preg_replace_callback(...)`

Suchen und Ersetzen von Übereinstimmungen, wobei der Return-Wert einer Callback-Funktion den Ersetzungstext bestimmt. [\[php.net\]](http://php.net)

PHP Hypertext Preprocessor

Funktionsbibliothek: reguläre Ausdrücke (Fortsetzung) [vgl. [Exkurs](#)]

Ein regulärer Ausdruck R kann wie folgt rekursiv zusammengesetzt sein. F , G bzw. $L(F)$, $L(G)$ bezeichnen reguläre Ausdrücke sowie die definierten Sprachen.

R	Erklärung
a	das Zeichen a
FG	Zusammenfügen von zwei Worten
$F G$	Alternativen
(F)	Klammerung
$F?$	F ist optional; gleiche Semantik wie $F \varepsilon$
$F+$	nicht-leere Folge von Worten aus $L(F)$
F^*	beliebig lange Folge von Worten aus $L(F)$
$F\{m, n\}$	Folge mit mindestens m und höchstens n von Worten aus $L(F)$
$F\{m\}$	Folge mit genau m Worten aus $L(F)$
$[abc]$	alternativ ein Zeichen aus der Klammer
$[\^abc]$	alternativ ein anderes Zeichen als die in der Klammer
$[a - zA - Z]$	alternativ ein Zeichen aus Zeichenbereichen
$.$	beliebiges Zeichen
$^$	Anfang der Zeichenfolge (nichts darf vorangehen)
$\$$	Ende der Zeichenfolge (nichts darf darauf folgen)

PHP Hypertext Preprocessor

Funktionsbibliothek: reguläre Ausdrücke (Fortsetzung)

```
<?php
    $pattern = "/d[a-z]f/";
    $str = "defabcdef";
    $num = preg_match_all($pattern, $str,
        $matches, PREG_OFFSET_CAPTURE);
?>

<pre>
    <?php print_r($matches); ?>
</pre>
```

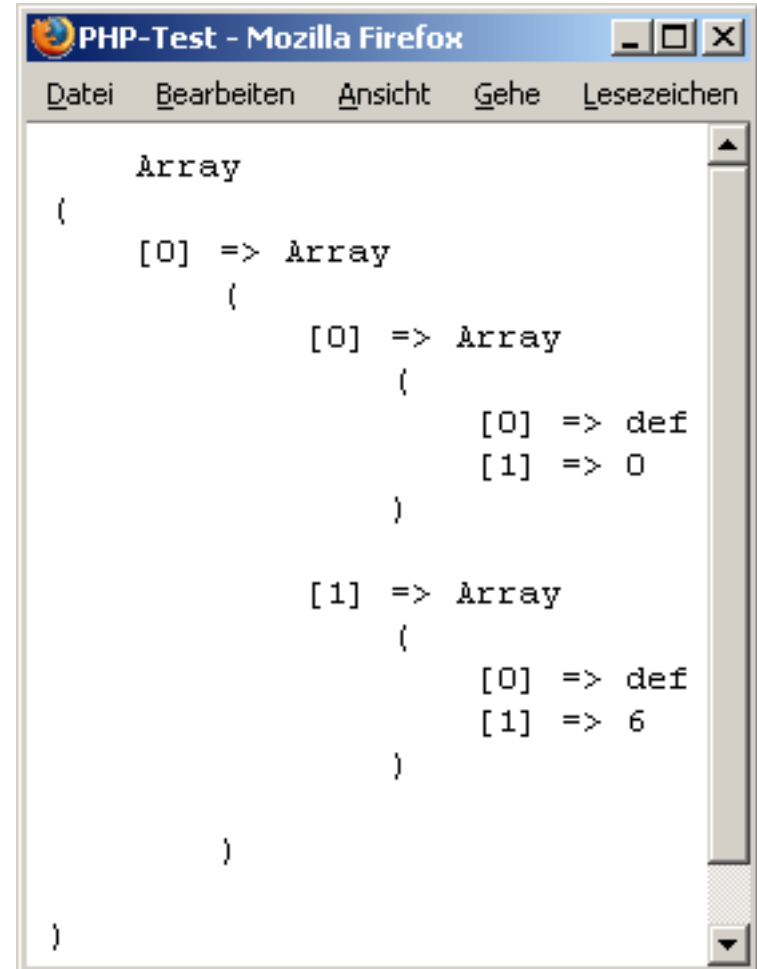
PHP Hypertext Preprocessor

Funktionsbibliothek: reguläre Ausdrücke (Fortsetzung)

```
<?php
    $pattern = "/d[a-z]f/";
    $str = "defabcdef";
    $num = preg_match_all($pattern, $str,
        $matches, PREG_OFFSET_CAPTURE);
?>

<pre>
    <?php print_r($matches); ?>
</pre>
```

[\[Demo\]](#)



```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [0] => def
                    [1] => 0
                )
            [1] => Array
                (
                    [0] => def
                    [1] => 6
                )
        )
)
```

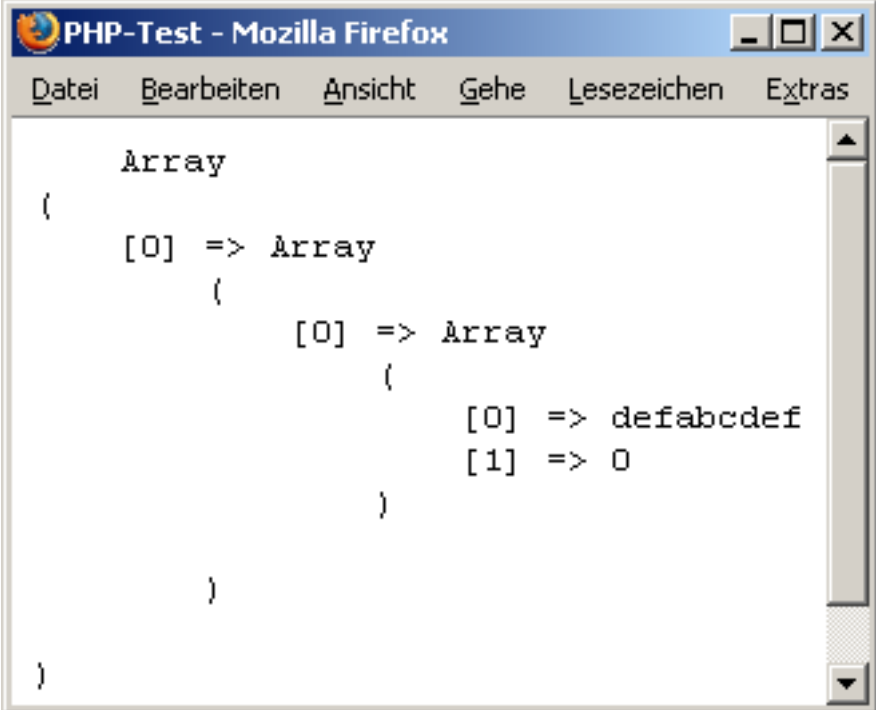
PHP Hypertext Preprocessor

Funktionsbibliothek: reguläre Ausdrücke (Fortsetzung)

```
<?php
    $pattern = "/d[a-z]*f/";
    $str = "defabcdef";
    $num = preg_match_all($pattern, $str,
        $matches, PREG_OFFSET_CAPTURE);
?>

<pre>
    <?php print_r($matches); ?>
</pre>
```

[Demo]



```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [0] => defabcdef
                    [1] => 0
                )
        )
)
```

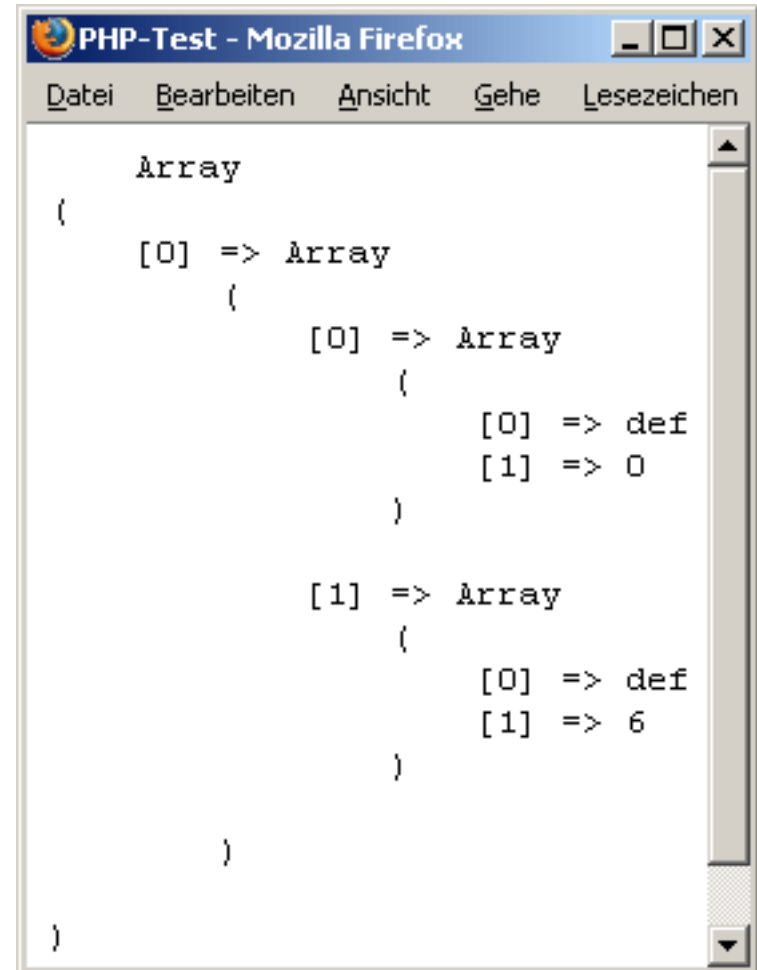
PHP Hypertext Preprocessor

Funktionsbibliothek: reguläre Ausdrücke (Fortsetzung)

```
<?php
    $pattern = "/d[a-z]*f/U";
    $str = "defabcdef";
    $num = preg_match_all($pattern, $str,
        $matches, PREG_OFFSET_CAPTURE);
?>

<pre>
    <?php print_r($matches); ?>
</pre>
```

[\[Demo\]](#)



```
Array
(
    [0] => Array
        (
            [0] => Array
                (
                    [0] => def
                    [1] => 0
                )
            [1] => Array
                (
                    [0] => def
                    [1] => 6
                )
        )
)
```

Bemerkungen:

- ❑ Innerhalb eines regulären Ausdrucks fungiert „\“ als Escape-Zeichen.
- ❑ Die Standardeinstellung für die Match-Bildung ist „gierig“ (*greedy*); hierbei wird der längste Match gesucht. Mit dem Ungreedy-Modifizierer „U“ wird in den Modus „nicht gierig“ umgeschaltet. [\[php.net\]](http://php.net)
- ❑ Unter Linux können reguläre Ausdrücke mit dem Shell-Programm `/usr/bin/pcretest` interaktiv getestet werden.

PHP Hypertext Preprocessor

Funktionsbibliothek: Datenbanken

Die PHP-Datenbankschnittstellen ermöglichen es, dynamisch HTML-Seiten basierend auf Datenbankinhalten zu generieren. Beispiele:

- ❑ Auswahl und Anzeige von Inhalten aus einer Produktdatenbank
- ❑ Benutzerverwaltung wie Abgleich von Benutzernamen und Passwörtern

Funktionalität der PHP-Datenbankschnittstellen:

- ❑ Erstellung und Verwaltung von Verbindungen zu Datenbank-Servern
- ❑ Auswahl von Datenbanken
- ❑ Generierung von SQL-Ausdrücken
- ❑ Zugriff auf die Ergebnisse von SQL-Anfragen

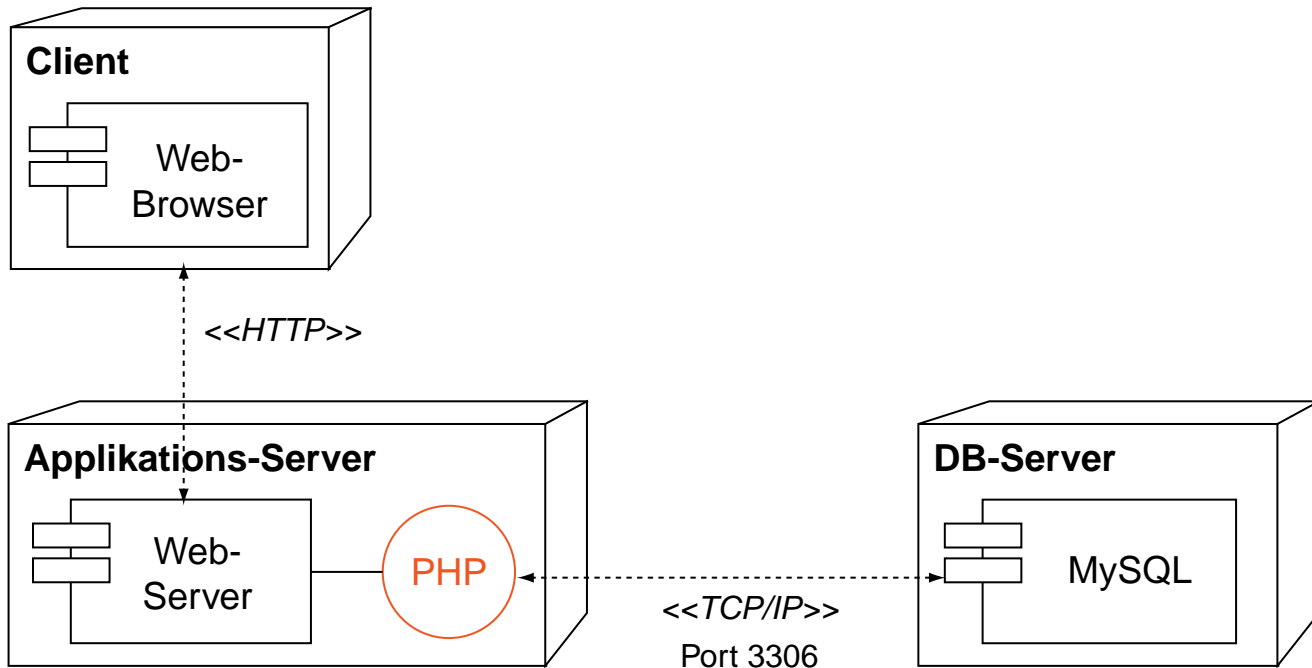
Zahlreiche Datenbanken werden unterstützt, u.a.: dBase-kompatible Formate, Berkeley-DB-Formate, Oracle, Sybase, MySQL, ODBC-Datenquellen. [\[php.net\]](#)

Bemerkungen:

- ❑ Eine weit verbreitete Software-Kombination zur Erstellung Web-basierter Datenbankanwendungen ist unter dem Schlagwort „Lamp“ bzw. „Wamp“ bekannt: Linux (bzw. Windows) + Apache + MySQL + PHP.

PHP Hypertext Preprocessor

Funktionsbibliothek: Datenbanken (Fortsetzung)



Beispielszenario:

- ❑ Auf einem Web-Server liegt eine Kundendatenbank, u.a. mit der `addresses`-Relation.
- ❑ Aufgabe: Realisierung eines Web-Interfaces zur Suche, Sortierung und Auflistung von Kundendaten.

PHP Hypertext Preprocessor

Funktionsbibliothek: Datenbanken (Fortsetzung)

1. Mit Datenbank-Server verbinden:

```
$dbhost = "webis2.uni-weimar.de";  
$dbuser = "peter";  
$dbpassword = "secret";  
$connection = mysql_connect($dbhost, $dbuser, $dbpassword)  
    or die("Could not connect: " . mysql_error());
```

2. Datenbank auswählen:

```
$dbname = "customers";  
mysql_select_db($dbname, $connection)  
    or die("Could not select database");
```

PHP Hypertext Preprocessor

Funktionsbibliothek: Datenbanken (Fortsetzung)

1. Mit Datenbank-Server verbinden:

```
$dbhost = "webis2.uni-weimar.de";  
$dbuser = "peter";  
$dbpassword = "secret";  
$connection = mysql_connect($dbhost, $dbuser, $dbpassword)  
    or die("Could not connect: " . mysql_error());
```

2. Datenbank auswählen:

```
$dbname = "customers";  
mysql_select_db($dbname, $connection)  
    or die("Could not select database");
```

3. SQL-Ausdruck konstruieren und auswerten:

```
$table = "addresses";  
$searchstring = $_REQUEST["searchstring"];  
$query = "SELECT lastname, firstname, phone FROM $table  
    WHERE lastname LIKE '%$searchstring%'  
    ORDER BY lastname, firstname";  
$result = mysql_query($query, $connection)  
    or die("Query failed: " . mysql_error());
```

PHP Hypertext Preprocessor

Funktionsbibliothek: Datenbanken (Fortsetzung)

4. Anzahl von Ergebniszeilen prüfen:

```
if (mysql_num_rows($result) == 0) {  
    echo "<h1>Kein Kunde mit dem Namen $searchstring  
vorhanden.</h1>";  
}
```

5. Datensätze aus assoziativem Array auslesen:

```
echo "<ol>";  
while ($row = mysql_fetch_assoc($result)) {  
    echo "<li>";  
    echo $row["lastname"] . ", ";  
    echo $row["firstname"] . " ";  
    echo $row["phone"];  
    echo "</li>";  
}  
echo "</ol>";
```

6. Speicher freigeben:

```
mysql_free_result($result);
```

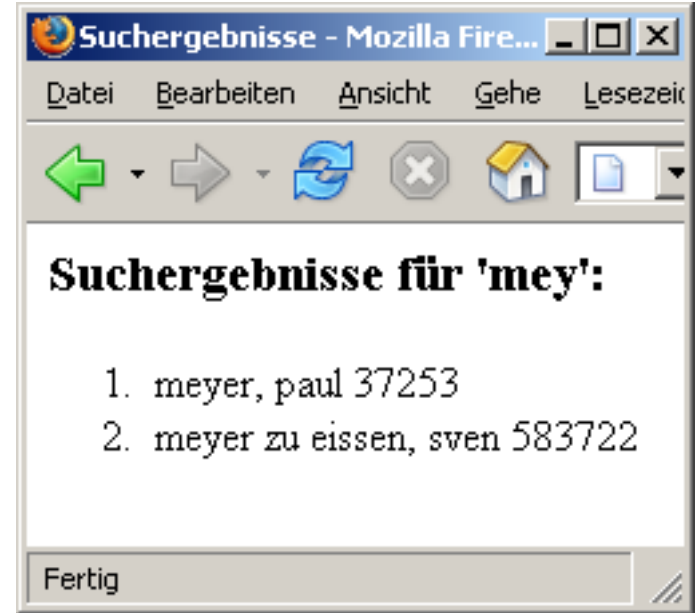
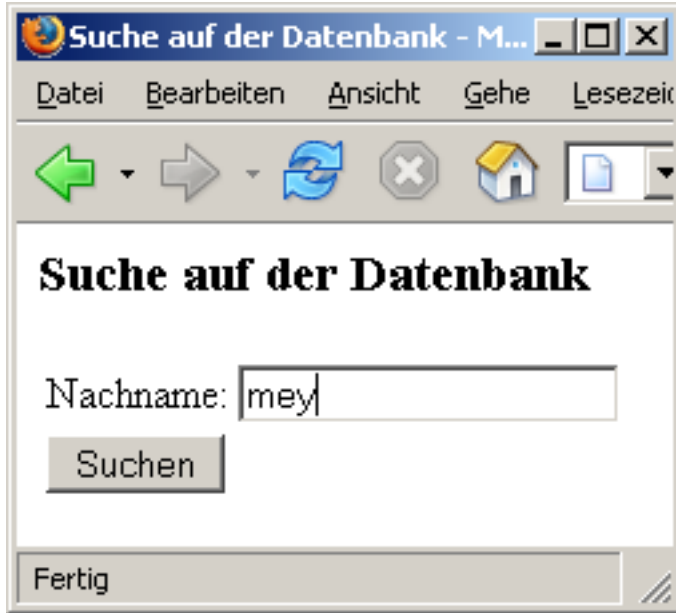
Bemerkungen:

- ❑ Wird `mysql_select_db` ohne `$connection`-Parameter aufgerufen, wird sich auf die zuletzt geöffnete Verbindung bezogen.
- ❑ `mysql_pconnect()`
Variante zum Aufbau einer persistenten Verbindung.
- ❑ `die(string Message)`
Ausgabe von *Message* und Beendigung des aktuellen Skripts.
- ❑ `mysql_fetch_assoc(resource Result)`
Gibt ein assoziatives Array zurück, das den nächsten Datensatz aus *Result* enthält. Sind keine weiteren Datensätze vorhanden, wird `FALSE` zurückgegeben.

PHP Hypertext Preprocessor

Funktionsbibliothek: Datenbanken (Fortsetzung)

[Demo]



Server-Technologien

Vergleich der Technologien

	Programmieraufwand	Anwendungen	Abhängigkeit vom Server	Abhängigkeit v. Prog.sprache
CGI	mittel	allgemeiner Zugriff auf Informationsquellen	keine	klein
Web-Server-Bibliotheken	hoch	komplexere, Performanz-kritische Anwendungen	hoch	meist C/C++
Servlets	mittel	"	mittel	nur Java
SSI	sehr gering	Wiederverwendung von Dokumentteilen	gering	keine
PHP	mittel	allgemeiner Zugriff auf Informationsquellen	hoch	klein

[vgl. Turau 1999]

Server-Technologien

Vergleich der Technologien (Fortsetzung)

Kriterium	CGI, Perl	PHP	Servlets
Wartbarkeit	+	++	+++++
Performance	+	+++	+++++
Skalierbarkeit	+	+++	+++++
Verfügbarkeit	+++++	++++	+++++
Plattformneutralität	+++++	+++	+++
Abstraktionslevel	+	+++	+++++
Erweiterbarkeit	+++	++	+++++
Dokumentation	++++	++++	+++
Support	+++++	+++++	+++
Tool-Unterstützung	+	+	++++

[vgl. Wöhr 2004]

Server-Technologien

Quellen zum Nachlernen und Nachschlagen im Web

- ❑ Leibniz-Rechenzentrum. *Reguläre Sprachen, reguläre Ausdrücke.*
www.lrz-muenchen.de/services/schulung/unterlagen/regul
- ❑ D. Enseleit. *SELFPHP.*
www.selfphp.info
- ❑ PHP-Dokumentationsgruppe. *PHP Documentation.*
www.php.net/docs.php
- ❑ J. Stärk. *PHP Tutorial.*
www.usegroup.de/software/phptutorial
- ❑ W3 Schools. *PHP Tutorial.*
www.w3schools.com/php
- ❑ V. Vaswani. *PHP 101: PHP For the Absolute Beginner.*
devzone.zend.com/node/view/id/627
- ❑ Zend. *Eclipse PHP Development Tools (PDT).*
www.zend.com/community/pdt