Kapitel WT:IV

IV. Server-Technologien

- Web-Server
- Common Gateway Interface CGI
- Web-Container und -frameworks
- Web-Template-Engines
- Exkurs: reguläre Ausdrücke
- PHP Hypertext Preprocessor
- PHP Funktionsbibliotheken

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.

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 Abstraktionsebenen. Die Ebene 1 behandelt die Notation von Grundsymbolen, die Ebene 2 behandelt die syntaktische Struktur der Sprache. [WT:V Exkurs: Programmiersprachen]
- 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} bzw. {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, $N \cap \Sigma = \emptyset$
- P endliche Menge von Produktionen bzw. Regeln

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

- S Startsymbol, $S \in N$

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, $N \cap \Sigma = \emptyset$
- P endliche Menge von Produktionen bzw. Regeln

$$P \subset \underbrace{(N \cup \Sigma)^* \, N \, (N \cup \Sigma)^*}_{A} \times \underbrace{(N \cup \Sigma)^*}_{Ab}$$

- S Startsymbol, $S \in N$

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: $w \to w'$
- Gegeben die Regel $w \to w'$, dann stehen w, w' in der sogenannten *Transitionsrelation* $\to_G$. 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:

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

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

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:

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

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

Beispiel:

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

$$\begin{array}{lcl@{\qquad}lcl} S & \to & ABS & BA & \to & AB \\ S & \to & \varepsilon & BS & \to & b \\ & & & Bb & \to & bb \\ & & & Ab & \to & ab \\ & & & Aa & \to & aa \end{array}$$

Bemerkungen:

- Es ist Konvention, die Nichtterminalsymbole mit Großbuchstaben und die Terminalsymbole mit Kleinbuchstaben zu bezeichnen.
- Zur Erzeugung einer rekursiv aufzählbaren Sprache (Typ 0, für die Regeln in P existieren keine Einschränkungen) existieren abzählbar unendlich viele Grammatiken.
- Eine andere Grammatik, die die gleiche Sprache wie im Beispiel erzeugt, ist:
  $N = \{S, A, B\}$, $\Sigma = \{a, b\}$, $P = \{S \to aSb, S \to \varepsilon\}$

Exkurs: reguläre Ausdrücke

Grundlagen: Chomsky-Hierarchie

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

- Typ 0.
- Typ 1 $\sim$ kontextsensitiv.
- Typ 2 $\sim$ kontextfrei.
- Typ 3 $\sim$ regulär.

Exkurs: reguläre Ausdrücke

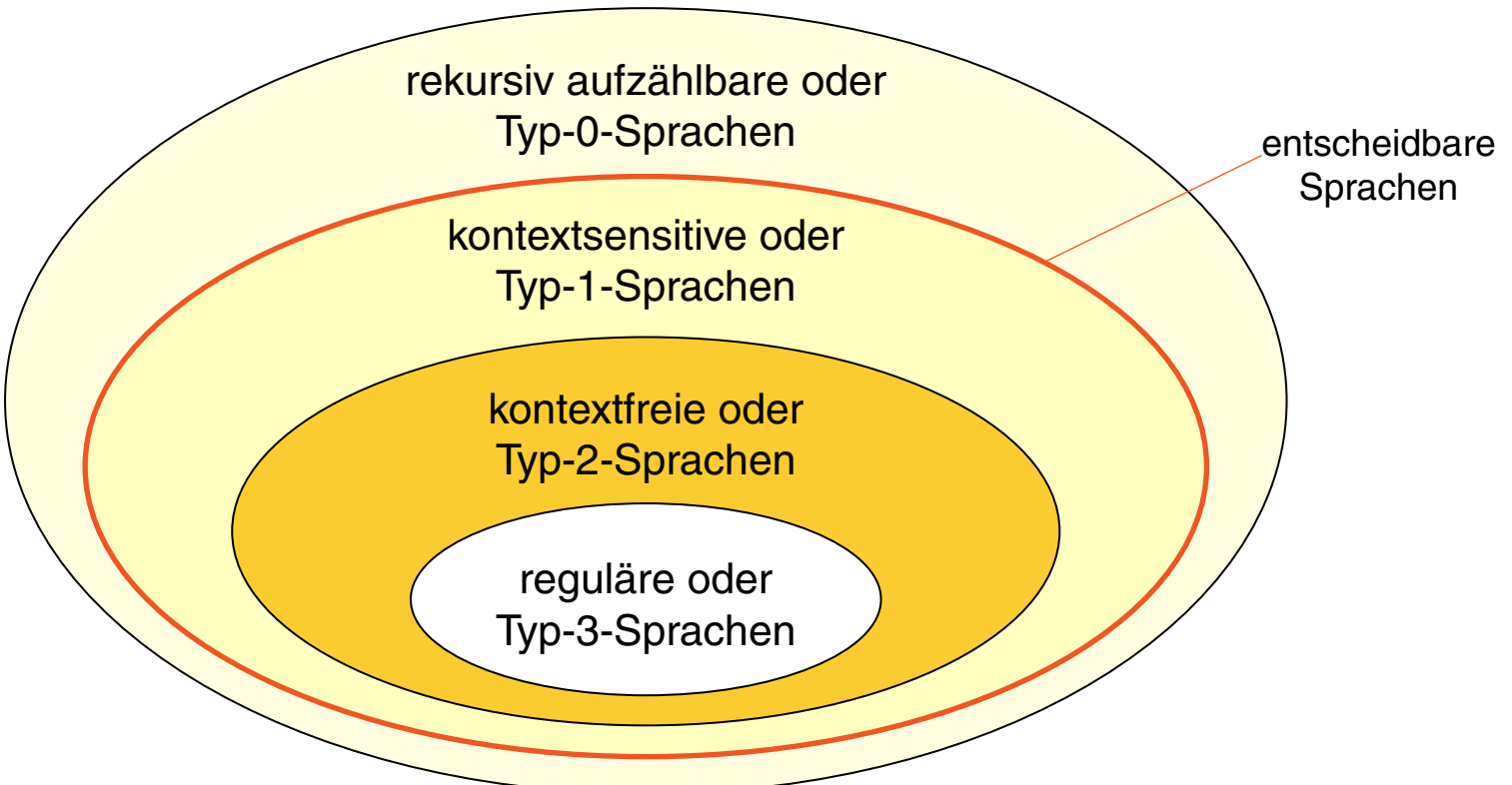
Grundlagen: Chomsky-Hierarchie

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 $w \to w' \in P$ gilt: $|w| \leq |w'|$
- Typ 2 $\sim$ kontextfrei.
  Für alle Regeln $w \to w' \in P$ gilt: w ist eine einzelne Variable; d.h., $w \in N$.
- Typ 3 $\sim$ regulär.
  Die Grammatik ist vom Typ 2 und für alle Regeln, $w \to w'$, gilt zusätzlich: $w' \in (\Sigma \cup \Sigma N)$, d.h., die rechten Seiten der Regeln bestehen entweder aus einem Terminalsymbol oder aus einem Terminalsymbol 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:
  - $\rightarrow$ Das Wortproblem für alle Sprachen vom Typ 1, 2 oder 3 ist entscheidbar.
  - $\rightarrow$ 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 (der Nerode-Relation)

Exkurs: reguläre Ausdrücke

Kalküle für reguläre Sprachen

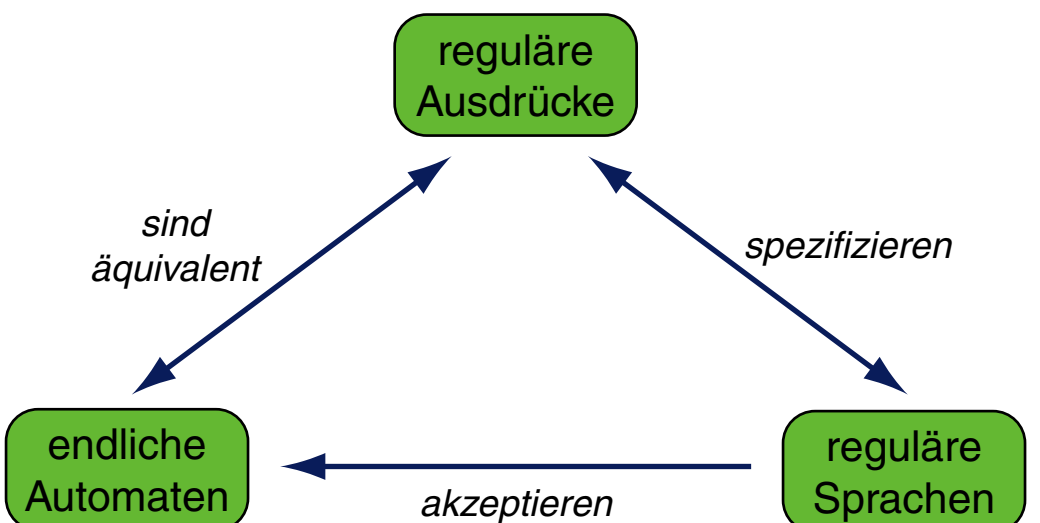
Verschiedene Kalküle zur Bildung von Worten einer regulären Sprache:

(a) endlichen Akzeptor bzw. Automat

(b) regulärer Ausdruck

(c) Typ-3-Grammatik

(d) Angabe endlich vieler Äquivalenzklassen (der Nerode-Relation)



[Haenelt 2005, Jurafski/Martin 2000]

Exkurs: reguläre Ausdrücke

Grundlagen: Zusammenfassung

	Kalküle zur Spracherzeugung
Typ 0	Typ-0-Grammatik
	Turingmaschine
Typ 1	kontextsensitive Grammatik
	linear beschränkte Turingmaschine [Wikipedia]
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
	Turingmaschine
Typ 1	kontextsensitive Grammatik
	linear beschränkte Turingmaschine [Wikipedia]
Typ 2	kontextfreie Grammatik
	Kellerautomat
Typ 3	reguläre Grammatik (Typ-3-Grammatik)
	deterministischer/nicht-deterministischer endlicher Automat
	regulärer Ausdruck

	Komplexität des Wortproblems [Wikipedia]
Typ 0	unentscheidbar
Typ 1	exponentielle Komplexität, NP-hard
Typ 2	$O(n^3)$
Typ 3	lineare Komplexität

Exkurs: reguläre Ausdrücke [Kastens]

Konstruktion [PHP: reguläre Ausdrücke]

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.	(F)	Klammerung
5.	F^+	nicht-leere Folge von Worten aus $L(F)$
6.	F^*	beliebig lange Folge von Worten aus $L(F)$
7.	F^n	Folge von n Worten aus $L(F)$
8.	ε	das leere Wort

Exkurs: reguläre Ausdrücke [Kastens]

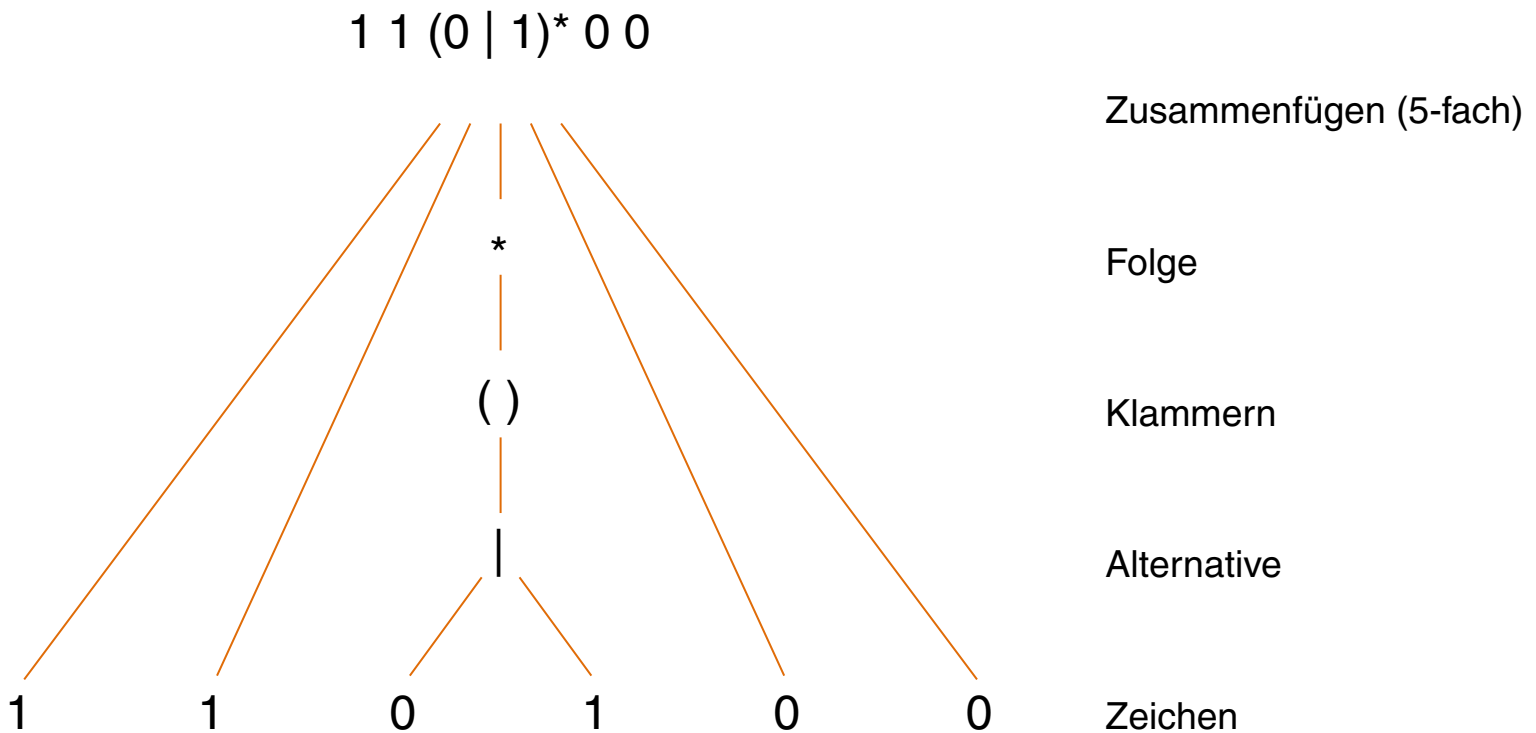
Konstruktion [PHP: reguläre Ausdrücke]

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.	(F)	$(L(F))$	Klammerung
5.	F^+	$\{f_1 f_2 \ldots f_n \mid f_i \in L(F),\ n \geq 1,\ i = 1, \ldots, n\}$	nicht-leere Folge von Worten aus $L(F)$
6.	F^*	$\{\varepsilon\} \ \cup \ L(F^+)$	beliebig lange Folge von Worten aus $L(F)$
7.	F^n	$\{f_1 f_2 \ldots f_n \mid f_i \in L(F),\ i = 1, \ldots, n\}$	Folge von n Worten aus $L(F)$
8.	ε	$\{\varepsilon\}$	das leere Wort

Exkurs: reguläre Ausdrücke [Kastens]

Beispiele



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 [Kastens]

Beispiele (Fortsetzung)

R	**Name der Sprache $L(R)$**	**Worte aus $L(R)$**
(a | b) (c | d | ε)	*Abc*	ac, bc, ad, bd, a, b
Sehr geehrte(r | ε) (Frau | Herr)	*Anrede*	Sehr geehrte Frau

Exkurs: reguläre Ausdrücke [Kastens]

Beispiele (Fortsetzung)

R	**Name der Sprache $L(R)$**	**Worte aus $L(R)$**
(a | b) (c | d | ε)	*Abc*	ac, bc, ad, bd, a, b
Sehr geehrte(r | ε) (Frau | Herr)	*Anrede*	Sehr geehrte Frau
0 | 1 | . . . | 9	*Digit*	7
a | b | . . . | z	*sLetter*	x
A | B | . . . | Z	*cLetter*	B
sLetter | *cLetter*	*Letter*	m, N

Exkurs: reguläre Ausdrücke [Kastens]

Beispiele (Fortsetzung)

R	**Name der Sprache $L(R)$**	**Worte aus $L(R)$**
(a | b) (c | d | ε)	*Abc*	ac, bc, ad, bd, a, b
Sehr geehrte(r | ε) (Frau | Herr)	*Anrede*	Sehr geehrte Frau
0 | 1 | . . . | 9	*Digit*	7
a | b | . . . | z	*sLetter*	x
A | B | . . . | Z	*cLetter*	B
sLetter | *cLetter*	*Letter*	m, N
Letter (*Letter* | *Digit*)*	*Bezeichner*	Maximum, min7, a
Digit $^{+}$.*Digit* ^{2}	*GeldBetrag*	23.95, 0.50
(*cLetter* | *cLetter* ^{2} | *cLetter* ^{3})– (*cLetter* | *cLetter* ^{2})– (*Digit* | *Digit* ^{2} | *Digit* ^{3} | *Digit* ^{4})	*KFZ*	PB–AS–0815
1^{3} (1 | 0)* 0^{3}	*Dual*	1111000, 1111101010000

Exkurs: reguläre Ausdrücke

Beispiele (Fortsetzung)

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

Beispiel: Darstellung aller Dateinamen der Form

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

- Unix-Shell.

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

- PHP.

```
$d = "[0-9]";
preg_match("/webtecI$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.

Kapitel WT:IV

IV. Server-Technologien

- Web-Server
- Common Gateway Interface CGI
- Web-Container und -frameworks
- Web-Template-Engines
- Exkurs: reguläre Ausdrücke
- PHP Hypertext Preprocessor
- PHP Funktionsbibliotheken

PHP Hypertext Preprocessor

Einführung [Einordnung]

Charakteristika:

- wie JSP und Jinja: dokumentenzentrierte (HTML) Programmierung
- prozedurale Sprache mit objektorientierten Erweiterungen
- wenige einfache Typen, dynamisch typisiert
- Notation an C und Perl orientiert
- umfangreiche Funktionsbibliotheken
- Open Source

PHP Hypertext Preprocessor

Einführung [Einordnung]

Charakteristika:

- wie JSP und Jinja: dokumentenzentrierte (HTML) Programmierung
- prozedurale Sprache mit objektorientierten Erweiterungen
- wenige einfache Typen, dynamisch typisiert
- Notation an C und Perl orientiert
- umfangreiche Funktionsbibliotheken
- Open Source

Anwendung:

- Programme, die Server-seitig ausgeführt werden
- kleine private bis große kommerzielle Projekte
- Schwerpunkt auf Datenbanken

PHP Hypertext Preprocessor

Einführung (Fortsetzung)

Einbindung von PHP-Code in HTML:

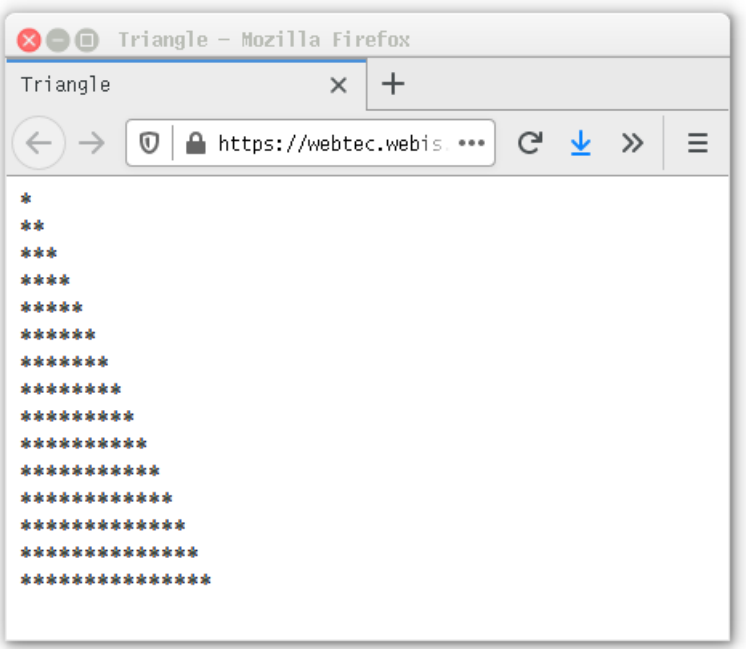
```
<!DOCTYPE html>
<html>
  <head> <title>Triangle</title> </head>
  <body>
    <?php
      $line = 1;
      while ($line < 16) {
        $col = 1;
        while ($col <= $line) {
          echo "*";
          $col = $col + 1;
        }
        echo "<br>\n";
        $line = $line + 1;
      }
    ?>
  </body>
</html>
```

PHP Hypertext Preprocessor

Einführung (Fortsetzung)

Einbindung von PHP-Code in HTML:

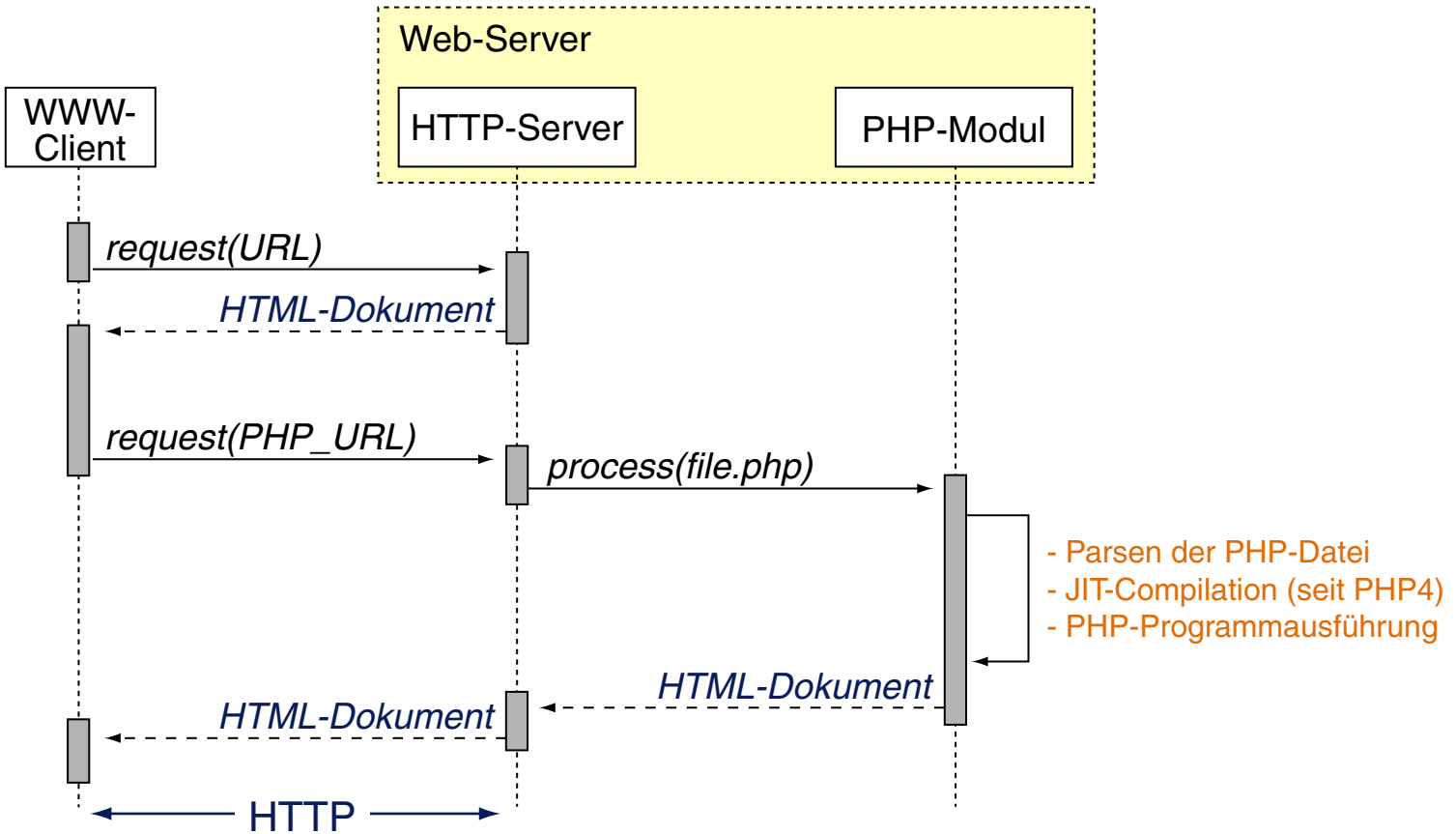
```
<!DOCTYPE html>
<html>
  <head> <title>Triangle</title> </head>
  <body>
    <?php
      $line = 1;
      while ($line < 16) {
        $col = 1;
        while ($col <= $line) {
          echo "*";
          $col = $col + 1;
        }
        echo "<br>\n";
        $line = $line + 1;
      }
    ?>
  </body>
</html>
```



[PHP: Aufruf]

PHP Hypertext Preprocessor

PHP: Sequenzdiagramm Seitenauslieferung [statisch, CGI, Servlet, JSP, PHP]



Bemerkungen:

- PHP kompakt:
  1. Historie
  2. Einbindung in HTML-Dokumente
  3. Grundlagen der Syntax
  4. Variablen
  5. Datentypen
  6. Kontrollstrukturen

PHP Hypertext Preprocessor

Historie [php.net] [Wikipedia]

1994 Rasmus Lerdorf entwickelt erste Version der “Personal Home Page Tools’.

1996 PHP/FI 2. Open Source Gemeinde steigt in die Weiterentwicklung ein (FI = Form Interpreter).

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”.

PHP Hypertext Preprocessor

Historie [php.net] [Wikipedia]

1994 Rasmus Lerdorf entwickelt erste Version der “Personal Home Page Tools’.

1996 PHP/FI 2. Open Source Gemeinde steigt in die Weiterentwicklung ein (FI = Form Interpreter).

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 Technologies durch [Ze]ev Suraski und A[nd]i Gutmanns. [zend.com]

2000 PHP 4. Leistungsfähiger Parser (Zend-Engine), modularer Basiscode, HTTP-Sessions, Ausgabepufferung, sicherere Benutzereingaben, umfangreiche Datenbankunterstützung.

2004 PHP 5. Zend Engine II, neues Objektmodell mit public-/private-/protected-Modifizierern, deutlich verbesserte Unterstützung der XML-Konzepte DOM, SAX, XSLT.

2009 PHP 5.3. Namespaces, Late Static Bindings, Closures und Lambda-Kalkül.

2015 PHP 7. Zend Engine 3, Performance-Verbesserungen, explizite/implizite Typ-Konversion.

2020 PHP 8. Just-In-Time (JIT)-Compilation, striktere Handhabung der Sprachsemantik.

2022 Aktuelle Version von PHP: [php.net]

Statistiken zur Verbreitung: [Server-side: w3techs.com] [Overall: tiobe.com]

Bemerkungen:

- PHP ist ein rekursives Akronym: [[[{ . . . | P }] HP]HP]HP Hypertext Preprocessor

PHP Hypertext Preprocessor

Einbindung in HTML-Dokumente

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

Syntaxalternativen zur Auszeichung:

1. Standard-Tags:

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

2. Sprachspezifische Script-Deklaration:

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

3. Kurzschreibweise einer SGML-Verarbeitungsanweisung:

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

Bemerkungen:

- Die Kurzschreibweise mit „`<?`“ muss in der Konfiguration von PHP aktiviert sein. Aus Sicht der Portabilität von PHP-Dateien ist sie nicht sinnvoll. [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 „Sprachkonstrukt“ [php.net] und wird hinsichtlich der Argumente besonders behandelt.

PHP Hypertext Preprocessor

Grundlagen der Syntax [JavaScript]

Bezeichner [php.net]

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

PHP Hypertext Preprocessor

Grundlagen der Syntax [JavaScript]

Bezeichner [php.net]

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

Anweisungen [php.net]

- Eine 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 [JavaScript]

- Variablen werden durch Initialisierung gleichzeitig definiert und deklariert.
- Eine Variable kann Werte beliebigen Typs annehmen.
- Unterscheidung von lokalen, statischen, globalen und vordefinierten/built-in (inklusive der superglobalen) Variablen. [php.net: statisch, vordefiniert, superglobal]
- Eine Variable ist global, wenn sie außerhalb des Bindungsbereiches einer Funktion steht. Vordefinierte Variablen sind per Default global.
- Im Bindungsbereich einer Funktion sind globale Variablen mit dem Schlüsselwort `global` sichtbar zu machen.

  Sonderfall: superglobale 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 superglobalen Variablen. Konstanten werden durch die Funktion `define()` definiert.

PHP Hypertext Preprocessor

Variablen [JavaScript]

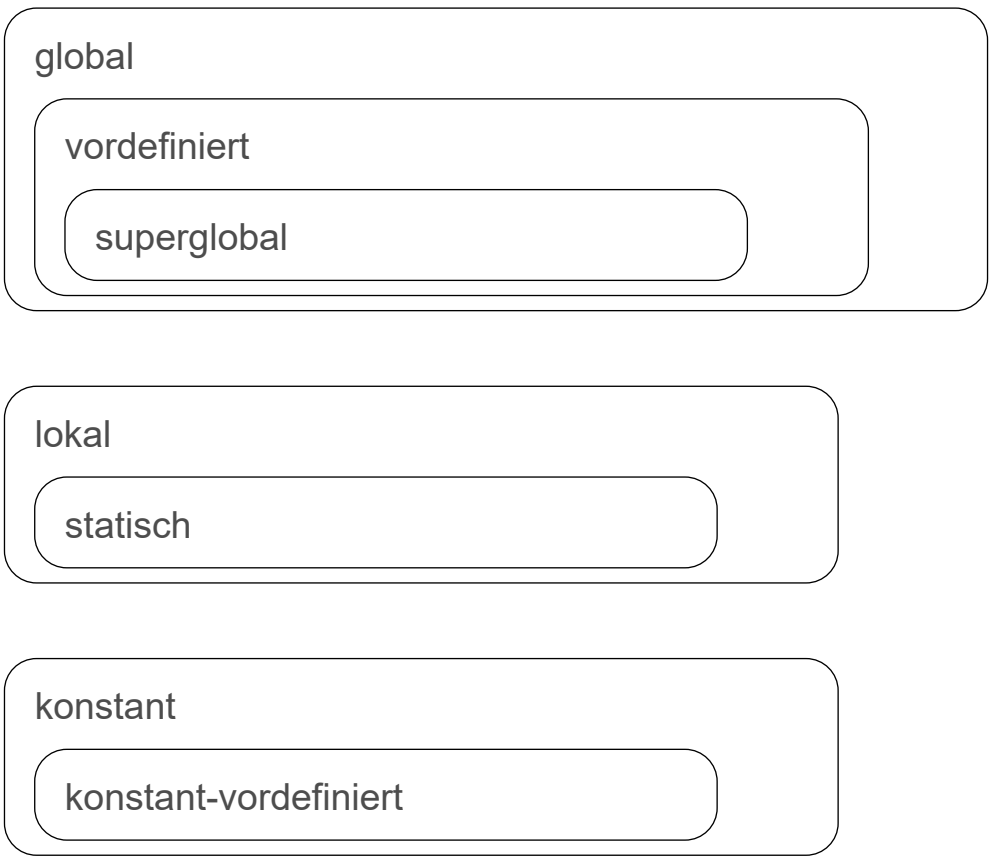
- Variablen werden durch Initialisierung gleichzeitig definiert und deklariert.
- Eine Variable kann Werte beliebigen Typs annehmen.
- Unterscheidung von lokalen, statischen, globalen und vordefinierten/built-in (inklusive der superglobalen) Variablen. [php.net: statisch, vordefiniert, superglobal]

- Eine Variable ist global, wenn sie außerhalb des Bindungsbereiches einer Funktion steht. Vordefinierte Variablen sind per Default global.
- Im Bindungsbereich einer Funktion sind globale Variablen mit dem Schlüsselwort `global` sichtbar zu machen.
  Sonderfall: superglobale 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 superglobalen Variablen. Konstanten werden durch die Funktion `define()` definiert.

PHP Hypertext Preprocessor

Variablen: Illustration von Geltungsbereichen

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

  function test() {
      echo $a;
  }

  test();


  $b = 1;
  $c = 2;

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

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

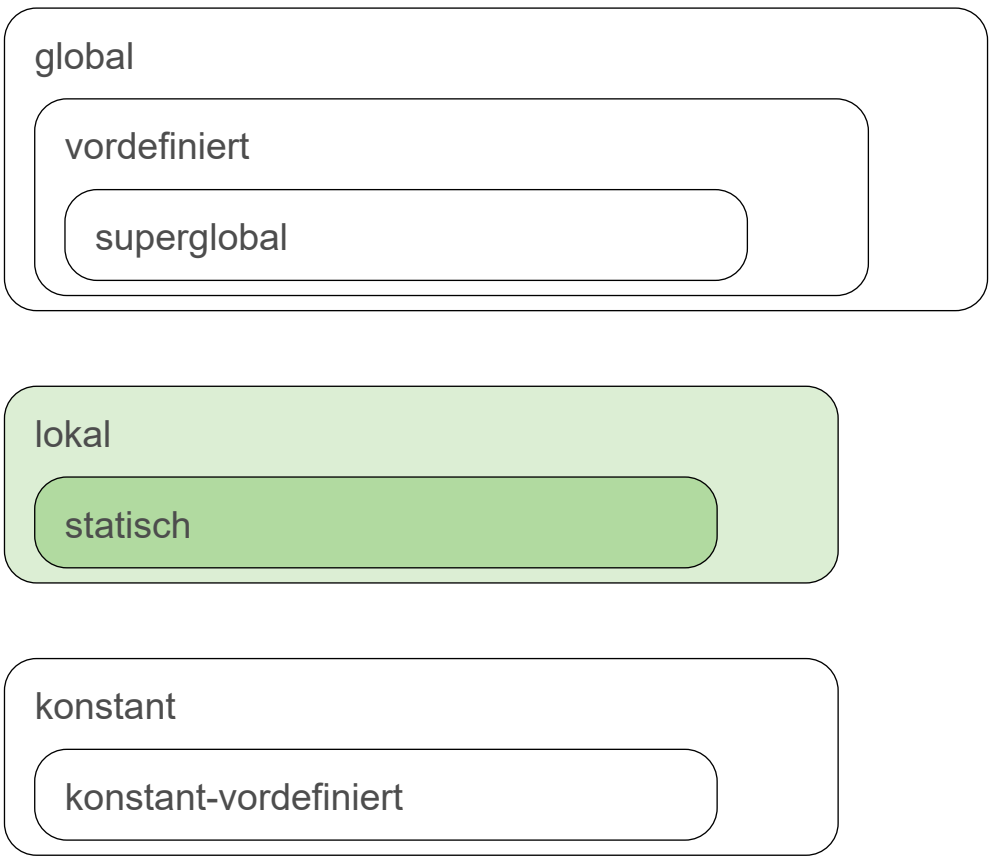


Geltungsbereich

PHP Hypertext Preprocessor

Variablen: Illustration von Geltungsbereichen

```
<?php
  $a = 1;

  function test() {
    echo $a;
  }

  test();


  $b = 1;
  $c = 2;

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

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

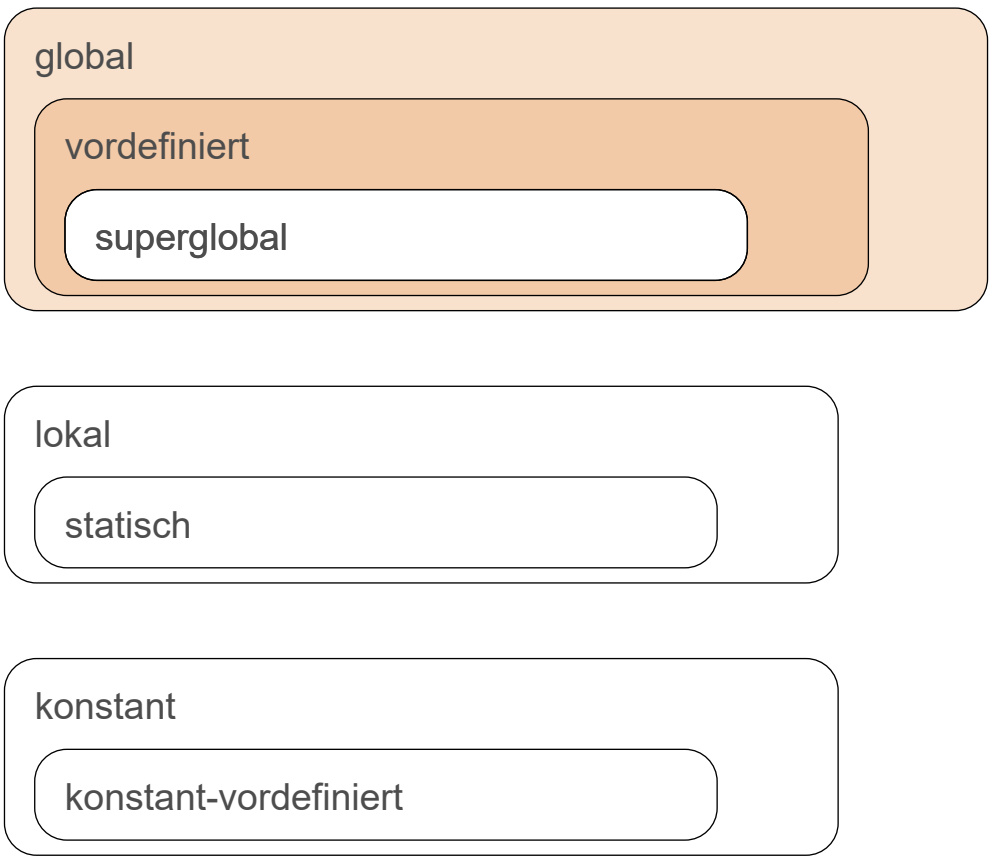


Geltungsbereich

PHP Hypertext Preprocessor

Variablen: Illustration von Geltungsbereichen

```php
<?php
   $a = 1;
   function test() {
       echo $a;
   }
   test();

   $b = 1;
   $c = 2;
   function Summe() {
     global $b, $c;
     $c = $b + $c;
   }
   Summe();
   echo $c;
?>
```

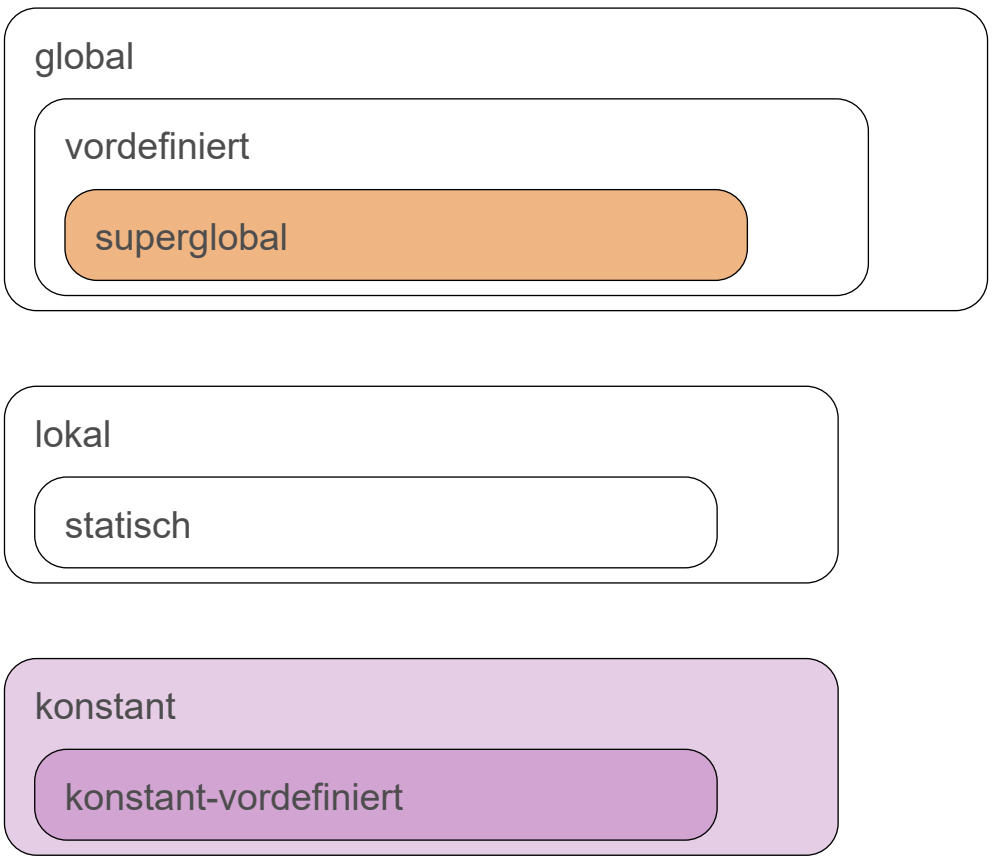


Geltungsbereich

PHP Hypertext Preprocessor

Variablen: Illustration von Geltungsbereichen

```php
<?php
   $a = 1;
   function test() {
      echo $a;
   }
   test();

   $b = 1;
   $c = 2;
   function Summe() {
     global $b, $c;
     $c = $b + $c;
   }
   Summe();
   echo $c;
?>
```



Geltungsbereich

PHP Hypertext Preprocessor

Variablen: Illustration von Geltungsbereichen (Fortsetzung)

```php
<?php
  $a = 1;     // globaler Bereich
  function test() {
    echo $a; // Referenz auf den Bindungsbereich von test()
  }
  test();
?>
```

PHP Hypertext Preprocessor

Variablen: Illustration von Geltungsbereichen (Fortsetzung)

```php
<?php
  $a = 1;     // globaler Bereich
  function test() {
    echo $a; // Referenz auf den Bindungsbereich von test()
  }
  test();
?>
```

```php
<?php
  $b = 1;
  $c = 2;
  function Summe() {
    global $b, $c;
    $c = $b + $c;
  }
  Summe();
  echo $c;
?>
```

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 [php.net]: `$$`*VarName*
  Der Inhalt von `$`*VarName* wird als Variablenname verwendet.

- Referenzen [php.net]: `$`*VarName2* `=` `&$`*VarName1*
  Neuer Variablenname (Alias) `$`*VarName2* für den Inhalt von `$`*VarName1*.

- Überprüfung, ob eine Variable definiert ist:
  `boolean` **`isset`**`($`*VarName*`)`

- Löschen einer Variablen:
  `void` **`unset`**`($`*VarName*`)`

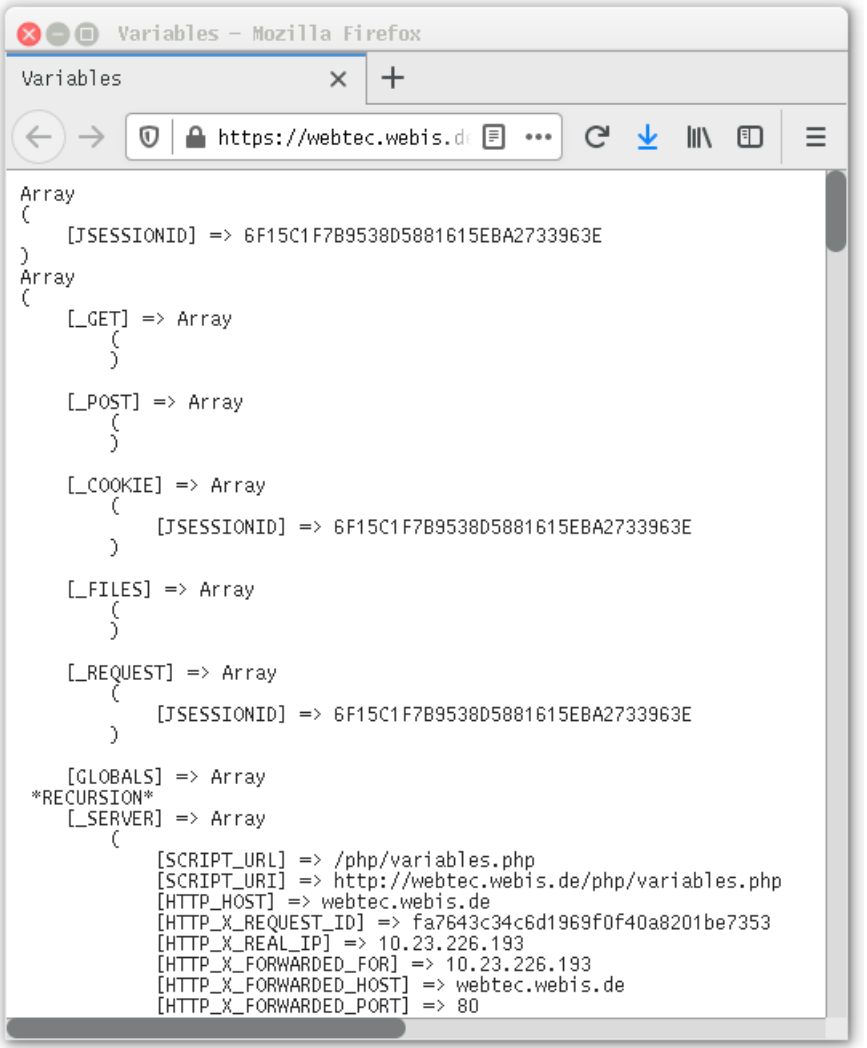
- Zeigt Informationen über eine Variable in lesbarer Form an:
  `boolean` **`print_r`**`($`*VarName*`)`

- Neun superglobale Variablen (vordefinierte Arrays) [php.net]:
  `$GLOBALS, $_SERVER, $_GET, $_POST, $_FILES, $_COOKIE, $_SESSION, $_REQUEST, $_ENV`

PHP Hypertext Preprocessor

Variablen: besondere Konzepte (Fortsetzung)

```
<!DOCTYPE html>
<html>
  <head>
    <title>Variables</title>
  </head>
  <body>
    <pre>
      <script language="php">
        print_r($_REQUEST);
        print_r($GLOBALS);
      </script>
    ...
```



[PHP: php-Datei, Aufruf]

Bemerkungen:

- 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.

  Referenzen in PHP sind vergleichbar mit Referenzen in C++. [php.net]

PHP Hypertext Preprocessor

Datentypen: Primitive [JavaScript]

`integer, float`

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

`string` [php.net]

- Einfache Anführungszeichen. Alle Zeichen stehen für sich selbst; nur »'« muss „escaped“ werden: `\'`
- Doppelte Anführungszeichen. Der String wird geparst 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:

`echo`	akzeptiert durch Kommata getrennte Folge von Ausdrücken [php.net]
`print`	akzeptiert nur einen einzelnen Ausdruck [php.net]

PHP Hypertext Preprocessor

Datentypen: Primitive (Fortsetzung)

`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, falls
  (a) ihr die Konstante `NULL` als Wert zugewiesen wurde,
  (b) ihr bis zum aktuellen Zeitpunkt kein Wert zugewiesen wurde, oder
  (c) sie mit `unset()` gelöscht wurde.

PHP Hypertext Preprocessor

Datentypen: Arrays [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 [php.net]:

(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 [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 [php.net]:

(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 [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 [php.net]:

(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 "Schlüssel-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 [JavaScript]

- Anweisungsfolge bzw. Block:
- Bedingte Anweisung:
- `while`-Schleife:
- `do-while`-Schleife [Cartoon]:
- `for`-Schleife:

PHP Hypertext Preprocessor

Kontrollstrukturen [JavaScript]

- Anweisungsfolge bzw. Block:

```
{ $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 [Cartoon]:

- `for`-Schleife:

PHP Hypertext Preprocessor

Kontrollstrukturen [JavaScript]

- Anweisungsfolge bzw. Block:

```
{ $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 [Cartoon]:

```
$i = 0; do {$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 [JavaScript]

- Anweisungsfolge bzw. Block:

```
{ $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 [Cartoon]:

```
$i = 0; do {$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 (in der Funktionsdefinition) ist eine Variable, die mit dem Wert des aktuellen Parameters (in einem Funktionsaufruf) initialisiert wird.
- Notiert man ein »&« vor dem formalen Parameter (in der Funktionsdefinition) oder vor dem aktuellen Parameter (in einem Funktionsaufruf), geschieht die Übergabe für diesen Parameter durch call-by-reference. [php.net]

Kapitel WT:IV

IV. Server-Technologien

- Web-Server
- Common Gateway Interface CGI
- Web-Container und -frameworks
- Web-Template-Engines
- Exkurs: reguläre Ausdrücke
- PHP Hypertext Preprocessor
- PHP Funktionsbibliotheken

PHP Funktionsbibliotheken

Es existiert eine große Bandbreite von ausgereiften Funktionsbibliotheken, die in jedem PHP-Programm verwendet werden können. Auswahl:

- Authentication
- Bitcoin
- Code Analysis
- (2) Database
- ElasticSearch
- Facebook
- Geolocation
- HTTP
- Image Manipulation
- JSON
- Keyword Extraction
- Logging
- Machine Learning
- NoSQL
- Office
- PDF
- QR Codes
- (1) Regular Expressions
- Search
- Text Processing
- URL Processing
- Virtual Machines
- WordPress
- XML Processing
- Yaml
- Zip

Übersichten im Web: [php.net] [libs.garden] [LibHunt]

PHP Funktionsbibliotheken

(1) Perl Regular Expressions PCRE

PHP verwendet eine Perl-ähnliche Syntax für reguläre Ausdrücke [php.net]:

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

Pattern

- Der Delimiter muss ein nicht-alphanumerisches Zeichen sein.
- Optionale Modifizierer beeinflussen die Match-Strategie. [php.net]
- Funktionen zum Suchen und Ersetzen regulären Ausdrücken [php.net]

PHP Funktionsbibliotheken

(1) Perl Regular Expressions PCRE (Fortsetzung)

PHP verwendet eine Perl-ähnliche Syntax für reguläre Ausdrücke [php.net]:

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

Pattern

- Der Delimiter muss ein nicht-alphanumerisches Zeichen sein.
- Optionale Modifizierer beeinflussen die Match-Strategie. [php.net]
- Funktionen zum Suchen und Ersetzen regulären Ausdrücken [php.net]

Beispiele:

```
echo preg_match("/def/" , "defabcdef" );    ⇝ 1
```

Pattern *String*

```
echo preg_match("=def=", "defabcdef");      ⇝ 1
```

PHP Funktionsbibliotheken

(1) Perl Regular Expressions PCRE (Fortsetzung)

- `int preg_match(string` *Pattern*, `string` *String*

  `[, array &`*Matches* `[, PREG_OFFSET_CAPTURE [, int` *Offset* `]]] )`

  Durchsucht *String* nach der ersten Übereinstimmung mit dem in *Pattern* definierten regulären Ausdruck (und füllt das Array *Matches*). [php.net]

PHP Funktionsbibliotheken

(1) Perl Regular Expressions PCRE (Fortsetzung)

- `int preg_match(string` *Pattern*, `string` *String*

  `[, array &`*Matches* `[, PREG_OFFSET_CAPTURE [, int` *Offset* `]]] )`

  Durchsucht *String* nach der ersten Übereinstimmung mit dem in *Pattern* definierten regulären Ausdruck (und füllt das Array *Matches*). [php.net]

- `int preg_match_all(...)`

  Sucht nach allen Übereinstimmungen. [php.net]

- `mixed preg_replace(...)`

  Suchen und Ersetzen von Übereinstimmungen. [php.net]

- `mixed preg_replace_callback(...)`

  Suchen und Ersetzen von Übereinstimmungen, wobei der Return-Wert einer Callback-Funktion den Ersetzungstext bestimmt. [php.net]

PHP Funktionsbibliotheken

(1) Perl Regular Expressions PCRE (Fortsetzung) [Exkurs: reguläre Ausdrücke]

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 [php.net]	**Erklärung**
1.	a	das Zeichen a
2.	FG	Zusammenfügen von zwei Worten
3.	$F \mid G$	Alternativen
4.	(F)	Klammerung
5.	$F+$	nicht-leere Folge von Worten aus $L(F)$
6.	$F*$	beliebig lange Folge von Worten aus $L(F)$
7.	$F\{n\}$	Folge von n Worten aus $L(F)$
	$F?$	F ist optional (gleiche Semantik wie $F \mid \varepsilon$)
	$F\{m,n\}$	Folge mit mindestens m und höchstens n von Worten aus $L(F)$
	$[abc]$	alternativ ein Zeichen aus der Klammer
	$[\hat{}abc]$	alternativ ein anderes Zeichen als die in der Klammer
	$[a-zA-Z]$	alternativ ein Zeichen aus Zeichenbereichen
	.	beliebiges Zeichen (außer Zeilenumbruch)
	$\hat{}$	Anfang der Zeichenfolge (nichts darf vorangehen)
	$	Ende der Zeichenfolge (nichts darf darauf folgen)

PHP Funktionsbibliotheken

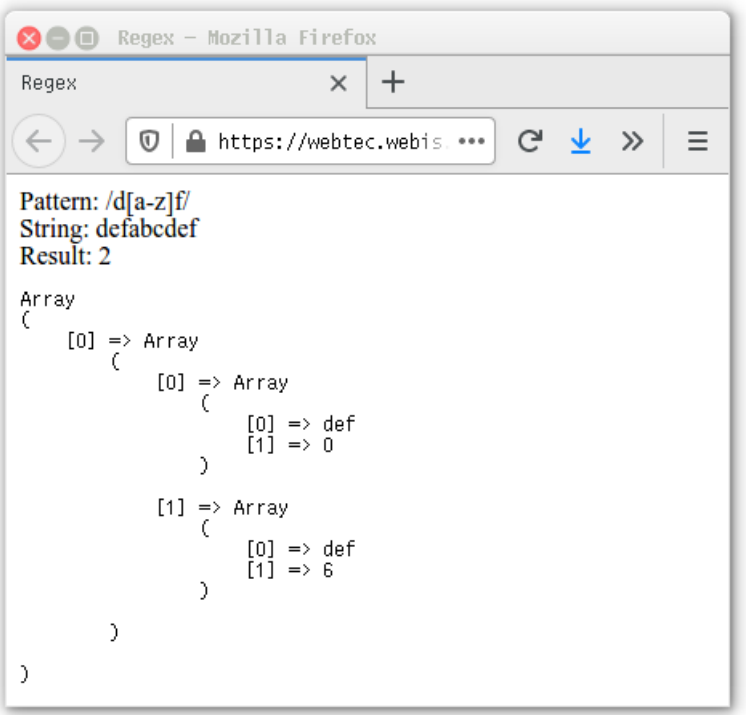
(1) Perl Regular Expressions PCRE (Fortsetzung)

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

  echo "Pattern: ", $pattern, "<br/>";
  echo "String: ", $str, "<br/>";
  echo "Result: ", $num, "<br/>";
?>

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

PHP Funktionsbibliotheken

(1) Perl Regular Expressions PCRE (Fortsetzung)

```php
<?php
  $pattern = "/d[a-z]f/";
  $str = "defabcdef";
  $num = preg_match_all($pattern, $str, $matches, PREG_OFFSET_CAPTURE);
  echo "Pattern: ", $pattern, "<br/>";
  echo "String: ", $str, "<br/>";
  echo "Result: ", $num, "<br/>";
?>
<pre>
  <?php print_r($matches); ?>
</pre>
```



[PHP: php-Datei, Aufruf]

PHP Funktionsbibliotheken

(1) Perl Regular Expressions PCRE (Fortsetzung)

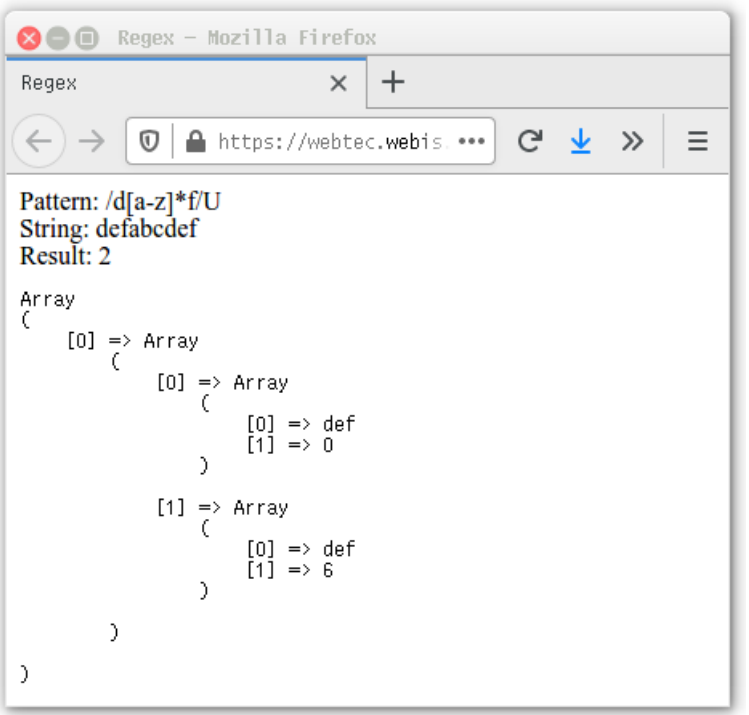
```
<?php
  $pattern = "/d[a-z]*f/";
  $str = "defabcdef";
  $num = preg_match_all($pattern, $str, $matches, PREG_OFFSET_CAPTURE);

  echo "Pattern: ", $pattern, "<br/>";
  echo "String: ", $str, "<br/>";
  echo "Result: ", $num, "<br/>";
?>

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

[PHP: php-Datei, Aufruf]

PHP Funktionsbibliotheken

(1) Perl Regular Expressions PCRE (Fortsetzung)

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

  echo "Pattern: ", $pattern, "<br/>";
  echo "String: ", $str, "<br/>";
  echo "Result: ", $num, "<br/>";
?>

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



[PHP: php-Datei, Aufruf]

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]
- Unter Linux wird mittels „`php -a`“ ein Command-Line-Interpreter (PHP-Shell) gestartet, mit dem sich PHP-Ausdrücke interaktiv evaluieren lassen.
- Bis zur Version PHP 7 stand noch die POSIX-Engine als Alternative für reguläre Ausdrücke zur Verfügung, die jedoch zwei Größenordnungen langsamer ist.

PHP Funktionsbibliotheken

(2) PHP Data Objects PDO

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 Passworten

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 unterstüzt, u.a.: dBase-kompatible Formate, Berkeley-DB-Formate, Oracle, Sybase, MySQL, ODBC-Datenquellen. [php.net]

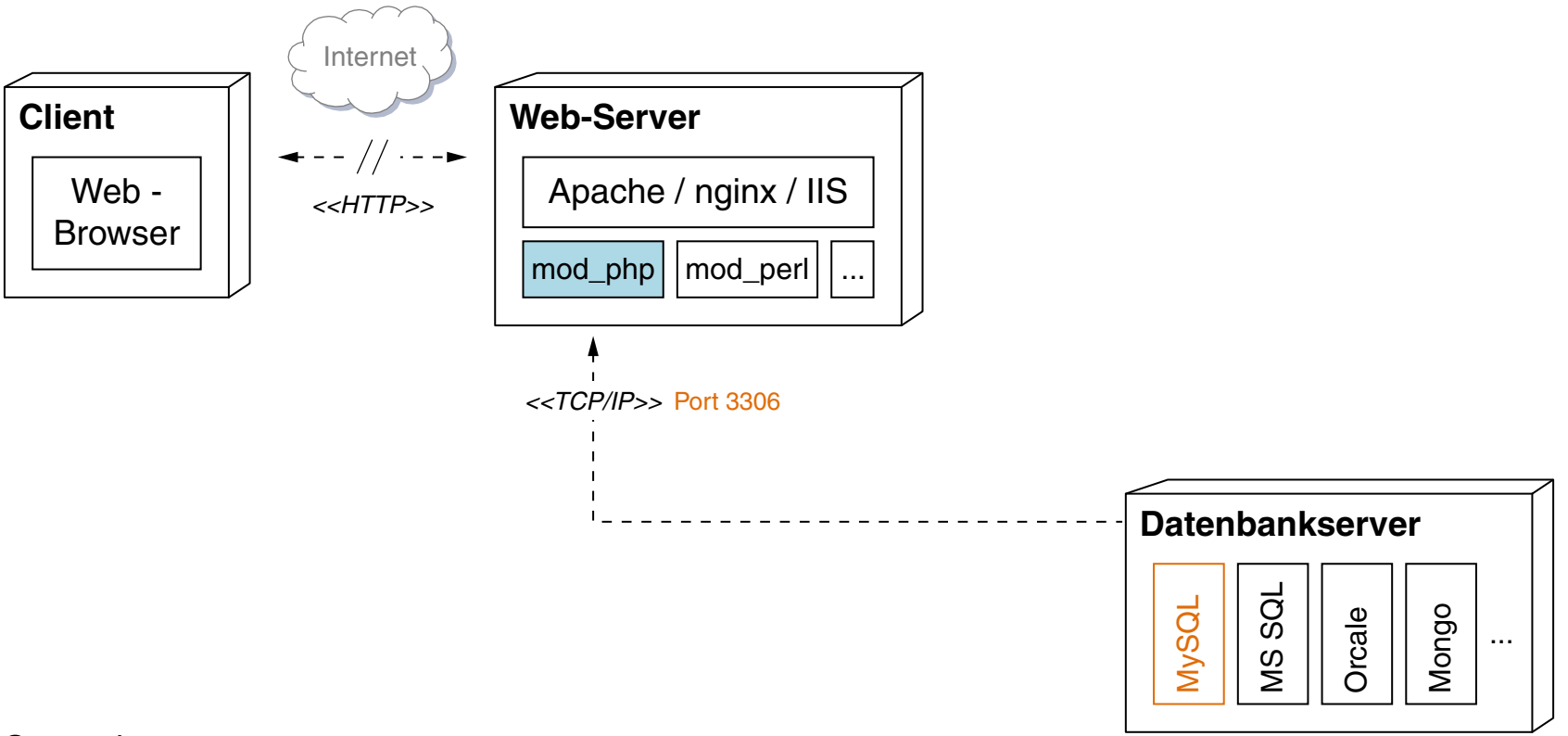
Bemerkungen:

- PHP Data Objects ist eine abstrakte Schnittstelle für verschiedene Datenbanken. [php.net]
- Eine weit verbreitete Software-Kombination zur Erstellung Web-basierter Datenbankanwendungen ist unter dem Schlagwort XAMPP (früher: LAMP bzw. WAMP) bekannt:

  Linux / Mac OS / Windows + Apache + MariaDB (früher: MySQL) + PHP + Perl

PHP Funktionsbibliotheken

(2) PHP Data Objects PDO (Fortsetzung)

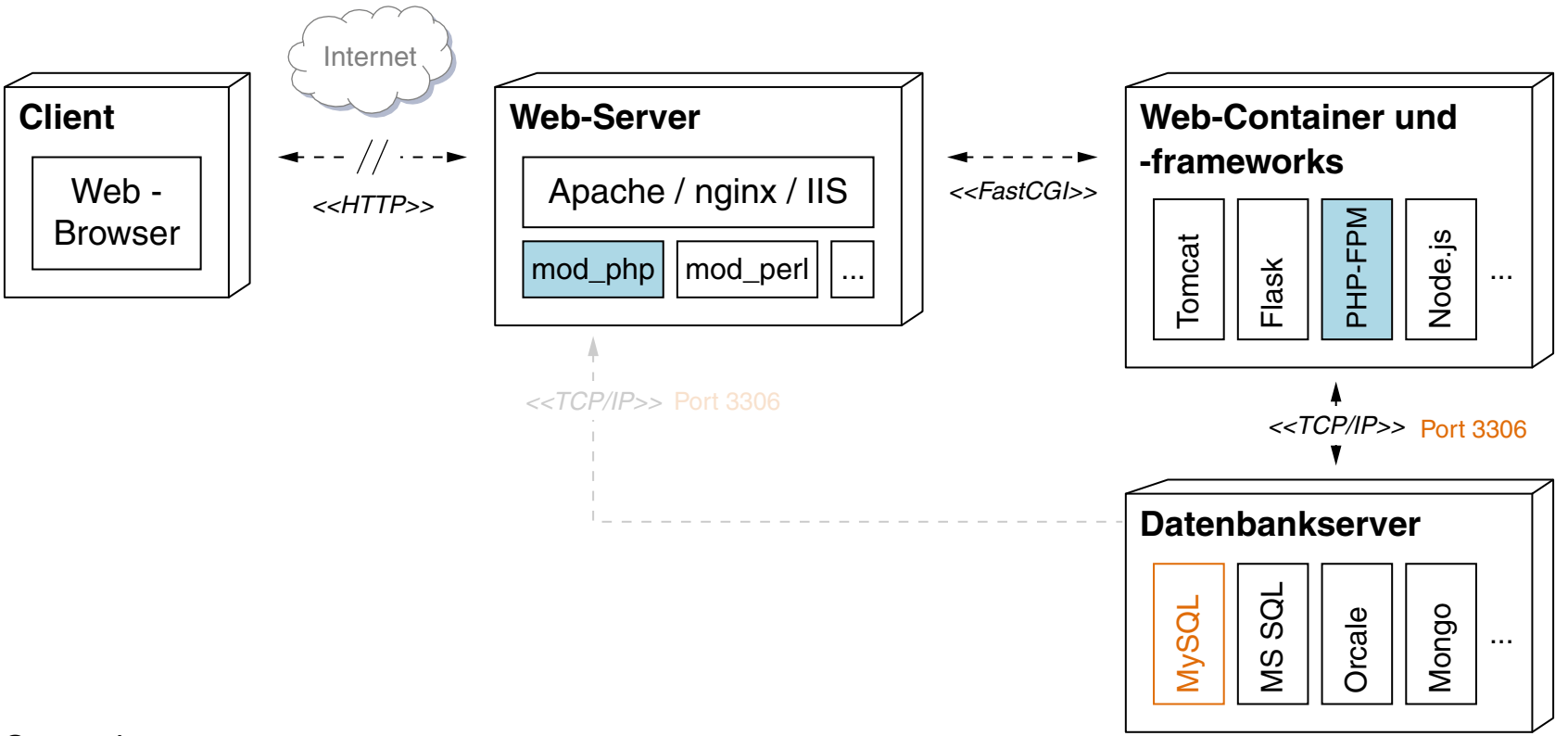


Szenario:

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

PHP Funktionsbibliotheken

(2) PHP Data Objects PDO (Fortsetzung)



Szenario:

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

PHP Funktionsbibliotheken

(2) PHP Data Objects PDO (Fortsetzung)

1. Mit Datenbank-Server verbinden und Datenbank auswählen:

```
$dbhost = "localhost";
$dbname = "customers";
$dbuser ="webtec";
$dbpassword = "secret";
try {
    $link = new PDO("mysql:host=" . $dbhost . ";dbname=" . $dbname,
        $dbuser, $dbpassword);
} catch (PDOException $exception) {
    die("Could not connect: " . $exception->getMessage());
}
```

PHP Funktionsbibliotheken

(2) PHP Data Objects PDO (Fortsetzung)

1. Mit Datenbank-Server verbinden und Datenbank auswählen:

```
$dbhost = "localhost";
$dbname = "customers";
$dbuser ="webtec";
$dbpassword = "secret";
try {
    $link = new PDO("mysql:host=" . $dbhost . ";dbname=" . $dbname,
        $dbuser, $dbpassword);
} catch (PDOException $exception) {
    die("Could not connect: " . $exception->getMessage());
}
```

2. 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 = $link->query($query)
    or die("Query failed: " . $link->errorInfo()[2]);
```

PHP Funktionsbibliotheken

(2) PHP Data Objects PDO (Fortsetzung)

3. Anzahl von Ergebniszeilen prüfen:

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

4. Datensätze aus assoziativem Array auslesen:

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

5. Speicher freigeben:

```
$link = null;
```

Bemerkungen:

- Eine persistente Verbindung zur Datenbank wird aufgebaut, wenn im Konstruktor von `PDO` als zusätzlicher Parameter `array(PDO::ATTR_PERSISTENT => true)` übergeben wird.

- `die(string` *Message*`)`

  Ausgabe von *Message* und Beendung des aktuellen Skripts.

- `$link->errorInfo()`

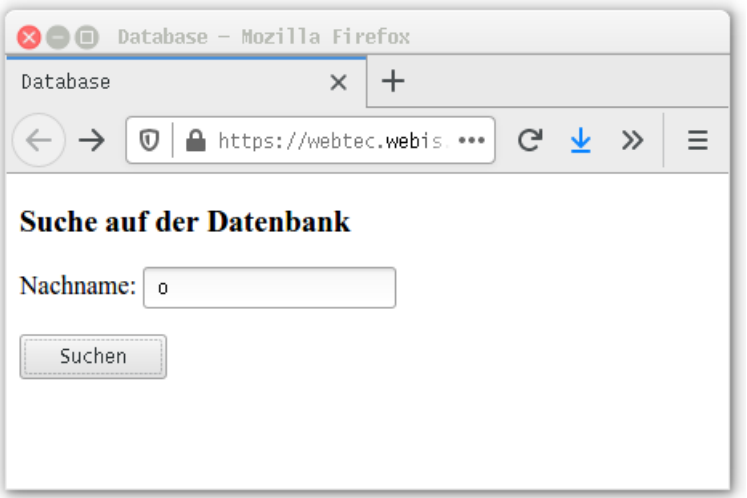
  Rückgabe eines Arrays mit Fehlerinformationen zum letzten Methodenaufruf von `$link`. Eintrag 1 enthält den SQL Fehlercode, Eintrag 2 einen treiberspezifischen Fehlercode und Eintrag 3 eine textuelle Fehlerbeschreibung.

- `foreach ($result as $row)`

  Iteriert über die Datensätze (= Zeilen, Tupel) aus `$result` mit den assoziativen Arrays `$row`.

PHP Funktionsbibliotheken

(2) PHP Data Objects PDO (Fortsetzung)



[PHP: Aufruf]

Server-Technologien

Quellen zum Nachlernen und Nachschlagen im Web

- Enseleit. *SELFPHP.*
  www.selfphp.info
- Kastens. *Einführung in Web-bezogene Sprachen.*
  Universität Paderborn.
- PHP-Dokumentationsgruppe. *PHP Documentation.*
  php.net/docs.php
- Vaswani. *PHP 101: PHP For the Absolute Beginner.*
  ss23.github.io/php-tutorial
- W3 Schools. *PHP Tutorial.*
  www.w3schools.com/php